გია სურგულაძე

ვიზუალური პროგრამება C# 2010

ენის ბაზაზე

„ტექნიკური უნივერსიტეტი"

საქართველოს ტექნიკური უნივერსიტეტი

გია სურგულაძე

ვიზუალური დაპროგრამება
C#_2010 ენის ბაზაზე

თბილისი
2011

უაკ 681.3.06

წიგნში გადმოცემულია მაიკროსოფთის ფირმის ახალი პროგრამული პლატფორმის ბაზაზე შექმნილი Ms Visual Studio .NET Framework 4.0/4.5 პაკეტის C# დაპროგრამების ენის ვიზუალური ინსტრუმენტული საშუალებანი.

წარმოდგენილია C#_2010/2011 Expression და Professional ვერსიებში არსებული მართვის ვიზუალური ელემენტები და კომპონენტები, ობიექტ-ორიენტირებული მიდგომით პროგრამული სისტემების რეალიზაციის ასპექტები. ინკაფსულაციის, პოლიმორფიზმის, მემკვიდრეობითობის და აბსტრაქციის თეორიული საფუძვლები და პრაქტიკული გამოყენების ამოცანების გადაწყვეტა. განიხილება C#_2010 ენის ვიზუალური კონსტრუქციები ADO.NET, MsAccess, MySQL და Ms SQL Server მონაცემთა ბაზებთან სამუშაოდ, აგრეთვე ვებ-აპლიკაციების აგების საშუალებები ASP.NET და XML. მოცემულია გრაფიკულ რეჟიმში მუშაობის (GDI+ და WPF) მეთოდები და ანიმაციური პროგრამების აგების ვიზუალური საშუალებები. თეორიულ საკითხებთან ერთად შემოთავაზებულია პრაქტიკულ და ლაბორატორიულ ამოცანათა ციკლი C# ენის ასათვისებლად ვინდოუს- და ვებ-აპლიკაციების პროექტების აგების მიზნით.

განკუთვნილია მართვის საინფორმაციო სისტემების (Management Information Systems) და პროგრამული ინჟინერიის (Software Engineering) სპეციალობის სტუდენტების, მაგისტრანტებისა და დოქტორანტები-სათვის, აგრეთვე სხვა სფეროს სპეციალისტებისთვის, რომელთაც სურვილი აქვთ შეისწავლონ დაპროგრამების ვიზუალური მეთოდები და ინსტრუმენტული საშუალებანი C# ენის ბაზაზე .NET გარემოში.

რეცენზენტები:

- აკად.დოქტორი, ასოც. პროფ. **ლ. ბეჟანიშვილი,**
- ტ.მ.კ., ასოც.პროფ. **ე. თურქია**

ISBN 978-9941- 20-021-2

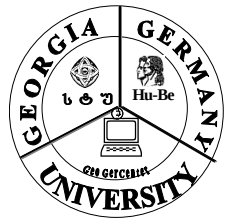
Georgian Technical University

GIA SURGULADZE

VISUAL PROGRAMMING

with C# 2010

Supported by DAAD
(Germany)



© Publication House "Technical University", Tbilisi, 2011
ISBN 978-9941- 20-021-2

ABSTRACT

This book sets out Visual modeling tools of C# programming language as part of Ms Visual Studio .NET Framework 4.0/4.5 package, developed based on a new program platform from Microsoft Corp.

The book presents visual elements and components of Control existent in the C#_2010/2011 Expression and Professional versions as well as the aspects of implementation of software systems using object-oriented approach. Also, theoretical basics and practical application problem solving are present for Incapsulation, Polymorphism, Inheritance and Abstraction.

The book discusses C#_2010 language visual designs to work with databases of ADO.NET, MsAccess, MySQL and Ms SQL Server. Web application design means ASP.NET and XML are also discussed. Methods of working under graphical regime (GDI+ and WPF) and visual means of creating animatio programs are set out.

Together with theoretical topics, practical and workshop problem cycle are present for learning C# language with the goal of designing Windows and web application projects.

The book is to primarily serve Management Information Systems (MIS) and Software Engineering (SA) students, for those doing Master or Doctorate studies. It will also be helpful for readers of other professions willing to study visual methods of programming and instrumental means based on C# language in .NET environment.

ვუძღვნი სტუ-ს ”მართვის ავტომატიზებული სისტემების” კათედრის დაარსების 41-ე წლისთავს და მისი დამაარსებლების, ჩემი მასწავლებლების, აკადემიკოს გოჩა ჩოგოვაძის და პროფესორ გიორგი გოგიჩაიშვილის დაბადების 70-ე წლისთავის იუბილეს

ავტორის წინასიტყვაობა

თანამედროვე საინფორმაციო ტექნოლოგიებში ვიზუალური დაპროგრამება ობიექტ-ორიენტირებული მოდელირების პროგრამული რეალიზაციის ერთ-ერთი აქტუალური და მუდმივად განახლებადი თეორიულ-პრაქტიკული მიმართულებაა. მისი მიზანია დიდი და რთული განაწილებული პროგრამული სისტემების დეველოპინგი უნიფიცირებული (სტანდარტული) ენებით (Software Engineering with UML).

მართვის კომპიუტერული სისტემების პროგრამული უზრუნველყოფის აგების პროცესების ასეთი სრულფასოვანი ავტომატიზაცია ვიზუალური მოდელირების სახელწოდებით დამკვიდრდა. იგი მოდელების გრაფიკულ წარმოდგენას ეყრდნობა და ფლობს მოქნილ რევერსულ ტექნოლოგიას (პროგრამული კოდი <---> დიაგრამები).

„მაიკროსოფთი” და სხვა ფირმები დღესაც განსაკუთრებით ზრუნავენ მომხმარებლისთვის ახალი ინტერფეისული ინსტრუმენტების შესათავაზებლად, ამიტომაც ბოლო წლებში კვლავაც გამოჩნდა მათი პროდუქტები .NET-პლატფორმის ტექნოლოგიის სახით, რომელშიც ახალი კონცეფცია - დაპროგრამების ენებისადმი დამოუკიდებლობის იდეაა ჩადებული.

განსაკუთრებით საინტერესოა C#.NET, VB.NET, F#.NET (2010 წლის ვერსიიდან), ADO.NET, ASP.NET და სხვა ინსტრუმენტები, როგორც ვინდოუსის, ისე ვებ-აპლიკაციების დასაპროგრამებლად. მნიშვნელოვანია, აგრეთვე, ამ ენებზე მომხმარებელთა ინტერფეისების აგების ამოცანათა გადაწყვეტა მონაცემთა დიდ ბაზებთან მუშაობისას (MsAccess, MySQL,SQL Server, Oracle და სხვ.).

ექსპერიმენტული კვლევის გარკვეული ნაწილი შესრულდა ნიურნბერგ-ერლანგენისა და ბერლინის ჰუმბოლდტის უნივერსიტეტების „მონაცემთა ბაზებისა და მენეჯმენტის“ და „დაპროგრამების ტექნოლოგიების” კათედრებზე, ჩემი სამეცნიერო მივლინებების დროს გერმანიაში 2007/2008 და 2010/11 წლებში.

მადლიერების გრძნობით მინდა მოვიხსენიო ჩვენი კათედრის აღზრდილები, ამჟამად კოლეგები, ახალგაზრდა მეცნიერ-პედაგოგები ე. თურქია, ნ. თოფურია და ი. ბულია, რომლებიც განსაკუთრებულ და მაღალპროფესიულ დახმარებას მიწევდნენ თანამედროვე Windows- და Web-დაპროგრამების და მონაცემთა ბაზების კომპლექსური გამოყენების ამოცანების პრაქტიკულ გადაწყვეტაში.

ამ წიგნის გამოცემის იდეის განხორციელებაში გარკვეული წვლილი მიუძღვით „მართვის ავტომატიზებული სისტემების“ სპეციალობის სტუდენტებს, რომელთა ჯგუფებშიც ვკითხულობდი ლექციებს ამ საგანში და რომელთა თხოვნითაც დაიწერა ეს ნაშრომი.

„მას”-კათედრაზე ჩემი მოღვაწეობის 40 წლის მანძილზე შესამჩნევად შეიცვალა ცხოვრების, ლექტორებისა და ახალგაზრდობის მოთხოვნილება ინფორმაციული ტექნოლოგიების სფეროში. კომპიუტერული ტექნიკისა და პროგრამული ტექნოლოგიების სწრაფმა განვითარებამ, მათ შორის ინტერნეტის გაბატონებამ მნიშვნელოვნად შეცვალა (ან ცვლის) ადამიანის მენტალიტეტი (საით, ეს სადისკუსიო თემაა !).

მახსენდება 1974-75 წლები, როდესაც მაგდებურგის უნივერსიტეტის მეცნიერ-სტაჟიორი, ორი თვე, თებერვალსა და მარტში თითქმის ყოველდღე მატარებლით მივემგზავრებოდი ლაიფციგის დიდ ბიბლიოთეკაში („Große Bücherei”), რათა საფუძვლიანად, ღრმად გამერჩია შემთხვევით მოძიებული ნაშრომები, პროფესორ ედგარ კოდის პირველი სტატიები „რელაციური მოდელებისა და ბაზების“ შესახებ (ამ პერიოდში არ არსებობდა ასლის გადამღები აპარატურა და ინტერნეტი). ეს სტატიები დაედო საფუძვლად დღეისათვის ფუნქციონირებად მონაცემთა ბაზების მართვის სისტემებს: Oracle, SQL Server, MySQL, MsAccess, Paradigm და სხვ. ე. კოდი ამერიკის საერთაშორისო ასოციაციამ დააჯილდოვა „ტიურინგის” პრემიით (ეს ნობელის პრემიის რანგის ჯილდოა კომპიუტერულ ტექნოლოგიებში).

1976 წელს მომზადდა ჩემი სადისერტაციო ნაშრომის თეორიული ნაწილი „მონაცემთა რელაციური ბაზების სტრუქტურების ავტომატიზებული დაპროექტება”, რომელიც რუსმა „მეგობარ-მეცნიერებმა დროებით დაბლოკეს“ (ასეთ საკითხებზე მუშაობა რუსეთში ჯერ კიდევ არ იყო დაწყებული).

მხოლოდ 1980 წელს, ჩემი სამეცნიერო ხელმძღვანელის, ბატონი გოჩა ჩოგოვაძის ხელშეწყობით მოხერხდა, რომ სანკტ-პეტერბურგის (ლენინგრადის) ელექტროტექნიკის უნივერსიტეტის სადისერტაციო საბჭოს სხდომაზე, აკადემიკოს ა. ვავილოვის თავმჯდომარეობით (ლენინგრადის უნივერსიტეტების რექტორთა საბჭოს თავმჯდომარე და გამოჩენილი მეცნიერი, „საწარმოო პროცესების ავტომატიზაციის“ კათედრის გამგე), მოიწონეს დისერტაცია და მომანიჭეს ტექნიკის მეცნიერებათა კანდიდატის სამეცნიერო ხარისხი (რელაციური ბაზების პრაქტიკული რეალიზაციის მიმართულებაში ეს იყო ერთ-ერთი პირველი დისერტაცია მაშინდელი „საბჭოთა კავშირის“ სივრცეში).

საერთაშორისო, ასევე განსაკუთრებით მოსკოვის უნივერსიტეტე-ბის ინიციატივით და ჩვენი კათედრის აქტიური მონაწილეობით, ხშირად იმართებოდა სამეცნიერო-ტექნიკური კონფერენციები და სკოლა-სემინარები (თბილისში, სოხუმში (1977), თელავში (1978), ბაკურიანსა (1981,83) და ა.შ.), რომელთა თემა იყო „მონაცემთა ინტელექტუალური ბანკები და სისტემები” (საფუძველი იყო რელაციური ბაზების კონცეფცია, ლოგიკური და სიტუაციური მართვის თეორია).

ჩემთან ერთად კათედრაზე ამ მიმართულებით მუშაბდნენ გ. ჩაჩანიძე, ვ. ქაჩიბაია (შემდეგ დრეზდენის უნივერსიტეტის ასპირანტი), გ. ღარიბაშვილი. ახალი თაობიდან - ე. თურქია (ბაიროითის უნივერსიტეტის მეცნიერ-სტაჟიორი), დ. გულუა (ბერლინის ჰუმბოლდტის უნივერსიტეტის მეცნიერ-სტაჟიორი, შემდეგ ამავე უნივერსიტეტის სერვერების მენეჯერი), ლ. პეტრიაშვილი და გ. ედიბერიძე (ერლანგენის უნივერსიტეტის მეცნიერ-სტაჟიორები), ნ. თოფურია (ობიექტ-როლური მოდელირება), მ. კაშიბაძე (ორგანიზაციული სისტემების ინფორმაციული რესურსების მართვა), მ. ოხანაშვილი (მარკეტინგული ბიზნეს პროცესების მონაცემთა ბაზები და იმიტაციური მოდელირება) და სხვ.

1981 წელს სტუდენტთა საკავშირო სამეცნიერო ნაშრომების კონკურსზე წარდგენილი იყო ჩვენი კათედრის სტუდენტების, ვ. რეტერის და ა. ჩიხლაძის ნაშრომი „რელაციური ბაზების სტრუქტურების დაპროექტების შემდგომი სრულყოფა” (ხელმძღვანელი გ. სურგულაძე). პირველად, კათედრის ისტორიაში, ეს ნაშრომი „ოქროს მედლით” დაჯილდოვდა.

მონაცემთა რელაციური ბაზების მიმართულების განვითარება, როგორც ეს ე. კოდის, ჯ. მარტინის, დ. დეიტის, ჯ. ულმანის და სხვა ცნობილ ამერიკელ, გერმანელ, რუს მეცნიერთა მიერ შექმნილ წიგნებში აისახებოდა, ხელს უწყობდა ჩვენს ქვეყანაში ახალგაზრდა მეცნიერთა კვალიფიკაციის დონის ამაღლებას. დღეს, ჩვენი კათედრის მრავალი კურსდამთავრებული საზღვარგარეთ მოღვაწეობს. მაგალითად, ვ. ქაჩიბაია კანადაში, ტორონტოს ერთ-ერთი უდიდესი საფინანსო ბანკის მონაცემთა ბაზის ადმინისტრატორია.

პირველი ქართულენოვანი წიგნები მონაცემთა რელაციური ბაზების შესახებ პროფ. გ. ჩოგოვაძის, გ. სურგულაძის და ვ. ქაჩიბაიას კალამს ეკუთვნის (1985-1990). ასევე მნიშვნელოვანი იყო ფუნდამენტური სახელმძღვანელო „მონაცემთა და ცოდნის ბაზების აგების საფუძვლები” (გ.ჩოგოვაძე, გ.სურგულაძე, ო.შონია-1996). ამ წიგნებით აღიზარდა სტუდენტების მრავალი თაობა.

90-იანი წლებიდან საინფორმაციო ტექნოლოგიების ბაზარზე ჩნდება მოთხოვნილება C/C++/Java დაპროგრამების ენებზე, რასაც განსაკუთრებული მნიშვნელობა ჰქონდა ქსელური სისტემებისა და ინტერნეტის განვითარების სფეროში. პირველი წიგნი ქართულ ენაზე „დაპროგრამების მეთოდები C/C++ ენების ბაზაზე” ჩვენ კათედრაზე გამოიცა (გ. გოგიჩაშვილი, გ. სურგულაძე, ო.შონია - 1997).

2001 წლიდან „მას” კათედრის სამეცნიერო და სასწავლო პროცესში ინერგება და ვითარდება უნიფიცირებული მოდელირების ენის UML-ტექნოლოგია, რომელიც დღეს პროგრამული ინჟინერიის თეორიული საფუძველია. ამ მიმართულებით, როგორც ბიზნეს-პროცესების ანალიტიკოსმა მსოფლიო ბანკის და USAID-ის არაერთ სამეცნიერო პროექტში მივიღე მონაწილეობა, რაც აისახა ჩემს სამეცნიერო ნაშრომებში.

წინამდებარე სახელმძღვანელოც, „ვიზუალური დაპროგრამება C#-2010 ენაზე” პირველი ქართულენოვანი წიგნია, რომელშიც ყურადღება გამახვილებულია ვიზუალურ-კომპონენტურ პროგრამირების ელემენტებზე და რელაციურ მონაცემთა ბაზების ერთობლივ გამოყენებაზე. ვიმედოვნებ, რომ წიგნი სასარგებლო და საინტერესო იქნება ჩვენი სტუდენტების და მკითხველებისთვის, რომლებიც ეუფლებიან უახლეს საინფორმაციო ტექნოლოგიებს.

პროფ. გია სურგულაძე
თბილისი 25.12.2011

ს ა რ ჩ ე ვ ი

შესავალი:

.NET ტექნოლოგიის არსი და ობიექტ-ორიენტირებული დაპროგრამების მეთოდის მოკლე მიმოხილვა

საინფორმაციო ტექნოლოგიები უახლესი კომპიუტერული ინდუსტრიის ბაზაზე განვითარების მაღალი ტემპებით ხასიათდება. მსოფლიო ბაზარზე გამოჩნდა არაერთი ახალი აპარატურული (Hardware) და პროგრამული (Software, Groupware) სისტემები. სულ უფრო ფართოვდება კომპიუტერული სისტემების მომხმარებელთა წრე, რაც აქტუალურს ხდის ამ სფეროში მობილური და მეგობრული ინტერფეისების და მეთოდური საშუალებების შექმნას.

წინამდებარე სახელმძღვანელოში გადმოცემულია მაიკროსოფთის ფირმის პროგრამული პროდუქტი C#.NET 2010, მისი ფუნქციურობის აღწერა და ამ პლატფორმის ბაზაზე Windows- და Web-აპლიკაციების აგების ვიზუალური ინსტრუმენტული საშუალებანი. შემოთავაზებულია C#.NET ენის ინტეგრირებული გარემო და მის საფუძველზე ASP.NET და ADO.NET პროგრამული კოდების მართვა.

შესავალში მოცემულია .NET პლატფორმის ზოგადი კონცეფცია და მუშაობის პრინციპები MsVisual Studio.NET Framework 4.0 გარემოში (2010). აგრეთვე ობიექტ-ორიენტირებული დაპროგრამების მეთოდის ძირითადი ცნებები და პრინციპები.

პირველ ნაწილში, რომელიც ათი თავისგან შედგება, ვრცლადაა გადმოცემული C#.NET ენის საფუძვლები და ვიზუალური დაპროგრამების მართვის ელემენტები. განიხილება ობიექტ-ორიენტირებული დაპროგრამების მეთოდის ძირითადი კომპონენტების (კლასები და ობიექტები) და პრინციპების (ინკაფსულაცია, მემკვიდრეობითობა, პოლიმორფიზმი, აბსტრაქცია) პროგრამული რეალიზაციის საშუალებანი.

მეორე ნაწილი ეხება C# ენის კავშირს მონაცემთა ბაზების მართვის სისტემებთან, კერძოდ ADO.NET დრაივერის ობიექტების, კოდების და მისი ფუნქციების აღწერას, Ms Access, SQL Server და სხვ. მონაცემთა ბაზებთან მუშაობის პრინციპების

აღწერას. C# ენის საფუძველზე ინტერფეისების დაპროექტებას და შესაბამისი საილუსტრაციო ამოცანების განხილვას, მათ შორის XML ენის ელემენტებთან კავშირშიც.

მესამე ნაწილში აღწერილია C# ენის ვიზუალური კომპონენტები ვებ-აპლიკაციების ასაგებად ASP.NET პაკეტის საფუძველზე. შემოთავაზებულია პრაქტიკული მაგალითები aspx და C# კოდირებისათვის Visual Studio .NET გარემოში.

მეოთხე ნაწილი ეძღვნება C# ენის ვიზუალური ელემენტების გამოყენების საკითხებს გრაფიკული რეჟიმისათვის. მოცემულია საილუსტრაციო მასალა ანიმაციური დაპროგრამების ამოცანების გადაწყვეტის პრინციპების ახსნის მიზნით. განიხილება WPF ტექნოლოგიის (Windows Presentation Fondation) საწყისები მომხმარებელთა ინტერფეისების ასაგებად, 2D და 3D განზომილებიანი გრაფიკების წარმოსადგენად და ა.შ.

წიგნი განკუთვნილია მართვის საინფორმაციო სისტემების (Management Information Systems) და პროგრამული ინჟინერიის (Software Engineering) სპეციალობის სტუდენტების, მაგისტრანტებისა და დოქტორანტებისთვის. აგრეთვე სხვა სფეროს სპეციალისტებისთვის, რომელთაც სურვილი აქვთ შეისწავლონ დაპროგრამების ვიზუალური მეთოდები და ინსტრუმენტული საშუალებანი Ms VisualStudio.NET Framework 4.0 (2010) ვერსიის C# ენის ბაზაზე.

➢ დაპროგრამების თანამედროვე პლატფორმები და ენები

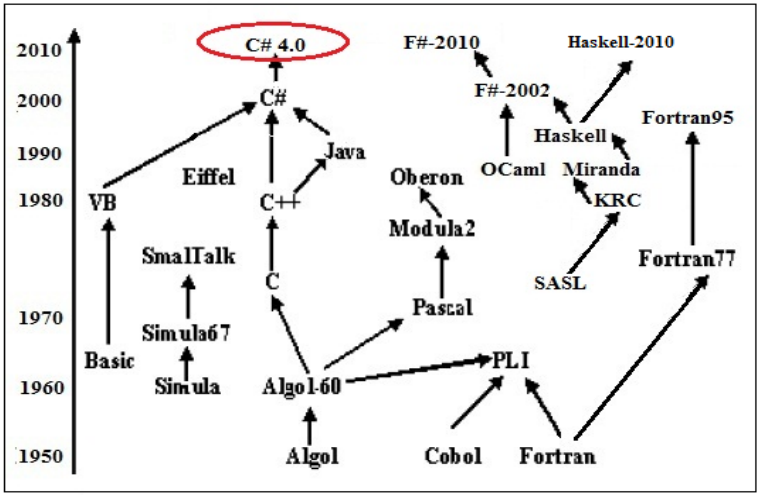
დაპროგრამების პლატფორმა და ენა სხვადასხვა ცნებაა. პლატფორმა მიეკუთვნება კომპიუტერის აპარატულ-პროგრამულ ტექნოლოგიას.

კომპიუტერული პლატფორმა განისაზღვრება მისი პროცესორის და შინების აპარატურული გადაწყვეტით, რომელიც პირდაპირ კავშირშია მის ოპერაციულ სისტემასთან (მაგალითად, Unix, MsWindows, Linux). ოპერაციულ სისტემებს აქვს აპლიკაციის დაპროგრამების ინტერფეისები (API), რომელთა თავსებადობითაც განისაზღვრება ოპერაციულ სისტემათა ოჯახი. ამრიგად, პროგრამული პლატფორმა ოპერაციული სისტემაა, რომლის

მეშვეობითაც რეალიზდება გამოყენებითი პროგრამული პაკეტები, ანუ მომხმარებელთა პროგრამული აპლიკაციები.

დაპროგრამების ენა ის ინსტრუმენტული საშუალებაა, რომელზეც იწერება როგორც პროგრამული პლატფორმები (ოპერაციული სისტემები), ასევე მომხმარებელთა პროგრამული აპლიკაციები. მაგალითად, C-ენაზე 1972 წელს დაიწერა პირველი ქსელური ოპერაციული სისტემა Unix და შეიქმნა ახალი „მაღალი საიმედოობით“ ცნობილი პლატფორმა, რომელსაც დღესაც ერთ-ერთი წამყვანი ადგილი უჭირავს პროგრამული ინჟინერიის ბაზარზე.

გამოთვლით მანქანებზე რეალიზებული დაპროგრამების ენების განვითარების ისტორია 1950 წლიდან იწყება (ნახ.1).



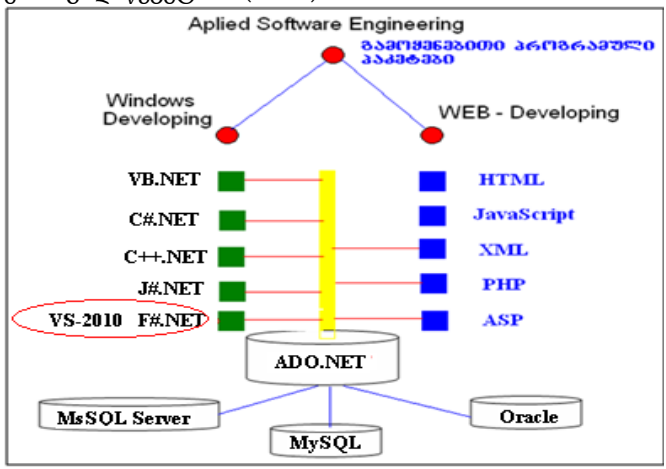
ნახ.1

დღემდე შექმნილია 2000-ზე მეტი ენა, მათგან კი მხოლოდ „ძლიერებმა“ მოაღწიეს ჩვენს საუკუნემდე და დღესაც ვითარდებიან. დაპროგრამების ენების საკონკურენციო ბაზარზე C-ენის „შთამომავლები“ (C++, Java, C#) ბატონობენ. ამას ხელი შეუწყო თვით Unix ოპერაციული სისტემის უნივერსალობამ და ანტივირუსულმა საიმედოობამ (MsWindows-თან შედარებით) [2].

ნახაზიდან ჩანს დაპროგრამების ენების (აქ ყველა არაა წარმოდგენილი ინფორმაციის დიდი მოცულობის გამო) ევოლუცია „პროგრამირების პარადიგმის“ თვალსაზრისით. პროგრამირების პარადიგმა არის ცნებათა და კონცეფციათა ერთობლიობა, რომლებიც პროგრამების დაწერის **სტილს** განსაზღვრავს. მას ენაში შეაქვს თვისობრივი ცვლილებები. დღეისათვის ცნობილია დაპროგრამების რამდენიმე სტილი: ოპერატორული, ფუნქციონალური, იმპერატიული, ლოგიკური, სტრუქტურული, ობიექტ-ორიენტირებული, ვიზუალური, კომპონენტური, სუბიექტური და ა.შ.

მაგალითად C-ენა სტრუქტურული დაპროგრამების სტილითაა ცნობილი, C#, C++, Visual Basic, Java ენები ობიექტ-ორიენტირებული სტილისაა და ა.შ. Haskell და F# მეტაფუნქციური ენებია, რომლებიც ობიექტ-ორიენტირებულ სტილსაც ფლობენ, ამიტომ აქტუალურია მათი გამოყენება ამერიკის და ევროპის უნივერსიტეტებში.

მაიკროსოფთმა Ms Visual Studio.NET 4.0 (2010) ვერსიაში პირველად წარმოადგინა F# ენა, რომლის გამოყენებასაც წარმატებული მომავალი ექნება მათემატიკური და საინჟინრო ამოცანების გადაწყვეტაში (ნახ.2).



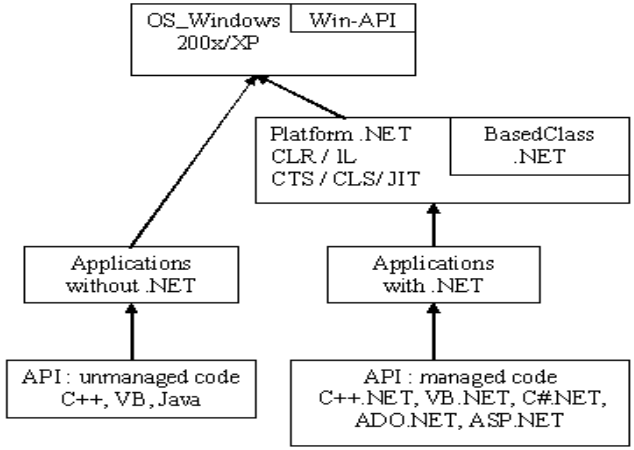
ნახ.2

➢ პლატფორმა .NET

Ms Visual Studio.NET Framework 4.0 პროგრამული პლატფორმაა, რომელიც Windows ოპერაციულ სისტემასა და გამოყენებით პროგრამულ აპლიკაციას შორისაა მოთავსებული (ნახ.3). ამ ინტეგრირებული პროგრამული პაკეტების გამოყენების მიზანია რთული სისტემების მოდელირება, კონსტრუირება და რეალიზაცია უნიფიცირებული პროგრამირების კონცეფციის გამოყენებით.

Windows-სისტემა უშუალოდ მუშაობს C++, VB, Java და სხვა ენებზე დაწერილ პროგრამულ API-დანართებთან (Application Programming Interface), რომლებიც რეალიზებულია როგორც უმართავი კოდები (unmanaged code). ამასთანავე იგი მუშაობს C#.NET, C++.NET, VB.NET და ა.შ., ზოგადად .NET-პლატფორმის მიერ მართვად (managed code) პროგრამულ დანართებთან.

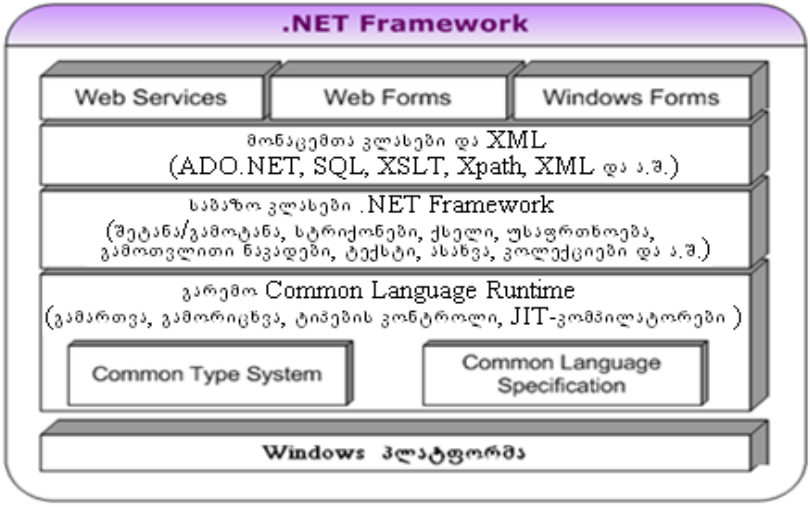
მართვაში იგულისხმება ის, რომ ეს კოდები ამუშავდება უშუალოდ .NET-ის მიერ, იმართება მათი პროცესები და მონაცემთა ნაკადები, მათ მიეწოდება შესასრულებლად საჭირო დამხმარე რესურსები და ა.შ.



ნახ.3

პრინციპში, .NET-პლატფორმა ასრულებს “ოპერაციული სისტემის” გარკვეულ ფუნქციებს და მოქნილად ფუნქციონი-

რებს Windows-თან. ამავე ნახაზზე საყურადღებოა თვით NET - პლატფორმის ბლოკი. რომელშიც ძირითადი ქვებლოკი Based Class.NET არის ამ პლატფორმის საბაზო კლასების ბიბლიოთეკა (უმრავლესობა დაწერილია C#-ენაზე). იგი სრულად ობიექტ-ორიენტირებულია, შედგება ობიექტთა ერთობლიობისგან, რომელთაგანაც თითოეულში რეალიზებულია განსაზღვრულ მეთოდთა ჯგუფები. მაგალითად, ფანჯრებისა და ფორმების ასახვა (Windows GUI), მონაცემთა ფაილებთან ურთიერთობა (ADO.NET), ვებ-გვერდების ორგანიზება და ინტერნეტთან კავშირი (ASP.NET) და სხვ. მე-4 ნახაზზე ნაჩვენებია .NET Framework-ის არქიტექტურა.



ნახ.4

ამავე ბლოკში ნაჩვენებია .NET-runtime - პლატფორმის სამუშაო გარემო (რომელშიც სრულდება პროგრამა), ანუ CLR(Common Language Runtime) და მას შესრულების საერთო გარემოსაც უწოდებენ. ესაა პროგრამული უზრუნველყოფა მომხმარებელთა გამოყენებითი პროგრამების შესასრულებლად.

CTS საერთო ტიპების სისტემაა (Common Type System), რომლის საფუძველზეც NET-პლატფორმა უზრუნველყოფს

დაპროგრამების სხვადასხვა ენის თავსებადობას. ამასთანავე CTS აღწერს მომხმარებელთა კლასების განსაზღვრის წესებსაც.

IL შუალედური გარდაქმნის ენაა (Intermediate Language). პროგრამები, რომელთა საწყისი კოდები დაწერილია, მაგალითად C#, C++, J++ ან VB ენებზე .NET-ში, კომპილატორი ამ მართვად კოდებს გადაიყვანს შუალედურ IL-ენაზე, რომელთაც შემდეგ CTS სწრაფად აკომპილირებს მანქანურ კოდში. ამგვარად, ობიექტური კოდები IL-ენის საშუალებით ისე მიიღება, რომ მათში არაა დაფიქსირებული, თუ რომელ ენაზეა დაწერილი საწყისი კოდი.

CLS ენის საერთო სპეციფიკაციაა (Common Language Specification), ანუ იმ სტანდარტების მინიმალური ერთობლიობა, რომელიც უზრუნველყოფს კოდებთან მიმართვას .NET-ის ნებისმიერი ენიდან. ამ ენების ყველა კომპილატორს გააჩნია CLS მხარდაჭერა.

JIT (Just-In-Time) ესაა შუალედური კოდის კომპილაციის ფაზა მანქანურ კოდში. სახელწოდება მიუთითებს იმაზე, რომ კოდის მხოლოდ იმ ცალკეული ნაწილების კომპილაცია ხდება, რომლებიც საჭიროა პროგრამის შესასრულებლად დროის მოცემულ მომენტში.

.NET Framework -ში გამოიყენება 2-ეტაპიანი კომპილაცია:

1. ეტაპი: კომპილაცია MSIL-ენაში; 2. ეტაპი: “just-in-time” კომპილაცია უშუალოდ შესრულების პროცესში.

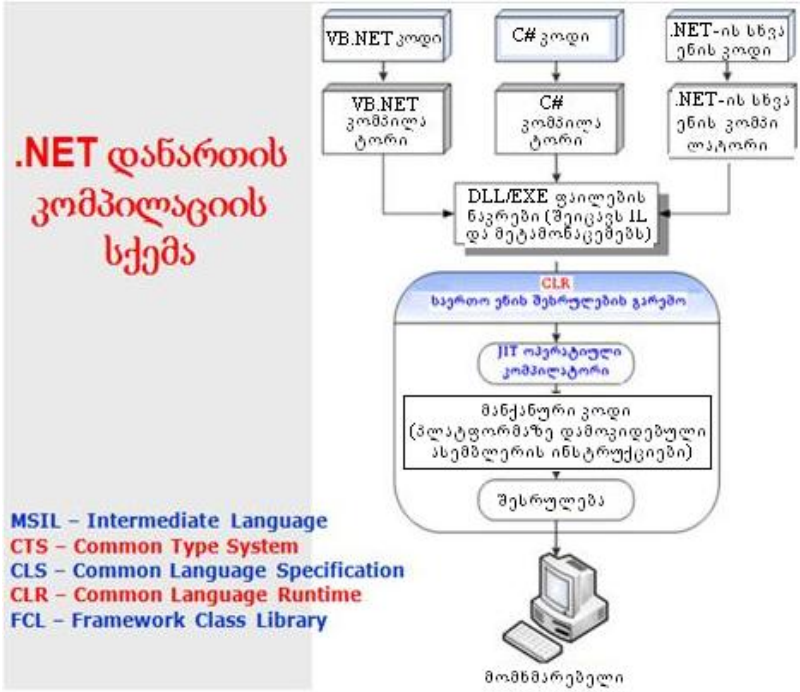
MSIL - ასემბლერული ენაა, რომელიც არაა დამოკიდებული მანქანაზე. ის სრულდება ყველგან, სადაც დაყენებულია CLR.

HTML-ისგან aspx-გვერდი განსხვავდება მასში სერვერული მართვის ელემენტების არსებობით, რომლებიც აღიწერება სპეც-ტეგებით. მე-5 ნახაზზე ნაჩვენებია სამომხმარებლო აპლიკაციის კომპილაციის სქემა .NET პლატფორმაზე.

➢ საერთო ტიპების სისტემა (CTS)

მაიკროსოფთის NET-პლატფორმისა და IL-შუალედური ენის არსებობის კონცეფციის საფუძველია ძირითადად ენის ობიექტ-

ორიენტულობა და საერთო ტიპების სისტემის არსებობა, რომელთაც კომპილატორები ფლობს.



ნახ.5

C# („სი შარფ”) ენა ობიექტ-ორიენტირებული ენების ერთ-ერთი ახალი და მძლავრი წარმომადგენელია, რომელიც შეიქმნა სპეციალურად NET-პლატფორმისათვის და თავსებადია Windows-ის თანამედროვე ვერსიებთან და ინტერნეტთან. ამ ენაზეა რეალიზებული NET-პლატფორმის უმრავლესი საბაზო კლასები.

როგორც ცნობილია, C++ ენა კომპილირდება ასემბლერულ კოდში, C#[1] ენა კი - შუალედურ IL-ენაში.

IL-ენის დანიშნულებაა პლატფორმული და ენობრივი დამოუკიდებლობის განხორციელება ობიექტ-ორიენტირებულ

გარემოში. Java-ენაც უზრუნველყოფს პლატფორმულ (Windows, Unix, Linux) დამოუკიდებლობას, მაგრამ მისი ბაიტ-კოდის შესრულების ეტაპზე იგი ინტერპრეტირდება (IL -კი კომპილირდება).

NET-პლატფორმისათვის ენობრივი თავსებადობა ხორციელდება IL ენაში არსებული ტიპების დიდი რაოდენობით, რომლებიც ორგანიზებულია ტიპთა იერარქიის ობიექტ-ორიენტირებული პრინციპებით. მე-6 ნახაზზე ნაჩვენებია ტიპთა ასეთი იერარქია მემკვიდრეობითობის კავშირის გამოყენებით.

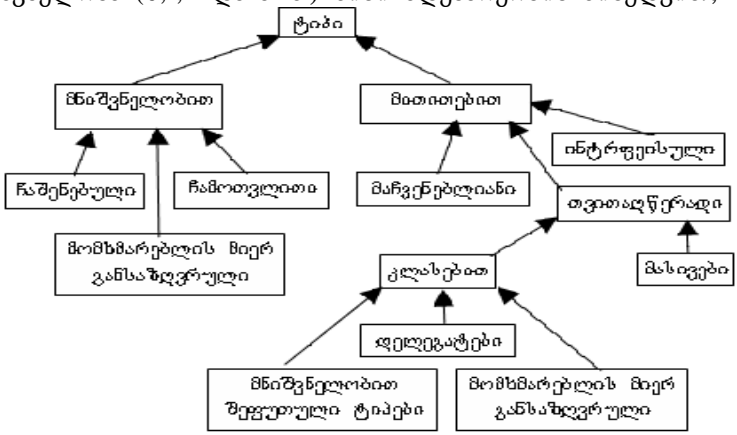
მოვიტანოთ ზოგიერთი კომენტარი, რომელიც ახსნის ნახაზს:

- **ტიპი** არის საბაზო კლასი, რომელიც ზოგადად ასახავს ნებისმიერ ტიპს;

- **ტიპი მნიშვნელობით** არის საბაზო კლასი, რომელიც ზოგადად ასახავს ნებისმიერ ტიპს მნიშვნელობით;

- **ჩაშენებული ტიპები მნიშვნელობით** არის სტანდარტული საბაზო ტიპები, რომლებიც აღწერს რიცხვებს, სიმბოლოებსა და ლოგიკურ მნიშვნელობებს;

- **ჩამოთვლით ტიპი** არის ჩამონათვალთა ერთობლიობა, რომელშიც თითოეულ მნიშვნელობას შეესაბამება რიცხვითი მნიშვნელობა (0,1,... და ა. შ.) მისი მდებარეობის მიხედვით;



ნახ.6. ტიპების ზოგადი სისტემა

- **მომხმარებლის მიერ განსაზღვრული ტიპი** არის საწყის კოდში (მომხმარებლის პროგრამაში) აღწერილი ტიპები, რომლებიც ინახება მნიშნელობებით (ესაა მაგალითად, სტრუქტურები).

- **ტიპი მითითებით** არის მონაცემთა ნებისმიერი ტიპები, რომელთანაც მიმართვა ხორციელდება მიმთითებლებით და ინახება ნაკადში;

- **თვითაღწერადი ტიპები** არის ტიპები, რომლებიც ასახავს ინფორმაციას თავიანთ შესახებ;

- **თვითაღწერადი ტიპები** არის ტიპები, რომლებიც ასახავს ინფორმაციას თავიანთ შესახებ;

- **მასივები** არის ნებისმიერი ტიპი, რომელიც შეიცავს ობიექტების მასივს;

- **ტიპები კლასებით** არის თვითაღწერადი ტიპები, რომლებიც არაა მასივები;

- **დელეგატები** - ტიპებია, რომლებიც დამუშავდა მიმთითებელთა შესანახად კლასის მეთოდებისათვის;

- **მნიშვნელობით შეფუთული ტიპები** არის ტიპები მნიშვნელობით, რომლებიც დროებით დაიყვანება ტიპებამდე მიმთითებლით, რათა შენახულ იქნას ნაკადში;

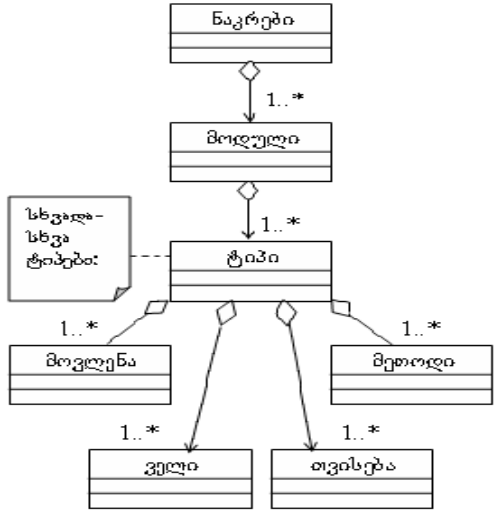
- **მომხმარებლის მიერ განსაზღვრული ტიპები მითითებით** არის ტიპები, განსაზღვრული საწყის კოდში, როგორც ტიპები მითითებით. პროგრამაში ესაა მაგალითად, ნებისმიერი კლასი.

➢ .NET-პლატფორმის კომპონენტები და მისი დანართის სტრუქტურა

როგორც აღვნიშნეთ, .NEთ პლატფორმის საბაზო კლასების დიდი ნაწილი დაწერილია C# ენის გამოყენებით, ამიტომაც საილუსტრაციო მაგალითებს ამ ენაზე გავუკეთებთ კომენტარს..

.NET პლატფორმის კომპონენტებიდან ერთ-ერთი მთავარი ბლოკია Assembly (ანაწყობი, ნაკრები), რომელიც ლოგიკურად აერთიანებს კოდს, რესურსებს და მეტამონაცემებს. იგი ლოგიკური და არა ფიზიკური ერთეულია, რადგან შეუძლია მოთავსდეს რამდენიმე ფაილში. ასეთ შემთხვევაში არსებობს

ერთი მთავარი ფაილი, რომელშიც ინახება ინფორმაცია დანარჩენებზე. მე-7 ნახაზზე ნაჩვენებია პროგრამული დანართის (Application) შესაბამისი ნაკრების ზოგადი იერარქიული სტრუქტურა აგრეგაციის კავშირის გამოყენებით.



ნახ.7. ნაკრების ზოგადი სტრუქტურა

ნახაზიდან ჩანს, რომ ნაკრები 1 ან რამდენიმე (1..*) მოდულისგან (მოდულე) შედგება. სწორედ მოდულში ინახება დანართის ან ბიბლიოთეკის კოდი, მისი მეტამონაცემებით. მოდულები შეიცავს ტიპებს. ესაა კოდის შაბლონები (კლასები), რომლებშიც ინკაფსულირებულია გარკვეული მონაცემები და მეთოდები. როგორც წინა პარაგრაფში ვახსენეთ, ტიპები ორი სახისაა: მიმთითებლებით (ანუ კლასები) და მნიშვნელობებით (ანუ სტრუქტურები).

ტიპებს აქვს ველები, თვისებები და მეთოდები. ველი გამოყოფს მეხსიერების ადგილს შესაბამის მონაცემთა ტიპისათვის. თვისებები ველების მსგავსია, ოღონდაც მათი დანიშნულებაა შესაბამის მონაცემთა საწყისი მნიშვნელობების განსაზღვრა და კონტროლი.

მეთოდები განსაზღვრავს მონაცემთა დასამუშავებლად კლასის ქცევას, ანუ რეაქციას გარედან შემოსულ შეტყობინებაზე (მოთხოვნაზე). შეტყობინება ინაფორმაციაა, რომელიც ამა თუ იმ მოვლენის შედეგად ფორმირდება.

პროგრამული ნაკრები შეიძლება იყოს ორი ტიპის: კერძო და საერთო გამოყენების. პირველ შემთხვევაში ნაკრები ინსტალირდება კერძო მომხმარებლს კატალოგში და მასთან სხვა მიმართვები გამორიცხულია.

საერთო გამოყენების ნაკრები შეიცავს პროგრამულ ბიბლიოთეკებს, რომელთაც იყენებს სხვადასხვა დანართი. აქ საჭიროა სპეციალური დაცვის მექანიზმების გამოყენება (სახელების კოლიზიისა და ნაკრებთა ვერსიების კონტროლის თვალსაზრისით).

კლასებს შორის სახელთა კოლიზიის აღმოფხვრის მიზნით .NET პლატფორმა იყენებს “სახელთა სივრცეს”.

სახელსივრცე (namespace): ესაა მონაცემთა ტიპების უბრალო დაჯგუფება. ყველა მონაცემთა ტიპის სახელს მოცემულ სახელთა სივრცეში ავტომატურად ემატება პრეფიქსი, რომელიც შედგენილია სახელსივრცის დასახელებისგან. ასევე შესაძლებელია ჩადგმული სახელსივრცეების შექმნა.

მაგალითად, საბაზო კლასების უმრავლესობისათვის, რომლებიც ზოგადი გამოყენებისთვისაა დანიშნული, მოთავსებულია სახელთა სივრცეში System, ვებ-გვერდებისათვის - System.Web და ა.შ.

C#-ის პროგრამის ტექსტის მაგალითზე შეიძლება შემდეგი კომენტარის გაკეთება:

```
namespace Magazia.Web // აქ სახელია Magazia.Web
{
  public class Checkout : PageBase
  {
    // და ა.შ.
```

დანართთა არეები (application area) არის .NET პლატფორმის მნიშვნელოვანი ელემენტი. მათი დანიშულებაა ერთდროულად და ერთმანეთთან მომუშავე დანართების იზოლაცია, რათა არ მოხდეს მონაცემთა არასასურველი დამუშავება.

პროგრამული დანართების იზოლაციისათვის Windows გამოიყენებს **„პროცესის”** ცნებას, რომელიც მისამართების სივრცეს ეხება. ყოველ პროცესს გამოეყოფა 4 გიგაბაიტი ვირტუალური მეხსიერება. ისინი დისკოზე სხვადასხვა ფიზიკური მისამართებითაა და არ გადაიკვეთება.

პროცესებს აქვს მინიჭებული განსაზღვრული პრივილეგიები და ოპერაციული სისტემა აკონტროლებს მათ, თუ რომელ ოპერაციას რომელი პროცესის გამოყენება შეუძლია.

დანართთა არეების გამოყენების იდეა მდგომარეობს იმაში, რომ პროცესებს შორის მოხერხდეს მონაცემთა გაცვლა. ამიტომაც პროცესი იყოფა რამდენიმე დანართის არედ. თითოეულ დანართის არეში თავსდება ერთი დანართის კოდი.

.NET პლატფორმის მნიშვნელოვანი საშუალებაა JIT (Just-In-Time) კომპილატორი. იგი ახორციელებს პროგრამული კოდის ცალკეული ნაწილის დროულად კომპილირებას (საჭიროების შემთხვევაში).

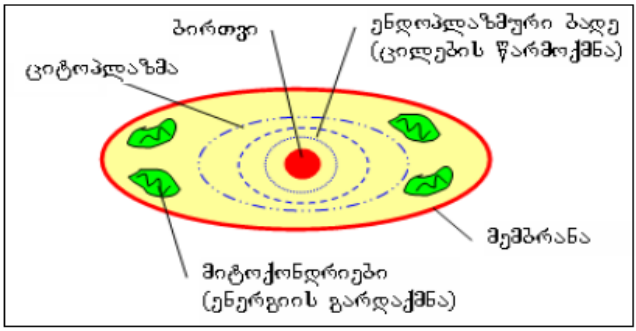
Visual Studio.NET არის პროგრამული სისტემების დამუშავების ინტეგრირებული გარემო, რომელშიც შესაძლებელია კოდების დაწერა, კომპილირება და გამართვა VB.NET, C++.NET, C#.NET, ASP.NET, ADO.NETO და სხვა ტექნოლოგიებით.

➢ ობიექტ-ორიენტირებული დაპროგრამების მეთოდი: კლასები და ობიექტები

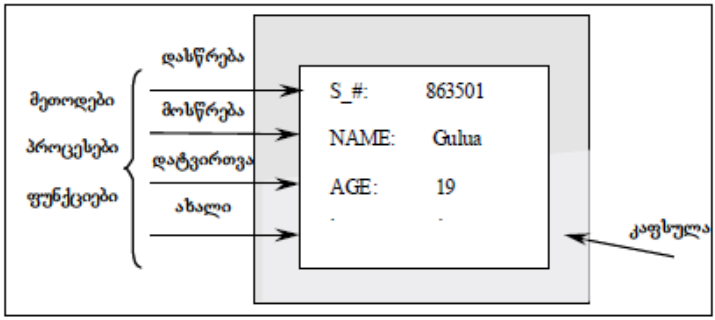
.NET გარემოში დაპროგრამება დაფუძნებულია ობიექტებზე. ობიექტი (object) – არის პროგრამული კონსტრუქცია, რომელშიც ინკაფსულირებულია ლოგიკურად დაკავშირებული მონაცემებისა და მეთოდების ერთობლიობა.

ობიექტი (Object) გაანიხილება, როგორც გარკვეული არსი (Entity), რომელიც ხასიათდება მდგომარეობით (მონაცემთა ერთობლიობა) და ქცევით (ფუნქციური პროგრამები). ობიექტის

ქცევა ანუ რეაქცია, რომლის დროსაც მისი ახალი მდგომარეობა განისაზღვრება, დამოკიდებულია გარედან მოსულ ინფორმაციაზე, შეტყობინებებზე. ვინაიდან ობიექტი უმთავრესი ცნებაა, იგი საწყისია ობიექტორიენტირებული დაპროგრამებისა, ამიტომ დიდი მნიშვნელობა აქვს მის სწორად გაგებას. განვიხილოთ კლასის და ობიექტის არსი ბიოლოგიური და ინფორმაციული უჯრედების მოდელების შედარების საფუძველზე (ნახ.8,9).



ნახ.8



ნახ.9

ბიოლოგიური უჯრედი კაფსულირებულია მემბრანის (გარსის) საშუალებით, რომლითაც ის გამოიყოფა სხვა უჯრედებისა და გარემოსაგან. მას უნარი აქვს გარემოსგან მიიღოს ქიმიური სახის ინფორმაცია და გადასცეს უჯრედს შიგნით. შუაგულში მოთავსებულია ბირთვი, რომელიც უჯრედის ძირითადი ინფორმაციის მატარებელია. იგი შედგება ქრომოსომებისგან,

რომლებიც გარკვეული გენეტიკის მქონეა. უჯრედის დაყოფის (გამრავლების) დროს ხდება მემკვიდრული თვისებების გადაცემა. ბირთვის გარშემო დაჯგუფებულია სხვადასხვა ფუნქციის ელემენტები. მაგალითად, ენდოპლაზმური ბადე – ცილების წარმოქმნის ფუნქცია, მიტოქონდრები - ენერგიის გარდაქმნის ფუნქცია, ციტოპლაზმა (თხევადი მოძრავი გარემო) – ტრანსპორტირების ფუნქცია და ა.შ.

როგორია “ინფორმაციული უჯრედის” აგებულება და რა ანალოგიაა ბიოლოგიურთან? მე-9 ნახაზზე განიხილება ობიექტი-სტუდენტი, რომელიც რეალური სამყაროს ნაწილია. იგი კაფსულირებულია, რომლის შიგნითაც მოთავსებულია ბირთვი ობიექტის თვისებების აღმწერ მონაცემთა ელემენტები S =(სტუდენტის ნომერი), NAME (გვარი, სახელი), AGE (ასაკი) და ა.შ.

ობიექტები ავტონომიური ელემენტებია, ისინი იქმნება შაბლონით (template), რომელსაც ობიექტ-ორიენტირებულ დაპროგრამების თეორიაში კლასს (class) უწოდებენ.

.NET-ის საბაზო კლასების ბიბლიოთეკა არის კლასთა ერთობლიობა, რომელიც გამოიყენება ობიექტების შესაქმნელად მომხმარებლის კერძო აპლიკაციაში. ამასთანავე პროგრამული პაკეტი Visual Studio საშუალებას იძლევა შეიქმნას საკუთარი კლასები კერძო პროგრამებისთვის.

- **ობიექტები, წევრები და აბსტრაგირება:**

ობიექტი – პროგრამული კონსტრუქციაა, რომელიც ასახავს განსაზღვრულ არსს (Entity). ესაა რეალურ სამყაროში არსებული ობიექტები, მაგალითად, ადამიანები, მანქანები, ფირმები, ცხოველები, ფრინველები, მცენარეები, ინსტიტუტები, კომპიუტერები და ა.შ. ყოველ ობიექტს აქვს განსაზღვრული, მისთვის დამახასიათებელი ფუნქციონალობა და თვისებები.

აპლიკაციაში ობიექტი შეიძლება იყოს ფორმა, მართვის ელემენტი (ღილაკი, კომბობოქსი, ბაზასთან შეერთება და სხვ.).

ამგვარად, ობიექტი დასრულებული ფუნქციონალური ერთეულია, რომელიც შეიცავს ყველა მონაცემს და ყველა ფუნქციას, რომლებიც აუცილებელია იმ ამოცანის გადასაწყვეტად, რომლისთვისაც ეს ობიექტი გამოიყენება.

რეალური სამყაროს ობიექტების წარმოდგენას (ასახვას) პროგრამული ობიექტებით უწოდებენ აბსტრაგირებას (abstraction).

- **კლასები, როგორც ობიექტთა შაბლონები:**

კლასი არის ერთგვაროვან ობიექტთა ერთობლიობა, რომელიც ამ ერთობლიობის სტრუქტურას ასახავს. კლასი განსაზღვრავს ობიექტთა წევრებს და მათ ყოფაქცევას, აგრეთვე საწყის მონაცემთა მნიშვნელობებს, საჭიროების შემთხვევაში.

კლასის ეგზემპლარის შექმნისას კომპიუტერის მეხსიერებაში შეიქმნება ამ კლასის ასლი. ასეთი სახით შექმნილ კლასის ეგზემპლარს უწოდებენ ობიექტს. დაპროგრამების ენაში მისი შექმნა ხდება ოპერატორით new. მაგალითად:

```
// გამოცხადდეს ცვლადი MyDataForm ტიპით DataForm
DataForm MyDataForm;
// შეიქმნას ობიექტის ეგზემპლარი DataForm და
// ჩაიწეროს იგი ცვლადში MyDataForm
MyDataForm = new DataForm( );
```

- **ობიექტები და წევრები:**

ობიექტები შედგება წევრებისგან, რომელთაც ეკუთვნის თვისებები, ველები, მეთოდები და მოვლენები. ისინი განსაზღვრავს ობიექტის მონაცემებს და ფუნქციონალობას.

ველები და თვისებები შეიცავს ობიექტის მონაცემებს, რომლებიც მის მდგომარეობას ასახავს. მეთოდები განსაზღვრავს მოქმედებებს, რომელთა შესრულება შეუძლია ობიექტს. მოვლენები კი წარმოადგენს შეტყობინებებს, რომელთაც მიიღებს ობიექტი ან გადასცემს სხვა ობიექტებს. მოვლენის დანიშნულებაა გააქტიუროს ობიექტის ესა თუ ის მეთოდი, რომელიც გადაამუშავებს მის მონაცემებს.

მაგალითად, ობიექტისთვის „ავტომობილი” თვისებები და ველებია: მოდელი, ფერი, ასაკი, საწვავის–ხარჯი და ა.შ. ეს მონაცემები ასახავს ობიექტის მდგომარეობას. მეთოდის მაგალითებია: სიჩქარის–გადართვა, დამუხრუჭება, რულის–მობრუნება და სხვ., ისინი ობიექტის ქცევას განსაზღვრავს. ავტომობილისთვის მოვლენები მიიღება შეტყობინების სახით გარედან (მაგალითად, ინფორმაცია გზის მოსახვევის შესახებ, ან სიჩქარის შეზღუდვის შესახებ) ან მანქანის მდგომარეობის

ამსახველი ნათურებიდან (წყლის გადახურება, საწვავის დამთავრება და ა.შ.).

- **ობიექტური მოდელები:**

მარტივ ობიექტებს აქვს რამდენიმე თვისება და მეთოდი, ასევე ერთი-ორი მოვლენა. რთულ ობიექტებს გააჩნია მეტი რაოდენობა თვისებების, მეთოდებისა და მოვლენების, აგრეთვე მათ შეიძლება ჰქონდეს “შვილი” ობიექტებიც, რომლებზეც შეუძლია პირდაპირი მიმართვის განხორციელება. მაგალითად, მართვის ელემენტს TextBox, აქვს თვისება Font, რომელიც არის Font-ტიპის ობიექტი. ამ თვალსაზრისით, ნებისმიერი Form კლასის ეგზემპლარი მოიცავს მასზე განთავსებულ მართვის ელემენტებს (Controls ერთობლიობა).

„მშობელი-შვილი” ჩადგმული ობიექტების იერარქია, რომელიც ქმნის ობიექტის სტრუქტურას, არის ობიექტური მოდელი (object model).

მაგალითად, ობიექტი „ავტომობილი” შედგება რამდენიმე „შვილი” ობიექტისგან: ძრავი, ბორბლები, ძარა და ა.შ. „ავტომობილი” ობიექტის ყოფაქცევა, რომელსაც „შვილი” ობიექტისთვის (ძრავი) თვისებაში აქვს ცილინდრების რაოდენობა 4 და 8, იქნება განსხვავებული.

„შვილი” ობიექტი შეიძლება შედგებოდეს თავის საკუთარი „შვილი” ობიექტებისგან და ა.შ. მაგალითად, ავტომობილი -> ძრავი -> სანთლები.

- **ინკაფსულაცია:**

ობიექტ-ორიენტირებული დაპროგრამების (ოოდ) ერთ-ერთი საბაზო პრინციპია ინკაფსულაცია, რომლის არსი მდგომარეობს ობიექტის რეალიზაციის გამოყოფაში მისი ინტერფეისისგან. ანუ პროგრამული დანართი (აპლიკაცია) ურთიერთქმედებს ობიექტთან მისი ინტერფეისის საშუალებით, რომელიც შედგება ღია თვისებებისა და მეთოდებისაგან.

ობიექტები ერთმანეთთან ურთიერთმოქმედებს თავიანთი ღია თვისებებით და მეთოდებით, ამიტომაც ობიექტს უნდა

ჰქონდეს ყველა აუცილებელი მონაცემი და მეთოდების ერთობლიობა, ამ მონაცემთა დასამუშავებლად.

ინტერფეისმა არ უნდა მისცეს „სხვას“ ობიექტის შიგა მონაცემებთან წვდომის უფლება, რაც გამორიცხავს ამ შემთხვევაში ობიექტის შიგა მონაცემებისთვის public-მოდიფიკატორის გამოყენებას (გამოიყენება private).

➢ პოლიმორფიზმი და მემკვიდრეობითობა

- **პოლიმორფიზმი:**

პოლიმორფიზმი მრავალფორმიანობას ნიშნავს და მას ობიექტ-ორიენტირებულ დაპროგრამებაში განსაკუთრებული როლი აქვს. პოლიმორფიზმის საშუალებით შესაძლებელია ერთიდაიგივე გახსნილი ინტერფეისის სხვადასხვაგვარი რეალიზაცია სხვადასხვა კლასში. ანუ პოლიმორფიზმის საშუალებით შესაძლებელია ობიექტის მეთოდებისა და თვისებების გამოძახება მათი რეალიზაციის მიუხედავად.

მაგალითად, ობიექტი *მძღოლი* ურთიერთქმედებს ობიექტთან *ავტომობილი* იმავე სახელის მქონე ღია ინტერფეისის საშუალებით. თუ სხვა ობიექტი, მაგალითად, *სატვირთო* ან *სპორტული–მანქანა* ფლობს ამ ღია ინტერფეისის მხარდაჭერას, მაშინ ობიექტს *მძღოლი* შეუძლია მათთანაც ურთიერთქმედება, მიუხედავად ინტერფეისის განსხვავებული რეალიზაციისა

პოლიმორფიზმის რეალიზაციის ორი ძირითადი მიდგომა არსებობს: ინტერფეისების და მემკვიდრეობითობის გამოყენებით. განვიხილოთ თვითოეული ცალ-ცალკე.

ინტერფეისი (interface) – ესაა შეთანხმება, რომელიც ობიექტის ყოფაქცევას განსაზღვრავს. იგი ადგენს კლასის წევრთა სიას, მაგრამ არაფერს ამბობს მათი რეალიზაციის შესახებ. ობიექტში დასაშვებია რამდენიმე ინტერფეისის რეალიზაცია, ხოლო ერთიდაიგივე ინტერფეისი შეიძლება რეალიზებულ იქნას სხვადასხვა კლასში.

ნებისმიერ ობიექტებს, რომლებშიც რეალიზებულია რომელიმე ინტერფეისი, შეუძლია ერთმანეთთან ურთიერთმოქმედება მისი საშუალებით. მაგალითად, ობიექტში *ავტომობილი*, რომელზეც ჩვენ ვსაუბრობთ, შეიძლება *IDrivable*-ინტერფეისის რეალიზაცია (ინტერფეისთა სახელები მიღებულია დაიწყოს “I” ასოთი), მეთოდებით: *მოძრაობა–წინ, მოძრაობა–უკან, გაჩერება.* იგივე ინტერფეისი შეიძლება რეალიზდეს სხვა კლასებშიც, როგორიცაა, მაგალითად, *სატვირთო–მანქანა* ან *კატერი.* შედეგად, ამ ობიექტებს შეუძლია ურთიერთმოქმედება ობიექტთან *მძღოლი.* ეს უკანასკნელი მთლიანად უხილავია ინტერფეისის რეალიზაციისთვის, რომელთანაც იგი მოქმედებს, მისთვის ცნობილია მხოლოდ ინტერფეისი.

მემკვიდრეობითობა იძლევა ახალი კლასების შექმნის საშუალებას არსებულის ბაზაზე. ამასთანავე ახალ კლასებში ნებადართულია ძველი კლასების სრული ფუნქციონალობის ჩართვა და, საჭიროების შემთხვევაში, შესაძლებელია მათი წევრების მოდიფიცირებაც.

კლასი, რომელიც გამოცხადებულია სხვა კლასის საფუძველზე, უწოდებენ წარმოებულ კლასს (derived class). ყოველ კლასს შეიძლება ჰქონდეს მხოლოდ ერთი პირდაპირი წინაპარი – მისი საბაზო კლასი (base class). წარმოებულ კლასს აქვს საბაზო კლასის წევრების ერთობლიობა. ასევე შესაძლებელია ახალი წევრების დამატება და ძველი წევრების (მემკვიდრეობით მიღებული) რეალიზაციის ცვლილებაც. წარმოებული კლასები ინარჩუნებს თავისი საბაზო კლასის ყველა მახასიათებელს და უნარი აქვს სხვა ობიექტებთან ისეთი ურთიერთმოქმედებისათვის, როგორც საბაზო კლასის ეგზემპლარებს.

მაგალითად, საბაზო კლასიდან *ავტომობილი* შეიძლება წსრმოებული კლასის *სპორტული–ავტომობილი* გამოცხადება, რომელიც თავის მხრივ, როგორც საბაზო კლასი, შეძლებს ახალი წარმოებული კლასის *სპორტული–კაბრიოლეტი* გამოცხადებას. ყოველ წარმოებულ კლასში დასაშვებია ახალი

წევრების (თვისებები, მეთოდები, მოვლენები) შემოტანა, ამსთანავე ისინი ინარჩუნებს საწყისი საბაზო კლასის *ავტომობილი* ფუნქციონალობას უცვლელი ფორმით.

➢ წევრების გადატვირთვა

გადატვირთვის შედეგად შესაძლებელია კლასის რამდენიმე წევრის შექმნა ერთიდაიმავე სახელით, ოღონდ განსხვავებული სიგნატურით (მოქმედებით). გადატვირთვას უფრო ხშირად იყენებენ მეთოდებისათვის. რეალიზებულია ასევე ოპერატორების გადატვირთვაც. ამ პარაგრაფში ჩვენი მიზანია კლასის გადატვირთული წევრების შექმნა.

მაგალითად, თუ კლასს სჭირდება ისეთი წევრი, რომელსაც შეუძლია სხვადასხვა პარამეტრების მიღება, ექნება შემდეგი სახე:

```
public void Display(int DisplayValue)
 {
   // . . .  }
```

ეს მეთოდი იფუნქციონირებს ნორმალურად, თუ მის ცვლადს მიეწოდა მთელი ტიპის მნიშვნელობა. სხვა ტიპის (მაგ., სტრიქონული ცვლადი), ან ერთზე მეტი პარამეტრის შემთხვევაში ის ვეღარ შეასრულებს თავის საქმეს.

ამიტომაც იყენებენ გადატვირთვას, ანუ რამდენიმე მეთოდის შექმნას ერთი სახელით.

გადატვირთულ მეთოდებს შეუძლია ერთნაირი ტიპის მნიშვნელობების დაბრუნება, მათი გამოცხადება ერთნაირი წვდომის მოდიფიკატორებით, მაგრამ გადატვირთული მეთოდების სიგნატურები უნდა იყოს განსხვავებული.

გადატვირთული მეთოდის გამოძახებისას შესრულების გარემო ამოწმებს გადაცემულ არგუმენტების ტიპებს, ადარებს არგუმენტების სიას არსებულ გადატვირთულ მეთოდთა სიგნატურებთან და იძახებს იმას, რომლის სიგნატურაც დაემთხვევა არგუმენტთა სიას. თუ არც ერთი არ დაემთხვა, მაშინ ფორმირდება შეცდომა.

- **გადატვირთული მეთოდების შექმნა:**

გადატვირთული მეთოდები იქმნება ისევე, როგორც ყველა სხვა მეთოდი. უპირველესად ყოვლისა საჭიროა მეთოდის გამოცხადება რაღაც სახელით, წვდომის მოდიფიკატორით, დასაბრუნებელი მნიშვნელობის ტიპით და არგუმენტების სიით.

გადატვირთულ მეთოდს უნდა ჰქონდეს იგივე სახელი, რაც აქვს არსებულ მეთოდს, ოღონდ მას ექნება სხვა სიგნატურა. მაგალითად:

```
// გადატვირთული მეთოდის მაგალითი
public void DisplayMessage(int I)
{
 MessageBox.Show(I.ToString());
}
// შემდეგ მეთოდს აქვს იგივე სახელი და სხვა სიგნატურა
public void DisplayMessage(string S)
{
  MessageBox.Show(S);
}
```

აქ განსაზღვრულია ორი მეთოდი ერთნაირი სახელით, ოღონდ სხვადასხვა სიგნატურით და რეალიზაციით. **DisplayMessage** მეთოდის გამოძახებისას შესრულების გარემო შეამოწმებს გადაცემულ არგუმენტის ტიპს. თუ იგი **String**-ია, ის გამოიძახებს მე-2 მეთოდს, ხოლო თუ **int**, მაშინ - 1-ელს.

ამგვარად, გადატვირთული მეთოდის შესაქმნელად საჭიროა ახალი მეთოდის გამოცხადება ძველი სახელით, ოღონდ სიგნატურა არ უნდა ემთხვეოდეს არცერთ ძველს. წვდომის მოდიფიკატორისა და დასაბრუნებელი ტიპისთვის ეს შეზღუდვები მოხსნილია. შემდეგ ახალი მეთოდისთვის იწერება რეალიზაცია.

- **ოპერატორების გადატვირთვა:**

მომხმარებლის მონაცემთა ტიპებთან მუშაობისას მოსახერხებელია არითმეტიკული და ლოგიკური ოპერატორების განსაზღვრა, აგრეთვე შედარების ოპერატორებისა, რომლებიც

მუშაობს აღნიშნულ ტიპებთან. განვიხილოთ შემდეგი სტრუქტურა:

```
public struct HoursWorked
 {
   float RegularHours;
   float OvertimeHours;  }
```

ასეთი მარტივი სტრუქტურის გამოყენება შესაძლოა ბუღალტრულ აპლიკაციაში სამუშაო საათებისა და ზედმეტი საათების (ზეგანაკვეთური სამუშაოს) აღრიცხვის მიზნით. მაგრამ ასეთი სტრუქტურის რამდენიმე ეგზემპლართან მუშაობისას შესაძლოა სირთულეების აღმოცენება. მაგალითად, დავუშვათ, რომ საჭიროა ასეთი სტრუქტურის ორი ეგზემპლარის მნიშვნელობათა შეჯამება. ასეთ შემთხვევაში აუცილებელია ახალი მეთოდის დაწერა, რომელსაც შეეძლება მისი გადაჭრა. თუ საჭიროა სამი ან მეტი ეგზემპლარის მნიშვნელობათა შეჯამება, მაშინ ეს ახალი მეთოდი უნდა გამოვიძახოთ რამდენჯერმე, რაც არაეფექტურია.

C# საშუალებას იძლევა განისაზღვროს ოპერატორის ქცევა მომხმარებელთა მონაცემეთა ტიპების დასამუშავებლად. მაგალითად, თუ საჭიროა სტრუქტურის შეჯამება ჩვენი მაგალითისთვის, შესაძლებელია "+" ოპერატორის გადატვირთული ვერსიის შექმნა, რაც მოითხოვს ამ ოპერატორში აუცილებელი ფუნქციონალობის ჩამატებას.

გადატვირთული ოპერატორები იქმნება გასაღებური სიტყვით operator, რომელსაც შემდეგი სინტაკსი აქვს:

```
public static type operator op (Argiment1[,  Argument2])
{
  // . . . რეალიზაცია
}
```

სადაც ***type*** ტიპია, რომელთანაც მუშაობს ოპერატორი. ის ამავდროულად არის ოპერატორის მიერ დაბრუნებული მნიშვნელობის ტიპი; **Argiment1** და **Argument2** ოპერატორის არგუმენტებია.

უნარული ოპერატორი ამუშავებს ერთ არგუმენტს ***type*** ტიპით. ბინარული ოპერატორები თხოულობს ორ არგუმენტს, რომელთაგან ერთი მაინც ***type*** ტიპისაა.

op - ეს თვით ოპერატორია, მაგალითად, { + , - , > , < , ! = და ა.შ.}.

გადატვირთული ოპერატორები უნდა გამოცხადდეს public და static მოდიფიკატორებით, რათა სხვა მომხმარებლებმა შეძლონ მათი გამოყენება.

გადატვირთული ოპერატორი უნდა გამოცხადდეს მომხმარებლის მონაცემთა ტიპის განსაზღვრის შიგნით, რომელთანაც უნდა იმუშაოს ოპერატორმა.

ჩვენ განვიხილეთ გადატვირთული ოპერატორის მაგალითი სტრუქტურასთან, მაგრამ იგი გამოიყენება ასევე კლასებთან.

ახლა განვიხილოთ გადატვირთული ოპერატორი “+”, რომელიც შეიქმნა ზემოთ HoursWorked სტრუქტურასთან სამუშაოდ:

```
public struct HoursWorked
{
 float RegularHours;
 float OvertimeHours;
  // გადატვირთული ოპერატორი გამოცხადებულ უნდა იქნას კლასის
  // განსაზღვრებაში, რომელთანაც მან უნდა იმუშაოს
public static HoursWorked operator + (HoursWorked  a,  HoursWorked  b)
  {
    HoursWorked Result = new HoursWorked( );
    Result RegularHours = a.RegularHours + b.RegularHours;
    Result  OvertimeHours = a.OvertimeHours + b.OvertimeHours;
    return Result;
  }
}
```

გადატვირთულ ოპერატორს პროგრამაში იყენებენ ისევე, როგორც სხვა ჩვეულებრივ ოპერატორს. ქვემოთ ნაჩვენებია კოდის მაგალითი, რომელიც ასრულებს **HoursWorked** სტრუქტურის ორი ეგზემპლარის შეკრებას.

```
 // გათვალისწინებულია, რომ ცვლადები Sunday და Monday
 // შეიცავს HoursWorked სტრუქტურის ეგზემპლარებს, რომლებიც
  // ინიცირებულია შესაბამისი მნიშვნელობებით.
HoursWorked  total = new HoursWorked();
total = Sunday + Monday;
```

გადატვირთული მეთოდების მსგავსად შესაძლებელია გადატვირთული ოპერატორების განსხვავებული რეალიზაციების შექმნა, იმ პირობით, რომ მათი სიგნატურები განსხვავებულია.

დავუშვათ, რომ **HoursWorked** სტრუქტურისათვის დაგვჭირდა int ტიპის მნიშვნელობის დამატება ველისათვის **NormaIHours** “+” ოპერატორის საშუალებით. ასეთი ამოცანის გადაჭრა შეიძლება კიდევ ერთი გადატვირთული “+” ოპერატორის ვერსიის შექმნით.

```
public struct HoursWorked
{
 float RegularHours;
 float OvertimeHours;
 // გადატვირთული ოპერატორის ძველი ვერსია (იხ. წინა მაგალითი)
public static HoursWorked operator + (HoursWorked  a,  HoursWorked  b)
  {
    HoursWorked Result = new HoursWorked( );
    Result RegularHours = a.RegularHours + b.RegularHours;
    Result  OvertimeHours = a.OvertimeHours + b.OvertimeHours;
    return Result;
  }
// გადატვირთული ოპერატორის ახალი რეალიზაციის ვერსია
public static HoursWorked operator + (HoursWorked  a,  int  b)
  {
    HoursWorked Result = new HoursWorked( );
    Result RegularHours = a. RegularHours + b;
    Result  OvertimeHours = a. OvertimeHours + B. OvertimeHours;
    return Result;
  }
}
```

ამგვარად, ვიზუალ C# გადატვირთული ოპერატორის შესაქმნელად საჭიროა შემდეგი მოქმედებები:

1. გამოცხადდეს გადატვირთული ოპერატორი კლასის ან სტრუქტურის განსაზღვრების შიგნით, რომელთანაც უნდა იმუშაოს მან, გასაღებური სიტყვით operator. გადატვირთული ოპერატორი აუცილებლად უნდა გამოცხადდეს public და static მოდიფიკატორებით. თუ ოპერატორი ბინარულია, მაშინ მის ერთ არგუმენტს მაინც უნდა ჰქონდეს იგივე ტიპი, რაც აქვს დასაბრუნებელ მნიშვნელობას.

2. დაიწეროს ოპერატორის რეალიზაცია.

➢ პოლიმორფიზმის რეალიზაცია ინტერფეისებით

ინტერფეისები განსაზღვრავს კლასის ყოფაქცევას. ერთი და იგივე ინტერფეისი შეიძლება რეალიზებულ იქნას სხვადასხვა კლასებში, რაც უზრუნველყოფს მათ ურთიერთმოქმედების შესაძლებლობას, ამგვარად პოლიმორ-ფიზმის კონცეფციის განხორციელებას.

ამ პარაგრაფში განვიხილავთ ინტერფეისების გამოცხა-დებისა და რეალიზების საკითხებს, აგრეთვე თუ როგორ ურთიერთმოქმედებს ობიექტები ინტერფეისების საშუალებით.

ჩვენ კონტექსტში ინტერფეისი არის გარკვეული შეთანხმება. ნებისმიერი ობიექტი, რომელშიც რეალიზებულია მოცემული ინტერფეისი, გარანტირებულად შეიცავს რეალიზა-ციას წევრებისათვის, რომლებიც გამოცხადებულია ამ ინტერფეისში. თუ ობიექტს აქვს რომელიმე ინტერფეისი, მაშინ ნებისმიერი სხვა ობიექტი, რომელშიც რეალიზებულია ეს ინტერფეისი, ფლობს შესაძლებლობას იმოქმედოს მასთან.

ინტერფეისი განსაზღვრავს მხოლოდ წევრებს, რომლებიც ობიექტშია რეალიზებული. თვით ინტერფეისის განსაზღვრებაში არაფერია ნათქვამი მისი წევრების რეალიზაციაზე. იგი შეიცავს მხოლოდ პარამეტრებისა და დასაბრუნებელ მნიშვნელობათა ტიპების აღწერას. ინტერფეისში გამოცხადებული წევრების

რეალიზაცია მთლიანად და სრულად ეკისრება კლასს, რომელშიც თვით ეს ინტერფეისია რეალიზებული.

ამგვარად, სხვადასხვა ობიექტებში ინტერფეისის ერთიდაიგივე წევრები რეალიზებულია სრულიად განსხვავებულად.

განვიხილოთ, მაგალითად, ინტერფეისი IShape, რომელიც განსაზღვრავს ერთადერთ მეთოდს – CalculateArea. ამასთანავე კლასი Circle, რომელშიც რეალიზებულია ეს ინტერფეისი, გაითვლის ფიგურის ფართობს სრულიად განსხვავებულად, ვიდრე კლასი Square, რომელშიც ასევე რეალიზებულია ეს ინტერფეისი. მიუხედავად ამისა, ობიექტს, რომელსაც სჭირდება ურთიერთქმედება IShape–ს გამოყენებით, შეუძლია მეთოდის CalculateArea გამოძახება Circle ან Square კლასიდან და მიიღოს კორექტული შედეგი.

- **ინტერფეისების განსაზღვრა:**

ინტერფეისი განისაზღვრება გასაღებური სიტყვით Interface. მაგალითად:

```
public interface IDrivable
{
 . . .
 }
```

აქ გამოცხადებულია **IDrivable** ინტერფეისი წევრების გარეშე.

ინტერფეისის წევრი-მეთოდები განისაზღვრება მეთოდის ჩვეულებრივი სიგნატურით, ოღონდ წვდომის მოდიფიკატორების (public, private, . . .) გარეშე. მოდიფიკატორი, რომელიც ინტერფეისისთვისაა მოცემული, ვრცელდება მის ყველა წევრზე. განვიხილოთ ინტერფეისის წევრი-მეთოდების გამოცხადების მაგალითი.

```
public interface IDrivable
{
  void GoForward(int Speed);
  void Halt();
   int DistanceTraveled( );
}
```

მეთოდების მსგავსად ინტერფეისში განისაზღვრება წევრი-თვისებებიც. ამისათვის გამოიყენება სპეციალური მეთოდები: მიმღები-მეთოდი (getter) და დამყენებელი-მეთოდი (setter). თვისებათა განსაზღვრა მთავრდება მნიშვნელობის ტიპის მითითებით, რომელსაც თვისება აბრუნებს. მაგალითად:

```
public interface IDrivable
{
  // სხვა წევრების განსაზღვრა აქ გამოტოვილია
  int FuelLevel
  {
    get;
      // ეს თვისება რომ იყოს მისაწვდომი მხოლოდ წასაკითხად, უნდა
      // გამოცხადდეს აქ მეთოდი set
  }
}
```

თვისებებისგან განსხვავებით, ველები არ შეიძლება იყოს ინტერფეისის წევრები. ესაა გარანტია იმისა, რომ კლასები, რომლებიც ინტერფეისების საშუალებით ურთიერთმოქმედებს, ვერ განახორციელებს მიმართვას ობიექტის შიგა მონაცემებთან.

ინტერფეისები განსაზღვრავს აგრეთვე მოვლენებს, რომლებსაც ის ობიექტები აგენერირებს, რომლებშიც ეს ინტერფეისია რეალიზებული. ნებისმიერი კლასი, რომელიც ინტერფეისს უკეთებს რეალიზაციას, ვალდებულია მისცეს რეალიზაციის საშუალება ამ ინტერფეისის ყველა წევრ-მოვლენას. მაგრამ ობიექტებისთვის, რომლებიც ამ ინტერფეისის საშუალებით ურთიერთმოქმედებს, არასავალდებულოა მასში გამოცხადებული ყველა მოვლენის დამუშავება. C# მოითხოვს მოვლენებისთვის ტიპი-დელეგატების დანიშვნას. მაგალითად:

```
public interface IDrivable
{
  // სხვა წევრების განსაზღვრა აქ გამოტოვილია
  event System.EventHandler OutOfFuel;
}
```

ინტერფეისის განსაზღვრა ხდება გასაღებური სიტყვით Interface და ემატება მისი წევრების: მეთოდების, თვისებების და მოვლენების სიგნატურები.

- **ინტერფეისები - პოლიმორფიზმის მიღწევის საშუალება:**

ნებისმიერი ობიექტი, რომელშიც რეალიზაბულია რომელიმე ინტერფეისი, შეუძლია ურთიერთქმედება სხვა ობიექტებთან, რომელთაც სჭირდებათ ეს ინტერფეისი. მაგალითად:

```
public void GoSomewhere(IDrivable v)
{
  // რეალიზაცია გამოტოვილია
}
```

აქ გამოცხადებულ მეთოდისთვის აუცილებელია, რომ მის არგუმენტში იყოს რეალიზებული **IDrivable** ინტერფეისი. შედეგად, ამ მეთოდს შეიძლება გადაეცეს ნებისმიერი ობიექტი, რომელშიც რეალიზებულია ეს ინტერფეისი, ამავდროულად ეს ეს ობიექტი არაცხადად გარდაიქმნება ამ ინტერფეისის ტიპად. ობიექტებისთვის, რომლებიც ურთიერთქმედებს ერთმენეთთან მათში რეალიზებული ინტერფეისის საშუალებით, ხელმისაწვდომია მხოლოდ ამ ინტერფეისის წევრები.

გარდა ამისა, ნებადართულია ობიექტების ცხადი სახით გარდაქმნა ინტერფეისის ტიპად, რომლებშიც რეალიზებულია რომელიმე ინტერფეისი. მაგალითში ნაჩვენებია Truck ობიექტის გარდაქმნა **IDrivable** ინტერფეისის ტიპად. იმისათვის რომ ეს პროცედურა შესრულდეს, საჭიროა Truck ობიექტში რეალიზებულ იყოს **IDrivable** ინტერფეისი.

```
Truck myTruck = new Truck();
Idivable myVehicle;
// Truck ობიექტის გარდაქმნა IDrivable ინტერფეისის ტიპად
myVehicle = (IDivable)myTruck;
```

- **ინტერფეისების რეალიზაცია:**

C#-ის პროგრამაში ინტერფეისის სარეალიზაციოდ გამოიყენება “ : “. მაგალითად,

```
public class Truck : IDrivable
{  // რეალიზაცია გამოტოვილია
}
```

კლასში შესაძლებელია რამდენიმე ინტერფეისის რეალიზაცია. ასეთი კლასის გამოცხადების დროს საჭიროა ინტერფეისების მითითება მძიმის საშუალებით:

```
public class Truck : IDrivable, IFuelBurning, ICargoCarrying
{
 // რეალიზაცია გამოტოვილია
}
```

თუ კლასში ან სტრუქტურაში რეალიზებულია რომელიმე ინტერფეისი, აუცილებელია მიეცეს რეალიზაციის შესაძლებლობა მის ყველა წევრს. ეს წესი ვრცელდება მაშინაც, თუ რელიზებულია რამდენიმე ინტერფეისი.

- **ინტერფეისის წევრების რეალიზაცია:**

C#-ის კლასებსა და სტრუქტურებში გამოცხადებული ინტერფეისის წევრების რეალიზაცია ხდება შემდეგნაირად. ამისათვის განსაზღვრავენ წევრს სახელით, რომელიც ემთხვევა ინტერფეისის წევრის სახელს. ამავდროულად, კლასის ეს წევრი ცხადდება წვდომის იმავე დონით, რომლითაც გამოცხადებულია ინტერფეისი. მომდევნო მაგალითი ასახავს მეთოდის რეალიზაციას, როგორც ინტერფეისის წევრისა:

```
public interface IDrivable
{
 void GoForward(int Speed);
}
public class Truck : IDrivable
{
   public  void GoForward(int Speed)
     {
      // რეალიზაცია გამოტოვილია        }
}
```

ასეთი სახით რეალიზებული ინტერფეისის წევრები მისაწვდომია როგორც ინტერფეისის, ასევე კლასისთვის. შედეგად, ამ წევრების გამოძახება შესაძლებელია ობიექტის

გარდაქმნის შემდეგ (რომელშიც ინტერფეისია რეალიზებული) მის საკუთარ ტიპში ან ამ ინტერფეისის ტიპში.

გარდა ამისა, ინტერფეისის რეალიზაცია კლასში შეიძლება ცხადად. ასეთი სახით რეალიზებული წევრები მისაწვდომია მხოლოდ ობიექტის გარდაქმნის შემდეგ ინტერფეისის ტიპში. ინტერფეისის წევრის ცხადი სახით რეალიზაციისთვის საჭიროა კლასში გამოცხადდეს იმავე სახელის წევრი, ინტერფეისის სრული სახელის მითითებით. ქვემოთ ნაჩვენებია ცხადად რეალიზებული GoForward მეთოდი, რომელიც არის IDrivable ინტერფეისის წევრი.

```
public class Truck : IDrivable
{
   void  IDrivable.GoForward(int Speed)
     {
      // რეალიზაცია გამოტოვილია
     }
}
```

ვინაიდან ეს წევრი რეალიზებულია ცხადად, მისი წვდომის დონე განისაზღვრება თვით ინტერფეისის მოდიფიკატორით.

ამგვარად, C#-ის პროგრამაში ინტერფეისის რეალიზაციის-თვის საჭიროა:

1. გამოცხადდეს კლასი საჭირო ინტერფეის(ებ)ით " : " -ის გამოყენებით;

2. დაიწეროს რეალიზაცია ინტერფეისის ყველა წევრისთვის:

- იმისათვის, რომ წევრი იყოს მისაწვდომი კლასისა და ინტერფეისისთვის, იგი უნდა გამოცხადდეს იმავე სახელით, წვდომის დონით და სიგნატურით, როგორიც აქვს ინტერფეისის შესაბამის წევრებს;

- იმისათვის, რომ წევრი გავხადოთ მისაწვდომი მხოლოდ ინტერფეისიდან, იგი უნდა გამოცხადდეს ინტერფეისის სრული სახელით (მოდიფიკატორი არაა სავალდებულო).

➢ პოლიმორფიზმის რეალიზაცია მემკვიდრეობითობით

მემკვიდრეობითობა საშუალებას იძლევა გამოცხადდეს ახალი კლასი არსებული კლასის საფუძველზე და გადასცეს ახალ კლასს მისი ყველა წევრი და ფუნქციონალობა. ამგვარად შესაძლებელია კლასების შექმნა, რომლებშიც რეალიზებულ იქნება საბაზო შესაძლებლობების ერთობლიობა, ხოლო შემდეგ გამოყენებულ იქნას ისინი წარმოებული კლასების გამოსაცხადებლად, სხვადასხვა, მაგრამ დაკავშირებული ფუნქციების შესასრულებლად.

წინამდებარე პარაგრაფში გავეცნობით მემკვიდრეობითობის გამოყენებას წარმოებული კლასების შესაქმნელად საბაზო კლასების საფუძველზე, ახალი რეალიზაციის შექმნას საბაზო კლასების წევრებისთვის, აგრეთვე აბსტრაქტული საბაზო კლასების გამოცხადებას.

- **მემკვიდრეობითობა:**

მემკვიდრეობითობა გვაძლევს საშუალებას შეიქმნას კლასები, რომლებიც რეალიზებას უკეთებს საერთო (ზოგად) შესაძლებლობათა ერთობლიობას. ამასთანავე, სპეციალიზებული კლასები, რომელთაც წარმოებულ კლასებსაც უწოდებენ, შთამომავლებია ერთი ზოგადი კლასის, რომელიც საბაზო კლასია. ამბობენ, რომ წარმოებული კლასები აფართოებენ საბაზო კლასის შესაძლებლობებს. საბაზო კლასში ინკაფსულირებულია ზოგადი ფუნქციონალობა, რომელიც გააჩნია ყველა მისგან წარმოებულ კლასს. ამასთანავე, წარმოებულ კლასებს აქვს დამატებითი შესაძლებლობები, რომლებიც უნიკალურია თვითოეული წარმოებული კლასისთვის.

მაგალითად, კლასი ავტომობილი ზოგადია და იგი ფლობს საბაზო კლასის ყველა ფუნქციონალობას (ობიექტებს, მეთოდებს, მოვლენებს). მის საფუძველზე შეიძლება შევქმნათ წარმოებული კლასი სატვირთო ავტომობილი, რომელიც მემკვიდრეობით შეიძენს საბაზოსგან მთლიან ფუნქციონალობას. დამატებითი შესაძლებლობების სახით მასში რეალიზებულ იქნება ისეთი თვისებები, რომლებიც მხოლოდ მისთვისაა დამახასიათებელი, მაგალითად, ტვირთის–სახე, ტვირთამწეობა და ა.შ.

- **პოლიმორფიზმი და წარმოებული კლასები:**

წარმოებული კლასი შეიძლება გავაიგივოთ მის საბაზო კლასთან. მაგალითად, კლასის *სატვირთო ავტომობილი* ყველა ობიექტი არის ამავდროულად ობიექტი კლასისა *ავტომობილი.*

პოლიმორფიზმი მემკვიდრეობითობის დროს მდგომარეობს იმაში, რომ წარმოებული კლასის ნებისმიერი ეგზემპლარი მზადაა შეასრულოს მისი საბაზო კლასის ფუნქციები. ნებისმიერი წარმოებული კლასი შეიძლება არაცხადად გარდაიქმნას თავისი საბაზო კლასის ობიექტში, ამასთანავე ყველა წევრი, რეალიზებული წარმოებულ კლასში მიუწვდომელი იქნება. მისაწვდომი იქნება მხოლოდ საბაზო კლასის წევრები.

- **წარმოებული კლასების შექმნა:**

წარმოებულ კლასებს აცხადებენ “ : “ სიმბოლოთი. მაგალითად:

```
public class PickupTruck : Truck
{
 // რეალიზაცია გამოტოვებულია
}
```

წარმოებულ კლასს მხოლოდ ერთი წინაპარი ჰყავს (საბაზო კლასი), მაგრამ მასში დამატებით დასაშვებია ერთი ან რამდენიმე ინტერფეისის რეალიზება. თუ ვაცხადებთ წარმოებულ კლასს, რომელშიც რამდენიმე ინტერფეისია რეალიზებული, მაშინ ისინი უნდა ჩამოითვალოს მიმდევრობით (“ , “-ებით) “ : საბაზო–კლასის–სახელი”-ის შემდეგ. მაგალითად:

```
public class FourwheeIDrivePickupTruck : PickupTruck,
IFourwheeIDrive
{
 // რეალიზაცია გამოტოვებულია
}
```

წარმოებული კლასის გამოცხადების შემდეგ შეიძლება დაემატოს მას ახალი წევრები, რომლებიც არასტანდარტულ შესაძლებლობებს არეალიზებს.

- **ჩაბეჭდილი კლასების შექმნა:**

ზოგჯერ საჭიროა ისეთი კლასები, რომლებიც ვერ შეასრულებს საბაზო კლასის როლს. მაგალითად, სპეციალიზებული კლასები, რომლებიც საჭიროა უშუალოდ მომხმარებლის აპლიკაციის კომპონენტების სახით გამოსაყენებლად. ისინი გამოუსადეგარი იქნება სხვა პროგრამისტებისთვის, ვინაიდან მათ არ შეუძლია გადასცეს თავის შთამომავლებს რაიმე სასარგებლო ფუნქციონალობა. ასეთი კლასები ცხადდება გასაღებური სიტყვით sealed. მაგალითად:

```
public sealed class AClass
{
 // რეალიზაცია გამოტოვებულია
}
```

- **მემკვიდრეობით გადაცემული წევრები:**

როგორც აღნიშნული იყო, წარმოებული კლასი ფლობს მისი საბაზო კლასის მთელ ფუნქციონალობას. წარმოებულ კლასში შეიძლება არა მხოლოდ ახალი წევრების ჩამატება, არამედ საბაზოდან მემკვიდრეობით მიღებული წევრების რეალიზაციის შეცვლა დასმული ამოცანების გადასაწყვეტად.

საბაზო კლასიდან მემკვიდრეობით მიღებული წევრების განსაზღვრების შეცვლით (override), ხერხდება მათი რეალიზაციის შეცვლა.

C# ენაში შესაძლებელია წევრების დამალვა (hide), რომლებიც მემკვიდრეობითაა მიღებული საბაზო კლასიდან. სამაგიეროდ რეალიზდება ახალი წევრი იმავე სახელით და სიგნატურით, ოღონდაც სხვა მახასიათებლებით განსხვავებული.

განსაზღვრების შეცვლა შეიძლება მხოლოდ თვისებებისა და მეთოდებისათვის, რომლებიც მიღებულია საბაზო

კლასიდან. ცვლადებისა და მოვლენებისათვის განსაზღვრების შეცვლა დაუშვებელია.

წევრის ახალ რეალიზაციაში სიგნატურა, დასაბრუნებელი მნიშვნელობის ტიპი და წვდომის დონე უნდა ემთხვეოდეს განსაზღვრებაშეცვლილ წევრის იმავე მნიშვნელობებს. მაგალითად:

```
// საბაზო კლასს აქვს GoForward მეთოდი
public class SportsCar : Car
{
  public override void GoForward(int Speed)
   {
     // რეალიზაცია გამოტოვებულია
   }
}
```

განსაზღვრებაშეცვლილი წევრის გამოძახებისას იმ წევრის მაგივრად, რომელიც საბაზო კლასიდანაა მემკვიდრეობით მიღებული, გამოიძახება ამ წევრის ახალი რეალიზაცია, გამოძახების კონტექსტისგან დამოუკიდებლად. მაგალითად, თუ წარმოებული კლასის ეგზემპლარის გარდაქმნის შემდეგ მისი საბაზო კლასის ტიპში გამოძახებულ იქნება განსაზღვრბაშეცვლილი მეთოდი, მაინც შესრულდება ამ მეთოდის ახალი რეალიზაცია. გამოძახებული წევრი განისაზღვრება ობიექტით და არა ცვლადის ტიპით.

საბაზო კლასის წევრის განსაზღვრების შესაცვლელად იგი უნდა მოინიშნოს გასაღებური სიტყვით virtual, წინააღმდეგ შემთხვევაში წევრები ითვლება ფიქსირებულად და მათი განსაზღვრების შეცვლა არ შეიძლება. მაგალითად:

```
public virtual void OverrideMe()
{
  // რეალიზაცია გამოტოვებულია
}
```

- **საბაზო კლასის წევრების დამალვა C#-ში.**

როგორც აღვნიშნეთ, C#-ის პროგრამაში შესაძლებელია საბაზო კლასიდან მემკვიდრეობით მიღებული წევრის შეცვლა

სხვა წევრით, რომელსაც სრულიად განსხვავებული რეალიზაცია ექნება. ამ ხერხს უწოდებენ “დამალვას” (hiding).

ახალი წევრის ტიპი და სიგნატურა უნდა იყოს იგივე, როგორიც ჩასანაცვლებელ დამალულ წევრს აქვს, მაგრამ წვდომის დონე, დასაბრუნებელ მნიშვნელობის ტიპი და რეალიზაცია შეიძლება განსხვავდებოდეს. თუ ახალი მეთოდის სახელი ემთხვევა არსებულიდან ერთ-ერთს, მაგრამ მათი სიგნატურები განსხვავდება, მაშინ ახალი მეთოდი განიხილება როგორც არსებული მეთოდის გადატვირთული ვერსია, ამასთანავე ეს უკანასკნელი რჩება მისაწვდომი. იმისათვის, რომ დავმალოთ საბაზო კლასიდან მემკვიდრეობითმიღებული წევრი, აუცილებელია გასაღებური სიტყვის new გამოყენება. მაგალითად:

```
public class MyBaseClass    // საბაზო კლასი
{
  public string MyMethod(int I)
    {
      // რეალიზაცია გამოტოვებულია
    }
}
public class MyInheritedClass : MyBaseClass // წარმოებული კლასი
{
  // ეს ფუნქცია ჩაანაცვლებს MyMethod მეთოდს საბაზო კლასიდან.
  // მას იგივე სიგნატურა, მაგრამ სხვა წვდომის დონე და
  // დასაბრუნებელი მნიშვნელობის ტიპი აქვს
  internal new int MyMethod(int  I)
    {
      // რეალიზაცია გამოტოვებულია
    }
}
```

- **თავსებადობის განხორციელება დამალულ წევრებზე:**

საბაზო კლასის დამალული წევრის რეალიზაცია ხდება მიუწვდომელი, მას ცვლის ახალი რეალიზაცია, შესაძლოა სულ სხვა მახასიათებლებით, რაც ზოგჯერ მნიშვნელოვან გავლენას ახდენს სხვა ობიექტებთან ურთიერთქმედების დროს.

ობიექტი, რომელიც იძახებს ფუნქციას MyMethod (იხ. წინა მაგალითი), ელოდება მნიშვნელობას string ტიპით, მაგრამ თუ ფუნქცია დააბრუნებს int–ს, მაშინ მოსალოდნელია შეცდომა. ამიტომაც, ძალზე საფრთხილოა დამალული წევრების გამოყენება და მხოლოდ მაშინ უნდა მივმართოთ მას, როცა უეჭველი გარანტიაა, რომ თავსებადობა არ დაირღვევა.

რიგ შემთხვევებში საბაზო კლასის დამალულ წევრების რეალიზაციაზე მიმართვა მაინც ხერხდება. საიდან იქნება წევრი გამოძახებული – საბაზოდან თუ წარმოებულიდან, დამოკიდებულია ცვლადის ტიპზე და არა ობიექტზე, როგორც ამას ადგილი ჰქონდა განსაზღვრებაშეცვლილი წევრის შემთხვევაში. მაგალითად:

```
// გამოიყენება კლასები MyBaseClass და MyInheritedClass
// (იხ. წინა კოდი)
MyInheritedClass  X = new  MyInheritedClass( );
MyBaseClass Y;
// ცვლადები X და Y მიუთითებს ახლა ერთიდაიმავე ობიექტს,
// მაგრამ მათი ტიპები განსხვავებულია.
Y = X;
// X -ის ცვლადის ტიპია MyInheritedClass, ამიტომ მოიმდეგნო
// სტიქონით გამოიძახება რეალიზაცია წარმოებული კლასიდან.
Y.MyMethod(42);
// Y -ის ცვლადის ტიპია MyBaseClass, ამიტომ მოიმდეგნო სტიქონით
// გამოიძახება რეალიზაცია საბაზი კლასიდან.
Y.MyMethod(42);
```

როგორც ვხედავთ, უშუალოდ ცვლადის ტიპი განსაზღვრავს, თუ რომელი წევრი იქნება გამოძახებული, დამალული – წარმოებულიდან, თუ საწყისი – საბაზოდან. ეს უზრუნველყოფს წევრის რეალიზაციის მოდიფიცირებას პოლიმორფიზმის პრინციპის დაურღვევლად.

წარმოებული კლასების გამოცხადებისას დამალული წევრების ახალი რეალიზაცია არ გადაიცემა მემკვიდრეობით, წარმოებულ კლასს გადაეცემა საბაზო კლასის შესაბამისი წევრების რეალიზაცია.

- **საბაზო კლასის წევრებზე წვდომის მიღება:**

განსაზღვრებაშეცვლილი ან დამალული წევრების გამოცხადებისას ზოგჯერ საჭიროა წვდომა საბაზო კლასის ამ წევრის რეალიზაციაზე. ამისათვის იყენებენ გასაღებურ სიტყვას base. მაგალითად:

```
// აქ დემონსტრირებულია საბაზო კლასის რეალიზაციის გამოძახება
// განსაზღვრებაშეცვლილი მეთოდიდან MyMethod( ).
public override void MyMethod( )
    {
      base.MyMethod( );
      // რეალიზაცია გამოტოვებულია
    }
```

- **დაცული წევრები:**

კლასების წევრებზე წვდომა, როგორც ადრე აღვნიშნეთ, ხორციელდება წვდომის დონეებით: public – შესაძლებელია მიმართვა აპლიკაციის ნებისმიერი ადგილიდან, აგრეთვე გარე კლასებიდანაც; internal – წვდომა ხორციელდება მხოლოდ წევრებზე ლოკალური ნაკრებიდან; private – წვდომა შესაძლებელია მხოლოდ კლასის შიგნით. შესძლებელია კიდევ წვდომის ორი დონე: protected და protected internal – რომლებიც არ განხილულა.

კლასების წევრთა ხილვადობის არეები, რომლებიც გამოცხადებულია გასაღებური სიტყვებით protected და private – მსგავსია, ერთი განსხვავებით, რომ პირველ შემთხვევაში წევრები მისაწვდომია წარმოებული კლასებისთვისაც. მაგალითად:

```
// საბაზო კლასში განსაზღვრულია ორი მეთოდი:
// წვდომის დონეებით protected და private
public class BaseClass
    {
      // მეთოდი private-წვდომის დონით არ შეიძლება
      // გამოვიძახოთ წარმოებული კლასებიდან
      private void  Method1( )
        {
          // რეალიზაცია გამოტოვებულია
        }
     // მეთოდი protected-წვდომის დონით შეიძლება
    // გამოვიძახოთ წარმოებული კლასებიდან
```

```
    protected void  Method2( )
     {
      // რეალიზაცია გამოტოვებულია
     }
}
 // კლასი, წარმოებული BaseClass -იდან
public class InheritedClass : BaseClass
{
 public void  Demo( )
   {
     // ასეთი გამოძახება დასაშვებია, რადგან protected არ კრძალავს
     // BaseClass–ის წევრებზე მიმართვას.
     this.Method2( );
     // ასეთი გამოძახება იწვევს კომპილაციის შეცდომას,
     // ვინაიდან გამოყენებულია private წვდომის დონე.
      this.Method1( );
   }
}
```

წვდომის დონე protected internal არის მოდიფიკატორების protected და internal ჰიბრიდი. ამიტომაც კლასის წევრი, რომელიც გამოცხადებულია protected internal –ით, მისაწვდომია ამ კლასის და სხვა კლასის წევრებისთვისაც, რომლებიც ამავე ნაკრებშია ამ კლასთან, აგრეთვე მისი წარმოებული კლასების წევრებისთვისაც.

- **აბსტრაქტული კლასები და წევრები:**

კომპონენტების შექმნის დროს ზოგჯერ საჭიროა კლასები, რომლებიც გვთავაზობს განსაზღვრულ შესაძლებლობათა ერთობლიობას, უცვლელი სახით გამოსაყენებლად. ხოლო ამ კლასის სხვა წევრების რეალიზაციაზე პასუხს აგებენ მათი წარმოებული კლასები. ასეთ შესაძლებლობას ფლობს აბსტრაქტული (აბსტრაცტ) კლასები, რომელთაც შეუძლია მხოლოდ საბაზო კლასების როლის შესრულება.

აბსტრაქტული კლასები ჰგავს ინტერფეისებს, მაგრამ მათ ბევრი საერთო აქვს ჩვეულებრივ კლასებთან. აბსტრაქტული კლასის ეგზემპლარის შექმნა არ შეიძლება, იგი გამოსადეგია მხოლოდ წარმოებული კლასების გამოცხადებისთვის. აბსტრაქტული კლასი იძლევა კლასის სრულ რეალიზაციას, ან მის ნაწილს ან საერთოდ არავითარ რეალიზაციას.

ინტერფეისებიც და აბსტრაქტული კლასებიც შეიცავს წევრების აღწერას, რომლებიც რეალიზებულ უნდა იქნას წარმოებულ კლასებში, ოღონდ ინტერფეისისგან განსხვავებით, მხოლოდ ერთ აბსტრაქტულ კლასს შეუძლია იყოს წარმოებული კლასის წინაპარი. აბსტრაქტული კლასები იძლევა მხოლოდ სრულად რეალიზებულ წევრებს, აგრეთვე იმ წევრებს, რომელთა რეალიზაციაზე პასუხისმგებელია წარმოებული კლასები.

➢ კითხვები და სავარჯიშოები

1. რას წარმოადგენს პროგრამული პლატფორმა და პროგრამული ენა, რა განსხვავებაა მათ შორის ?
2. როგორია პროგრამული ენების განვითარების ტენდენცია ?
3. რა არის დაპროგრამების მეთოდი და სტილი ?
4. დაახასიათეთ .NET პლატფორმა.
5. როგორი კავშირია Windows-ოპერაციულ სისტემასა და .NET პლატფორმებს შორის ?
6. როგორია .NET პლატფორმის არქიტექტურა ?
7. რა არის CTS (Common Types System) ?
8. რა არის CLS (Common Language Specification) ?
9. რა არის IL (Intermediate Language) ?
10. რა არის JIT (Just-In-Time) ?

11. როგორია .NET პლატფორმაზე პროგრამული აპლიკაციის ნაკრების (Assembly) სტრუქტურა ?

12. რა არის სახელსივრცე (namespace) ?

13. რა არის დანართის არე (Application area) და პროცესი ?

14. რომელი პროგრამული პაკეტების ინტეგრირებული .NET Framework 4.0 -ში (2010 ვერსიაში) ?

15. რას წარმოადგენს ობიექტ-ორიენტირებული დაპროგრამების მეთოდი, რა განსხვავებაა კლასსა და ობიექტს შორის ?

16. რა განსხვავებაა ობიექტის წევრებს შორის: თვისებები, მეთოდები, მოვლენები ?

17. რა არის ობიექტური მოდელი ?

18. რა არის ინკაფსულაცია ?

19. რა არის მემკვიდრეობითობა ?

20. რა არის პოლიმორფიზმი ?

21. როგორ ხდება კლასის წევრების გადატვირთვა ?

22. როგორ ხდება ოპერატორის გადატვირთვა ?

23. რა არის ობიექტის ინტერფეისი ?

24. როგორ ხდება პოლიმორფიზმის რეალიზაცია ინტერფეისებით ?

25. როგორ ხდება პოლიმორფიზმის რეალიზაცია მემკვიდრეობითობით ?

26. რა არის საბაზო და წარმოებული კლასები ?

27. როგორ იქმნება წარმოებული კლასი ?

28. როგორ ხდება საბაზო კლასის წევრების დამალვა ?

29. როგორ ხდება საბაზო კლასის წევრებზე წვდომა ?

30. რას წარმოადგენს აბსტრაქტული კლასები და წევრები ?

I ნაწილი: Visual C#.NET 2010 - მართვის ელემენტები

თავი 1. შესავალი ვიზუალური დაპროგრამების ენაში - C#.NET

Visual C# 2010 არის ვიზუალური, ობიექტ-ორიენტირებული ენა, რომელიც მაიკროსოფთის ფირმამ .NET Framework 4.0 ინტეგრირებულ გარემოსთან ერთად, ბოლო ვერსიაში წარმოადგინა, Visual Basic.NET, Visual C++.NET და F#.NET ენებთან ერთად.

Visual C# 2010 (შემდგომში C#) ენა მთლიანად მოიცავს მის წინამორბედს - Visual C# 2008 და ახალ გაფართოებებს.

შესავალში აღწერილია ვიზუალური C# ენის საწყისები და პირველი პროგრამული კოდის აგების ელემენტები ვინდოუს-აპლიკაციისთვის.

პრაქტიკული ექსპერიმენტებისთვის შესაძლებელია Visual C# 2010 Express Edition პაკეტის გამოყენება, რომელიც უფასოა და ინტერნეტიდან მისაწვდომი:
http://www.microsoft.com/visualstudio/en-us/products/2010-editions/visual-csharp-express

იგი მოიცავს ტექსტურ რედაქტორს პროგრამული კოდის ასაგებად, კომპილატორს - მის გასამართად და დებაგერს - შეცდომების აღმოსაჩენად.

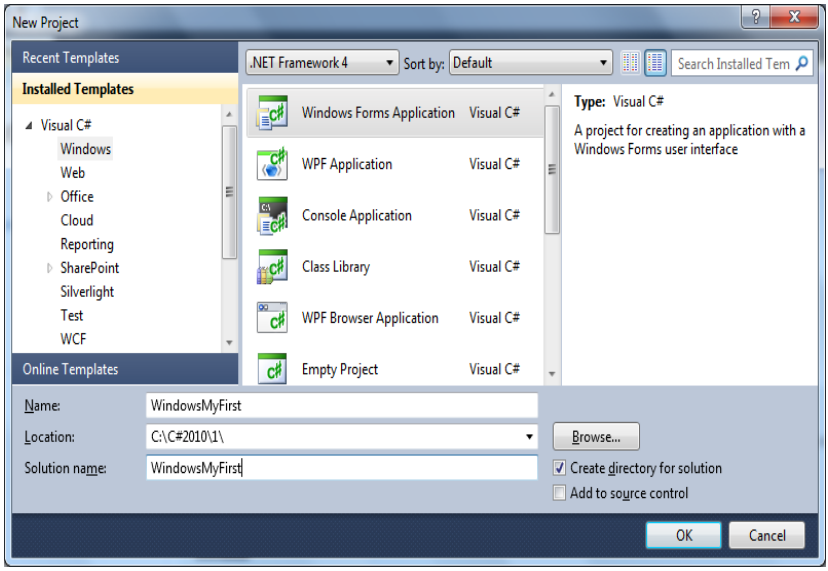
1.1. სამუშაო გარემო და პირველი ვინდოუს-პროგრამა

სამუშაო გარემოდან (ნახ.1.1) აირჩევა New Project (თუ ახალი პროგრამა იქმნება), განისაზღვრება პროექტის დასახელება და შენახვის კატალოგი (ნახ.1.2).

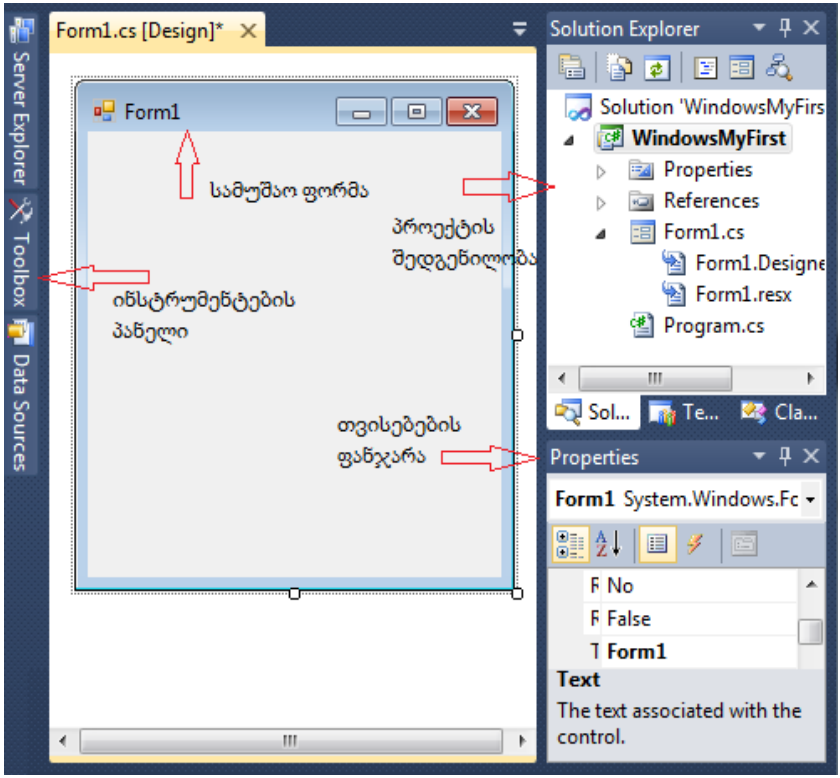
შედეგად მიიღება მომხმარებლის სამუშაო ფორმა-Form1 (ნახ.1.3) და ToolBox-ინსტრუმენტების პანელი (ნახ.1.4), საიდანაც ხდება პროგრამისთვის საჭირო მართვის ელემენტის გადატანა ფორმაზე. 1.3 ნახაზის ქვედა მარჯვენა ნაწილში მოთავსებულია ფორმის ელემენტების თვისებათა (Properties) ფანჯარა (იხ. დამატება_1.1). ის ცალსახად აღწერს ფორმის ან მისი მონიშნილი ელემენტ(ებ)ის თვისებებს.


Начальная страница
Start Page
Server Explorer
Toolbox
Microsoft
Visual Studio 2010 Professional
Connect To Team Foundation Server
New Project...
Open Project...
Recent Projects
Get Started
Guidance and Resources
Latest News
Welcome
Windows
Web
Cloud
Office
SharePoint
Data
What's New in Windows Development
Creating Windows Applications
Using the right Windows technology will de
different technologies to find the one that
Overview of Windows Client Applications
Creating WPF Applications
Creating Windows Forms Applications

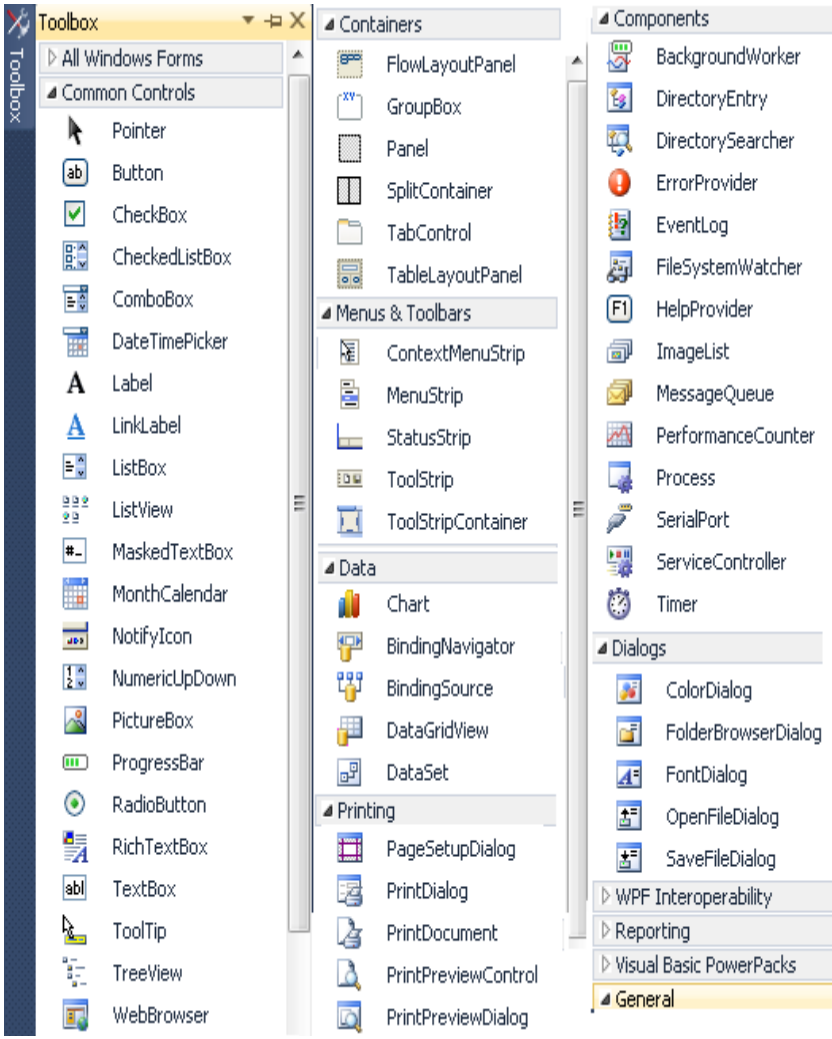
ნახ.1.1


New Project
Recent Templates
Installed Templates
Visual C#
Windows
Web
Office
Cloud
Reporting
SharePoint
Silverlight
Test
WCF
Online Templates
.NET Framework 4
Sort by:
Default
Search Installed Tem
Windows Forms Application
WPF Application
Console Application
Class Library
WPF Browser Application
Empty Project
Visual C#
Type: Visual C#
A project for creating an application with a
Windows Forms user interface
Name:
WindowsMyFirst
Location:
C:\C#2010\1\
Solution name:
WindowsMyFirst
Browse...
Create directory for solution
Add to source control
OK
Cancel

ნახ.1.2

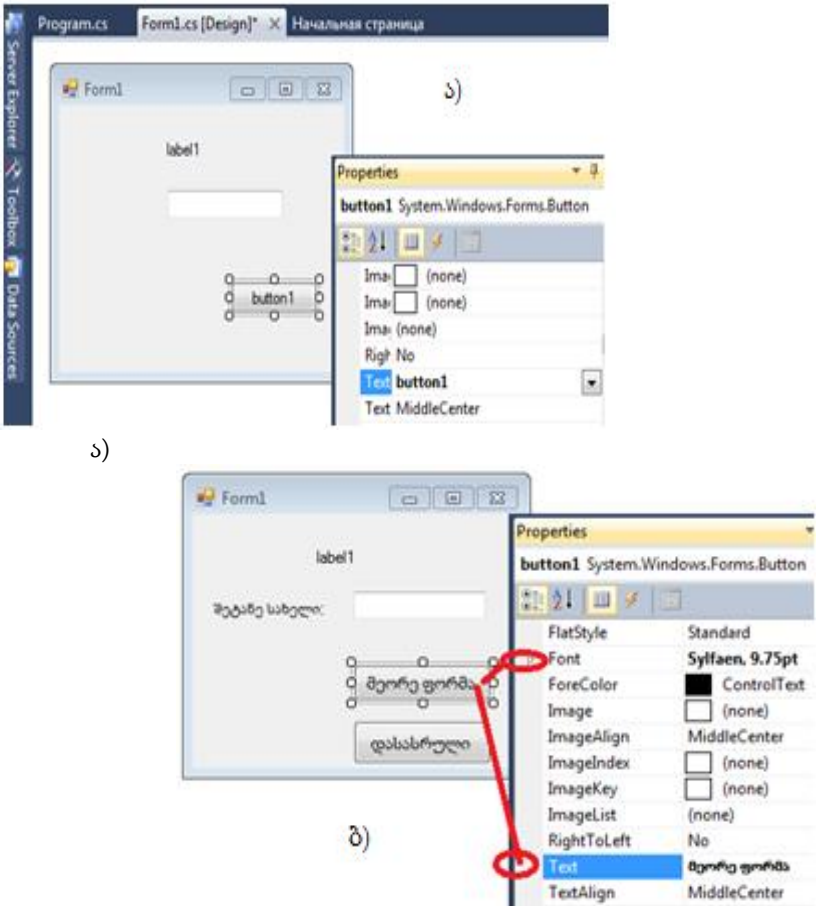

Server Explorer
Toolbox
Data Sources
Form1.cs [Design]*
Form1
სამუშაო ფორმა
პროექტის
იწსტრუმენტების
პანელი
თვისებების
ფანჯარა
Solution Explorer
Solution 'WindowsMyFirs
WindowsMyFirst
Properties
References
Form1.cs
Form1.Designe
Form1.resx
Program.cs
Sol...
Te...
Cla...
Properties
Form1 System.Windows.Fc
No
False
Form1
Text
The text associated with the
control.

ნახ.1.3


Toolbox
Toolbox
All Windows Forms
Common Controls
Pointer
Button
CheckBox
CheckedListBox
ComboBox
DateTimePicker
Label
LinkLabel
ListBox
ListView
MaskedTextBox
MonthCalendar
NotifyIcon
NumericUpDown
PictureBox
ProgressBar
RadioButton
RichTextBox
TextBox
ToolTip
TreeView
WebBrowser
Containers
FlowLayoutPanel
GroupBox
Panel
SplitContainer
TabControl
TableLayoutPanel
Menus & Toolbars
ContextMenuStrip
MenuStrip
StatusStrip
ToolStrip
ToolStripContainer
Data
Chart
BindingNavigator
BindingSource
DataGridView
DataSet
Printing
PageSetupDialog
PrintDialog
PrintDocument
PrintPreviewControl
PrintPreviewDialog
Components
BackgroundWorker
DirectoryEntry
DirectorySearcher
ErrorProvider
EventLog
FileSystemWatcher
HelpProvider
ImageList
MessageQueue
PerformanceCounter
Process
SerialPort
ServiceController
Timer
Dialogs
ColorDialog
FolderBrowserDialog
FontDialog
OpenFileDialog
SaveFileDialog
WPF Interoperability
Reporting
Visual Basic PowerPacks
General

ნახ.1.4

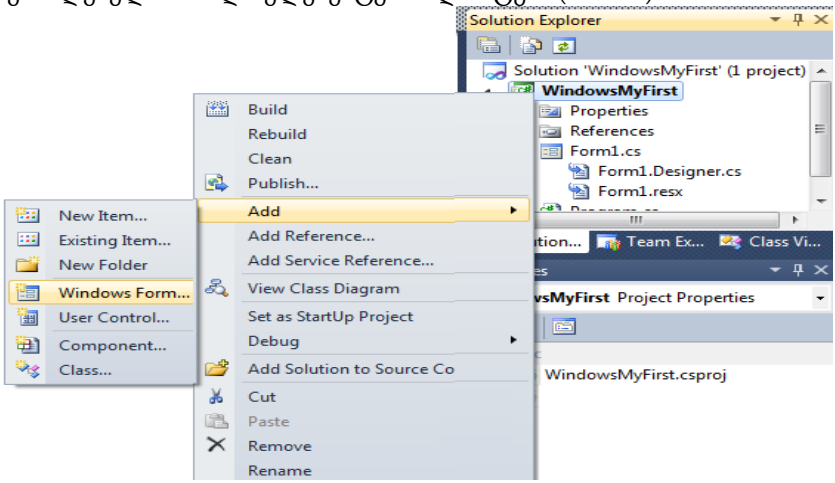
მაგალითად, 1.5-ა და ბ ნახაზებზე ილუსტრირებულია ფორმაზე გამოსატანი ტექსტის (label1), შესატანი სტრიქონული ველის (textBox1) და ღილაკის (button1) შესაბამისი ელემენტები, საწყისი (ა) და თვისებებიდან შეცვლილი მნიშვნელობებით (ბ).



ნახ.1.5. ა-საწყისი და ბ-საბოლოო მდგომარეობები

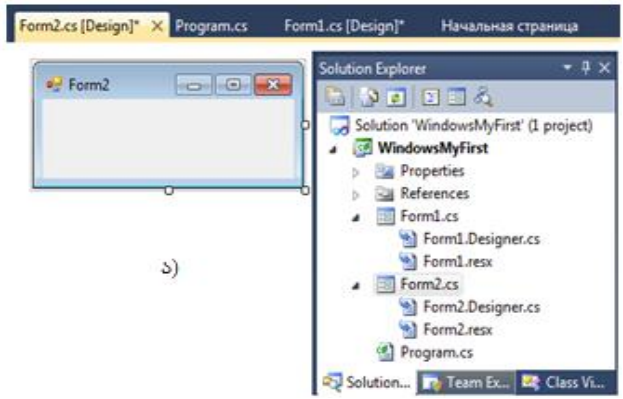
პროექტის შედგენილობა ასახულია Solution Explorer - ფანჯარაში, სადაც იერარქიული ხის სტრუქტურით

განთავსებულია მისი შემადგენელი კომპონენტები: ფორმები, რესურსები, პროგრამული კოდები (Program.cs) და ა.შ. აქ შესაძლებელია ახალი ელემენტების დამატება (ნახ.1.6).



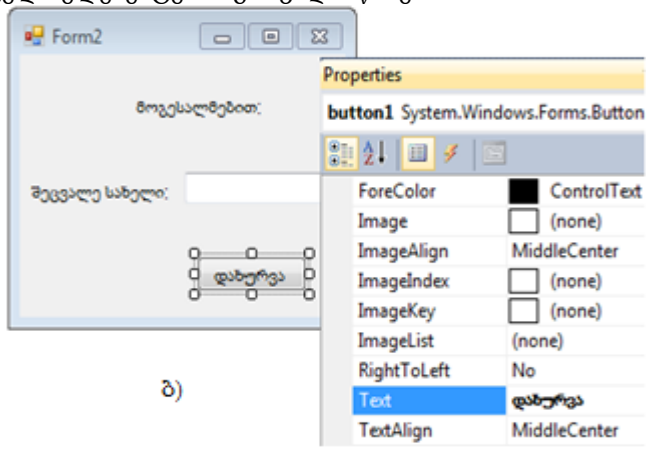
ნახ.1.6

ჩვენს შემთხვევაში პროექტს დაემატება Windows Form ტიპის Form2 ელემენტი, რომელსაც Solution Explorer-ში ექნება ასეთი სახე (ნახ.1.7).



ნახ.1.7. ა-საწყისი მდგომარეობა

ამგვარად, Windows-ფორმა (Form) ხილული ზედაპირია („ფანჯარა“), რომელზეც აისახება ინფორმაცია მომხმარებლისთვის. Windows-ფორმების აპლიკაცია (დანართი) იქმნება მმართველი ელემენტების (ნახ.1.4) გადმოტანით ფორმაზე. პროექტის პროგრამული ფრაგმენტები იქმნება ავტომატურად, არჩეული მმართველი ელემენტების გათვალისწინებით.



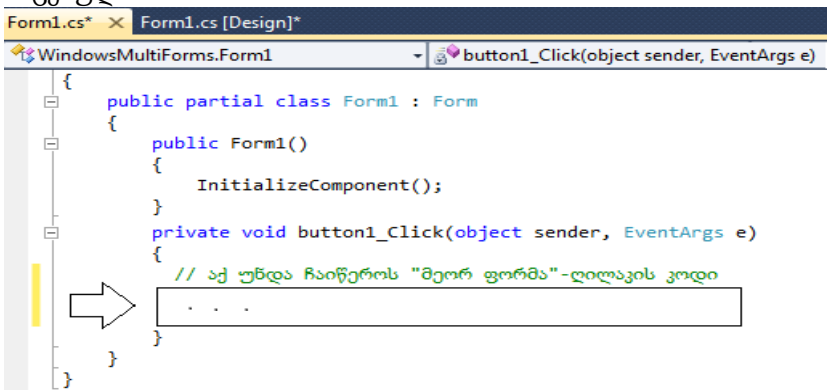
ნახ.1.7. ბ-საბოლოო: დაემატა ახალი ელემენტები

ამ ელემენტებიდან, ზოგიერთი, მაგალითად, „ღილაკი“ თხოულობს მისთვის შესაბამისი რეაგირების კოდის ხელით ჩაწერას, ანუ რა პროცედურა უნდა შესრულდეს ამ ღილაკზე მაუსის დაწკაპუნებით.

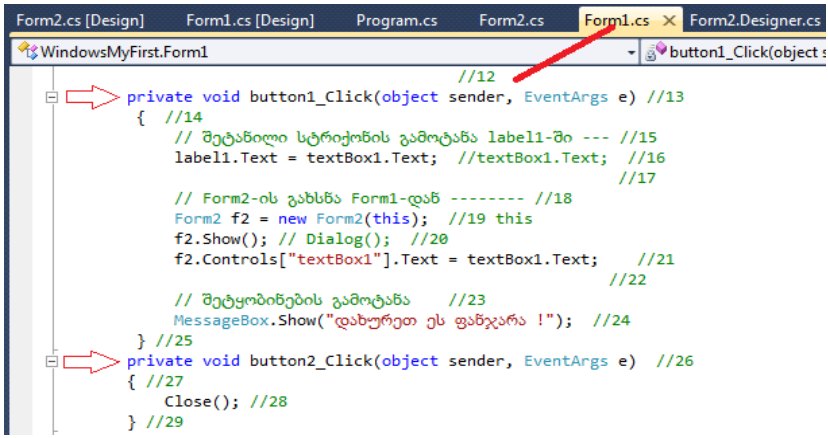
მაგალითად, დავსვათ ასეთი ამოცანა: Form1-დან „მეორ ფორმა“-ღილაკის დაჭერით label1-ში გამოჩნდეს textBox-ში ხელით შეტანილი სტრიქონის მნიშვნელობა, გაიხსნას Form2, მის textBox-ში აისახოს Form1-ის სტრიქონის მნიშვნელობა (ანუ მოხდეს ერთი ფორმიდან მეორეში მონაცემის გადაცემა) და ცალკე, შეტყობინების ფანჯარაში (MessageBox) გამოჩნდეს ტექსტი „დახურეთ ეს ფანჯარა“. MessageBox ფანჯრის დახურვის შემდეგ აქტიური ხდება Form2 ფანჯარა. აქ შესაძლებელია textBox-ში სტრიქონის მნიშვნელობის ხელით კორექტირება (ახალი სიტყვის ჩაწერაც), შემდეგ „დახურვა“-ღილაკის ამოქმედებით მართვა უბრუნდება

Form1-ს და მის textBox-ში გამოჩნდება Form2-ში კორექტირებული სტრიქონი (ანუ ხდება მონაცემის უკან დაბრუნება). Form1-ის „დასასრული“-ღილაკით პროგრამა ამთავრებს მუშაობას.

ამ ამოცანის გადასაწყვეტად გადავიდეთ პროგრამული კოდის ფანჯარაში, რისთვისაც Form1-ის „მეორე ფორმა“-ღილაკზე დავაწკაპუნოთ (ნახ.1.5-ბ). მისი შესაბამისი კოდის ფანჯარა 1.8-ა ნახაზზეა მოცემული. იგი ჯერ ცარიელია და აქ ხელით უნდა ჩავამატოთ საჭირო სტრიქონები, როგორც ეს 1.8-ბ ნახაზზეა მოცემული.



ნახ.1.8-ა



ნახ.1.8-ბ

აქვე, button2_Click() შეესაბამება ღილაკს „დასასრული“.

ახლა განვიხილოთ Form1-ში „მეორე ფორმა“-ღილაკის დაწკაპუნებით გამოტანილი პროგრამული კოდის სრული ლისტინგი_1.1:

```
//  ლისტინგი 1.1 --- WindowsMyFirst -------  // 1  კომენტარი
using System;  // 2  სისტემური სახელსივრცე, მოიცავს სისტემის
               // ფუნდამენტურ კლასთა ერთობლიობას
using System.Windows.Forms; //3  გრაფიკული ვინდოუს-აპლიკაციის
                // საჭირო საბაზო კლასების  სახელსივრცე. ინახება
                // System.Windows.Forms.dll ფაილში
namespace WindowsMyFirst   //4  ჩვენს მიერ შექმნილი სახელსივრცე,
                //აერთიანებს ჩვენს მეთოდებს და  გამორიცხავს
                // კონფლიქტს სხვა სახელსივრცის იგივე დასახელების
                        //  მეთოდებისთვის
{  //5
  public partial class Form1 : Form  //6-„წარმოებული : საბაზო“
                                       // კლასები
  {  //7
    public Form1() //8
    {   // 9
     InitializeComponent(); // 10  მეთოდი უზრუნველყოფს
                            // პროგრამის მართვის ელემენტების
                            // ასახვას  და ქცევას
    } // 11
      //12 ცარიელი სტრიქონი
   private void button1_Click(object sender, EventArgs e) // 13
                                          //ღილაკი „მეორე ფორმა“
    {  //14
      // შეტანილი სტრიქონის გამოტანა label1-ში --- // 15
     label1.Text = textBox1.Text;   // 16
                                    // 17
         // Form2-ის გახსნა Form1-დან --------  //18
```

```
    Form2 f2 = new Form2(this);  // 19 this - მაჩვენებელი,
                          // გადასცემს Form2-ს  მიმდინარე ობიექტს
                            // პარამეტრის სახით (იხ. დამატება_1.4)
    f2.Show();  // 20 ეკრანზე გამოჩნდება მეორე ფორმა
    f2.Controls["textBox1"].Text = textBox1.Text; // 21
                                    // გამოჩნდება შეტანილი ტექსტი
                                 // 22  ცარიელი სტრიქონი
        // შეტყობინების  გამოტანა    // 23
MessageBox.Show(textBox1.Text+",\nდახურეთ  ეს ფანჯარა  !"); // 24
   }  //25  იხურება  //14

private void button2_Click(object sender, EventArgs e) // 26
                                                 // ღილაკი  „ დასასრული“
   {  // 27
      Close();// 28  Form1-ის დახურვა
   } //29
  }   // 30  იხურება  // 7
}   // 31 იხურება  // 5
```

Form2-ის ღილაკის შესაბამისი პროგრამის ტექსტი მოცემულია 1_2 ლისტინგში:

```
// ლისტინგი 1.2   --- WindowsMyFirst --- Form 2 ------
using System;
using System.Windows.Forms;
namespace WindowsMyFirst
{ // 1
  public partial class Form2 : Form  //6
   { //2
     Form1  f1; // 8   f1 არის Form2-კლასის თვისება,  მიუთითებს
                // Form1-კლასის ობიექტზე
```

```
    public Form2(Form1 mainForm) //10 კონსტრუქტორია და
                    // ელოდება მიმთითებელს პარამეტრის სახით
                // Form1-კლასის ობიექტზე. ეს პარამეტრია mainForm
     {
       f1 = mainForm;  //11 ამ ბრძანებით უკვე შესაძლებელია
                                    // Form2-დან Form1-ის წვდომა
       InitializeComponent();  //12 Form2-ის კომპონენტების
                                     // ინიციალიზება
     }
private void Form2_Load(object sender, EventArgs e) //15
                                  // Form2-ის ჩატვირთვისას ხდება
 {  // Form1-ის ელემენტების მნიშვნელობათა წვდომა
   textBox1.Text=f1.Controls["textBox1"].Text; //17
                              // გამოიტანება Form1-ის ტექსტბოქსის
 }       // მნიშვნელობა. "textBox1" ასრულებს ინდექსის როლს
         // კოლექციაში, Text-თვისებით შესაძლებელია ამ
         // ელემენტის მნიშვნელობის ცვლილება
private void button1_Click(object sender, EventArgs e)
                                                          //20
 {
   f1.Controls["textBox1"].Text = textBox1.Text; //22
                              // Form1 მიიღებს Form2-ში შეცვლილი
                              // ტექსტბოქსის მნიშვნელობას
    Close(); //23 Form2-ის დახურვა. მართვა გადაეცემა Form1-ს.
  }
}} // იხურება 2 და 1
```

პროგრამის შესრულების პროცესში (ტრანსლაცია, კომპილაცია) შესაძლოა შეცდომების არსებობა, რომლებიც ასახული იქნება კოდის ფანჯრის ქვედა ნაწილში შესაბამისი კომენტარებით (ნახ.1.9). პროგრამა ჩერდება, საჭიროა ამ შეცდომების შესწორება და კოდის ხელახალი ამუშავება.

უშეცდომო, მუშა კოდის მიღების შემდეგ სასურველია მისი ტესტირება, ანუ ამ პროგრამის მრავალჯერადი გამოძახება და მისი ელემენტების და მოვლენათა მეთოდების ფუნქციონალობის შემოწმება სხვადასხვა საწყისი მნიშვნელობებისათვის.

საბოლოოდ მიიღება შესრულებადი კოდი, ანუ .exe ფაილი, რომელიც შემდგომში შეიძლება დამოუკიდებლად იქნას ამუშავებული C#-ის სამუშაო გარემოს გარეშე.

პროექტის დასრულების შემთხვევაში პროგრამული კოდები და სხვა თანმხლები რესურსული ფაილები შეინახება .sln ფაილში, რომელიც შემდგომში იქნება გამოყენებული სისტემის მიერ მისი ხელახალი ამუშავებისას.

1.2. ფორმაზე ვიზუალურ ელემენტებთან მუშაობა

ვინდოუს-ფორმის დიზაინის სრულყოფისა და მმართველი ელემენტების ეფექტურად ასაგებად შესაძლებელია სისტემის სხვადასხვა საშუალებების გამოყენება. მაგალითად:

- ელემენტების ფორმატირება: ვერტიკალურად ან ჰორიზონტალურად თანაბარი (ან ჩვენთვის საჭირო) მანძილებით. ყველა ელემენტი შეიძლება ერთბაშად მოინიშნოს (Shift/Ctrl) და შემდეგ ჩატარდეს მათი თვისებების დაყენება. მაგალითად, ყველა ტექსტბოქსში ერთი ზომისა და ფერის ქართული ფონტის დაყენება;

```
private void button1_Click(object sender, EventArgs e) //13
{  //14
    // შეტანილი სტრიქონის გამოტანა label1-ში --- //15
    label1.Text = textbox1.Text;  //textBox1.Text;  //16
                                                    //17
    // Form2-ის გახსნა Form1-დან -------- //18
    Form2 f2 = new Form2();  //19 this
    f2.Show(); // Dialog();  //20
    f2.Controls["textBox1"].Text = textBox1.Text;    //21
                                                    //22
    // შეტყობინების გამოტანა    //23
    MessageBox.Show("დახურეთ ფანჯარა !");  //23
} //24
```

100 %

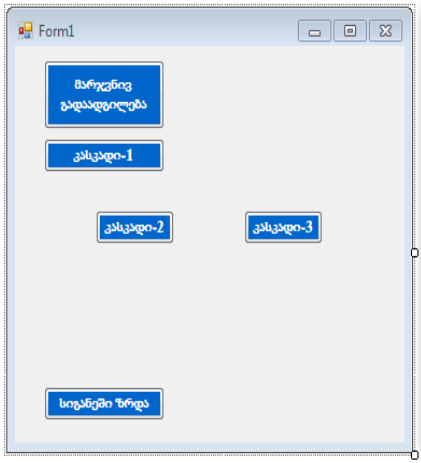
Error List

2 Errors | 0 Warnings | 0 Messages

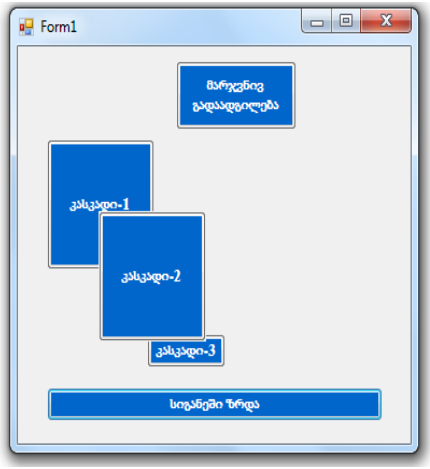
	Description	File	Line	Column	Project
1	The name 'textbox1' does not exist in the current context	Form1.cs	16	27	WindowsMyFirst
2	'WindowsMyFirst.Form2' does not contain a constructor that takes 0 arguments	Form1.cs	19	24	WindowsMyFirst

ნახ.1.9. შეცდომების ასხვის პანელი

- ელემენტების კოპირება: პროექტის სწრაფად ასაგებად ხელსაყრელია ერთხელ მომზადებული მმართველი ელემენტის მრავალჯერადი გამოყენება კოპირების საშუალებით (Ctrl+C და Ctrl+V). ამ დროს ელემენტის ყველა თვისება გადაიტანება ახალ, კოპირებულ ელემენტში და აღარაა საჭირო მათი ხელმეორედ დაყენება Properties-ში;
- ელემენტთა თვისებების შეცვლა შესრულების პროცესში: ფორმაზე ელემენტებს აქვს თვისესბები Size: Width/Height (ზომა: სიგანე/სიმაღლე) და Location: X/Y (მდებარეობა: კოორდინატები ფორმის ზედა-მარცხენა კუთხიდან). სიდიდეები პიქსელებში მოიცემა. 1.10-ა,ბ ნახაზებზე მოცემულია ფორმა ხუთი ღილაკით (ა-რედაქტირების რეჟიმი, ბ-მუშაობის რეჟიმი) და კოდის ლისტინგი_1.3 შესაბამისი ღილაკებისთვის, რომელთა საშუალებითაც შესაძლებელია ბუტონების ზომის და მდებარეობის ცვლილება მუშაობის რეჟიმში;



ნახ.1.10-ა. რედაქტირების რეჟიმი



ნახ.1.10-ბ. მუშაობის რეჟიმი

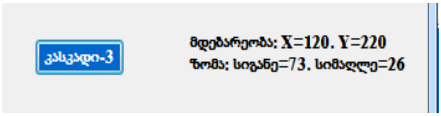
- ელემენტთა სახელები: ყველა ელემენტს თავისი საკუთარი სახელი უნდა ჰქონდეს. რეკომენდებულია სახელის წინ სამი ასოს გამოყენება, რომელიც ელემენტის ტიპს და ფუნქციონალობას მიუთითებს. მაგალითად, Label-სთვის lbl…, TextBox-სთვის txt…, Button-სთვის btn და ა.შ.;

```
// ლისტინგი_1.3--ელემენტების ზომის და მდებარეობის ცვლილება
using System;
using System.Drawing;
using System.Windows.Forms;
namespace WindowsMultiButtons
{ // 1
  public partial class Form1 : Form
   { // 2
    public Form1()
    { InitializeComponent(); }

private void Form1_Load(object sender, EventArgs e) { }
private void button1_Click(object sender, EventArgs e)
  {  // კასკადი-1
    button1.Size = new Size(100, 100);
   }
private void button2_Click(object sender, EventArgs e)
 { // გადაადგილება მარჯვნივ 20 პიქსელით
    button2.Location = new Point(button2.Location.X +
                               20, button2.Location.Y);
 }
private void button3_Click(object sender, EventArgs e)
  { // კასკადი-3
    button3.Location = new Point(120, 220);
 }
private void button4_Click(object sender, EventArgs e)
{ // სიგანეში გაფართოება 20-პიქსელით მაუსის ერთ წკაპზე
   button4.Size = new Size(button4.Size.Width + 20,
                            button4.Size.Height);
 }
private void button5_Click(object sender, EventArgs e)
   { // კასკადი-2
    button5.Size = new Size(100,100);
```

```
   }
  } // იხურება 2
} // იხურება 1
```

- ტექსტურ ელემენტში მრავალსტრიქონიანი ტექსტის გადაბმა და ეკრანზე გამოტანა: მაგალითად, label1-ელემენტში, რომლის სახელია lblText (Properties: Name), უნდა გამოვიტანოთ კასკადი-3 ბუტონის ფორმაზე მდებარეობის და ზომის ამსახველი სტრიქონები. 1_4 ლისტინგში ნაჩვენებია კოდის ფრაგმენტი, რომელშიც „+“ სიმბოლო გამოიყენება სტრიქონების გადასაბმელად (concatenation). 1.11-ა ნახაზზე ნაჩვენებია „კასკადი-3“ ღილაკი, რომლის დაწკაპუნებით მიიღება ტექსტური შედეგი, რომელიც label1-ელემენტშია გამოტანილი (ბ).

```
// ლისტინგი 1_4: --- concatenation ”+” -----
private void button3_Click(object sender, EventArgs e)
 {
   button3.Location = new Point(120, 220);
   lblText.Text = "მდებარეობა: X=" + button3.Location.X+
                  ". Y=" + button3.Location.Y + "\n" +
                  "ზომა: სიგანე=" + button3.Size.Width +
                 ". სიმაღლე=" + button3.Size.Height;   }
```



ნახ.1.11-ა. ფორმის ფრაგმენტი ნახ.1.11-ბ. შედეგი

1.3. დამატებითი ინფორმაცია:

დამატება_1.1) Form კლასის მნიშვნელოვანი თვისებები (Properties)

- BackColor – მართვის ელემენტის (მაგ., Form1) ფონის ფერი
- BackgroundImage – მართვის ელემენტის ფონური გამოსახულება
- ClientSize – მართვის ელემენტის ზომები (სიმაღლე, სიგანე)
- ContextMenu – მართვის ელემენტის კონტექსტური მენიუ

- ControlBox – ფორმის სატიტულო სტრიქონში MinimizeBox, MaximizeBox, ShowIcon გამოჩენა (true) ან დამალვა (false)
- Controls – ფორმაზე ყველა მართვის ელემენტის მიღება
- DesktopBounds – ფორმის ზომის და მდებარეობის განსაზღვრა სამუშაო მაგიდაზე
- FormBorderStyle – ფორმის საზღვრების (ჩარჩოს) სტილი
- HelpButton – ღილაკი გამოჩნდეს (true) ან არ გამოჩნდეს (false)
- Name – ფორმის სახელი (გამოიყენება კოდში)
- Opacity – პროგრამის შესრულების დროს ფორმის გამჭვირვალობა (სიფერმკრთალე)
- Size – ფანჯრის (ფორმის) ზომა
- Text – ფორმის სახელი ფანჯარაზე
- Visible – ფორმის გამოჩენა (true) ან არ გამოჩენა (false)
- WindowState – ფანჯრის ზომის მდგომარეობა: მინიმალური, მაქსიმალური ან ნორმალური

დამატება_1.2) Form კლასის მნიშვნელოვანი მეთოდები (Methods)

- Activate() – მართვის ელემენტის გააქტიურება
- Close() – ფორმის დახურვა
- GetStyle() / SetStyle() – ფანჯრის სტილის და ქცევის დადგენა / განსაზღვრა
- Hide() / Show() – მართის ელემენტის დამალვა / გამოჩენა
- ShowDialog() – დიალოგის ველის გამოტანა

დამატება_1.3) Form კლასის მნიშვნელოვანი მოვლენები (Events)

- BackColorChanged / BackgroundImageChanged – ფორმის ფონის შეცვლა / ფონის გამოსახულების შეცვლა
- Closed – მოვლენა აქტიურდება ფორმის დახურვის შემდეგ
- Closing – მოვლენა აქტიურდება ფორმის დახურვის პროცესში (შეიძლება პროცესის შეჩერება cancel-ით)

- HelpButtonClicked / HelpRequested – ფორმის სატიტულო სტრიქონში დახმარების ღილაკის არსებობა
- Load – აქტიურდება ფორმის ჩატვირთვისას
- Paint – აქტიურდება, როცა ფორმა ხელახლა იხაზება
- ResizeBegin / Resize / ResizeEnd / SizeChanged – კლიენტის ინფორმირება ფანჯრის ცვლილებების შესახებ
- Shown – აქტიურდება, როცა ფანჯარა პირველად გამოჩნდება
- VisibleChanged - აქტიურდება ფანჯრის ხილვადობის შეცვლისას

დამატება_1.4) საკვანძო სიტყვა this - მაჩვენებელი

საკვანძო სიტყვა this გამოიყენება მიმდინარე ობიექტის მისათითებლად. ობიექტის თითოეული მეთოდი ავტომატურად ფლობს this მაჩვენებელს, რომელიც მიუთითებს იმ ობიექტს, რომლისთვისაც იყო გამოძახებული ეს მეთოდი. აგრეთვე გამოიყენება როგორც გაფართოებული კლასის პირველი პარამეტრის მოდიფიკატორი.

მაგალითები:

- დამალული წევრების მიღება მსგავსი დასახელებით:

```
public Employee(string name, string alias)
{
   // Use this to qualify the fields, name and alias:
   this.name = name;
   this.alias = alias;
}
```

- ობიექტის გადაცემა სხვა მეთოდებზე პარამეტრის სახით:

```
CalcTax(this);
```

- ინდექსატორების გამოცხადება:

```
public int this[int param]
{
   get { return array[param]; }
   set { array[param] = value; }
}
```

this მაჩვენებელი არ გააჩნია სტატიკურ ფუნქცია-წევრებს, ვინაიდან ისინი კლასის დონეზე არსებობს და არა როგორც ობიექტის ნაწილი. სტატიკურ მეთოდში this-ის მითითება შეცდომაა.

ქვემოთ მაგალითში this გამოიყენება Employee კლასის წევრების განსაზღვრისთვის, name და alias, რომლებიც დამალულია მსგავსი სახელებით. იგი ასევე გამოიყენება ობიექტის გადასაცემად CalcTax მეთოდზე, რომელიც ეკუთვნის სხვა კლასს.

```
class Employee
    {
      private string name;
      private string alias;
      private decimal salary = 3000.00m;

      // Constructor:
      public Employee(string name, string alias)
      {
        // Use this to qualify the fields, name and alias:
        this.name = name;
        this.alias = alias;
      }
      // Printing method:
      public void printEmployee()
      {
        Console.WriteLine("Name: {0}\nAlias: {1}", name, alias);
        // Passing the object to the CalcTax method by using this:
        Console.WriteLine("Taxes: {0:C}", Tax.CalcTax(this));
      }

      public decimal Salary
      {
        get { return salary; }
      }
    }
  class Tax
    {
      public static decimal CalcTax(Employee E)
      {
        return 0.08m * E.Salary;
      }
    }
```

```
class MainClass
 {
    static void Main()
    {
       // Create objects:
       Employee E1 = new Employee("Mingda Pan", "mpan");

       // Display results:
       E1.printEmployee();
    }
 }
```

1.4. კითხვები და სავარჯიშოები:

1.1. რას წარმოადგენს C#.NET პროგრამული აპლიკაციის პროექტი, როგორია მისი სტრუქტურა ?

1.2. სად და როგორ ხდება C#.NET პროექტების შენახვა და პოვნა_?

1.3. რა არის Solution Explorer და როგორია მისი სტრუქტურა ?

1.4. რა არის Properties ?

1.5. რა ნაწილებისგან შედგება C#.NET აპლიკაციის პროექტის პროგრამული კოდი ?

1.6. რა ტიპის ვიზუალური მართვის ელემენტები არსებობს C#.NET -ის ინსტრუმენტების პანელზე ?

1.7. როგორ იქმნება ახალი Windows-ფორმა ?

1.8. სად ინახება პროგრამულად Windows-ფორმაზე მოთავსებული მართვის ელემენტები ?

1.9. რა განსხვავებაა label და textBox ელემენტებს შორის ?

1.10. როგორ აიგება და ფუნქციონირებს button ელემენტი ?

1.11. როგორ იხსნება ერთი ფორმიდან მეორე ფორმა და როგორ გადეცემა მას მონაცემები, რა არის “this” ?

1.12. როგორ შეიტანება/გამოიტანება მონაცემები ფორმაზე ქართული შრიფტით ?

1.13. რას წარმოადგენს MessageBox და როგორ მუშაობს იგი ?

1.14. Form კლასის რომელი მეთოდით ხდება ფორმის გახსნა და დახურვა ?

1.15. რას წარმოადგენს `InitializeComponent()` კლასის მეთოდი ?

1.16. რას წარმოადგენს `Form2_Load` () მეთოდი ?

1.17. როგორ მიიღება პროგრამული აპლიკაციის .exe კოდი ?

1.18. როგორ ხდება სინტაქსური შეცდომების პოვნა და გასწორება ?

1.19. როგორ ხდება ფორმაზე მართვის ელემენტებთან მუშაობა ?

1.20. როგორ ხდება ფორმაზე მართვის ელემენტისთვის სასურველი მდებარეობის განსაზღვრა (ვიზუალურად და პროგრამულად) ?

1.21. როგორ ხდება ფორმაზე მართვის ელემენტისთვის სასურველი ზომის განსაზღვრა (ვიზუალურად, პროგრამულად) ?

1.22. როგორ ხდება ტექსტური სტრიქონების გადაბმა და ეკრანზე ასახვა ?

1.23. Form კლასის რომელ თვისებებს იცნობთ ?

1.24. Form კლასის რომელ მეთოდებს იცნობთ ?

1.25. Form კლასის რომელ მოვლენებს იცნობთ ?

1.26. ააგეთ პროგრამა, რომელიც ფორმის პანელს ამოძრავებს ოთხი მიმართულებით მასზე მოთავსებულ ელემენტებიანად: ორი ბუტონი, ორ ტექსტბოქსი და ერთი ლაბელი.

1.27. ააგეთ კოდი, რომელიც ლურჯი ფერის ფორმაზე (ფონი) გაატარებს თეთრი, წითელი და მწვანე ფერის „ბილკებს“ (პანელით, მარცხნიდან მარჯვნივ).

1.28. ააგეთ პროგრამა „ავტორალი“: 7 მანქანა 7 ბილიკზე უნდა მივიდეს სასტარტოდან საფინიშო ხაზამდე. გადაადგილება ხდება შემთხვევით რიცხვთა გენერატორის დახმარებით.

თავი 2. Windows-პროგრამების ძირითადი ელემენტები

გადმოცემულია Visual_C# 2010 ენის ხშირად გამოყენებადი ვიზუალური კომპონენტები. ძირითადი სტრუქტურული და ობიექტ-ორიენტირებული კონსტრუქციები, მათ შორის მონაცემები და მათი ტიპები, ობიექტები, ოპერაციები, ციკლები, განშტოებები, გადამრთველები და სხვა მმართველი ელემენტები.

2.1. მონაცემთა საბაზო ტიპები და მათი გარდაქმნის მეთოდები

მონაცემი შეიძლება იყოს კონსტანტა, რომლის მნიშვნელობა პროგრამაში (ან მეთოდის „მოქმედების_არის“ შიგნით {...}) უცვლელია. მაგალითად, შენობის_სიმაღლე=100 მ., პიროვნების გვარი, π=L/D და სხვ. მათთვის მითითებული უნდა იყოს:

```
const int Building_Height=100;   // მთელრიცხვა ტიპი
const double PI=3.1415926535897; // ნამდვილრიცხვა ტიპი
                                 // მცოცავი წერტილით
const string Last_Name=”დოლიძე”; // სტრიქონული ტიპი
```

მონაცემი ცვლადია, თუ მისი მნიშვნელობა პროგრამის მუშაობის პროცესში შეიძლება შეიცვალოს. ცვლადის სახელი იწყება ასო-ნიშნით და შედგება ასოების, ციფრების და „_“ ნიშნისგან.

დაუშვებელია სახელის დაყოფა „space“-ს საშუალებით (მაგალითად, Last Name - შეცდომაა). პროგრამის („მოქმედების_არის“ შიგნით {...}) არ შეიძლება ორი სხვადასხვა ცვლადისთვის ერთი სახელის გამოყენება.

მნიშვნელობის მინიჭებამდე ცვლადი უნდა იქნეს გამოცხადებული თავისი ტიპით და სახელით.

2.1 ცხრილში მოცემულია C# ენაში გამოყენებული მონაცემთა ტიპები და მათი მნიშვნელობების დიაპაზონები:

C#- ენის ცვლადებისა და კონსტანტების ტიპები. ცხრ.2.1

ტიპები	თანრიგი (ბიტი)	დიაპაზონი
bool	1	true, false
byte	8	0 .. 255
char	16	0 .. 65535
decimal	128	1.0E-28 .. 7.9E28
double	64	5.0E-324 .. 1.9E308
enum	ჩამოთვლითი ტიპი	იღებს: byte, int, short,long ტიპების მნიშვნელობებს
float	32	1.5E-45 .. 3.4E38
int	32	-2 147 483 648 .. 2 147 483 648
long	64	- 9 223 372 036 854 775 808 . . 9 223 372 036 854 775 807
short	16	-32 768 .. 32 767
sbyte	8	-128 .. 127
struct	ტიპების კონტეინერი	შეიძლება ერთდროულად შეიცავდეს სხვადასხვა ტიპებს
uint	32	0 .. 4 294 967 295
ulong	64	0.. 18 446 744 073 709 551 615
ushort	32	0 .. 65 535

არსებობს მონაცემთა ტიპების გარდაქმნის არაცხადი (implicit) და ცხადი (explicit) საშუალებანი. პირველის შემთხვევაში მონაცემთა გარდაქმნა ხდება პროგრამისტის ჩარევის გარეშე. ცხრილში ნაჩვენებია ეს შესაძლებლობანი.

ტიპების არაცხადი გარდაქმნის ცხრილი. **ცხრ.2.2**

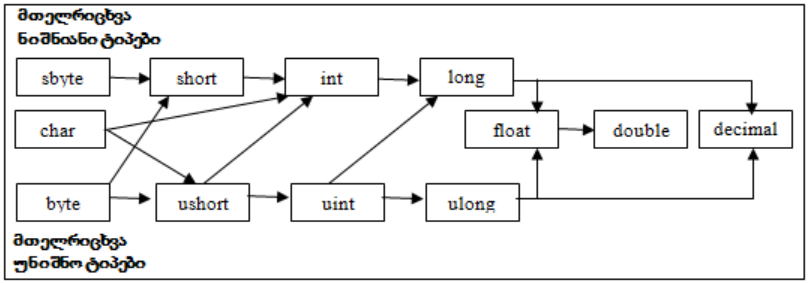
საწყისი ტიპი	გადაყვანის ტიპი
sbyte	short, int, long, float, double, ან decimal
byte	short, ushort, int, uint, long, ulong, float, double, ან decimal
short	int, long, float, double, ან decimal
ushort	int, uint, long, ulong, float, double, ან decimal
int	long, float, double, ან decimal
uint	long, ulong, float, double, ან decimal
long	float, double, ან decimal
char	ushort, int, uint, long, ulong, float, double, ან decimal
float	double
ulong	float, double, ან decimal

ტიპების ცხადი გარდაქმნის ცხრილი. **ცხრ.2.3**

საწყისი ტიპი	გადაყვანის ტიპი
sbyte	byte, ushort, uint, ulong, ან char
byte	sbyte or char
short	sbyte, byte, ushort, uint, ulong, ან char
ushort	sbyte, byte, short, ან char
int	sbyte, byte, short, ushort, uint, ulong, ან char

uint	sbyte, byte, short, ushort, int, ან char
long	sbyte, byte, short, ushort, int, uint, ulong, ან char
ulong	sbyte, byte, short, ushort, int, uint, long, ან char
char	sbyte, byte, ან short
float	sbyte, byte, short, ushort, int, uint, long, ulong, char, ან decimal
double	sbyte, byte, short, ushort, int, uint, long, ulong, char, float, ან decimal
decimal	sbyte, byte, short, ushort, int, uint, long, ulong, char, float, ან double

2.1 ნახაზზე მოცემულია ტიპების **არაცხადი** (ნაჩვენებია „->“-ისრით, რომელსაც C# კომპილატორი ავტომატურად ასრულებს) და **ცხადი** (ისრის საწინააღმდეგო მიმართულება, რომელიც პროგრამისტმა უნდა მიუთითოს აუცილებლად) გარდაქმნის ნაკადების სქემა.



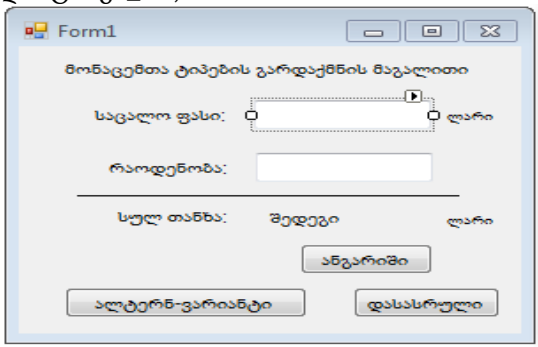
ნახ.2.1

2.2 ნახაზზე ნაჩვენებია ფორმა, რომლის ორ ტექსტბოქსში ხელით შეიტანება პროდუქციის „საცალო ფასის“ (textBox1.Text) და შესასყიდი „რაოდენობის“(textBox2.Text) შესაბამისი რიცხვითი მნიშვნელობები (იხ. ლისტინგი_2.1). ვინაიდან ისინი TextBox-ში იწერება და სტრიქონული ცვლადებია, რეალურად ფიქსირდება სიმბოლური რიცხვები („50.75“ და „2“), რომელთაც მათემატიკურ

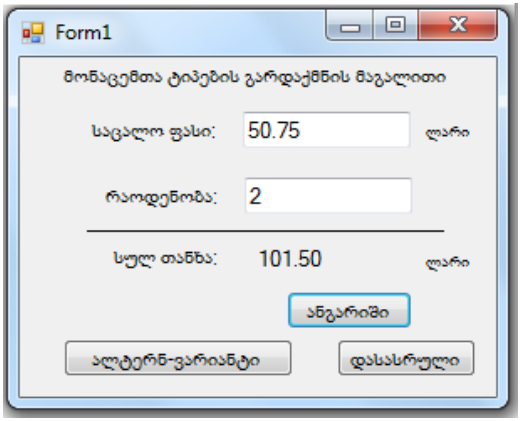
ოპერაციებში მონაწილეობა არ შეუძლია. ისინი უნდა გარდაიქმნას ნამდვილ რიცხვებად. ასეთ გარდაქმნას ასრულებს Convert-კლასის ToDecimal() და ToInt32() მეთოდები:

Convert.ToDecimal(Sacalo_Fasi) და Convert.ToInt32(Raodenoba).

მიიღება: რიცხვები სიმბოლური ბრჭყალების გარეშე („50.75“ -> 50.75 და „2“ -> 2). ეს რიცხვები მრავლდება ერთმანეთზე და მიიღება რიცხვი 101.75, რომლის ეკრანზე გამოსატანად label5-ში (შედეგი) საჭიროა მისი ისევ სტრიქონულ ტიპში გარდაქმნა, რომელსაც ასრულებს ToString() მეთოდი (23-ე სტრიქონი, ლისტინგი_2.1).



ნახ.2.2-ა



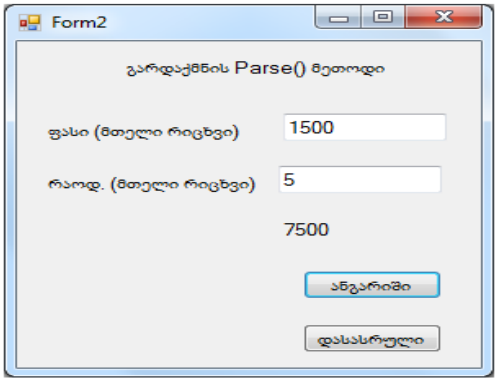
ნახ.2.2-ბ: შედეგი

```
// მონაცემთა ტიპების გარდაქმნის პროგრამის ლისტინგი_2.1 -------
using System;
using System.Windows.Forms;
namespace WinFormDataConvert
{
 public partial class Form1 : Form
 {
  public Form1() { InitializeComponent();  }
 private void button1_Click(object sender, EventArgs e)
  {   // რიცხვების შეტანა ტექსტბოქსებში
    String Sacalo_Fasi = textBox1.Text;
    String Raodenoba = textBox2.Text; //სტრიქონული ტიპით
     // Convert-ით სტრიქონული ტიპი გარდაიქმნება
     // Decimal და Int32 რიცხვით ტიპებად
     // არითმეტიკული *-ოპერაციის შესრულებისთვის.
     // ბოლოს ToString() მეთოდი გამრავლების შედეგს
     // გარდაქმნის ისევ სტრიქონულ ტიპად,
     // რათა იგი ეკრანზე გამოიბეჭდოს label5-ში
    label5.Text= (Convert.ToDecimal(Sacalo_Fasi) *
          Convert.ToInt32(Raodenoba)).ToString(); //23
   }
  }
}
```

არსებობს ალტერნატიული Parse(String) მეთოდი, რომელიც სიმბოლურ რიცხვს გარდაქმნის მთელ რიცხვად (იხ. მაგალითი ლისტინგი_2.2).

```
private void button1_Click(object sender, EventArgs e)
 {
  int a = 0;
  String sacalo = textBox1.Text;
  String raodenoba = textBox2.Text;
  a = int.Parse(sacalo) * int.Parse(raodenoba);
  label1.Text = a.ToString();
 }
```

შედეგი ასახულია 2.3 ნახაზზე:



ნახ.2.3

ტიპების კონვერტაციისთვის გამოიყენება აგრეთვე

cast(ტიპი) <გამოსახულება>

გარდაქმნა (casting type - ტიპების დაყვანა). ეს ცხადი სახის გარდაქმნაა. მაგალითად, კოდის ფრაგმენტში ქვემოთ მე-4 სტრიქონში ნაჩვენებია ეს შემთხვევა: არასწორი (ა) და სწორი (ბ) ვარიანტებით.

```
// . . . --- ა ---
float Sacalo_Fasi;
int Sum;
Sacalo_Fasi=250f;
Sum= Sacalo_Fasi;    // შეცდომაა
```

```
// . . . --- ბ ---
float Sacalo_Fasi;
int Sum;
Sacalo_Fasi=250f;
Sum= (int) Sacalo_Fasi;
                // სწორია Cast(int)
```

Cast(int) გარდაქმნისას რიცხვის მეათედი ნაწილი იკარგება, არ ხდება ავტომატური დამრგვალება. მაგალითად, თუ a=25,99 , მაშინ b=(int) a; // 25. დაიკარგა 0.99

2.2. ცვლადების მოქმედების დიაპაზონი

ცვლადებისათვის, რომლებიც კლასის მეთოდის შიგნითაა გამოცხადებული და აქ გამოიყენება, განისაზღვრება მოქმედების

დიაპაზონი (არე) ამ მეთოდის ფარგლებში (მაგალითად, ცვლადი int a; ლისტინგი_2.2). მეთოდის გარეთ ცვლადის სახელიც და მნიშვნელობაც უცნობია. მათ **ლოკალურ** ცვლადებს უწოდებენ. მეთოდის ხელმეორედ გამოძახებისას ცვლადის გამოცხადება ხდება თავიდან. დაუშვებელია ერთ მეთოდში ორი ლოკალური ცვლადის არსებობა ერთიდაიმავე სახელით.

ცვლადები, რომლებიც კლასის მეთოდების გარეთაა მოქმედებაში, ამ კლასის ფარგლებში გამოიყენება. მის გარეთ იგი უცნობია. მაგალითად, Form1-კლასისთვის, როცა ეს ფორმა გახსნილია, ყველა ცვლადი აქტიური და მისაწვდომია ამ ფორმის მეთოდებითვის, მის დახურვამდე.

ცვლადების გამოცხადება ხდება აგრეთვე საკვანძო სიტყვებით: private, public, protected. პირველი მათგანი ცვლადებს მალავს და კლასის გარედან მათ ვერავინ გამოიყენებს. public სიტყვა პირიქით, ცვლადების „ღიად წვდომის“ უფლებას იძლევა და შესაძლებელია მათი მიღება სხვა კლასებისთვისაც. protected-ით (საბაზო) კლასის ცვლადები დამალულია ყველასთვის, გარდა მისგან წარმოებული (შვილობილი) კლასებისა.

განვიხილოთ მაგალითი, ორი ცვლადის შემთხვევაში. პირველი კლასის ცვლადია (ClasDiap, ანუ კლასის ფარგლებში არის გლობალური). მეორე ლოკალური ცვლადია (LocalDiap), რომელიც მოქმედებს button1_Click(…) მეთოდის შიგნით.

განსაკუთრებით უნდა გამოვყოთ button2_Click(…) მეთოდის შიგნით მოქმედი ცვლადი ClasDiap, რომელსაც იგივე სახელი აქვს, რაც ჰქონდა Form3 კლასის ცვლადს მე-8 სტრიქონში, ანუ ClasDiap. ეს ორი ცვლადი ერთიდაიმავე სახელით სინამდვილეში სხვადასხვა ცვლადებია. ერთი გლობალური, კლასის ცვლადია (თავისი მნიშვნელობით), მეორე კი ლოკალური ცვლადია button2_Click(…) მეთოდის შიგნით და აქვს თავის მნიშვნელობა. ისინი მეხსიერების სხვადასხვა ადგილასაა შენახული, სხვადასხვა რიცხვებია.

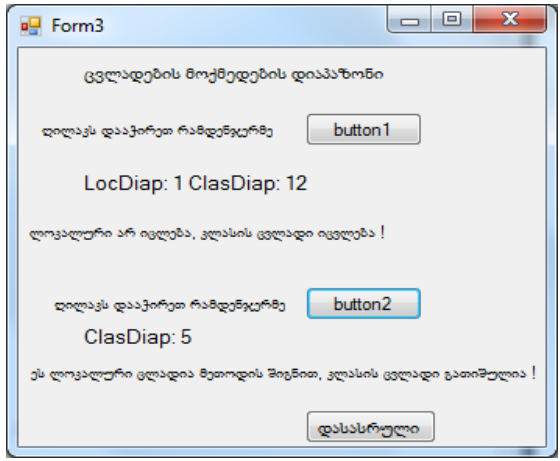
ვინაიდან button1_Click(…) მეთოდს არ აქვს გამოცხადებული შიგნით ClasDiap, ამიტომ ის იყენებს კლასის ცვლადს და მისი მნიშვნელობა ღილაკზე დაჭერისას

მონოტონურად იზრდება. აქ ლოკალური ცვლადი LocalDiap ღილაკის დაწკაპუნებისას ყოველთვის იღებს საწყის მნიშვნელობას (0-ს) და შემდეგ +1.

ვინაიდან button2_Click(…) მეთოდს აქვს გამოცხადებული შიგნით ClasDiap ცვლადი (ე.ი. ლოკალური ცვლადი, რომელიც ბლოკავს ერთსახელა-გლობალურს), ამ ღილაკზე დაწკაპუნებისას ეს ლოკალური ცვლადიც იღებს საწყის მნიშვნელობას 0-ს და შემდეგ +5.

ჩვენი მაგალითის პროგრამის ტექსტი მოცემულია 2_3 ლისტინგში, ხოლო მისი შედეგები ასახულია 2.3 ნახაზზე.

```
// ლისტინგი_2.3 --- ცვლადების მოქმედების დიაპაზონი ----
using System;
using System.Windows.Forms;
namespace WinFormDataConvert
{
 public partial class Form3 : Form
 {
  int ClasDiap = 0; // 8    ცვლადი კლასის დიაპაზონით
  public Form3() {InitializeComponent(); }
 private void button1_Click(object sender, EventArgs e)
  {
   int LocalDiap = 0;  // ცვლადი ლოკალური დიაპაზონით
   ClasDiap = ClasDiap + 1; // ცვლადი კლასის დიაპაზონით
  LocalDiap = LocalDiap + 1; // ცვლ. ლოკალური დიაპაზონით
  label1.Text = "LocDiap: " + LocalDiap +
                " ClasDiap: " + ClasDiap;
  }
private void button2_Click(object sender, EventArgs e)
 {
  int ClasDiap = 0; // ცვლადი ლოკალური დიაპაზონითაა !!!
        // ეს სხვა ClasDiap -ცვლადია, ვიდრე მე-8 სტრიქონში,
        // ლოკალური ცვლადი თიშავს გლობალურს
  ClasDiap = ClasDiap + 5;
  label2.Text = "ClasDiap: " + ClasDiap;
  }
}}
```



ნახ.2.3

2.3. ჩამოთვლითი ტიპები - enum

ჩამოთვლითი ტიპები გამოიყენება სიმბოლური კონსტანტების სასრული სიმრავლეების (ან ერთობლიობების) შესაქმნელად მომხმარებლის მიერ. მაგალითად, თვეები (იანვარი, თებერვალი,..., დეკემბერი), კვირის_დღეები (ორშაბათი, სამშაბათი,..., კვირა), ფერები (შავი, წითელი, მწვანე,...,ყვითელი), მეგობრები (გიორგი, დავითი,..., ეკატერინე) და სხვ.

საკვანძო სიტყვა enum არის System.Enum ტიპის ფსევდონიმი, რომელიც მოთავსებულია .NET Framework კლასთა ბიბლიოთეკაში. მას გააჩნია მრავალი მეთოდი ჩამოთვლით ტიპებთან სამუშაოდ. ასეთი ფუნქციონალური შესაძლებლობებია, მაგალითად:

- სტრიქონების გარდაქმნა ჩამოთვლით მნიშვნელობებად;
- ჩამოთვლითი მნიშვნელობების გარდაქმნა ტექსტად;
- შემოწმების ჩატარება, არის თუ არა მოცემული მნიშვნელობა ჩამოთვლის ელემენტი;
- შემოწმდეს ჩამოთვლის საბაზო ტიპი;
- განისაზღვროს ჩამოთვლის ელემენტის სიმბოლური სახელი, რომელიც ასახავს მოცემულ მნიშვნელობას და სხვ.

ჩამოთვლის გამოცხადების სინტაქსი ასეთია:

[<წვდომის_სპეციფიკატორი>] enum <ჩამოთვლის_სახელი>[:<მთელრიცხვა_ტიპი>]
{ <ჩამოთვლითი_ელემენტის_სახელი_1>
[=<მთელრიცხვა_მნიშვნელობა_1>],
<ჩამოთვლითი_ელემენტის_სახელი_2>
[=<მთელრიცხვა_მნიშვნელობა_2>], ...
}

სადაც:

- წვდომის სპეციფიკატორებია: public, private, protected, internal, protected internal;

- მთელრიცხვა_ტიპი: byte, sbyte, short, ushort, int, uint, long, ulong;

მაგალითად:

public enum Color : int {Black, Whyte, Red=4, Blue, Green=10, Yellow=20, Pink}

ჩამოთვლის შექმნის შემდეგ შესაძლებელია ამ ტიპის ცვლადის გამოცხადება და მისთვის მნიშვნელობის მინიჭება აღწერილი სიმრავლიდან:

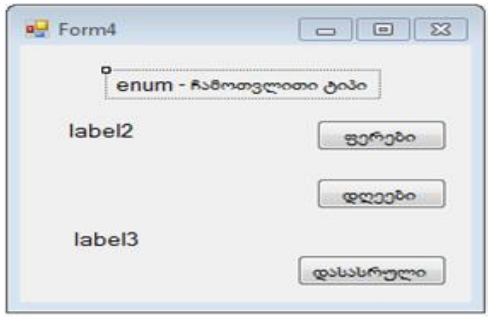
Color myColor;
myColor=Color.Red; // არ შიძლება myColor=4; შეცდომაა !!!

ახლა განვიხილოთ 2_4 ლისტინგით მოცემული პროგრამის კოდი, რომელშიც ილუსტრირებულია ჩამოთვლითი ტიპის გამოყენება:

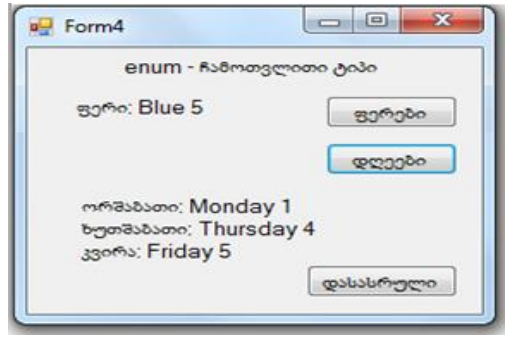
```
// ლისტინგი 2_4 --- enum ჩამოთვლითი ტიპი ----------------
using System;
using System.Windows.Forms;
namespace WinFormDataConvert
{
 public partial class Form4 : Form
 {
  enum Color : int // „ფერები“
   {
    Black, Whyte, Red=4, Blue, Green=10, Yellow=20, Pink
    // 0      1      4       5      10          20      21
   }
   public Form4() {  InitializeComponent();  }
```

```
private void button1_Click(object sender, EventArgs e)
{
 label2.Text="ფერი:" + Color.Blue + "  "+ (int) Color.Blue;
}
private void button2_Click(object sender, EventArgs e)
{
label3.Text="ორშაბათი:" + DayOfWeek.Monday + " " +
                    (int)DayOfWeek.Monday + "\n" +
              "ხუთშაბათი:  " + DayOfWeek.Thursday + "  " +
                    (int)DayOfWeek.Thursday + "\n"+
              "კვირა: "+DayOfWeek.Friday+" "+
                    (int) DayOfWeek.Friday;
}
private void button3_Click(object sender, EventArgs e)
 { Close();  }
}
```

2.4 ნახაზზე მოცემულია პროგრამის მუშაობის შედეგები ღილაკების „ფერები“ და „დღეები“ ამოქმედების შემდეგ.



ნახ.2.4



2.4. სტრიქონული ტიპი string და String კლასი

სტრიქონებთან მუშაობას განსაკუთრებული ადგილი უჭირავს გამოყენებით ინფორმატიკაში, კერძოდ ლინგვისტურ, სტატისტიკურ და სხვა სფეროს კომპიუტერულ სისტემებში. მონაცემთა ბაზების, ექსპერტული და გადაწყვეტილების მიღების სისტემებში მის საფუძველზე არაერთი მნიშვნელოვანი ფუნქციონალური ამოცანები წყდება. წინამდებარე პარაგრაფში შემოთავაზებულია String კლასის არსი, მისი მეთოდები და თვისებები, მათი გამოყენების ინსტრუქციები.

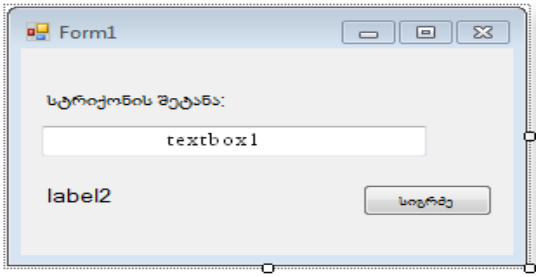
სტრიქონების დამახსოვრება ხდება String კლასით, რომლის სინონიმია string მონაცემთა ტიპი. String კლასის ობიექტები, ანუ სტრიქონები ფლობენ მრავალ თვისებას და მეთოდს, რომლებიც მასივის (Array) კლასის მსგავსია.

String კლასის ობიექტის კოპირებისას იქმნება სტრიქონის ასლი ცვლადის მსგავსად (არა ობიექტის მსგავსად). ანუ გვაქვს ორი ერთმანეთისგან დამოუკიდებელი სტრიქონი, ერთში ცვლილება არ იწვევს მეორის ცვლას.

ახლა განვიხილოთ String კლასის თვისებები და მეთოდები პრაქტიკული მაგალითების საფუძველზე.

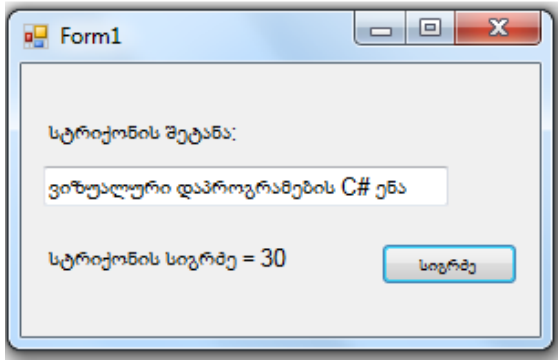
თვისება Length გამოიყენება სტრიქონის სიგრძის დასადგენად, ანუ რამდენი სიმბოლოსგან შედგება იგი.

დავდოთ ფორმაზე სტრიქონის შესატანი textbox1 ველი და label2 - სტრიქონის სიგრძის მნიშვნელობის გამოსატანი ველი. ღილაკით „სიგრძე“ მოხდეს სტრიქონის სიგრძის დათლა და მიშვნელობის გამობეჭდვა (ნახ.2.5-ა). ღილაკის კოდი მოცემულია 2.5_ლისტინგზე, ხოლო შედეგი - 2.5-ბ ნახაზზე.



ნახ.2.5-ა

```
// ლისტინგი_2.5 --- String ----
private void button1_Click(object sender, EventArgs e)
   {
      string Striqoni;
      Striqoni = textBox1.Text;
      label2.Text = "სტრიქონის სიგრძე = "+Striqoni.Length;
   }
```



ნახ.2.5-ბ

ამგვარად, შეტანილი სტრიქონი დამახსოვრებულ იქნა string ტიპის Striqoni ცვლადში. მისი სიგრძის ანგარიში კი მოხდა Striqoni.Length თვისების დახმარებით.

სტრიქონში სიმბოლოები ჩალაგებულია მასივის მსგავსად, ანუ ერთი სიმბოლო ერთი ელემენტია და თავისი ინდექსი აქვს (იხ.ცხრილში [ქეთო და კოტე]).

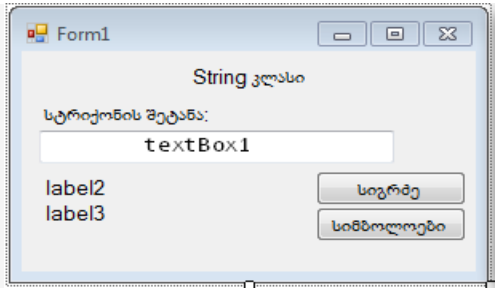
ქ	ე	თ	ო		დ	ა		კ	ო	ტ	ე
0	1	2	3	4	5	6	7	8	9	10	11

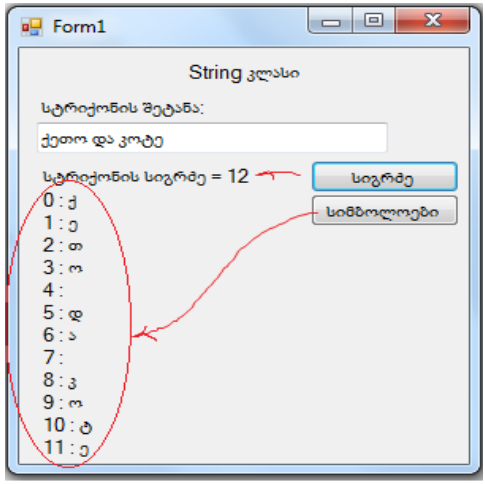
დავუმატოთ 2.5 ფორმას ერთი ღილაკი „სიმბოლოები“ და მივაბათ მას კოდი 2.6 ლისტინგით.

```
// ლისტინგი_2.6 -- სიმბოლოები -----
private void button2_Click(object sender, EventArgs e)
  {
    string Striqoni;
```

```
    char simbolo;  // ერთი სიმბოლო
    int i;   // ინდექსი
    Striqoni = textBox1.Text;
    label2.Text="სიმბოლოები";
    label3.Text="";
    for (i = 0; i < Striqoni.Length; i++)
      {
        simbolo = Striqoni[i];
        label3.Text += i.ToString()+" : "+ simbolo
+"\n";
      }
  }
```

შედეგი მოცემულია 2.6 ნახაზზე.





ნახ.2.6

2.4.1. სტრიქონის დამუშავების მეთოდები: Trim() და Replace()

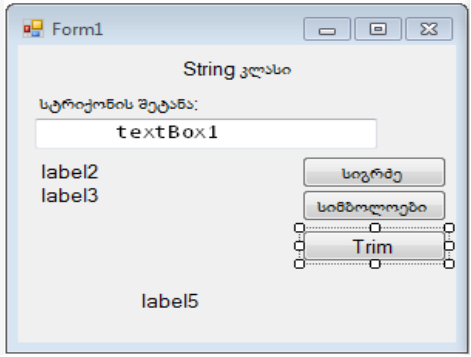
სტრიქონებთან მუშაობისას ხშირად ვხვდებით ტესქტში ცარიელ სიმბოლოებს (პრობელებს), რომელთა დამუშავება (ამოყრა, შემცირება და ა.შ.) აუცილებელია. ამისათვის ბიბლიოთეკაში რამდენიმე მეთოდია:

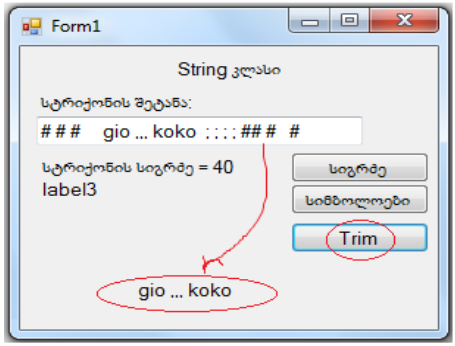
Trim() – არასასურველი სიმბოლოების ამოყრა სტრიქონის თავიდან ან ბოლოდან, მათ შორის პრობელებისაც;

TrimStart() – არასასურველი სიმბოლოების ამოყრა მხოლოდ სტრიქონის თავიდან;

TrimEnd() - არასასურველი სიმბოლოების ამოყრამხოლოდ სტრიქონის ბოლოდან.

ტექსტის შუაგულიდან არასასურველი სიმბოლოების (პრობელებისაც) ამოსაყრელად ან შესაცვლელად გამოიყენება მეთოდი Replace(). 2.7 ნახაზზე ნაჩვენებია Trim მეთოდის მაგალითი და მისი კოდის ფრაგმენტი 2.7_ლისტინგში.

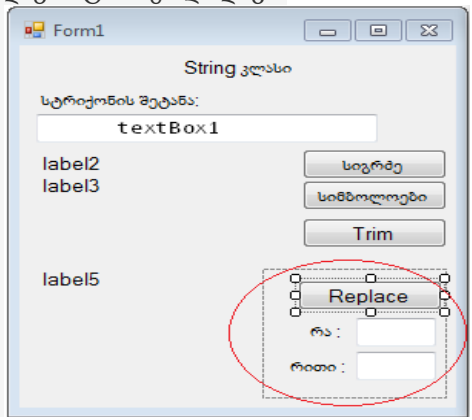


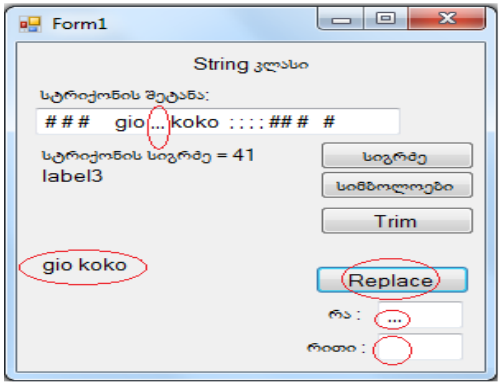


ნახ.2.7

```
// ლისტინგი_2.7 ---  Trim -----
private void button3_Click(object sender, EventArgs e)
 {
  string Striqoni, SuftaStriqoni;
  Striqoni = textBox1.Text;
   SuftaStriqoni = Striqoni.Trim(' ', ';', '#');
   label5.Text = SuftaStriqoni;
 }
```

როგორც შედეგიდან ჩანს, სტრიქონის თავში და ბოლოში აღარაა ზედმეტი სიმბოლოები. დარჩა მხოლოდ „ ,,, „ სტრიქონის შუაში. ახლა Replace() მეთოდი გამოვიყენოთ, რისთვისაც ფორმაზე დავამატოთ ეს ღილაკი.





ნახ.2.8

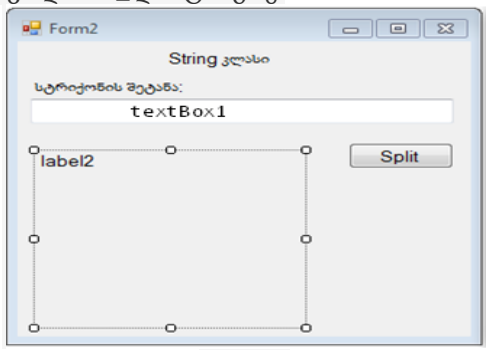
```
// ლისტინგი_2.8 ---  Trim + Replace -----
private void button4_Click(object sender, EventArgs e)
  {
   string Striqoni, SuftaStriqoni, Ra, Riti;
   Striqoni = textBox1.Text;
   Ra = textBox2.Text;
   Riti = textBox3.Text;
   SuftaStriqoni = Striqoni.Trim(' ', ';', '#');
   SuftaStriqoni = SuftaStriqoni.Replace(Ra, Riti);
   label5.Text = SuftaStriqoni;
  }
```

Replace() მეთოდი მოითხოვს ორი დამატებითი ტექსტბოქსის გამოყენებას. რომლებშიც მიეთითება სტრიქოინის შიგნით „რა“ უნდა შეიცვალოს „რითი“.

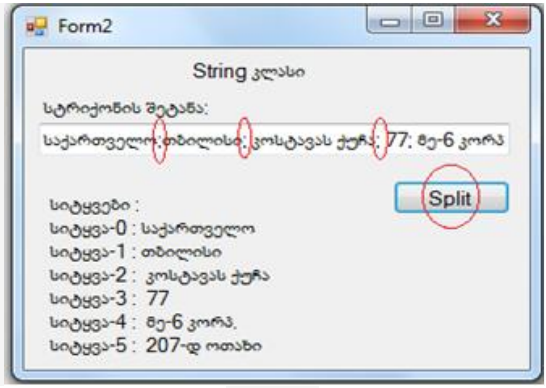
2.4.2. სტრიქონის დაყოფის მეთოდი : Split()

სტრიქონების დამუშავების Split() მეთოდი გამოიყენება მაშინ, როცა საჭიროა რაიმე წინადადების დაშლა ცალკეულ სიტყვებად. ამ დროს უნდა მიეთითოს გამყოფი სიმბოლო, რომლითაც მოხდება სტრიქონის დაშლა. ეს სიმბოლო შეიძლება იყოს „ ; “ ან „ , “ ან სხვ., მაგალითად „ცარიელი სიმბოლო“ (პრობელი).

განვიხილოთ მაგალითი (ნახ.2.9) და შესაბამისი პროგრამული კოდი 2.9_ლისტინგზე.



ნახ.2.9-ა



ნახ.2.9-ბ

```
// ლისტინგი_2.9 ---  Split () -----
private void button1_Click(object sender, EventArgs e)
 {
   string Striqoni = textBox1.Text;
   string[] Sityva;
   int i;
   Sityva = Striqoni.Split(';');
   label2.Text = "სიტყვები :" + "\n";
   for (i = 0; i < Sityva.Length; i++)
      label2.Text += "სიტყვა-" + i + " : " +
                                    Sityva[i] + "\n";
 }
```

სტრიქონის დასამუშავებლად საჭიროა for-ციკლი Sityva.Length თვისების კონტროლით. შედეგში კარგად ჩანს დაშლილი სტრიქონის ნაწილები: სიტყვები ინდექსებით.

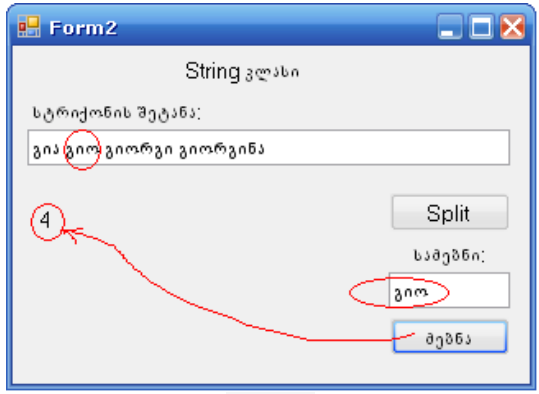
2.4. 3. სტრიქონში ძებნის მეთოდები: IndexOf(), LastIndexOf() და IndexOfAny()

სტრიქონებთან მუშაობისას ხშირად საჭირო ხდება იმის დადგენა, არის თუ არა ერთი სტრიქონი (ან ქვესტრიქონი) მეორე სტრიქონის ნაწილი. ამ დროს გამოიყენება მეთოდები: IndexOf(), LastIndexOf() და IndexOfAny(). განვიხილოთ მაგალითი. ფორმაზე ძირითად სტრიქონთან (textBox1) ერთად მოვათავსოთ საძებნი

ქვესტრიქონი (textBox2) და ღილაკზე ”ძებნა” მივაბათ 2.10_ლისტინგის კოდი.

```
// ლისტინგი_2.10 --- IndexOf() ----
private void button2_Click(object sender, EventArgs e)
{
 string Striqoni, Sazebni;
 int pozicia;
 label2.Text = "";
 Striqoni = textBox1.Text;
 Sazebni = textBox2.Text;
 pozicia = Striqoni.IndexOf(Sazebni);
 label2.Text += pozicia;
}
```

2.10 ნახაზზე ნაჩვენებია ”გია გიო გიორგი გიორგინა” სტრიქონში (Striqoni) საძებნი ქვესტრიქონის ”გიო” (Sazebni) მდებარეობის პოზიცია (=4). ესაა საძებნი ქვესტრიქონის პირველი ნაპოვნი პოზიცია. თუ პოზიცია 0-ია, მაშინ ის მიუთითებს ძირითადი სტრიქონის დასაწყისზე, ხოლო თუ იგი ”-1”, მაშინ ასეთი ქვესტრიქონი ვერ მოიძებნა.

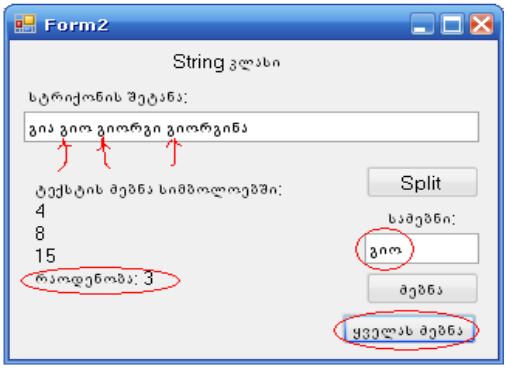


ნახ. 2.10

თუ საჭიროა საძებნი ქვესტრიქონის ძირითად სტრიქონში არსებობის ყველა შემთხვევის დაფიქსირება, მაშინ ძებნის

ალგორითმი მიიღებს 2.11_ლისტინგში მოცემული კოდის სახეს, ხოლო შედეგები იქნება 2.11 ნახაზზე მოცემული.

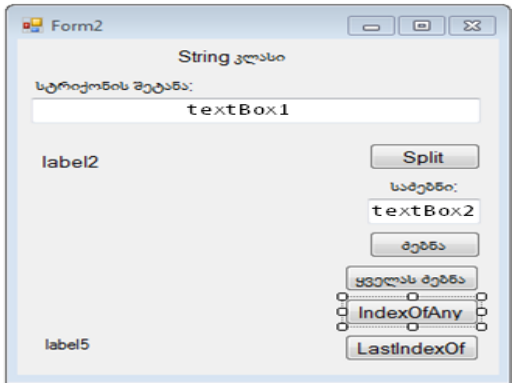
```
   // ლისტინგი_2.11 --- All IndexOf ( ) ----
private void button3_Click(object sender, EventArgs e)
 {
   string Striqoni, Sazebni;
    int pozicia, zebnisDackeba=0, raod=0;
    label2.Text = "";
    Striqoni = textBox1.Text;
    Sazebni = textBox2.Text;
    label2.Text = "ტექსტის ძებნა სიმბოლოებში: " + "\n";
 do
    {
     pozicia = Striqoni.IndexOf(Sazebni, zebnisDackeba);
     zebnisDackeba = pozicia + 1;
     if (pozicia != -1)
       {
        label2.Text += pozicia + "\n";
        raod++;
       }
    }
     while (pozicia != -1)
              ;
 label2.Text += "რაოდენობა: " + raod;
}
```

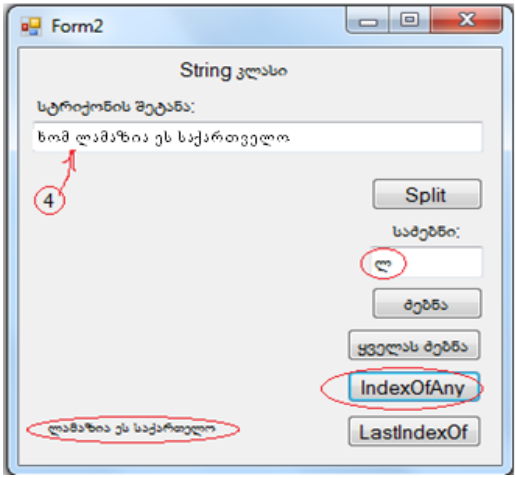


ნახ. 2.11

როგორც ვხედავთ, do…while ციკლის შიგნით ხდება საძებნი ქვესტრიქონის მრავალჯერადი მოძიება ძირითად სტრიქონში. საწყისი საძიებო პოზიცია 0-ია, შემდეგი კი განისაზღვრება ყოველი ახალი ნაპოვნი პოზიციის შემდეგ. პროგრამა იმახსოვრებს ასევე ნაპოვნი შემთხვევების რაოდენობას.

განვიხილოთ მაგალითი `IndexOfAny()` მეთოდისთვის, რომლის დანიშნულებაცაა სტრიქონში პირველი ქვესტრიქონის პოვნა მითითებული სიმბოლოთი. 2.12 ნახაზზე ნაჩვენებია ეს შემთხვევა, ხოლო 2.12_ლისტინგში `IndexOfAny` ღილაკის კოდი.

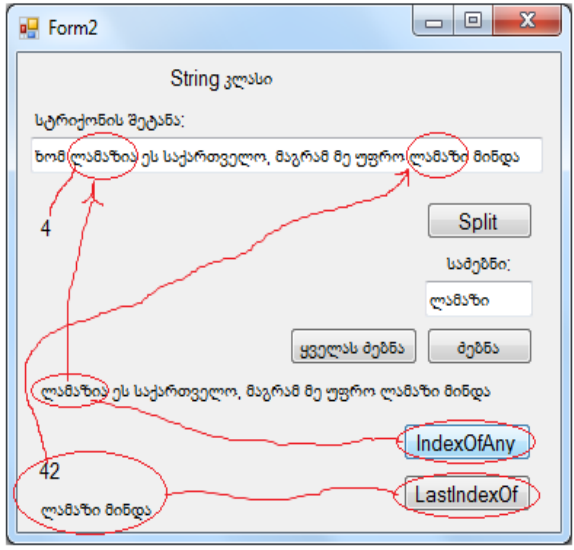




ნახ.2.12

```
  // ლისტინგი_2.12 --- IndexOfAny ----
private void button4_Click(object sender, EventArgs e)
 {
  string Striqoni;
  int pozicia;
  label2.Text = "";
  Striqoni = textBox1.Text;
  textBox2.Text = "ლ";  // ან "ს"
  pozicia = Striqoni.IndexOfAny(new char[] { 'ლ', 'ს' });
  //pozicia = Striqoni.IndexOfAny(new char[] { 'კ', 'ს' });
  label2.Text += pozicia;
  label5.Text = Striqoni.Substring(pozicia);
 }
```

ახლა განვიხილოთ LastIndexOf () მეთოდი. იგი ძირითად სტრიქონში პოულობს მითითებული სიმბოლოს ან სიტყვის მიხედვით ქვესტრიქონს, ოღონდ ბოლოდან (არა პირველს მარცხნიდან). შესაბამისი კოდი მოცემულია 2.13_ლისტინგში, შედეგები კი 2.13 ნახაზზე. აქ კარგად ჩანს განსხვავება IndexOfAny() და LastIndexOf () მეთოდებს შორის.



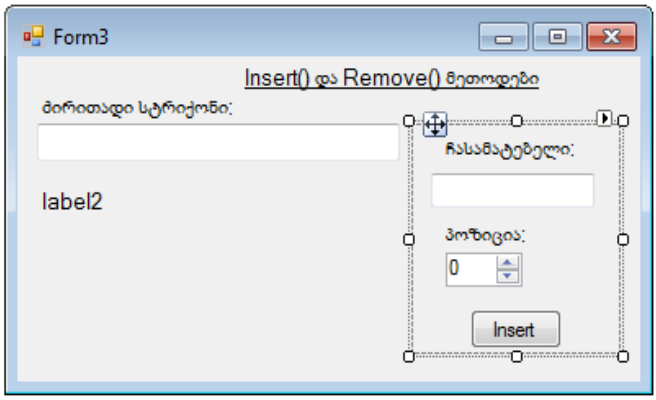
ნახ.2.13

```
// ლისტინგი_2.13 --- LastIndexOf ----
private void button5_Click(object sender, EventArgs e)
 {
   string Striqoni;
   int pozicia;
 label2.Text = ""; label6.Text = ""; label7.Text = "";
    Striqoni = textBox1.Text;
    textBox2.Text = "ლამაზი";
    pozicia = Striqoni.LastIndexOf("ლამაზი");
    label6.Text += pozicia;
    label7.Text += Striqoni.Substring(pozicia);
 }
```

2.4. 4. სტრიქონებთან მუშაობა : დამატება - Insert () , წაშლა - Remove(), ქვესტრიქონი - Substring()

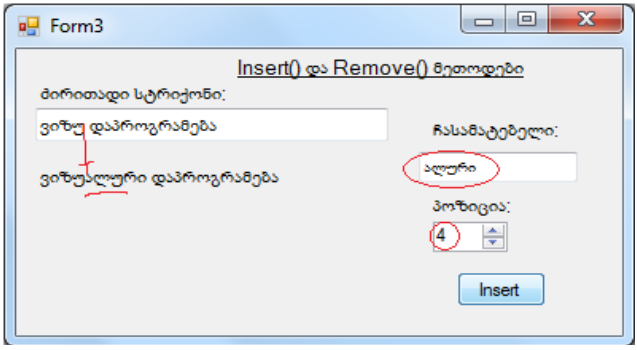
- Insert() მეთოდი უზრუნველყოფს ერთი სტრიქონის შეტანას მეორე სტრიქონში. მაგალითად, ძირითად სტრიქონში (textBox1) შეტანილია რაიმე წინადადება. ჩასამატებელ სტრიქონში პანელზე (textBox2) შეიტანება სიმბოლო ან სიმბოლოთა მწკრივი, რომელიც უნდა ჩაჯდეს პოზიციაში მითითებულ რიცხვის მიხედვით (numericUpDown1). თავიდან numericUpDown1 ელემენტის Properties-ში Maximum=0, Minimum=0 და Value=0. ე.ი. ასეთ დროს ქვესტრიქონი მიუერთდება ძირითადს მხოლოდ დასაწყისში. თუ პოზიციას დავაყენებთ „4“-ზე, მაშინ მივიღებთ სწორ შედეგს. პროგრამის კოდი მოცემულია 2.14_ლისტინგში.

```
// ლისტინგი_2.14 --- Insert() ----------
private void button1_Click(object sender, EventArgs e)
  {
   string Striqoni, qveStriqoni;
   Striqoni = textBox1.Text;
   qveStriqoni=textBox2.Text;
   label2.Text=Striqoni.Insert(
            (int)numericUpDown1.Value,qveStriqoni);
  }
```



ნახ.2.13

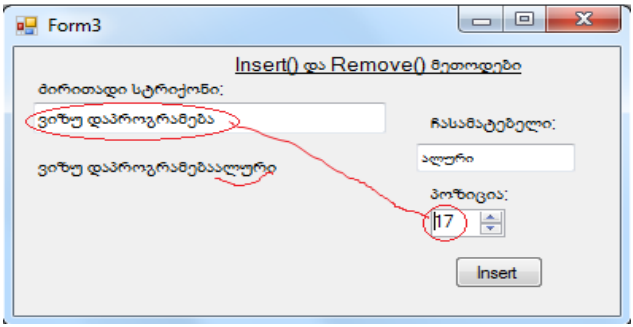
შედეგი 2.14 ნახაზზეა ასახული. აქ ლებელში ჩანს სიტყვა „ვიზუალური“, რომელიც Insert() მეთოდით განხორციელდა.



ნახ.2.14

თუ პოზიციას შევცვლით, მაგალითად „15“-ით, მერე „17“ და მივიღებთ 2.15 ნახაზზე მიღებულ უაზრო შედეგს. ამის მეტი მთვლელი არ გაზრდის მნიშვნელობას, ვინაიდან ის ძირითადი სტრიქონის სიგრძეა.

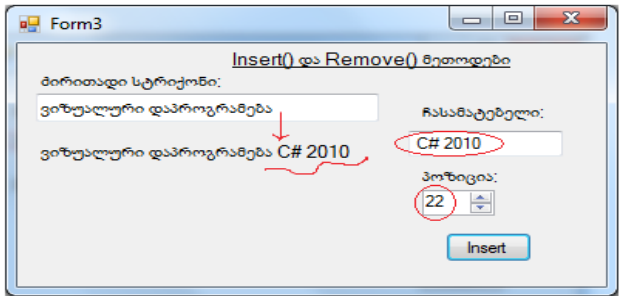
იმისათვის, რომ განხორციელდეს ძირითადი სტრიქონის შეცვლის შესაძლებლობა და მისი სიგრძის პარამეტრის ცვლილება, საჭიროა კოდი, რომელიც 2.15_ლისტინგზეა მოცემული.



ნახ.2.15

```
// ლისტინგი_2.15 ---
private void textBox1_TextChanged(object sender, EventArgs e)
 {
   string Striqoni;
  Striqoni = textBox1.Text;
  numericUpDown1.Maximum = Striqoni.Length;
 }
```

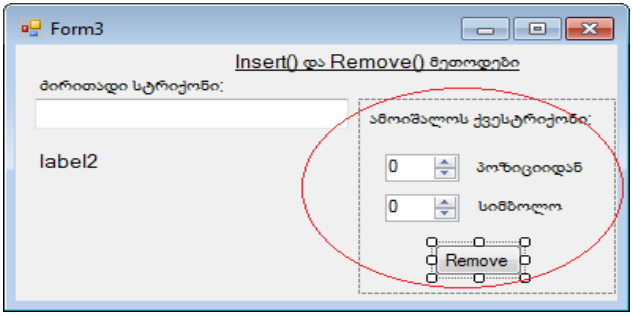
როგორც კი დაიწყება მირითდი სტრიქონის textBox1-ში ტექსტის ცვლილება, მაშინვე იმუშავებს ეს კოდი და შეიცვლება `numericUpDown1.Maximum`-ში სტრიქონის სიგრძის შემზღუდველი რიცხვი (ნახ.2.16).



ნახ.2.16

- Remove() მეთოდი უზრუნველყოფს სტრიქონიდან ქვესტრიქონის ამოშლას, მითითებული პოზიციისა და

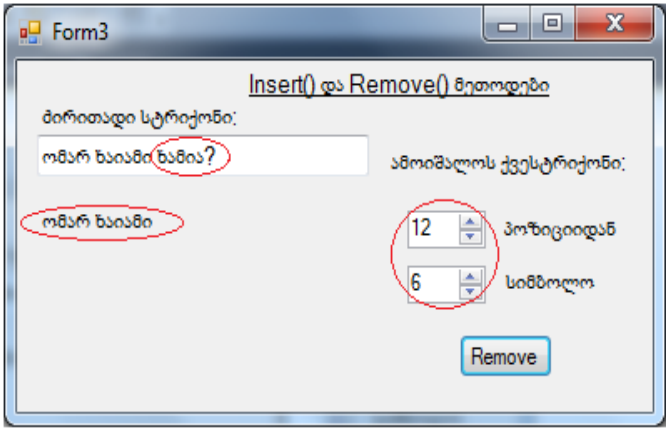
სიმბოლოების რაოდენობის მიხედვით. დავამატოთ ფორმაზე ეს ელემენტები (ნახ.2.17).



ნახ.2.17

```
   // ლისტინგი_2.17 --- Remove() ------
private void button2_Click(object sender, EventArgs e)
 {
   string Striqoni;
   Striqoni = textBox1.Text;
   label2.Text=Striqoni.Remove(
                  (int)numericUpDown1.Value,
                  (int)numericUpDown2.Value);
 }
private void textBox1_TextChanged(object sender, EventArgs e)
 {
   string Striqoni;
   Striqoni = textBox1.Text;
   numericUpDown1.Maximum = Striqoni.Length-1;
   numericUpDown2.Maximum = Striqoni.Length;
 }
private void numericUpDown1_ValueChanged(object sender,
                                          EventArgs e)
 {
   string Striqoni = textBox1.Text;
   numericUpDown2.Maximum = Striqoni.Length -
                            numericUpDown1.Value;  }
```

შედეგები ასახულია 2.18 ნახაზზე.



ნახ.2.18

`numericUpDown1_ValueChanged()` მეთოდის დანიშნულებაა „პოზიციის“ და „სიმბოლოების“ მრიცხველებში დასაშვები რიცხვების კონტროლი. მაგალითად, ჩვენ შემთხვევაშია 12 და 6. ბოლო სიტყვა მთლიანად ამოშლილია.

თუ პოზიციაში ჩავწერთ 13-ს, მაშინ სიმბოლოში ავტომატურად გამოჩნდება 5, თუ 14, მაშინ 4 და ა.შ., ბოლოს 17-ზე გვექნება 1. მეტს ვეღარ შევცვლით პოზიციის მეტობისკენ. პირიქით პოზიციის შემცირება დასაშვებია, მაგალითად დავაყენოთ 5. მაშინ სიმბოლოების რაოდენობა, რომელიც შეიძლება წავშალოთ მაქსიმალურად იქნება 13 და შედეგად მივიღებთ სიტყვას „ომარ“.

2.5. C# ენის ოპერატორები

ენაში გამოიყენება სხვადასხვა ტიპის ოპერატორები, კერძოდ არითმეტიკული, ლოგიკური, სტრიქონების გადაბმის, ინკრემენტ-დეკრემენტის, წაძვრის, რელაციური, ობიექტების შექმნის და სხვ. ქვემოთ ცხრილში მოცემულია ეს ოპერატორები.

C# Operators

Operator category	Operators
Arithmetic	+ - * / %
Logical (boolean and bitwise)	& | ^ ! ~ && || true false
String concatenation	+
Increment, decrement	++ --
Shift	<< >>
Relational	== != < > <= >=
Assignment	= += -= *= /= %= &= |= ^= <<= >>=
Member access	.
Indexing	[ ]
Cast	( )
Conditional	?:
Delegate concatenation and removal	+ -
Object creation	new
Type information	as is sizeof typeof
Overflow exception control	checked unchecked
Indirection and Address	* -> [ ] &

შენიშვნა: ოპერატორების დეტალური განმარტებები შეიძლება იხილოთ ლიტერატურულ წყაროებში [4,6,10].

2.6. კითხვები და სავარჯიშოები

2.1. C#.NET ენის მონაცემთა რომელ საბაზო ტიპებს იცნობთ, როგორია მათი მნიშვნელობათა დიაპაზონები ?

2.2. რას ნიშნავს ტიპების ცხადი და არაცხადი გარდაქმნის პროცედურა ?

2.3. როგორ გამოიყენება Convert კლასი ?

2.4. როგორ შევკრიბოთ ორ textBox-ში შეტანილი ორი რიცხვი და ჩავწეროთ მესამე textBox -ში ?

2.5. ააგეთ ფორმა „ვალუტის გაცვლა“: ლარების გადასაყვანად დოლარებსა და ევროებში, მითითებული სავალუტო კურსით.

2.6. რა არის გლობალური და ლოკალური ცვლადები ?

2.7. ააგეთ C# კოდი, რომელშიც ორი button-ით მოხდება დოლარების და ევროების ორ textBox-ში მითითებული თანხების გადაყვანა ლარებში, ხოლო მესამე button-ით შედეგი ჩაიწერება label-ში.

2.8. რას წარმოადგენს ჩამოთვლითი enum-ტიპი და როგორ გამოიყენება იგი ?

2.9. ააგეთ პროგრამა, რომელიც შექმნის „Month“ (12 თვე დასახელებებით) ჩამოთვლითი enum-ტიპის გამოყენებით. button-ით გამოიტანეთ label-ში ნომერი და თვის დასახელება (განახორციელეთ ყველა თვის მიმდევრობით გამოტანა).

2.10. ააგეთ კოდი 2.9 სავარჯიშოს მიხედვით, ოღონდ label-ში გამოიტანეთ მხოლოდ კენტი თვეების დასახელებები.

2.11. რას წარმოადგენს სტრიქონული ტიპი ?

2.12. როდის და როგორ ვიყენებთ string სპეციფიკატორს და String კლასს ?

2.13. String კლასის რომელ თვისებებს იცნობთ ?

2.14. String კლასის რომელ მეთოდებს იცნობთ ?

2.15. ააგეთ კოდი, რომელიც მოძებნის ტექსტურ ფაილში მითითებულ სიტყვას და დაითვლის ამ სიტყვის რაოდენობას.

2.16. ჩაამატეთ სტრიქონის მითითებულ ადგილას ახალი ქვესტრიქონი.

2.17. ჩაანაცვლეთ ტექსტურ ფაილში რამდენიმე ადგილას ”ძველი სტრიქონი” მითითებული ”ახალი ქვესტრიქონით”.

თავი 3. ვიზუალური ელემენტები და პროგრამირება

Windows-დაპროგრამება C# ენის ბაზაზე ორი ნაწილისგან შედგება:

- ვიზუალური მართვის ელემენტების გამოყენება და
- პროგრამირება უშუალოდ ენის საფუძველზე.

ამ თავში ჩვენ ორივე ნაწილს განვიხილავთ პარალელურად. ყურადღება გამახვილდება System.Windows.Forms სახელსივრცის Control საბაზო კლასიდან წარმოებულ ვიზუალური მართვის ელემენტებზე, როგორიცაა Panel, TextBox, NumericUpDown, ListBox, ComboBox, CheckBox, RadioButton, Timer, PictureBox და სხვ.

წინასწარ განვიხილოთ Control კლასის თვისებები, მეთოდები და მოვლენები, რომლებიც საერთოა აღნიშნული ელემენტებისათვის.

3.1. Control საბაზო კლასის თვისებები

- BackColor / BackgroundImage – მართვის ელემენტის ფონის ფერი / მართვის ელემენტის ფონური გამოსახულება
- BackgroundImageLayout – ფონური გამოსახულების ადგილმდებარეობა მართვის ელემენტზე, რომელიც მოცემულია ჩამოთვლით ტიპში
- Cursor – როგორ აისახება კურსორი მაუსის მიმთითებლის მიტანით მართვის ელემენტზე
- Font – რომელი ფონტი (შრიფტი) გამოიყენება მართ[ის ელემენტზე
- ForeColor – მართვის ელემენტზე გამოსატანი ტექსტის ფერი
- RightToLeft – მიუთითებს ტექსტის გამოტანის მიმერთულების ცვლილებაზე
- Text – მართვის ელემენტზე გამოსატანი ტექსტი
- UseWaitCursor – მოლოდინის კურსორი გამოვიყენოთ (true) თუ არა (false)
- Tag – მომხმარებლის მიერ განსაზღვრული მონაცემთა შენახვა კონკრეტული მართვის ელემენტის მიხედვით

• Anchor / Dock / Location / MaximumSize / MinimumSize / Size – კონტეინერის საზღვრებია, რომელშიც მოთავსებულია მართვის ელემენტი და შეუძლია ცვლილება მისი ”მშობლის” ცვლილებასთან ერთად / ავტომატური ცვლილების განხორციელება / მართვის ელემენტის ზედა-მარცხენა კუთხის კოორდინატები კონტეინერთან მიმართებაში / მართვის ელემენტის ზომები (მაქსიმალური, მინიმალური)

• Margin / Padding – ცარიელი ადგილი მართვის ელემენტებს შორის / განსაზღვრავს მართვის ელემენტის შევსებას

• Enabled – მართვის ელემენტის მოქმედება ნებადართულია თუ არა (false)

• TabIndex/TabStop – მართვის ელემენტების მიმდევრობითობა კონტეინერის შიგნით. თუ ელემენტს TabStop-ში აქვს false, მაშინ ის არ შედის მიმდევრობითობაში

• Visible – მართის ელემენტი ხილვადია (true) ან არა (false).

3.2. Control - საბაზო კლასის მეთოდები

• BeginInvoke() / Invoke() / EndInvoke() – უზრუნველყოფს InvokeRequired თვისების მნიშვნელობის true-ში გადაყვანას, ობიექტის დელეგატის მიერ მეთოდის გამოსაძახებლად სხვა ნაკადიდან

• Contains() – ამოწმებს, არის თუ არა მართვის ელემენტი (გამოძახებული მეთიდისთვის) „შვილობილი“

• CreateGraphics() – უზრუნველყოფს Graphics ობიექტის მიღებას მართვის ელემენტისთვის

• Dispose()- ათავისუფლებს ყველა რესურსს, რომელსაც იყენებს Control-ობიექტი

• FindForm() – მიიღება ფორმა, რომელზეც არის მართვის ელემენტი

• Focus() – თუ მართის ელემენტი ხილვადია (Visible==true) ან აქტივირებადია (Enabled==true), მაშინ მას ექნება „ფოკუსის“ თვისება

• GetChildAtPoint() – მიიღება „შვილობილი“ მართვის ელემენტი მითითებული კოორდინატებით. თუ აქ არაა მართვის ელემენტი, მაშინ „ცარიელი მიმართვაა“ (nullNothingnullptr)

• GetContainerControl() – აბრუნებს ContainerControl-ის მომდევნო ობიექტს მართვის ელემენტების ჯაჭვში

• GetNextControl() – იძახებს მომდევნო ან წინა მართვის ელემნტს Tab-მიმდევრობაში (იხ.TabIndex)

• Hide() / Show() – მალავს / გამოაჩენს მართვის ელემენტს

• GetStyle() / SetStyle() – განსაზღვრავს მართვის ელემენტის წარმოდგენის სტილს

• Invalidate() - ხდის უმოქმედოს მართვის ელემენტის მთლიან ზედაპირს და ამზადებს ხელახალი აგებისთვის (Update)

• Refresh() - იძულებით ქმნის პირობებს, რომლის დროსაც მართვის ელემენტი ბლოკავს კლიენტის არეს და სასწრაფოდ განაახლებს თავის გარემოს

• Update() – მართვის ელემენტი განაახლებს კლიენტის არეებს

3.3. Control საბაზო კლასის მოვლენები

• Click / DoubleClick – მაუსის მარცხენა ღილაკზე რეაქცია (EventHandler დელეგატის გამოყენებაზეა ბაზირებული). აქ მუშაობს Enter-იც

• KeyDown / KeyUp / KeyPress – კლავიატურის ღილაკზე რეაქცია

• MouseClick / MouseDoubleClick – მხოლოდ მაუსისთვის (MouseEventHandler დელეგატის გამოყენებაზეა ბაზირებული)

• MouseDown / MouseUp – რეაქცია მაუსის ღილაკის დაჭერაზე / აშვებაზე (MouseEventHandler დელეგატის გამოყენებაზეა ბაზირებული)

• MouseEnter / MouseHover / MouseLeave / Mousewheel / MouseMove – მაუსის კურსორის მიტანით მართვის ელემენტზე ჩაირთვება მოვლენა MouseEnter, შემდეგ შეიძლება განხორციელდეს ამ ელემენტის გადატანა, ამ ელემენტზე გამოჩნდეს ინფორმაცია და ა.შ.

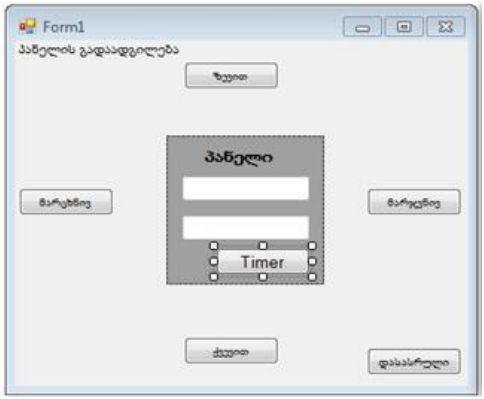
• Move – მართვის ელემენტის მდებარეობის შეცვლის მოვლენა ფორმაზე

• Paint – მართვის ელემენტის ხელახალი გამოხაზვის მოვლენა

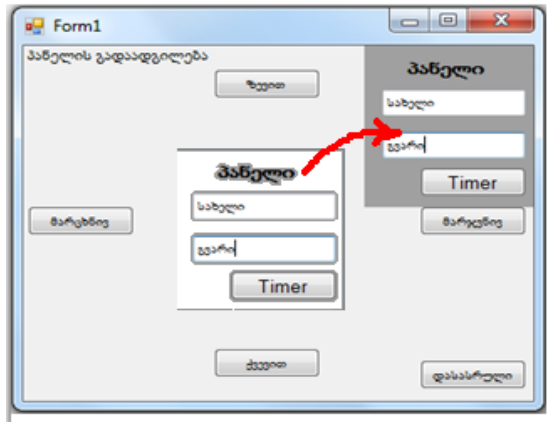
• Resize – მართვის ელემენტის ზომების შეცვლის მოვლენა

3.4. მართვის კონტეინერული ელემენტი Panel

ვინდოუსის ფორმაზე მართვის ელემენტის - პანელის (Panel) გამოყენებას განსაკუთრებული მნიშვნელობა აქვს. იგი კონტეინერია, რომელზედაც შეიძლება მოთავსდეს სხვა ელემენტები (მაგალითად, label, textBox, comboBox და ა.შ.). პანელის გადაადგილებით ფორმაზე იგი თან გაიყოლებს მასზე მოთავსებულ კომპონენტებსაც, რაც მეტად მოხერხებულია. ასევე შესაძლებელია პანელის კოპირება თავის ელემენტებიანა სხვა ფორმაზე. 3.1 ნახაზზე ილუსტრირებულია პანელის და ოთხი ღილაკის (მარცხნივ, მარჯვნივ, ზევით და ქვევით) მაგალითი.



Properties-ში პანელის პარა-მეტრებია (თვისებები):
BackColor= ControlDark
//პანელის ფონის ფერი
Location=120,80
// პოზიცია ფორმაზე
X=120, Y=80
Size=125,125 // პანელის
// ზომა პიქსელ.:
// სიგანე, სიმაღლე



ნახ.3.1

3_1 ლისტინგში მოცემულია პროგრამის კოდი, რომელშიც პანელის გადაადგილება ფორმაზე ხდება ღილაკების გამოყენებით. ნახაზზე ნაჩვენებია პანელისა და მისი „შიგთავსი“ ელემენტების საბოლოო მდებარეობა ფორმის ზედა მარჯვენა კუთხეში.

```
// ლისტინგი_3.1 – Panel-ის გადაადგილება ფორმაზე
using System;
using System.Drawing;
using System.Windows.Forms;
namespace WinFormPanel
{
 public partial class Form1 : Form
  {
    public Form1() { InitializeComponent(); }
 private void button1_Click(object sender, EventArgs e)
  { // Top
   panel1.Location = new Point(panel1.Location.X,
                               panel1.Location.Y - 10);
  }
 private void button2_Click(object sender, EventArgs e)
  { // Bottom
   panel1.Location = new Point(panel1.Location.X,
                               panel1.Location.Y + 10);
  }
 private void button3_Click(object sender, EventArgs e)
  { // Left
   panel1.Location = new Point(panel1.Location.X-10,
                               panel1.Location.Y);
  }
 private void button4_Click(object sender, EventArgs e)
  { // Right
   panel1.Location = new Point(panel1.Location.X + 10,
                               panel1.Location.Y);
  }
 private void button5_Click(object sender, EventArgs e)
  {
    Close();  }
}}
```

პროგრამის ტექსტში კონსტრუქტორი Point() ასრულებს კლასის ეგზემპლარის ინიციალიზებას ახალი X,Y კოორდინატებით, რომლის შემდეგაც ობიექტი panel1 გადაადგილდება ფორმაზე 10 პიქსელით მითითებული მიმართულებით.

3.5. მართვის ელემენტი Timer

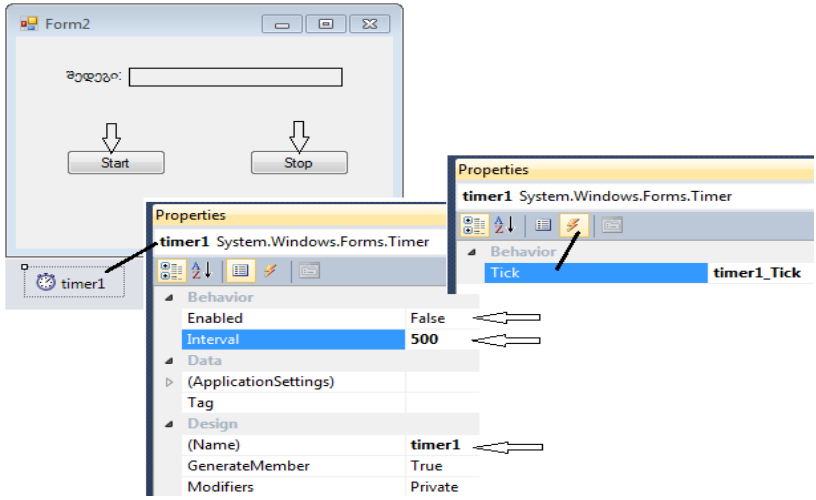
კლასი Timer უზრუნველყოფს მოვლენის ამოქმედებას მომხმარებლის მიერ განსაზღვრულ ინტერვალში. ის გამოიყენება ვინდოუსის ფორმების აპლიკაციებში. მოვლენის სახელია Tick და ის ხდება მაშინ, როდესაც მოცემული დროის პერიოდი გავიდა და ტაიმერი ჩართულია.

3.1 ნახაზზე წარმოდგენილ პანელზე ჩანს ღილაკი Timer, რომლის ამუშავებითაც გამოიძახება Form2 და მომზადდება მასზე ორი ღილაკი: Timer-ის მოვლენის ამოქმედების (Start) და მოვლენის შეჩერების (Stop). ამ ღილაკების მანიპულირებით „შედეგში“ გამოიტანება „G” სიმბოლოს კონკაქტენაციით მიღებული სტრიქონი - ინტერვალით 500, რაც შეესაბამება 0.5 წამს.

3.2 ნახაზზე ნაჩვენებია Form2 თავისი ელემენტებით და თვისებებით. Timer გადმოიტანება კომპონენტების პანელიდან, იგი თავსდება ავტომატურად ფორმის ქვემოთ. მისი მონიშვნის შემდეგ Properties-ში დაყენდება საჭირო მნიშვნელობები.

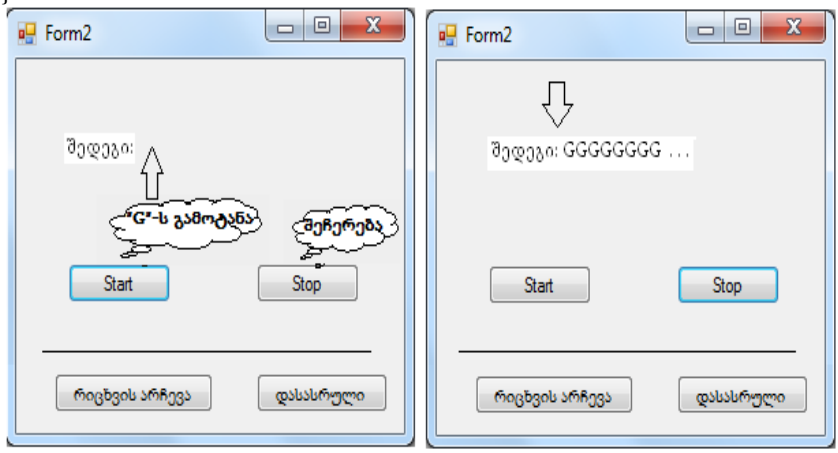
პროგრამის ტექსტი მოცემულია 3.2 ლისტინგში, სადაც Form2-ზე განლაგებული Start, Stop ღილაკებიცაა აღწერილი.

3.3 ნახაზზე ნაჩენებია საბოლოო შედეგის ამსახველი ფორმა.



ნახ.3.2

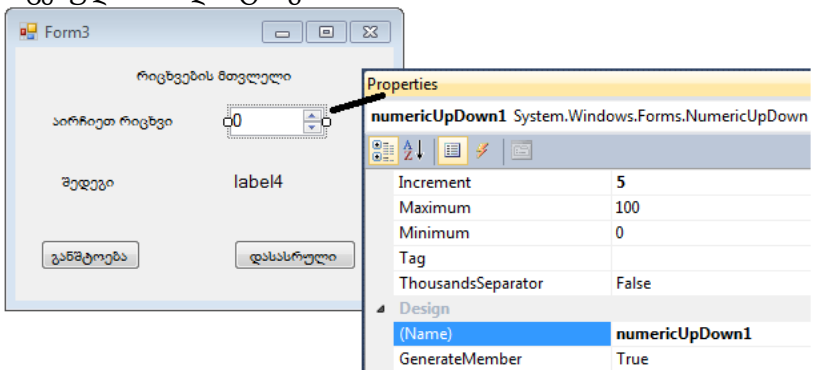
```
// ლისტინგი_3.2 ---- Timer - ელემენტი --------
using System;
using System.Drawing;
using System.Windows.Forms;
namespace WinFormPanel
{
 public partial class Form2 : Form
  {
    public Form2()  { InitializeComponent();   }
    private void button1_Click(object sender, EventArgs e)       // Start
      {
        timer1.Enabled = true;
      }
    private void button2_Click(object sender, EventArgs e)       // Stop
      {
        timer1.Enabled = false;
      }
    private void timer1_Tick(object sender, EventArgs e)   // მოვლენის
      {                                    // ამუშავება დაშედეგის გამოტანა
        label1.Text += "G";
      }
    private void button3_Click(object sender, EventArgs e)
      {   Close();  }
  }
}
```



ნახ.3.3. საწყისი და საბოლოო შედეგის ფანჯრები

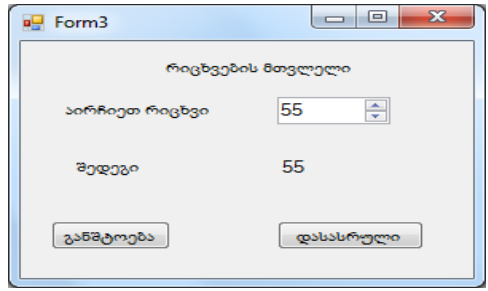
3.6. მართვის ელემენტი - რიცხვების არჩევა: NumericUpDown

ელემენტი NumericUpDown საშუალებას იძლევა შევარჩიოთ საჭირო რიცხვი მთვლელის რეჟიმში (პატარა ისრები მარჯვენა მხარეს, რომლებითაც ხდება საწყისი რიცხვის (Value) მატება ან კლება Increment-თვისებაში მითითებული ბიჯით) ან ავკრიფოთ იგი ხელით კლავიატურიდან (ნახ.3.4). Properties-ში მიეთითება აგრეთვე რიცხვის სავარაუდო Maximum და Minimum მნიშვნელობები. შედეგი აისახება label4-ში. კოდის ფრაგმენტი მოცემულია 3.3 ლისტინგში.



ნახ.3.4

```
// ლისტინგი_3.3 – ValueChanged მოვლენა ----
private void numericUpDown1_ValueChanged(object sender,
                                         EventArgs e)
  {
     label4.Text = numericUpDown1.Value.ToString();
  } // საბოლოო შედეგი მოცემულია 3.5 ნახაზზე.
```



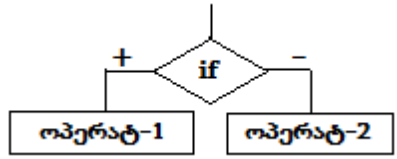
ნახ.3.5

3.7. განშტოებები: if / if…else / if…else if…

როგორც ცნობილია, პროგრამის შესრულება ხდება მისი ოპერატორების ფიზიკურად დალაგებული მიმდევრობით, ან ლოგიკური პირობის დაკმაყოფილების მიხედვით, ან უპირობო გადასვლის ოპერატორით (მაგალითად, goto - ოპერატორი, რომელიც სტრუქტურული დაპროგრამების პრინციპებიდან გამომდინარე სასურველია არ იქნას გამოყენებული).

პროგრამის მმართველი სტრუქტურებიდან ორი ტიპის კონსტრუქცია გამოიყენება: განშტოებები და ციკლები. აქ ჩვენ განვიხილავთ პირველ მათგანს, ხოლო ციკლებს 3.9 პარაგრაფში დავუბრუნდებით. განშტოებას აქვს შემდეგი სინტაქსი (ნახ.3.6):

```
if (პირობა)      // პირობა არის :
                 // true - ჭეშმარიტი (+)
                 // ან false -მცდარი (-)
  {
     ოპერატორები-1;  // სრულდება,
       // თუ პირობა არის true
  }
[else     // თუ არა, მაშინ
  {
     ოპერატორები-2;  // სრულდება,
       // თუ პირობა არის false
  } ]
```



ნახ.3.6

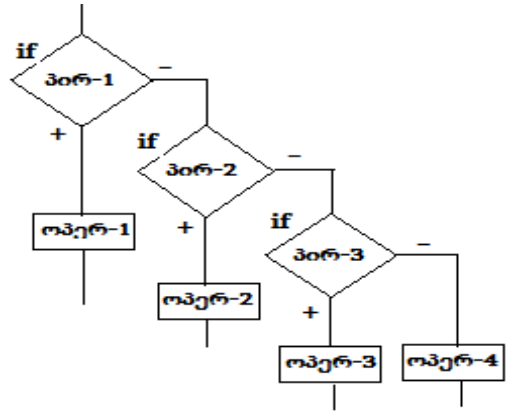
სინტაქსში „[ ]“ კვადრატული ფრჩხილები ნიშნავს, რომ მისი შიგთავსი ყოველთვის არაა აუცილებელი. ამგვარად, **„ოპერატორები-1“** ყოველთვის შესრულდება (true-ს დროს), ხოლო false-ს დროს მართვა გადაეცემა if-ბლოკის გარეთ მომდევნო ოპერატორს. თუ **„ოპერატორები-1“** მხოლოდ ერთი ოპერატორია, მაშინ „{ }“-ფიგურული ფრჩხილები არაა საჭირო.

if კონსტრუქციაში „**პირობა“**, ანუ „შედარების გამოსახულება“ შეიძლება იყოს რთული, შედგებოდეს რამდენიმე შედარებისგან. ასეთ განშტოებას „ჩადგმული“ ეწოდება და რამდენიმე ერთმანეთის შიგნით მოთავსებული if-ბლოკისგან შედგება (ნახ.3.7).

განვიხილოთ შესაბამისი პროგრამული რეალიზაციის მაგალითი მმართველი ვიზუალური ელემენტების გამოყენებით. ავაგოთ Form4 (იხ.ნახ.3.8), რომელსაც გამოიძახებს ღილაკი „განშტოება“ (ნახ.3.5).

სინტაქსი:

```
if (პირობა-1)
  {ოპერატ-1}
 else
  if (პირობა-2)
    {ოპერატ-2}
   else
     if (პირობა-3)
      {ოპერატ-3}
     else
      {ოპერატ-4}
```

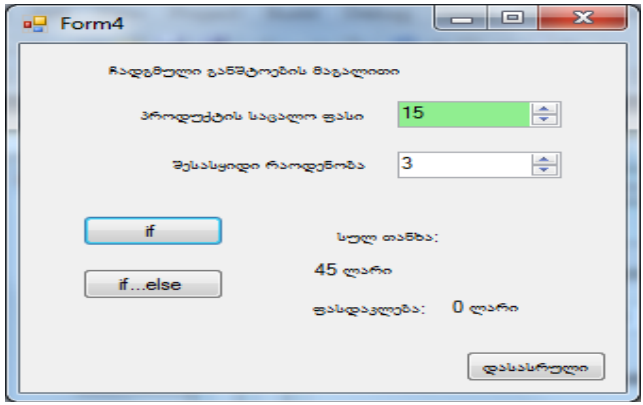


ნახ.3.7

ამოცანა_3.1: მდგომარეობს შემდეგში, კლიენტმა უნდა აირჩიოს შესაძენი პროგრამული პროდუქტი მისთვის მისაღები საცალო ფასით X; შემდეგ უნდა განსაზღვროს შესასყიდი პროდუქციის რაოდენობა Y; მონაცემთა ასარჩევად გამოყენებულია მთვლელები (NumericUpDown). პროდუქციის მთლიანი ღირებულება განისაზ-ღვრება გამოსახულებით: P=X*Y, სადაც X=f(Y) ანუ საცალო ფასდაკლება განისაზღვრება სკალით:

N	რაოდენობა (Y)	საცალო ფასდაკლება (fd %)
1	1-4	0
2	5-20	10
3	21 -მეტი	25

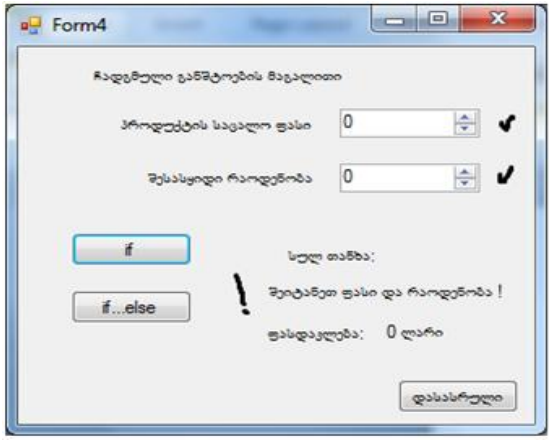
პროგრამამ უნდა გაიანგრიშოს მთლიანი ღირებულება და გამოიტანოს Form4 ფანჯრის შედეგის ველში (ნახ.3.8). ღილაკის კოდი მოცემულია 3.4 ლისტინგში.



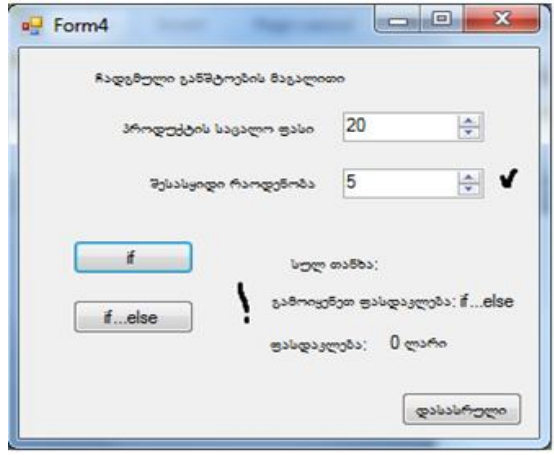
ნახ.3.8

```
// ლისტინგი_3.4  ----- if-ღილაკის კოდის ფრაგმენტი ----
using System;
using System.Drawing;
using System.Windows.Forms;
namespace WinFormPanel
{
 public partial class Form4 : Form
  {
   public Form4() {InitializeComponent(); }
 private void button1_Click(object sender, EventArgs e)
  { // if - ღილაკი
    int X = (int)numericUpDown1.Value; // ტიპის გარდაქმნა
                                       // cast(int) -ით
    int Y = (int)numericUpDown2.Value;
    int z=0;
    label5.Text = "";
    numericUpDown1.BackColor = Color.White;
  // მაგ.:1
  if (X == 0 || Y == 0)
    label5.Text = "შეიტანეთ ფასი და რაოდენობა !";
  // მაგ.:2
  if (X > 0 && Y > 4)
    label5.Text = "გამოიყენეთ ფასდაკლება: if...else";
  // მაგ.:3
  if (X > 0 && Y > 0 && Y < 5)
    {
```

```
        z = X * Y;
        label5.Text = z.ToString() +" ლარი";
        label7.Text = "0 ლარი";
        numericUpDown1.BackColor = Color.LightGreen;
      }
    }
private void button4_Click(object sender, EventArgs e)
  { Close(); }
 }
} // კოდში გამოიყენება შეტყობინებები მომხმარებლისთვის (ნახ.3.9):
```



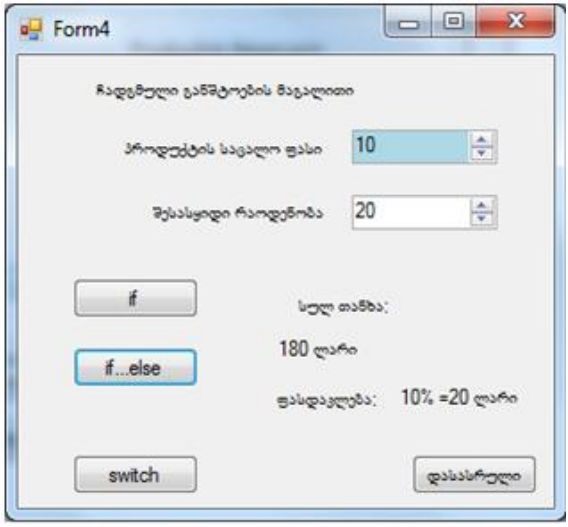
ნახ.3.9

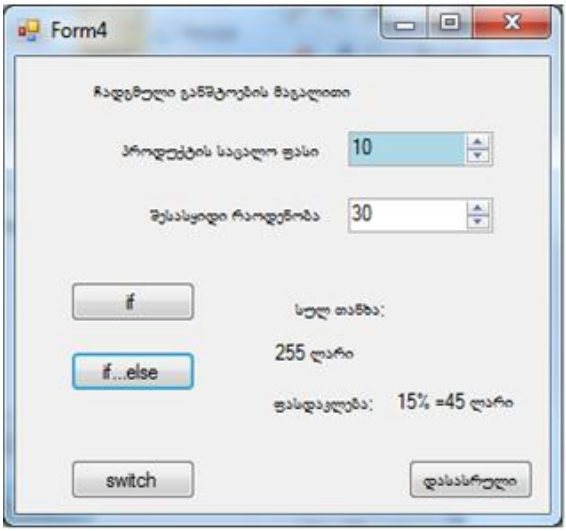


ახლა განვიხილოთ if…else ღილაკის შემთხვევა, ანუ იგი მუშაობს მაშინ, როცა კლიენტი ირჩევს პროდუქციის რაოდენობას 5-20 ფარგლებში (10%-ანი ფასდაკლება), ან 21-დან ზევით (15%-ანი ფასდაკლება). k-ფასდაკლების კოეფიციენტი განისაზღრება if…else ბლოკიდან. კოდის ტექსტი მოცემულია 3.5 ლისტინგში.

```
// ლისტინგი_3.5  ----- if…else ღილაკის კოდის ფრაგმენტი ----
private void button2_Click(object sender, EventArgs e)
 {
  int X = (int)numericUpDown1.Value;
  int Y = (int)numericUpDown2.Value;
  int z = 0, fd = 0, k = 0; // z-სულ თანხა; fd-ფასდაკლება;
                            // k-ფასდაკლების კოეფიციენტი
  label5.Text = "";
  numericUpDown1.BackColor = Color.White;
  if (X == 0 || Y == 0)
    label5.Text = "შეიტანეთ ფასი და რაოდენობა !";
  if (X > 0 && Y > 20)
    label5.Text = "გამოიყენეთ ფასდაკლება: if.else if";

  if (X > 0 && Y < 5)
    label5.Text = "ფასდაკლება არაა, გამოიყენეთ if ბლოკი !";
   else
    {
     if (X>0 && Y<=20)
        k = 10; // 10 %-იანი ფასდაკლება
      else
        k=15;   // 15 %-იანი ფასდაკლება, თუ Y>20
     fd=(X*Y)*k/100;
     z = X * Y - fd;
     label5.Text = z.ToString() + " ლარი";
     label7.Text = k.ToString() + "% =" +
                   fd.ToString() + " ლარი";
     numericUpDown1.BackColor = Color.Blue;
  } }
```

პროგრამის მუშაობის შედეგები, როცა კლიენტი ირჩევს პროდუქციის მეტ რაოდენობას, მოცემულია 3.10 ნახაზზე.

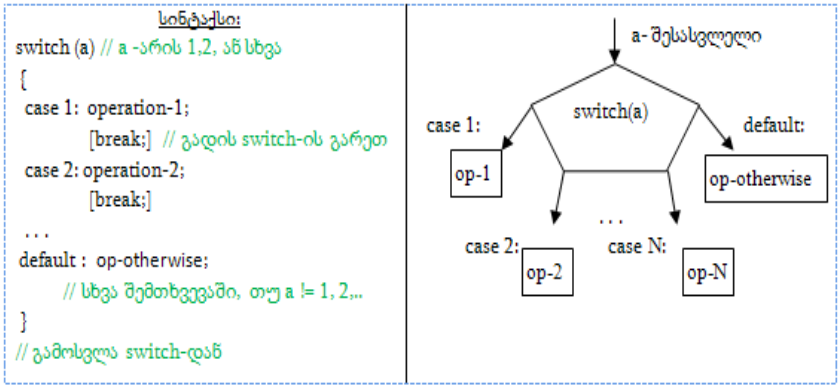




ნახ.3.10

3.8. განშტოება - გადამრთველი: switch…case

3.7 ნახაზზე ჩვენ განვიხილეთ if…else if… ჩადგმული განშტოებების შემთხვევა (აქ იყო ნაჩვენები 3 if ბლოკი). პრაქტიკული ამოცანების გადაწყვეტისას ხშირად გვხვდება შემთხვევები, როდესაც ასეთი ჩადგმული ბლოკების რაოდენობა 10 ან მეტია (მაგალითად, მენიუს პუნქტები 99 სტრიქონით). ასეთ შემთხვევაში if…else if… კონსტრუქციის გამოყენება მოუხერხებელია. ამისათვის დაპროგრამების ენაში შემოიტანეს გადამრთველის switch…case კონსტრუქცია (ნახ.3.11).



ნახ.3.11

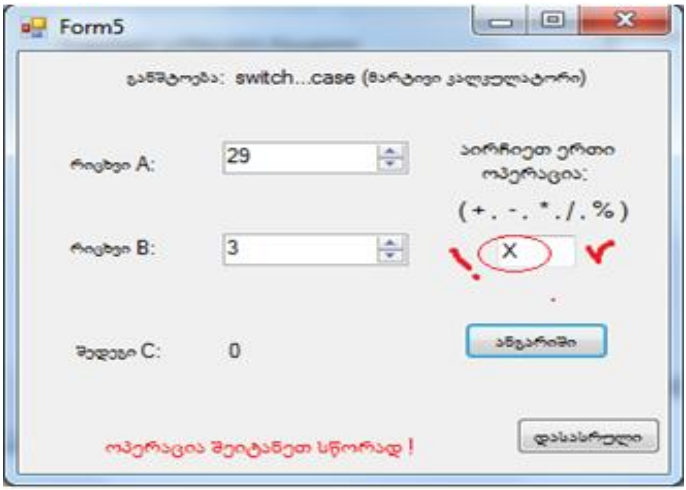
ამოცანა_3.2: ავაგოთ მარტივი კალკულატორი, რომელიც ვიზუალური ელემენტების გამოყენებით საშუალებას მოგვცემს ავირჩიოთ ორი მთელი რიცხვი და მათემატიკური ოპერაცია +, -, *, /, % (მოდული). პროგრამის კოდი მოცემულია 3.6 ლისტინგში, ხოლო 3.12 ნახაზზე ასახულია კალკულატორის ფორმა და შედეგები.

```
// ლისტინგი_3.6 --- switch…case ----------
using System;
using System.Drawing;
using System.Windows.Forms;
namespace WinFormPanel
{
  public partial class Form5 : Form
   {
    public Form5() { InitializeComponent(); }
```

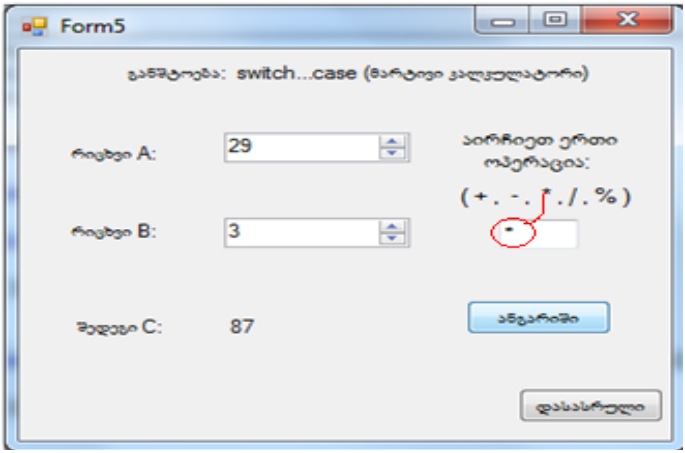
```
private void button1_Click(object sender, EventArgs e)
  { // ანგარიში
    string op = "";
    int A = (int)numA.Value;
    int B = (int)numB.Value;
    int C=0;
    label8.Text = " ";
    op = textBox1.Text;
    switch (op)
      {
       case "+": C = A + B;
                  break;
       case "-": C = A - B;
                 break;
       case "*": C = A * B;
                 break;
       case "/":
              if (B != 0)  // გამყოფში 0-ის გამორიცხვა
                 C = A / B;
               else
                label8.Text = "B არ შეიძლება იყოს 0 !";

              break;
       case "%": C = A % B;
                 break;
       default:// თუ ოპერაციის სიმბოლო  არასწორად იქნა არჩეული
            label8.Text = "ოპერაცია შეიტანეთ სწორად !";
             break;
      }
      NumC.Text = C.ToString();
    }
private void button6_Click(object sender, EventArgs e)
   {Close();}
 }
}
```

პროგრამაში გათვალისწინებულია მომხმარებლის მიერ შესაძლო გაუთვალისწინებელი შეცდომების აღმოფხვრის საკითხები. მაგალითად, „რიცხვი-B“ არ შეიძლება იყოს „0“, თუ დაგეგმილია გაყოფის ოპერაციის გამოყენება. ამ შემთხვევაში debugger აჩერებს პროგრამას და გამოაქვს წყვეტის შეტყობინება. ეს

რომ არ მოხდეს `case "/":` ბლოკში ჩასმულია კონტროლის if(B!=0) … else ოპერატორი. ასევე, ოპერაციის არჩევისას Form5-ზე (+,-,*,/,%) სიმბოლოები ზუსტად უნდა იყოს არჩეული, ან გამოიცემა შესაბამისი შეტყობინება `default:`-ში, თუ კლავიატურაზე არასწორადაა არჩეული სიმბოლო.



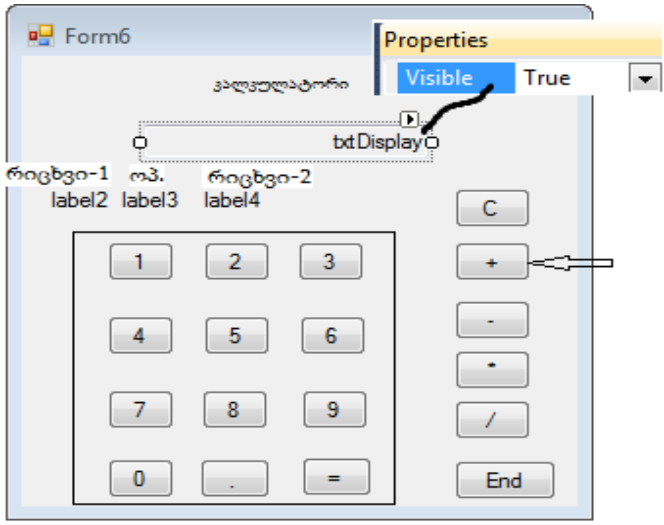
ნახ.3.12-ა



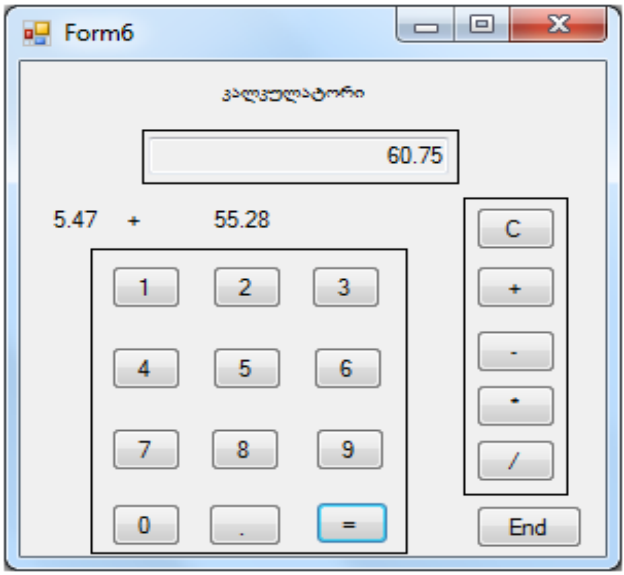
ნახ.3.12-ბ

ამოცანა_3.3: განვიხილოთ კალკულატორის შედარებით რეალური მოდელის აგება (ნახ.3.13-ა). ელემენტებად გამოყენებულია დისპლეის ერთი textBox ელემენტი (Name=txtDisplay), რიცხვებისთვის 0,1,..9 button, +, -, *, /, % - ოპერაციებისთვის ოთხი button, „მცოცავი“ წერტილის „ .“ – button, შედეგის ფიქსირების „=“ – button და დისპლეის გასუფთავების “C” (Clear) button.

ჩვენს შემთხვევაში txtDisplay.Text ელემენტის თვისება Visibile არჩეულია True, რაც გამორიცხავს ამ ბლოკში სიმბოლოების ხელით ჩაწერას (შემთხვევითი შეცდომების გამოსარიცხად). შესატანი რიცხვები უნდა აირჩეს „0-9“ და „.“ ღილაკების საშუალებით.



ნახ.3.13. ა-საწყისი მდგომარეობა



ნახ.3.13. ბ-საბოლოო მდგომარეობა

ღილაკების თავზე „რიცხვი-1“ „ ოპ.“ „რიცხვი-2“, რომლებიც label 2,3,4-ში თავსდება, გამოიყენება შეტანილი რიცხვების, ოპერაციების და შედეგების საკონტროლოდ (ეს დამხმარე ვიზუალური ელემენტებია, რომლებიც შეიძლება არ იყოს კალკულატორზე).

მუშაობის ალგორითმი ასეთია:

1. პროგრამის ამუშავება და Form6 კალკულატორის ინტერფეისის გამოტანა;

2. C ღილაკით დისპლეის გასუფთავება;

3. პირველი რიცხვის შეტანა 0-9 და „.“ ღილაკების მიმდევრობითი არჩევით;

4. ერთი ოპერაციის არჩევა +, -, *, /, % ღილაკებიდან (მაგალითად, „+“. ამ დროს დისპლეის ტექსტბოქსი სუფთავდება, რიცხვი დამახსოვრებულია და კონტროლის სტრიქონი შეივსება პირველი რიცხვით და ოპერაციის სიმბოლოთი);

5. მეორე რიცხვის შეტანა 0-9 და „.“ ღილაკების მიმდევრობითი არჩევით;

6. „=“ ღილაკით არჩეული ოპერაციის შესრულება ორ რიცხვზე და შედეგის ასახვა დისპლეის ეკრანზე (ნახ.3.13-ბ).

მესამე , მეოთხე და ა.შ. შესაკრები რიცხვების დასამატებლად საჭიროა იგივე პროცედურების: „+“, რიცხვის შეტანა, „=“ ... „+“, რიცხვის შეტანა, „=“ გამოყენება.

3.7 ლისტინგში ნაჩენებია ზემოაღწერილი ალგორითმის შესაბამისად მომუშავე კალკულატორის პროგრამის კოდის ფრაგმენტი „+“ ოპერაციისთის.

```
// ლისტინგი_3.7 -- კალკულატორი
using System;
using System.Drawing;
using System.Windows.Forms;
namespace WinFormPanel
{
 public partial class Form6 : Form
 {
    double total1=0, total2=0;
    public Form6()
 {
   InitializeComponent();
  }
private void button1_Click(object sender, EventArgs e)
   {
     txtDisplay.Text = txtDisplay.Text + "1";
   }
private void button2_Click(object sender, EventArgs e)
```

```
    {
      txtDisplay.Text = txtDisplay.Text + "2";
    }

private void button3_Click(object sender, EventArgs e)
    {
      txtDisplay.Text = txtDisplay.Text + "3";
    }

private void button4_Click(object sender, EventArgs e)
    {
      txtDisplay.Text = txtDisplay.Text + "4";
    }

private void button5_Click(object sender, EventArgs e)
    {
      txtDisplay.Text = txtDisplay.Text + "5";
    }

private void button6_Click(object sender, EventArgs e)
    {
      txtDisplay.Text = txtDisplay.Text + "6";
    }

private void button7_Click(object sender, EventArgs e)
    {
      txtDisplay.Text = txtDisplay.Text + "7";
    }

private void button8_Click(object sender, EventArgs e)
    {
      txtDisplay.Text = txtDisplay.Text + "8";
    }
```

```
private void button9_Click(object sender, EventArgs e)
   {
     txtDisplay.Text = txtDisplay.Text + "9";
   }

private void button10_Click(object sender, EventArgs e)
   {
     txtDisplay.Text = txtDisplay.Text + "0";
   }

private void button18_Click(object sender, EventArgs e)
   { // დისპლეის გაწმენდა
     txtDisplay.Clear();
     total1=0.0;            // ცვლადების განულება
     total2=0.0;
   }

private void button15_Click(object sender, EventArgs e)
   {
       txtDisplay.Text = txtDisplay.Text + ".";
    }

private void button11_Click(object sender, EventArgs e)
  { // +
   total1 = total1 + double.Parse(txtDisplay.Text);
   label2.Text = txtDisplay.Text;
   label3.Text = "+";
   txtDisplay.Clear();
  }

private void button16_Click(object sender, EventArgs e)
  { // =
   label4.Text = txtDisplay.Text;
```

```
    total2 = total1 + double.Parse(txtDisplay.Text);
    txtDisplay.Text = total2.ToString();
    total1 = 0;
   }

private void button17_Click(object sender, EventArgs e)
   {
     Close();
   }
 }
}
```

დავალება: გააფართოეთ პროგრამის ფუნქციონალობა, ანუ დაწერეთ პროგრამის კოდი „ +, - , * , / , %“ ოპერაციებისთის.

3.9. ტერნარული განშტოების ოპერაცია „ ? : „

პირობითი განშტოების ოპერაცია

„ ? : „

C# ენაში ერთადერთია, რომელიც სამ გამოსახულებას აერთიანებს:

y=(გამოსახულება_პირობა) ? (გამოსახულება_1) : (გამოსახულება_2);

თუ პირობა არის true, მაშინ y = <გამოსახულება_1>,
თუ false, მაშინ y=<გამოსახულება_2>.

3.10. კითხვები და სავარჯიშოები:

3.1. რას წარმოდგენს კონტეინერული ელემენტები ? ჩამოთვალეთ მათი სახეები.

3.2. ააგეთ პანელი ფორმის ცენტრში, მოათავსეთ მასზედ სხვა ელემენტები, მაგალითად, textBox და label და ამოძრავეთ ოთხივე მიმართულებით. ეკრანის ნაპირებთან მისვლისას გაჩერდეს (არ დატოვოს ეკრანი).

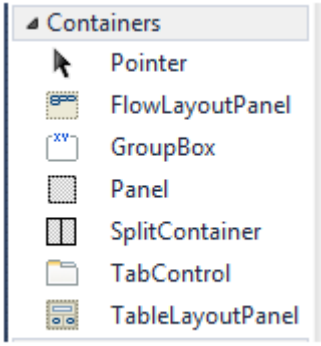
3.3. რას წარმოადგენს Timer ელემენტი და როგორ მუშაობს იგი ?

3.4. რას წარმოადგენს NumericUpDown ელემენტი და როგორ მუშაობს იგი ?

3.5. ააგეთ პროგრამა კალკულატორის დიზაინით და ფუნქციებით. შეასრულეთ არითმეტიკული ოპერაციები ნამდვილი ტიპის რიცხვებზე.

თავი 4. კონტეინერული და მართვის ვიზუალური ელემენტები

წინა თავში განხილულ იყო განშტოების პროცესის პროგრამული რეალიზების საშუალებები (if…else, switch). ამჯერად განვიხილავთ C# ენის განშტოების ვიზუალურ ელემენტებს (CheckBox, RadioButton) და მათი გამოყენების შესაძლებლობებს კონტეინერულ ვიზუალურ ელემენტებთან ერთად (Panel, FlowLayoutPanel, TableLayoutPanel, GroupBox) ნახ.4.1.



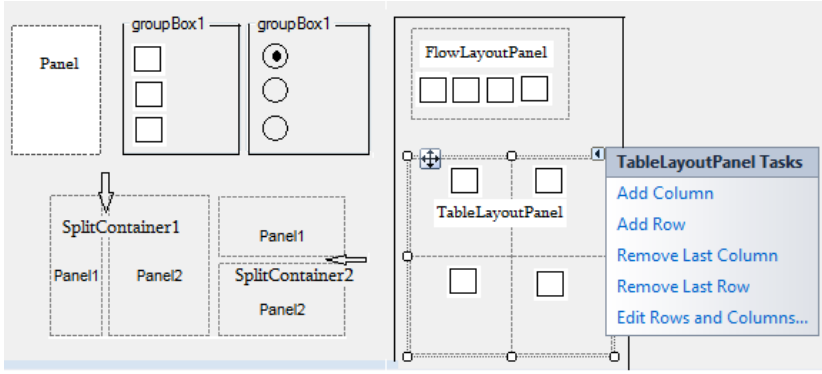
ნახ.4.1

☐ / ☒- CheckBox - საკონტროლო უჯრა ან ველია, რომელიც ცარიელი (გამორთულია) ან მონიშნულია (ჩართულია). რამდენიმე CheckBox-ის შემთხვევაში შეიძლება ჩართული იყოს 0, 1 ან ყველა.

 - RadioButton - გადამრთველი ღილაკი ან ველია, რომლის ჩართვისას მასში თავსდება მრგვალი მარკერი. რამდენიმე RadioButton-ის შემთხვევაში მხოლოდ ერთია აქტიური, რომელიც მონიშნულია (2 ან მეტი არ შეიძლება).

Panel (განხილულია წინა თავში), FlowLayoutPanel, TableLayoutPanel - კონტეინერული კლასის ელემენტებია, რომლებზეც თავსდება მართვის ვიზუალური ელემენტები, მაგალითად, რამდენიმე CheckBox, RadioButton ან სხვ. (ნახ.4.2).

GroupBox - არის აგრეთვე Container კლასის ჯგუფური საკონტროლო უჯრა, შემოსაზღვრული ჩარჩოთი, რომელშიც შეიძლება მოთავსდეს რამდენიმე CheckBox, RadioButton ან სხვა ვიზუალური ელემენტი. ის აერთიანებს ერთგვაროვან მონაცემებს, რომელთა ტექსტური იდენტიფიკაცია ხდება მისი ჩარჩოს ზედა-მარცხენა კუთხეში (ნახ.4.2).



ნახ.4.2

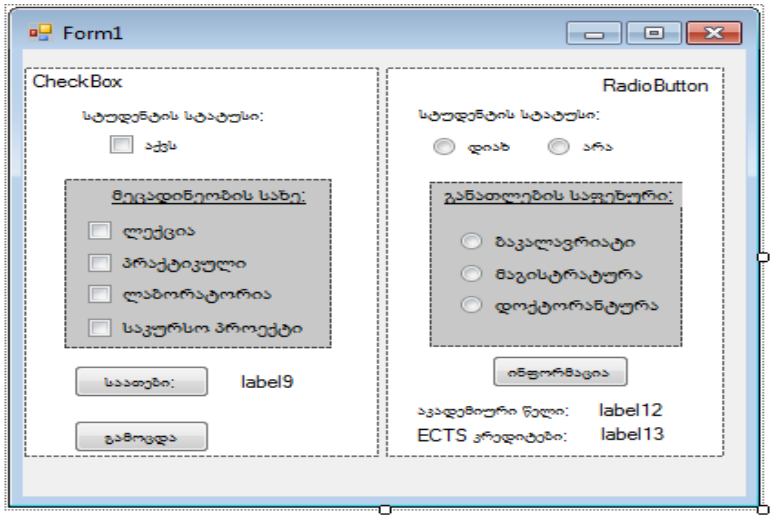
ნახაზიდან ჩანს კონტეინერულ ელემენტებს შორის განსხვავებაც. მაგალითად, SplitContainer ელემენტი, ინსტრუმენტების პანელიდან ფორმაზე გადმოტანისას, ქმნის ორ Panel-ს (ვერტიკალურს ან ჰორიზონტალურს), რომელთა ზომების შეცვლა მაუსით ხორციელდება. პანელის სპეციალური სახეა FlowLayoutPanel და TableLayoutPanel კონტეინერები. პირველი მათგანი მართვის ელემენტებს ალაგებს ერთ სტრიქონში (მაგ., ნახაზზე რამდენიმე checkBox). როცა ივსება ფორმის პანელის სიგანე, მომდევნო ელემენტი გადმოდის ახალ, მეორე სტრიქონზე. TableLayoutPanel-ის შემთხვევაში სისტემა მართვის ელემენტებს განალაგებს ცხრილში (ავტომატურად პანელი იყოფა 4 ნაწილად). როგორც ნახაზიდან ჩანს აქ შესაძლებელია ახალი სტრიქონების და სვეტების დამატება, ან არსებულის წაშლა.

მომდევნო პარაგრაფებში დეტალურად შევეხებით აღნიშნულ ვიზუალურ ელემენტებს მათი პრაქტიკულ ამოცანებში გამოყენების თვალსაზრისით.

4.1. მართვის ელემენტები: ჩამრთველი CheckBox, გადამრთველი RadioButton და თვისება checked

ამოცანა_4.1: საჭიროა ავაგოთ ვინდოუს-ფორმა, რომელსაც აქვს 4.3 ნახაზზე ნაჩვენები სახე. გამოიყენეთ ვერტიკალური SplitContainr ორი პანელით. მარცხენაზე მოათავსეთ პანელი 4

checkBox-ით (მეცადინეობის სახე), 2 ღილაკი (საათები - ითვლის მეცადინეობის ჯამურ საათებს; გამოცდა - იძახებს Form2 ფორმას ახალი ქვეამოცანისათვის).

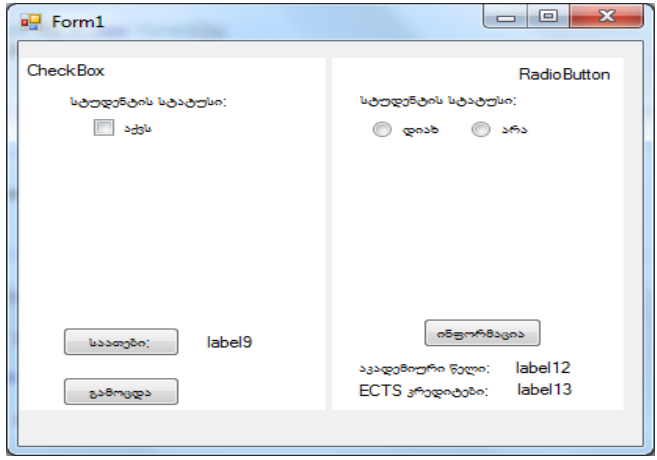


ნახ.4.3

მარცხენა პანელის ზედა ნაწილში ჩანს checkBox – „აქვს“, რომელიც ახორციელებს სტუდენტის სტატუსის განსაზღვრას, ანუ თუ ეს checkBox გამორთულია, მაშინ სუბიექტი არაა სტუდენტი და მისთვის პანელი ”მეცადინეობის სახე“ გამორთულია (ნახ.4.4). თუ checkBox ჩართულია, მაშინ სუბიექტი სტუდენტია და შესაძლებელია პანელზე „მეცადინეობის სახე“ მაუსის დახმარებით ჩაირთოს 1,2 ან ყველა checkBox. ბოლოს საათების ღილაკის დახმარებით label9-ში გამოჩნდება სემესტრში ამ კონკრეტული საგნის საათების ჯამური რიცხვი. თუ რომელიმე checkBox-ს გამოვრთავთ, მაშინ ჯამური რიცხვი შემცირდება შესაბამისად.

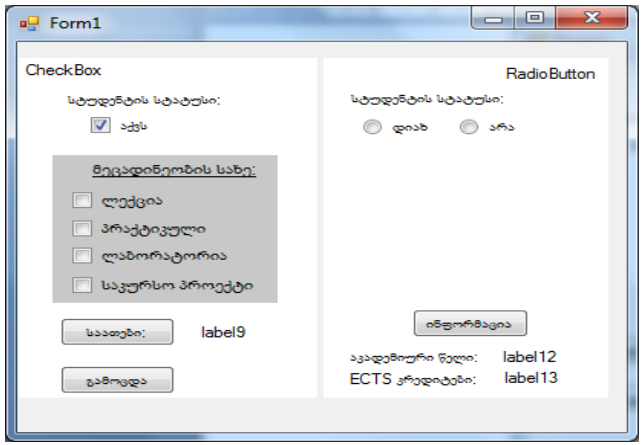
მარჯვენა პანელზე ნაჩვენებია radioButton გადამრთველის მაგალითი. თუ სტუდენტის სტატუსის „დიახ“ ბუტონში არაა მარკერი, მაშინ პანელი „განათლების საფეხური“ სამი radioButton-ით გამორთულია (ნახ.4.4). თუ არის მარკერი, მაშინ ეს პანელი ხილვადია და შეიძლება ერთ-ერთი ბუტონის არჩევა. შემდეგ

„ინფორმაცია“ - ღილაკის ამოქმედებით label12 და label13 უჯრებში გამოჩნდება შესაბამის მონაცემთა მნიშვნელობები: სასწავლო წლებისა და ECTS-კრედიტების საერთო რაოდენობა. radioButton-ების გადართით შესაძლებელია სხვა ინფორმაციის მიღებაც.

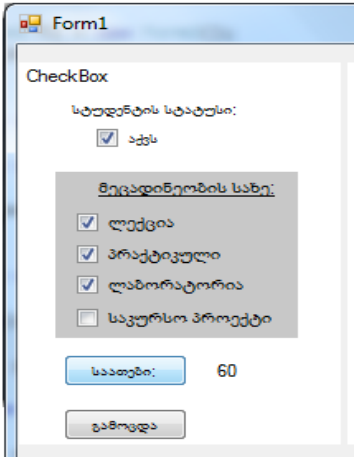


ნახ.4.4

მარცხენა პანელზე ჩავრთოთ სტუდენტის სტატუსის checkBox. 4.5 ნახაზზე გამოჩნდება შედეგი:



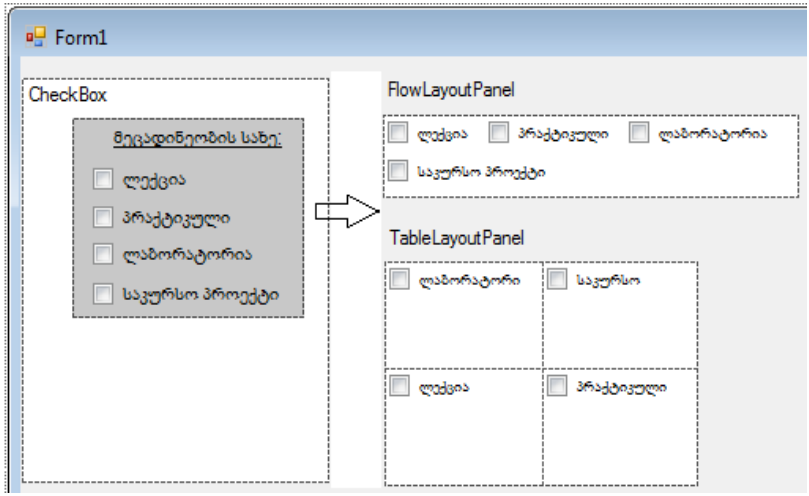
ნახ.4.5



პანელზე „მეცადინეობის სახე“ ჩავრთოთ checkBox-ები „ლექცია“, „პრაქტიკული“ და „ლაბორატორია“. შემდეგ ღილაკით „საათები“ label9-ში გამოჩნდება რიცხვი (ნახ.4.6).

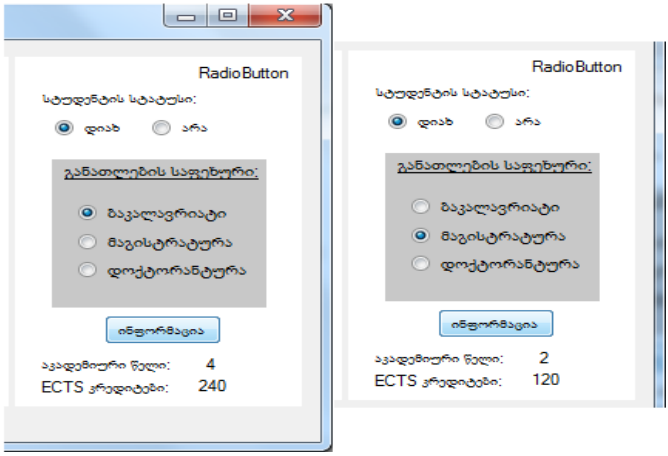
ნახ.4.6

4.7 ნახაზზე ნაჩვენებია checkBox-ების განლაგების ვარიანტები RowLayoutPanel და TableLayoutPanel კონტეინერული ელემენტების გამოყენებით.



ნახ.4.7

ამოცანა_4.2: იგივე პროცედურები ჩავატაროთ radioButton-ის მაგალითზე პანელის მარჯვენა ნაწილში. აქ (ნახ.4.4) ჩავსვათ მარკერი „დიახ“-ში და გამოჩენილ „განათლების საფეხურების“ პანელზე ავირჩიოთ რომელიმე ბუტონი. შემდეგ ღილაკით „ინფორმაცია“. მივიღებთ 4.8 ნახაზზე ნაჩვენებ სურათს.



ნახ.4.8

აღნიშნული ამოცანის რეალიზაციის კოდი მოცემულია 4.1 ლისტინგზე.

```
// ლისტინგი_4.1 --- CheckBox, RadioButton, Checked,
// CheckedChanged, SplitContainer ----
using System;
using System.Drawing;
using System.Windows.Forms;
namespace WinFormChekRadio
{
  public partial class Form1 : Form
   {
    public Form1() { InitializeComponent(); }
private void button2_Click(object sender, EventArgs e)
   {
     Form2 f2 = new Form2();
     f2.Show();
   }
```

```
private void splitContainer1_Panel1_Paint(object
                          sender, PaintEventArgs e) { }
private void button1_Click(object sender, EventArgs e)
 { Close(); }
// მოვლენა CheckedChanged - ცვლის ელემენტების
// მდგომარეობას ---
private void checkBox1_CheckedChanged(object sender,
                                              EventArgs e)
{   // checkBox-ის ჩართვით გამოჩნდება პანელი
 if (checkBox1.Checked) // checkBox-ის checked თვისება
                        // მდგომარეობის საკონტროლოდ
   {
    panel2.Visible = true;
   }
  else    // checkBox-ის გამორთვით დაიმალება პანელი
   {
    panel2.Visible = false;
   }
   // საათების ანგარიში
private void button4_Click(object sender, EventArgs e)
  {
   int s = 0;
   if (checkBox2.Checked)
      s += 15;
   if (checkBox3.Checked)
      s += 15;
   if (checkBox5.Checked)
      s += 15;
   if (checkBox4.Checked)
      s += 30;
   label9.Text = s.ToString();
  }
// რადიო-ბუტონის საილუსტრაციო კოდი
private void radioButton1_CheckedChanged(object sender,
                                              EventArgs e)
{
 if (radioButton1.Checked)  // radioButton-ში მარკერის
                            //  ჩასმით გამოჩნდება პანელი
  {
   panel1.Visible = true;
  }
```

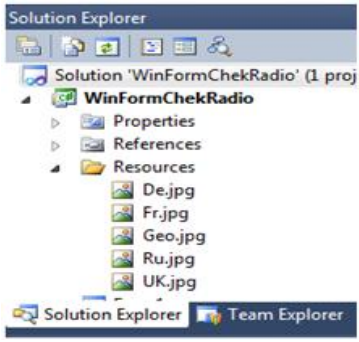
```
 else //radioButton-დან მარკერის ამოშლით დაიმალება პანელი
  {
   panel1.Visible = false;
  }
 }
  // ღილაკი „ინფორმაცია“ გამოაქვს შედეგი ----
private void button3_Click(object sender, EventArgs e)
{
  if (radioButton2.Checked)  // ჩადგმული if…else if…else
    {
     label12.Text = "  4";
     label13.Text = "240";
    }
   else if (radioButton3.Checked)
         {
           label12.Text = "  2";
           label13.Text = "120";
           }
          else
           {
            label12.Text = "  3";
            label13.Text = "180";
           }
         }
// ერთი რადიო-ბუტონიდან მეორეზე გადასვლისას სუფთავდება
// ძველი შედეგის მონაცემები
private void radioButton2_CheckedChanged(object sender,
                                         EventArgs e)
 { label12.Text = "   ";
   label13.Text = "   ";
 }
private void radioButton3_CheckedChanged(object sender,
                                         EventArgs e)
 { label12.Text = "   ";
   label13.Text = "   ";
 }
private void radioButton5_CheckedChanged(object sender,
                                         EventArgs e)
 { label12.Text = "   ";
   label13.Text = "   ";
  }
}}
```

4.2. საკონტროლო კონტეინერი GroupBox

GroupBox კონტეინერი მსგავსია ჩვენ მიერ ადრე განხილული Panel ელემენტისა, რომელზეც თავსდებოდა სხვა ვიზუალური ელემენტები (ნახ.4.3). აქვე „გამოცდა“ ღილაკის საშუალებით გავხსნათ Form2 ფანჯარა და GroupBox, ChekBox, RadioButton ელემენტების გამოყენებით გადავწყვიტოთ პროგრამული კოდის (პროექტის) აგების ამოცანა.

ამოცანა_4.3: ავაგოთ კოდი „საგამოცდო საგნების შერჩევა“. ამასთანავე გათვალისწინებულ უნდა იქნას რამდენიმე ენაზე მუშაობის შესაძლებლობა. საწყისი ფორმა მოცემულია 4.9-ა ნახაზზე.

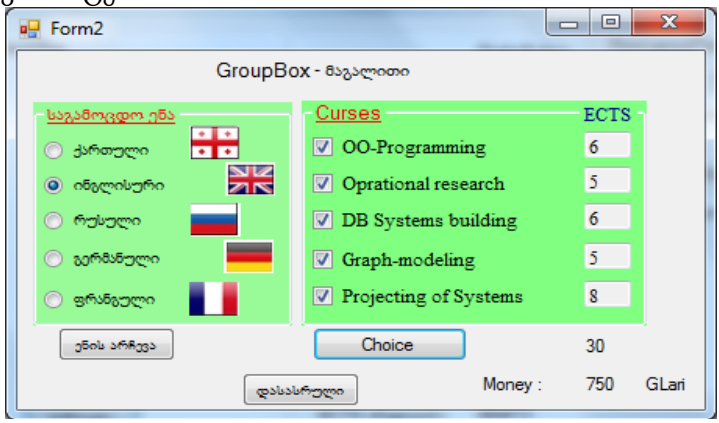
ნახ.4.9-ა



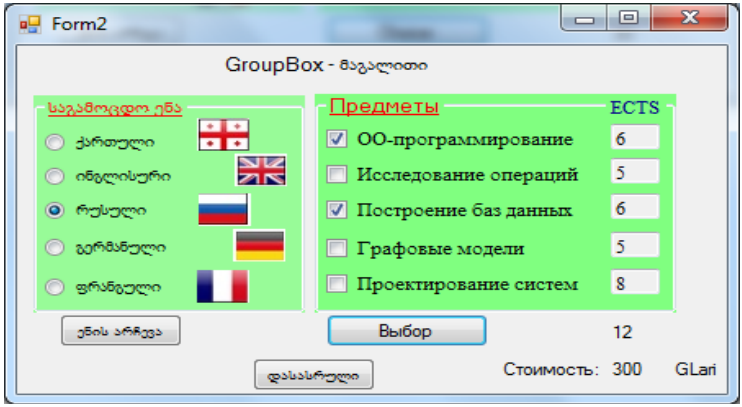
ნახ.4.9-ბ

ნახაზზე დროშებისთვის (გრაფიკული ელემენტი) გამოი-ყენება PictureBox მართვის ელემენტები, რომლის Resources ფაილებში (იხ. Solution_Explorer ნახ.4.9-ბ) ჩასმულია წინასწარ მომზადებული შესაბამისი .jpg-ფაილები. ელემენტები GroupBox-ის შიგნით განთავსებულია ამ ჯგუფის დასახელების შესაბამისად.

ნახაზზე ნაჩვენებია ორი ჯგუფური ფანჯარა: „საგამოცდო ენა“ და „საგამოცდო საგნები“. საჭირო ენის შესაბამისი რადიო-ბუტონის არჩევის შემდეგ ღილაკით „ენის არჩევა“ გადავალთ საგამოცდო საგნების არჩევაზე, გავააქტიურებთ ჩვენთვის საჭირო CheckBox-ებს. თუ ენა შეიცვალა არა-ქართულით, მაშინ საგამოცდო საგნების GroupBox-ის და მისი თანმხლები სხვა კომპონენტების წარწერებიც შეიცვლება არჩეული ენის შესაბამისად. მაგალითად, 4.10 და 4.11 ნახაზებზე ნაჩვენებია ინგლისური და რუსული ენების ვარიანტები.



ნახ.4.10



ნახ.4.11

label3-ში გამოიტანება სტუდენტის მიერ არჩეული საგნების ჯამური ECTS-კრედიტების რაოდენობა, label4-ში კი შესაბამისი გადასახდელი თანხის ოდენობა. შესაძლებელია საგნების მოკლება (checkBox-ის გამორთვით) ან ახლის დამატება (checkBox-ის ჩართვით). label3 და label4 უჯრებში მომენტალურად მოხდება გადაანგარიშება.

აღწერილი სისტემის შესაბამისი ელემენტების და ფუნქციონალობის პროგრამული კოდი მოცემულია 4.2 ლისტინგში.

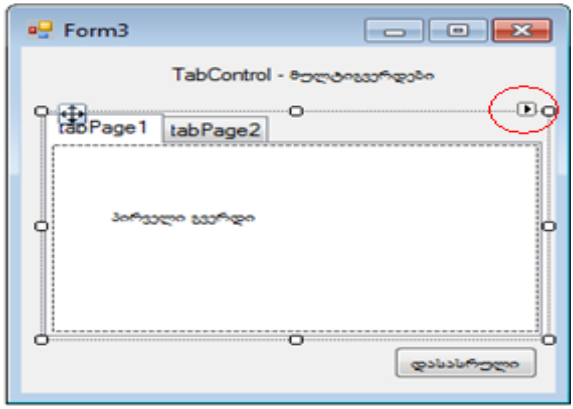
```
// ლისტინგი_4.2 --- GroupBox, CheckBox, RadioButton, ImageBox ---
using System;
using System.Drawing;
using System.Windows.Forms;
namespace WinFormChekRadio
{
  public partial class Form2 : Form
   {
    public Form2() { InitializeComponent(); }
    private void checkBox1_CheckedChanged(object sender,
                                          EventArgs e){ }
private void button1_Click(object sender, EventArgs e)
        // ქართული ენის არჩევა
 {
  if (radioButton1.Checked)
    {
     groupBox2.Text = "საგამოცდო საგნები";
     button2.Text = "აირჩიეთ";
     checkBox1.Text="ოო-პროგრამირება";
     checkBox2.Text="ოპერაციათა კვლევა";
     checkBox3.Text="მონაცემთა ბაზების აგება";
     checkBox4.Text="გრაფული მოდელები";
     checkBox5.Text="სისტემების დაპროექტება";
     label5.Text = "თანხა :";
     label6.Text = "ლარი";
    }
  else if (radioButton2.Checked) //ინგლისური ენის არჩევა
     {
       groupBox2.Text = "Curses";
       button2.Text = "Choice";
```

```
        checkBox1.Text="OO-Programming";
        checkBox2.Text="Oprational research";
        checkBox3.Text="DB Systems building";
        checkBox4.Text="Graph-modeling";
        checkBox5.Text="Projecting of Systems";
        label5.Text = "Money :";
        label6.Text = "GLari";
       }
    else if(radioButton5.Checked)  // რუსული ენის არჩევა
       {
        groupBox2.Text = "Предметы";
        button2.Text = "Выбор";
        checkBox1.Text="ОО-программирование";
        checkBox2.Text="Исследование операций";
        checkBox3.Text="Построение баз данных";
        checkBox4.Text="Графовые модели";
        checkBox5.Text="Проектирование систем";
        label5.Text = "Стоимость:";
        label6.Text = "GLari";
       }
    else       // გერმანული ენის არჩევა
      {
        groupBox2.Text = "Fachdisziplinen";
        button2.Text = "Wählen Sie bitte";
        checkBox1.Text = "OO-Programmierung";
        checkBox2.Text = "Operationsforschung";
        checkBox3.Text = "Entwicklung von Datenbanken";
        checkBox4.Text = "Graphische Modelen";
        checkBox5.Text = "Systemsentwurf";
        label5.Text = "Kosten :";
        label6.Text = "GLari";
       }
      }
private void button2_Click(object sender, EventArgs e)
  {     // კრედიტების ანგარიში
    int s = 0, Sum=0 ;
    if (checkBox1.Checked)
       s += 6;
    if (checkBox2.Checked)
      s += 5;
    if (checkBox3.Checked)
```

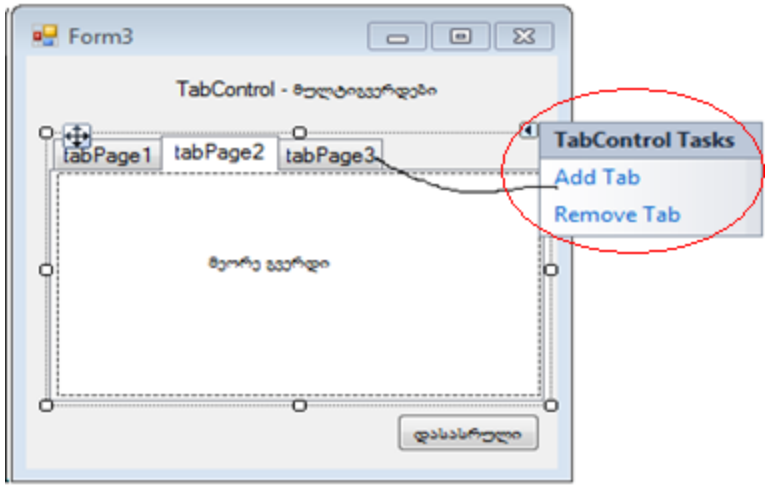
```
      s += 6;
    if (checkBox4.Checked)
      s += 5;
    if (checkBox5.Checked)
      s += 8;
    // კრედიტების შესაბამისი თანხის ანგარიში
    Sum = s * 25;
    label3.Text = s.ToString();
    label4.Text = Sum.ToString();
   }

private void button3_Click(object sender, EventArgs e)
  {
   Close();
  }
}}
```

4.3. კონტეინერი TabControl

კონტეინერული ელემენტი TabControl გამოიყენება ფორმაზე მრავალგვერდიანი დიალოგის ორგანიზებისთვის ურთიერთ-გადაფარვის პრინციპით (ნახ.4.12-ა,ბ). ახალი გვერდის დამატება ხდება Add Tab, ხოლო ძველის წაშლა Remove Tab სტრიქონებით კონტექსტურ მენიუდან, რომელიც გამოიტანება მარჯვენა კუთხეში პატარა ისარზე მაუსის მარჯვენა ღილაკით. თითოეული გვერდის სახელის ჩაწერა ხდება შესაბამისი გვერდის Properties-დან Text-ში.



ნახ.4.12-ა



ნახ.4.12-ბ

4.4. კითხვები და სავარჯიშოები

4.1. რა არის და როგორ გამოიყენება მართვის ვიზუალური ელემენტი ChackBox ?

4.2. რა არის და როგორ გამოიყენება მართვის ვიზუალური ელემენტი RadioButton ?

4.3. რა არის და როგორ გამოიყენება კონტეინერული მართვის ვიზუალური ელემენტი GroupBox ?

4.4. რა არის და როგორ გამოიყენება კონტეინერული მართვის ვიზუალური ელემენტი SplitContainer ?

4.5. რა არის და როგორ გამოიყენება კონტეინერული მართვის ვიზუალური ელემენტი FlowLayoutPanel ?

4.6. რა არის და როგორ გამოიყენება checked თვისება ?

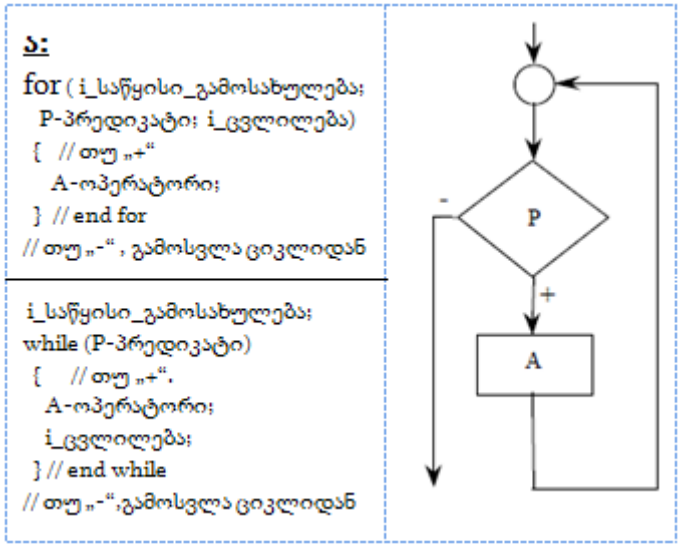
4.7. რა არის და როგორ გამოიყენება კონტეინერული მართვის ვიზუალური ელემენტი TabControl ?

4.8. აგეთ ფორმა და პროგრამა კონტეინერული ელემენტების გამოყენებით.

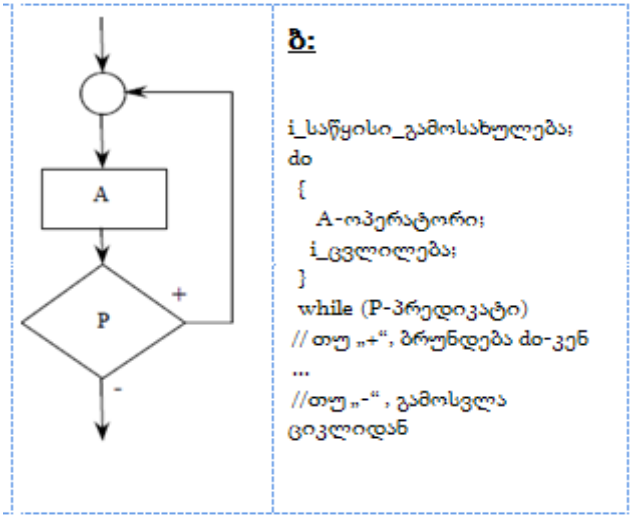
თავი 5. ციკლები, რეკურსია და შემთხვევით რიცხვთა გენერატორი

პროგრამული აპლიკაციების აგებისას, განშტოებისა და გადამრთველების მსგავსად, განსაკუთრებული ადგილი უჭირავს სტრუქტურული მართვის ისეთი საბაზო ელემენტებს, როგორიცაა ციკლები: for, while, do…while და foreach…in. წინამდებარე თავში განვიხილავთ ამ ელემენტებს და მათი საშუალებით პროგრამული პროექტების აგების საილუსტრაციო მაგალითებს.

ციკლის დანიშნულებაა ერთიდაიმავე პროცედურ(ებ)ის, ოპერატორების და მეთოდების რამდენჯერმე გამეორება საწყის მონაცემთა სხვადასხვა მნიშვნელობებისათვის, რომელთა ცვლა გარკვეული კანონზომიერებით ხდება. ციკლების გამოყენება პროგრამაში საგრძნობლად ამცირებს კოდის მოცულობას (სტრიქონების რაოდენობას) და პროგრამა ხდება ეფექტური. 5.1 ნახაზზე მოცემულია C# ენაში ციკლების კოდის აგების სინტაქსი და ზოგადი გრაფიკული ილუსტრაცია.



ნახ.5.1-ა: for და while ციკლები



ნახ.5.1-ბ: do…while ციკლი

foreach ციკლის ოპერატორი არ ყოფილა C, C++ და Java ენებში. მას აქვს შემდეგი სინტაქსი:

foreach (type identifier **in** expression**)**
{
// ოპერატორები;
}

სადაც expression არის კოლექციის (მასივის) დასახელება, რომლის შიგნითაც ხორციელდება ციკლური მოძრაობა. მაგალითად:

```
foreach (string Name in ListOfNames)
 {
  Console.WriteLine(“\t{0}”, Name);
 }
```

foreach...in ციკლის ოპერატორს მასივების შესწავლის დროს დავუბრუნდებით (თ.6). ახლა გადავიდეთ ციკლების და ვიზუალური ელემენტების გამოყენებით კოდის აგების პრაქტიკულ ამოცანებზე.

5.1. ციკლები და რეკურსიული ფუნქციები

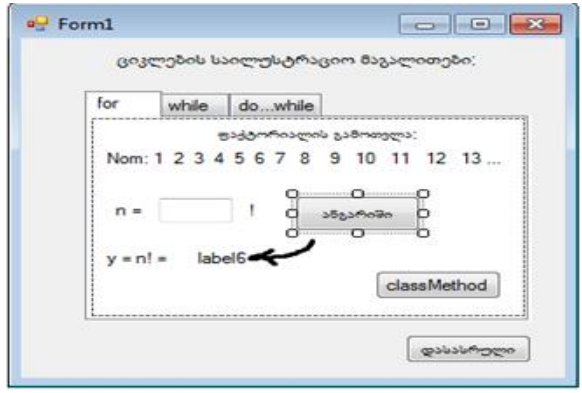
ამოცანა_5.1: ავაგოთ პროგრამული კოდი, რომელიც დაითვლის n! (ფაქტორიალის) მნიშვნელობას წინასწარ განსაზღვრული n-რიცხვისათვის. როგორც ცნობილია,

y = n! = 1 * 2 * 3 *…*n, სადაც n მოცემული მთელი რიცხვია.

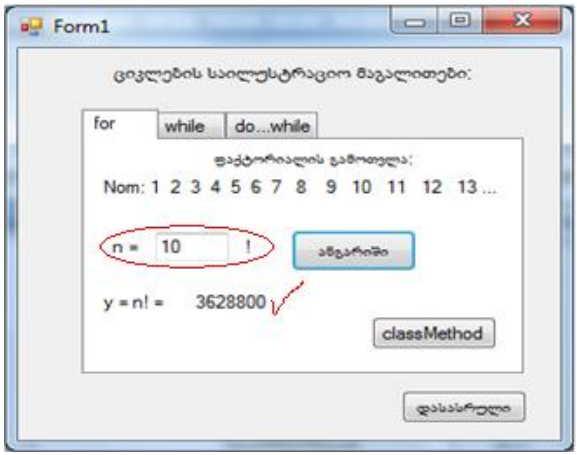
ფაქტორიალის გამოთვლის ალგორითმული გადაწყვეტა შესაძლებელია ციკლით და რეკურსიული პროცედურით. ამჯერად for-ციკლი გამოვიყენოთ:

```
// ლისტინგი_5.1 --- ფაქტორიალის ანგარიში for ციკლით ----
private void button2_Click(object sender, EventArgs e)
  {
   int i; double Fac;
   string ns = textBox1.Text;
   int n = Convert.ToInt32(ns);
   for (i = n, Fac = 1.0; i > 0; i--) // კლებადი ციკლი
      {
        Fac *= i;
      }
      label6.Text = Fac.ToString();
   }
```

5.1-ა ნახაზზე მოცემულია ფორმაზე TabControl კლასით შექმნილი მულტიგვერდების სურათი, რომლის გადამრთველებზეც მითითებულია ციკლის ტიპის (for, while, do…while) დასახელებები. შედეგი ასახულია 5.1-ბ ნახაზზე.



ნახ.5.1-ა



ნახ.5.1-ბ. შედეგი

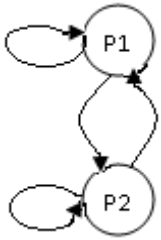
ახლა განვიხილოთ იგივე ამოცანის გადაწყვეტა რეკურსიის გამოყენებით. რეკურსია ნიშნავს ისეთ პროცედურას, როდესაც ერთ პროგრამას (ან მეთოდს) შუძლია გამოიძახოს მეორე პროგრამა (ან მეთოდი) და პირიქითაც, ან შეუძლია თავის თავის გამოძახება ციკლურად (ნახ.5.2).

ამოცანა_5.2: აიგოს ფაქტორიალის გამოთვლის პროგრამა. ამ შემთხვევაში კლასის შიგნით შეიქმნას სპეციალური მეთოდი, რომელიც გაიანგარიშებს მისთვის გარედან მიწოდებილი რიცხვის (n) ფაქტორიალს **რეკურსიის პრინციპის** საფუძველზე და დააბრუნებს შედეგს return-ოპერატორით.

ამოცანის გადაწყვეტის კოდის ფრაგმენტი მოცემულია 5_2 ლისტინგზე, ხოლო შედეგები გამოიტანება classMethod-ღილაკით (ნახ.5.1) Form2-ფანჯარაში (ნახ.5.3-ა). ბ-ნახაზზე ნაჩვენებია მაქსიმალური შესაძლო შედეგი.

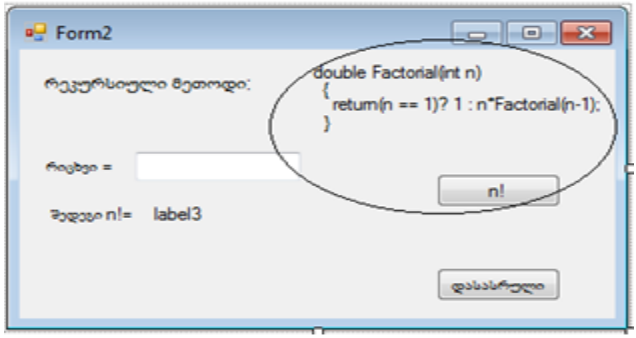
კომენტარი: `button1_Click` მოვლენით (ღილაკის ამოქმედება) გამოძახებულ იქნა მეთოდი (ფუნქცია) Factorial(int n) გადასაცემი n-არგიმენტით. მართვა გადაეცა double Factorial()-მეთოდს, რომლის return-ოპერატორში რეალიზებულია რეკურსია, ანუ Factorial(n-1) იძახებს „თავის-თავს“, მანამ, სანამ n არ

შემცირდეაბა 1-მდე. რეალურად, თუ n=5, ციკლში ხორციელდება: 5*4*3*2*1 და ეს ნამრავლი, ანუ ფაქტორიალის მნიშვნელობა უბრუნდება Fac-ცვლადს.

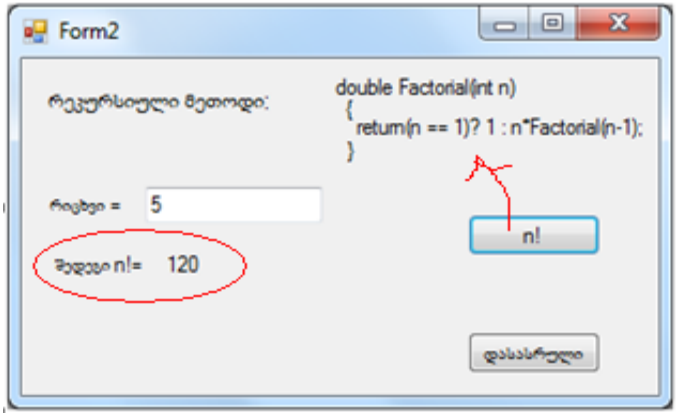


ნახ.5.2

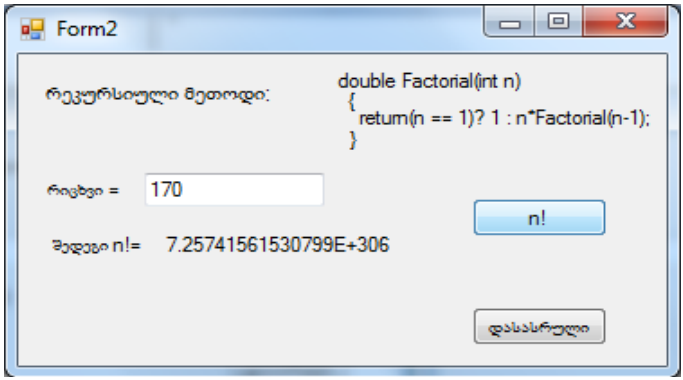
```
// ლისტინგი_5.2 -- რეკურსიული მეთოდი ---
using System;
using System.Windows.Forms;
namespace WinFormCycle
{
  public partial class Form2 : Form
   {
     public Form2()
         {InitializeComponent(); }
private void button1_Click(object
                  sender, EventArgs e)
 {
  double Fac;
  string ns = textBox1.Text;
  int n = Convert.ToInt32(ns);
  Fac=Factorial(n); // მეთოდის გამოძახება
  label3.Text = Fac.ToString();
 }
// რეკურსიული მეთოდი ფაქტორიალის
// საანგარიშოდ ---
double Factorial(int n)
 {  // რეკურსია !
  return(n == 1)? 1:n*Factorial(n-1);
 }
}
}
```



ნახ.5.3-ა1



ნახ.5.3-ა2. შედეგი



ნახ.5.3-ბ. n=170 მაქსიმალური შესაძლო რიცხვი

ამოცანა_5.3: ავაგოთ პროგრამული კოდი, რომელიც:

ა) გაითვლის ნებისმიერი მითითებული მთელი რიცხვისათვის **ფიბონაჩის** მწკრივში მის მომდევნო ”ფიბონაჩის რიცხვის” მნიშვნელობას (მაგალითად, რიცხვი=50, რომელია მის მარჯვნივ მდგარი უახლოესი ფიბონაჩის რიცხვი ?);

ბ) გაითვლის ფიბონაჩის რიცხვის მნიშვნელობას მე-n ელემენტისათვის ფიბონაჩის რიცხვთა მწკრივში. (მაგალითად, რას უდრის მწკრივში რიგით მე-15 ფიბონაჩის რიცხვი ?);

გ) გაითვლის ფიბონაჩის მწკრივის რიცხვთა ჯამს მითითებული მე-n ელემენტის ჩათვლით (მაგ., რას უდრის ფიბონაჩის მწკრივის 1-20 რიცხვების ჯამი ?).

როგორც ცნობილია, ფიბონაჩის რიცხვთა მწკრივი შემდეგი სახისაა:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, …

ანუ, რიცხვთა მწკრივის აგების წესი ასეთია: მომდევნო რიცხვი არის წინა ორი რიცხვის ჯამი.

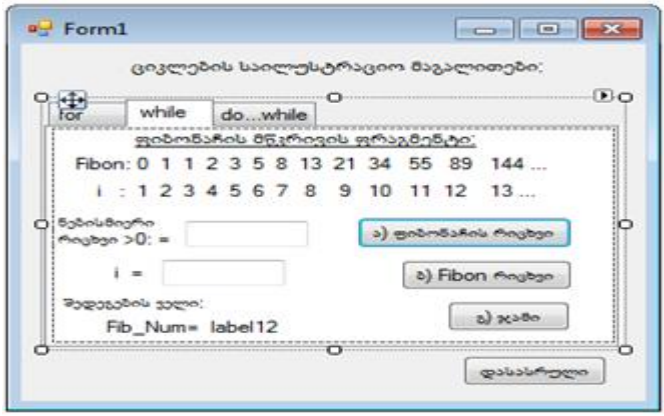
შენიშვნა: „ფიბონაჩის მწკრივი“ მათემატიკურ სამყაროში 1200 წელს შეიქმნა ლეონარდო პიზანსკის (ფსევდონიმი „ფიბონაჩი“) მიერ, თუმცა ასეთი მწკრივი ძველ ინდოეთშიც ყოფილა ცნობილი (იხ. მასალები ვიკიპედიაში). ფიბონაჩის მწკრივს მრავალი პრაქტიკული გამოყენება აქვს დღესაც სხვადასხვა დარგებში (ეს ცალკე თემაა). მისი ასეთი აქტუალურობის და მნიშვნელობის გამო ჩვენ განვიხილავთ ციკლური და რეკურსიული ოპერაციების პროგრამული კოდების მაგალითებს ფიბონაჩის რიცხვებისათვის.

5.4-ა ნახაზზე ნაჩვენებია მულტიგვერდი „while“, რომელზეც განლაგებულია ფიბონაჩის მწკრივის ფრაგმენტი (Fibon), ნატურალურ რიცხვთა მწკრივი (i), ჩვენი სამი ამოცანის სამი ბუტონი (ა,ბ,გ), საწყისი მნიშვნელობების შესატანი ორი ტექსტბოქსი („რიცხვი>0: =“ და „i=“) და შედეგის გამოსატანი ველი (Fib_Num=label12).

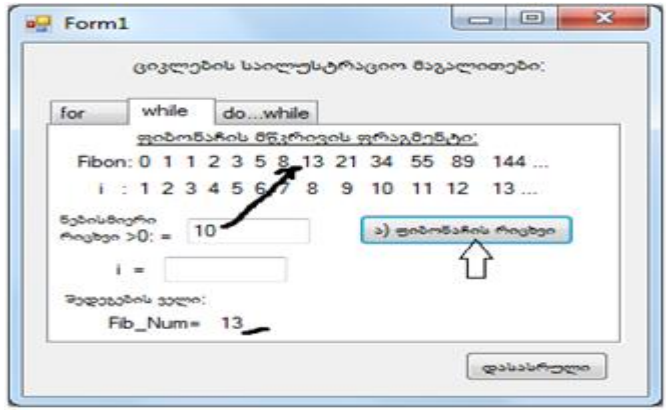
<u>2-ა) ამოცანის გადაწყვეტა:</u>

პროგრამის ამუშავების და „while” გვერდზე გადართვის შემდეგ საწყისი მონაცემის შეტანა უნდა განხორციელდეს ტექსტბოქსში (ნახ.5.4-ბ). როგორც კი მოხდება ამ ტექსტბოქსის მნიშვნელობის შეცვლა, მაშინვე ფორმაზე დაიმალება ბ და გ ბუტონები. ა-ბუტონი მზადაა დააფიქსიროს შედეგი. მაგალითად, რიცხვისთვის 10, ფიბონაჩის რიცხვის მნიშვნელობა იქნება 13. რიცხვისთვის 100, იქნება 144 და ა.შ.

while() - ციკლის გამოყენებით შექმნილი პროგრამის კოდი, რომელიც ამუშავდება ა-ბუტონის, როგორც მოვლენის ამოქმედებისას, მოცემულია 5_3 ლისტინგში:



ნახ.5.4-ა



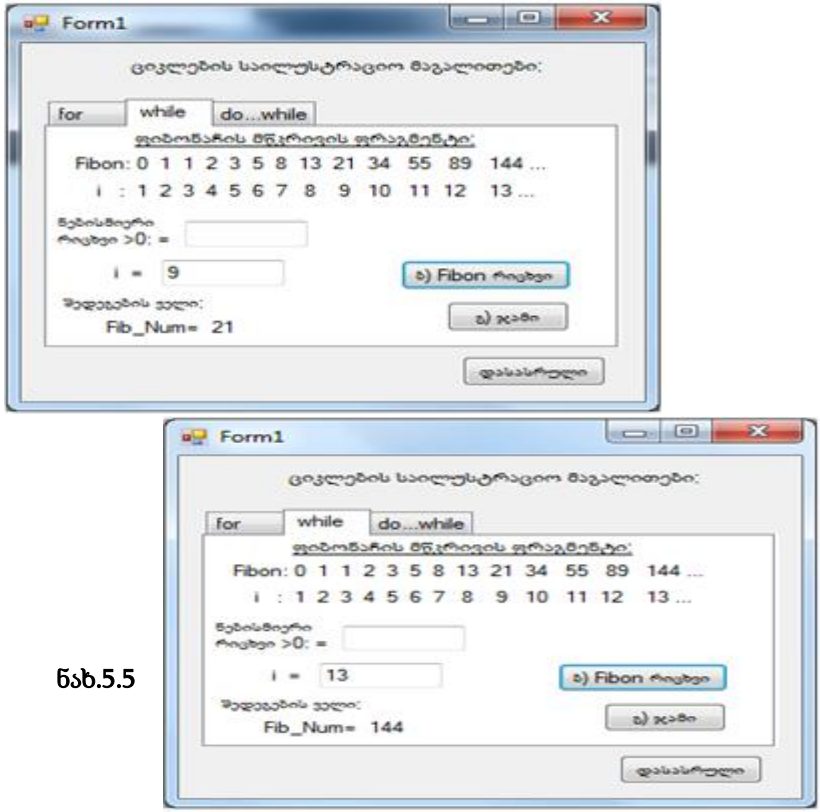
ნახ.5.4-ბ

```
// ლისტინგი_5.3 – while ციკლი + ფიბონაჩი-2-ა ----
private void button3_Click(object sender, EventArgs e)
 { // ა-ღილაკი
   int Previous, Next;  // ფიბონაჩის მწკრივის „წინა“ და
                        // „მომდევნო“ რიცხვები
   Previous = -1;
   Next = 1;
   string ns = textBox2.Text; // საწყისი მონაცემის შეტანა
   int n = Convert.ToInt32(ns);   // ტიპის გარდაქმნა
  int fib = 1;
```

```
while (fib <= n)  // ციკლი !!!
 {
   fib = Previous + Next;
   Previous = Next;
   Next = fib;
  label12.Text = " " + fib.ToString(); // შედეგის გამოტანა
   }
}
```

2-ბ) ამოცანის გადაწყვეტა:

თუ საწყისი მნიშვნელობა შეიტანება მეორე ტექსტბოქსში (ნახ.5.5), მაშინ იცვლება გვერდის მდგომარეობა, ანუ გააქტიურდება შესაბამისი ამოცანების ბ და გ ღილაკები და დაიმალება არასაჭირო ა-ღილაკი.



ნახ.5.5

5.4_ლისტინგში ნაჩვენებია ბ-ღილაკის კოდი, რომელიც ფიბონაჩის რიცხვის გასათვლელად გამოიყენებს რეკურსიულ მეთოდს (Fibonacci(int n) - ფუნქციას),

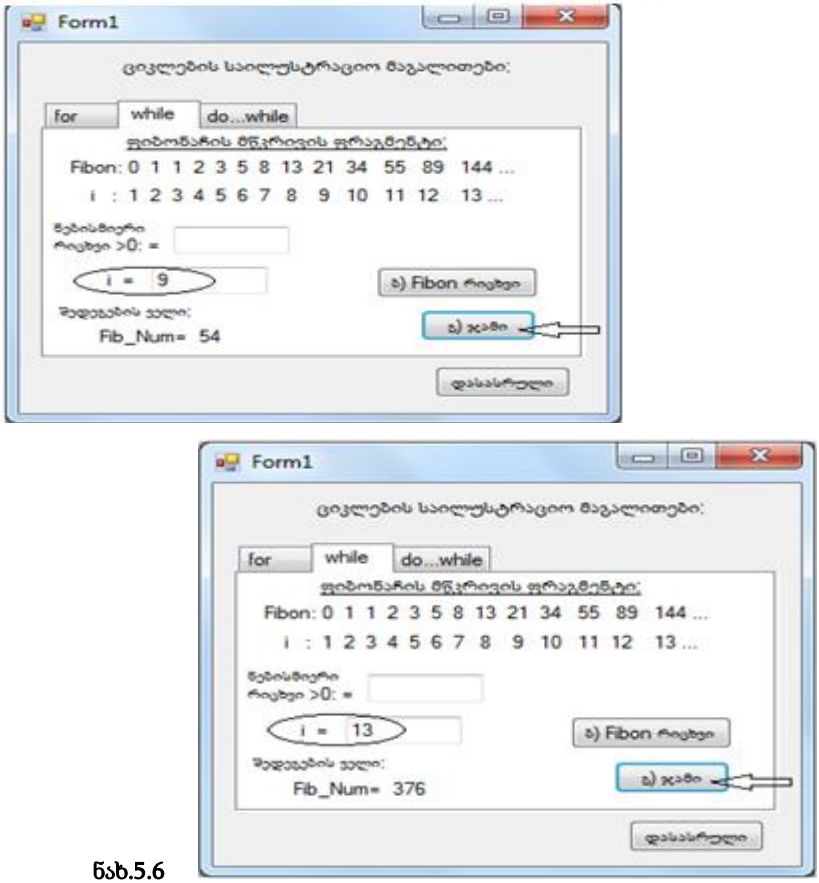
```
// ლისტინგი_5.4-- ფიბონაჩის რიცხვის გამოთვლა რეკურსით --
private void button5_Click(object sender, EventArgs e)
 {
  int Fibon;
  string ns = textBox3.Text;
  int n = Convert.ToInt32(ns);
  Fibon=Fibonacci(n-1);
  label12.Text = Fibon.ToString();
 }
int Fibonacci(int n)  // რეკურსიული მეთოდი
 {
   return n > 1 ? Fibonacci(n-1) + Fibonacci(n-2) : n;
 }
```

პროგრამაში გათვალისწინებული სამი ბუტონის გამოჩენა/არგამოჩენა ფორმაზე რეალიზებულია მათი შესაბამისი ტექსტბოქსის ელემენტების ხილვადობის პარამეტრების მართვით, რაც 5_5 ლისტინგშია აღწერილი.

```
 // ლისტინგი_5.5 ---  Visible თვისების მართვა -----
  private void textBox2_TextChanged(object sender,
                                EventArgs e) // ერთი textBox
  {
    if (textBox2.Text != "")
      {
       button3.Visible = true;  // ჩაირთო button3
       button5.Visible = false; // გამოირთო button5
       button6.Visible = false; // გამოირთო button6
      }
  }
private void textBox3_TextChanged(object sender,
                               EventArgs e) // მეორე textBox
 { if (textBox3.Text != "")
    {
     button5.Visible = true;  // გამოირთო  button5
     button6.Visible = true;  // გამოირთო button6
     button3.Visible = false; // გამოირთო button3
    }
 }
```

2-გ) ამოცანის გადაწყვეტა:

5.6 ნახაზზე ნაჩვენებია გ-ღილაკის ამოქმედებით არჩეული რიცხვებისთვის მწკრივის ჯამის გაანგარიშების შედეგები.



ნახ.5.6

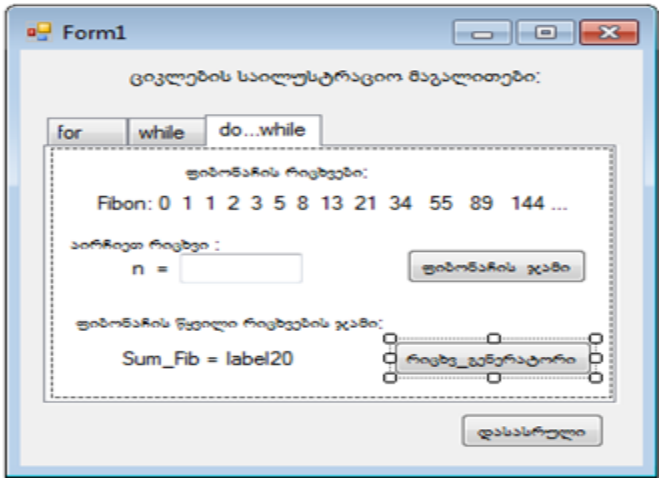
```
// ლისტინგი_5.6-გ ფიბონაჩის მწკრივის რიცხვთა ჯამის გამოთვლა ---
private void button6_Click(object sender, EventArgs e)
  {
   string ns = textBox3.Text;
   int n = Convert.ToInt32(ns);
   int zinaFib = 0, momdevnoFib = 1;
```

```
int SumFib = 0;
int i = 1;
while (i <=n)
    {
      int temp = momdevnoFib;
       momdevnoFib += zinaFib;
       SumFib += zinaFib;
       zinaFib = temp;
       i++;
     }
    label12.Text = SumFib.ToString();
}
```

5.2. ციკლები და შემთხვევით რიცხვთა გენერატორი

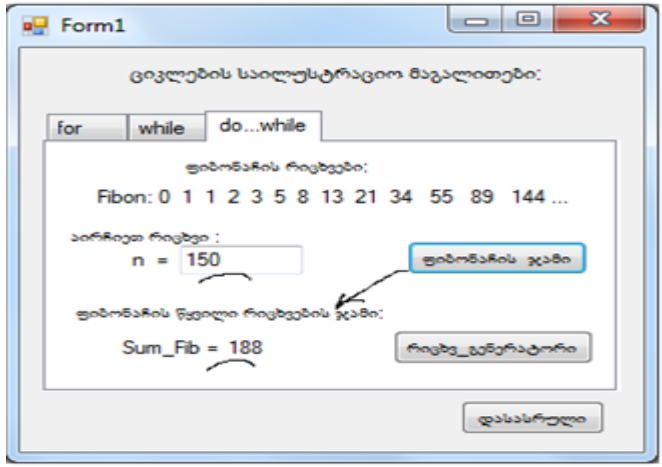
ახლა გადავიდეთ do…while ციკლის განხილვაზე (ნახ.5.7).

ამოცანა_5.4: საჭიროა პროგრამული კოდის აგება, რომელიც იანგარიშებს do…while ციკლის გამოყენებით ფიბონაჩის რიცხვთა მწკრივის ლუწი რიცხვების ჯამს. მწკრივის სიგრძე, ანუ მისი ელემენტების რაოდენობა უნდა მიეთითოს ტექსტბოქსში შეტანილი n-რიცხვით, მაგალითად, 150. შედეგი გამოტანილ უნდა იქნეს label20 -ში.



ნახ.5.7-ა

ნახ.5.7-ბ: შედეგი



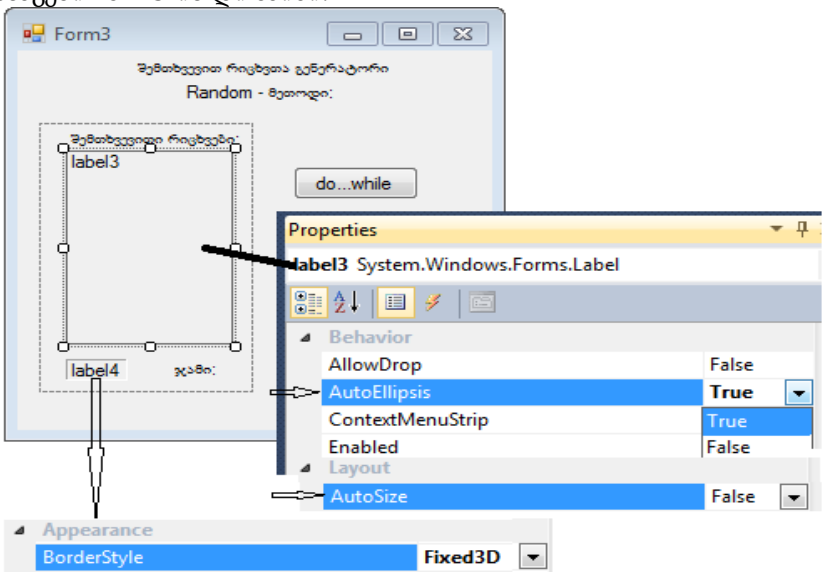
ფიბონაჩის მწკრივის რიცხვთა ჯამის განსაზღვრის სარეალიზაციო კოდი მოცემულია 5.7_ლისტინგში.

```
// ლისტინგი_5.7 – do…whle ციკლი ფიბონაჩისთვის ----
private void button7_Click(object sender, EventArgs e)
  {
   string ns = textBox4.Text;
   uint n = Convert.ToUInt32(ns);
   uint SumFib = 0;
   uint momdevnoFib = 1;
   uint zinaFib = 0;

 do
  {
   uint tmp = momdevnoFib;
   momdevnoFib = momdevnoFib + zinaFib;
   zinaFib = tmp;
   if (momdevnoFib % 2 == 0 && momdevnoFib < n)
                                        // წყვილობაზე შემოწმება
    SumFib += momdevnoFib;
  } while (momdevnoFib < n)
                    ;   // ცარიელი ოპერატორი
```

```
        label20.Text = SumFib.ToString();
}
```

5.7 ნახაზზე ჩანს ღილაკი „შემთხვევით_რიცხვთა გენერატორი“, რომელითაც იხსნება Form3 ფანჯარა და შესაბამისი ინტერფეისით.

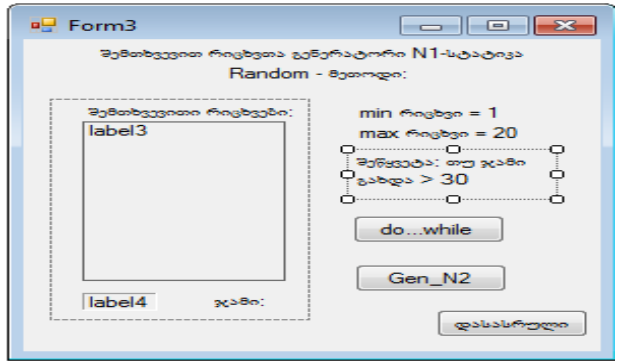
ამოცანა_5.5: საჭიროა პროგრამის კოდის აგება, რომელიც შემთხვევით რიცხვთა გენერატორის საშუალებით (Random-კლასის ობიექტით) ციკლურად ამოიღებს რიცხვებს მითითებულ დიაპაზონში (Next(min,max-1)) და მოათავსებს label3-ში. უნდა განისაზღვროს ამ რიცხვთა ჯამი label4-ში. 5.8 ნახაზზე ნაჩვენებია ასაგები Form3-ის დიზაინი.



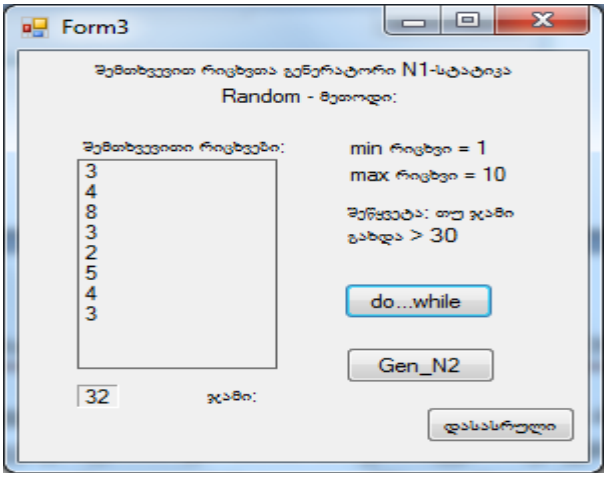
ნახ.5.8

label3-ის თვისებების შეცვლით AutoEllipsis=True და AutoSize+False შესაძლებელია უჯრის გაფართოება და მასზე რამდენიმე სტრიქონის ან ტესტის გამოტანა. label4 გავხადეთ 3-განზომილებიანი: BorderStyle=Fix3D.

```
// ლისტინგი_5.8 --do…while ციკლი შემთხვევით რიცხვთა
                        // გენერატორისთვის (სტატიკური კოდი)--
using System;
using System.Drawing;
using System.Windows.Forms;
namespace WinFormCycle
{
 public partial class Form3 : Form
 {
  Random r = new Random(); //Random-კლასი -შრ-გენერატორი
  public Form3() { InitializeComponent(); }
 private void button1_Click(object sender, EventArgs e)
    { // გენერატორის ამოქმედება
      int summe = 0, z;
      label3.Text = "";
      do
       {
        z=r.Next(1, 10); // Next მეთოდია Random-კლასის
        summe += z;
        label3.Text += z.ToString() + "\n";
       }
        while (summe < 30)
                ;
     label4.Text = summe.ToString();
   }
 private void button2_Click(object sender, EventArgs e)
     { Close(); }
  }
}
```



ნახ.5.8-ა



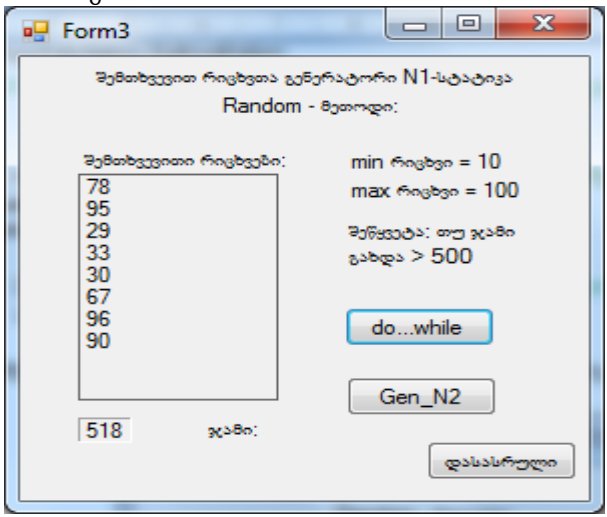
ნახ.5.8-ბ: შედეგი

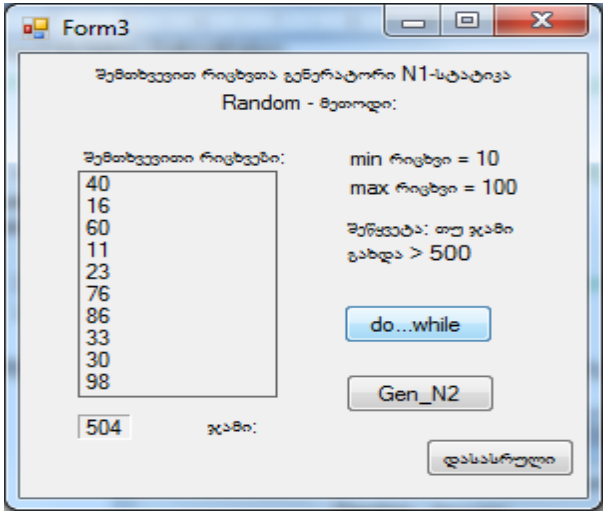
5.8 ნახაზზე ნაჩვენებია შედეგები. “do…while” ღილაკის ყოველ ახალ ამოქმედებაზე მუშაობას იწყებს შემთხვევით რიცხვთა გენერატორი და label3-ში გამოაქვს რიცხვები 1-10 დიაპაზონიდან, როგორც ეს Next(1,10) არის მითითებული. while(summe<30) ოპერატორი ამოწმებს ამ რიცხვების ჯამი ხომ არაა მეტი 30-ზე. თუ არაა მეტი, მაშინ სრულდება „ ; “ - ცარიელი ოპერატორი, რომელიც აბრუნებს პროცესს do { } - ციკლში. თუ ჯამი მეტია 30-ზე, მაშინ ციკლი მთავრდება და label4-ში იწერება semme-შედეგი.

შემთხვევით რიცხვთა დიაპაზონის შესაცვლელად პროგრამაში შევცვალოთ Next, როგორც ეს 5.9_ლისტინგის ფრაგმენტშია ნაჩვენები:

```
// ლისტინგი_5.9---შრ-გენერატორისთვის min და max
                                        // საზღვრების შეცვლა ----
do
{
  z = r.Next(10, 100); // მინ-მაქს რიცხვები შეიცვალა
  summe += z;
  label3.Text += z.ToString() + "\n";
}
 while (summe < 500)    // საკონტროლო ჯამი შეიცვალა
                ;
```

ახალი ექსპერიმენტისთვის შედეგები გამოტანილია 5.9 ნახაზზე.

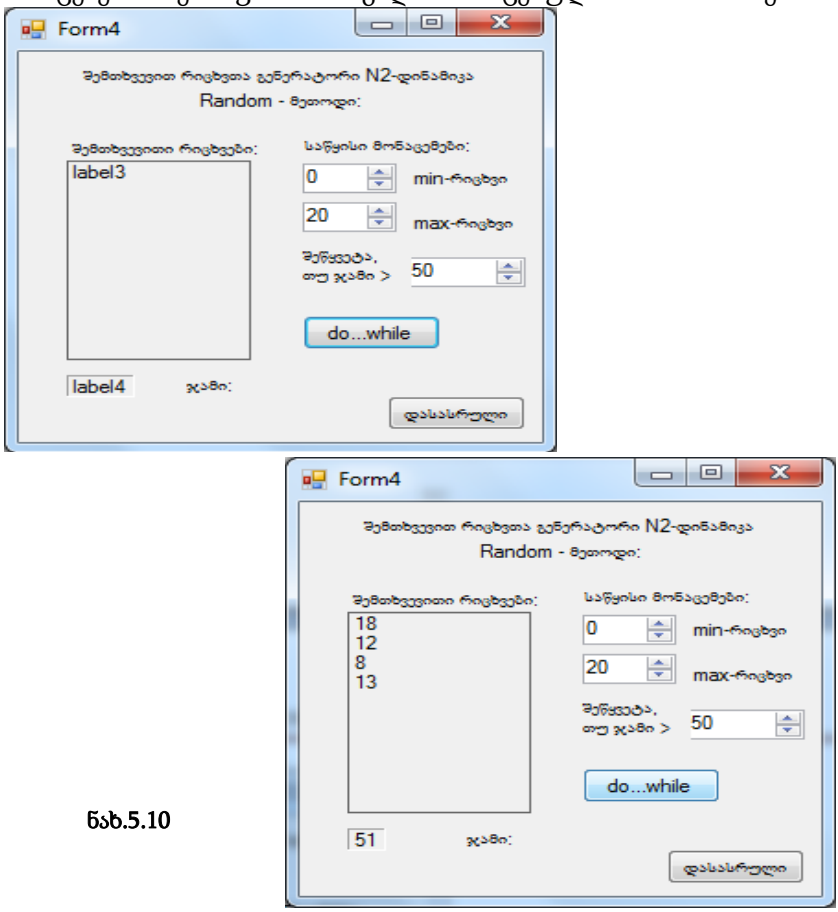




ნახ.5.9

საწყისი მონაცემების ასეთი ცვლილება, რომელიც ზემოთ ჩავატარეთ, სტატიკურია ანუ ხისტია და მოითხოვს პროგრამის კოდის შესაბამისი ცვლადების, აგრეთვე Properties-ში ფორმის ზოგიერთი მონაცემის ხელით შეცვლას, რაც მთლიანობაში არასასურველია, განსაკუთრებით პროგრამის მომხმარებლისთვის, რომელსაც შეიძლება ჰქონდეს დახურული კოდი.

პროგრამის დინამიკური ვარიანტი (ნახ.5.9 ღილაკი Gen_N2) მისცემს მომხმარებელს უფლებას თვითონ განსაზღვროს საწყისი მონაცემები. ასეთი ფორმის მაგალითი მოცემულია 5.10 ნახაზზე.



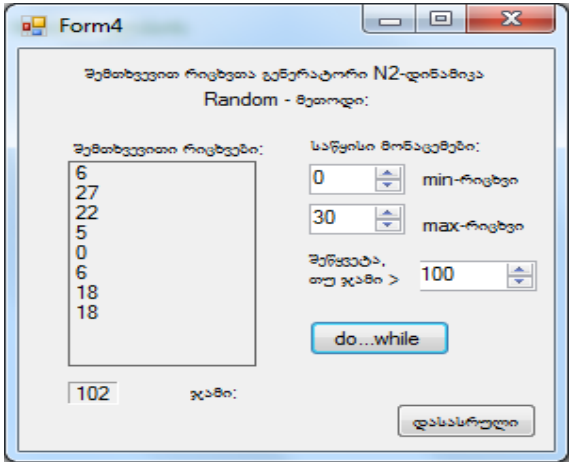
ნახ.5.10

გამოყენებულია სამი მართვის ვიზუალური ელემენტი numericUpDown: 1-მინიმალური რიცხვის, 2-მაქსიმალური რიცხვის და 3-ჯამის ანგარიშის ციკლის შეწყვეტისათვის. პროგრამულ კოდში შეტანილ იქნება ცვლილებები ერთხელ და შემდგომში ის აღარ მოითხოვს ცვლილებებს. პროგრამა იმუშავებს დინამიკურად საწყისი მონაცემებისთვის. კოდის ტექსტი მოცემულია 5.10_ ლისტინგში.

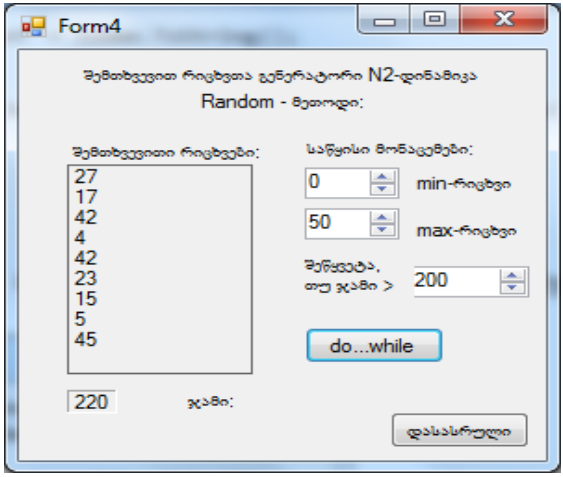
```
// ლისტინგი_5.10 --- do…while ციკლი შემთხ-რიცხვთა
                        // გენერატორისთვის (დინამიკური კოდი)--
using System;
using System.Drawing;
using System.Windows.Forms;
namespace WinFormCycle
{
  public partial class Form4 : Form
  {
   Random r = new Random();   // Random კლასი - შრგ
   public Form4() { InitializeComponent();  }
private void button1_Click(object sender, EventArgs e)
  { // ტიპის გარდაქმნა  cast(int) -ით
    int minNum = (int)numericUpDown1.Value;
    int maxNum = (int)numericUpDown2.Value;
    int sumControlNum = (int)numericUpDown3.Value;
    int summe = 0, z;
    label3.Text = "";
    do
     {  // Next არის მეთოდი Random-კლასის
       z=r.Next(minNum, maxNum);
       summe += z;
       label3.Text += z.ToString() + "\n";
     }
         while (summe < sumControlNum)
                    ;
       label4.Text = summe.ToString();
     }
```

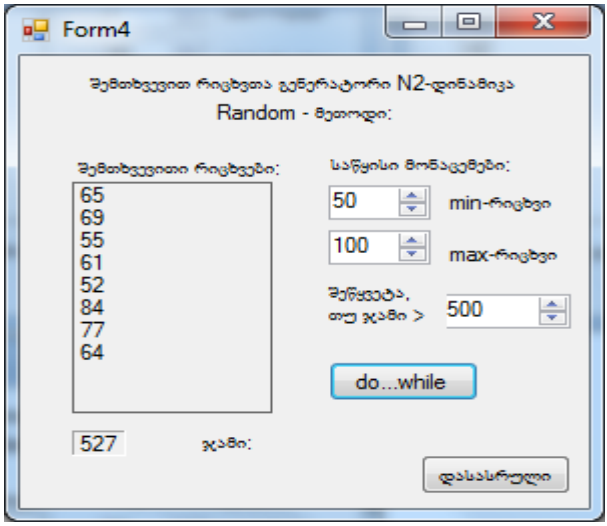
```
 private void button2_Click(object sender, EventArgs e)
   {  Close(); }
  }
}
```

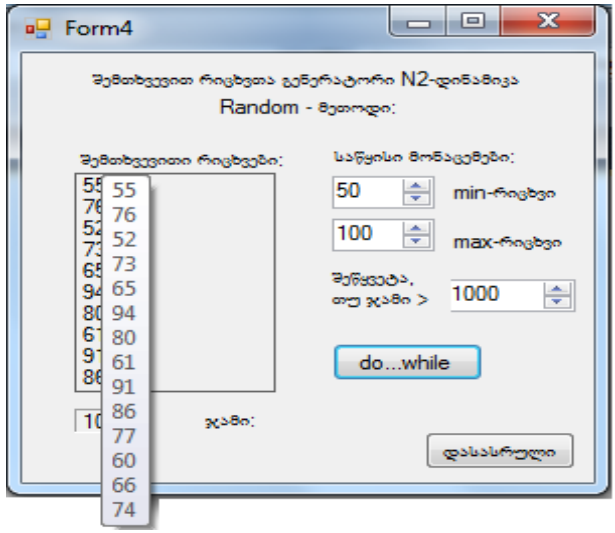
5.11 ნახაზზე მოცემულია პროგრამის მუშაობის შედეგები სხვადასხვა საწყისი მონაცემებისთვის:



ნახ.5.11-ა




Form4
შემთხვევით რიცხვთა გენერატორი N2-დინამიკა
Random - მეთოდი:
შემთხვევითი რიცხვები:
65
69
55
61
52
84
77
64
საწყისი მონაცემები:
50
min-რიცხვი
100
max-რიცხვი
შეწყვეტა,
თუ ჯამი >
500
do...while
527
ჯამი:
დასასრული


Form4
შემთხვევით რიცხვთა გენერატორი N2-დინამიკა
Random - მეთოდი:
შემთხვევითი რიცხვები:
55
76
52
73
65
94
80
61
91
86
77
60
66
74
საწყისი მონაცემები:
50
min-რიცხვი
100
max-რიცხვი
შეწყვეტა,
თუ ჯამი >
1000
do...while
ჯამი:
დასასრული

ნახ.5.11-ბ

მეოთხე მაგალითზე შემთხვევით რიცხვთა გენერატორის რიცხვები ვერ ეტევა label-ის ჩარჩოში. ასეთ დროს მაუსის კურსორის მიტანით label-ზე გამოჩნდება ყველა რიცხვის სვეტი.

5.3. კითხვები და სავარჯიშოები:

5.1. ციკლის რომელი ოპერატორები გამოიყენება C# ენაში და როგორია მისი სინტაქსი ?

5.2. როდის და როგორ გამოიყენება foreach ოპერატორი ?

5.3. რა არის რეკურსიული ფუნქცია ? მოიტანეთ მაგალითი.

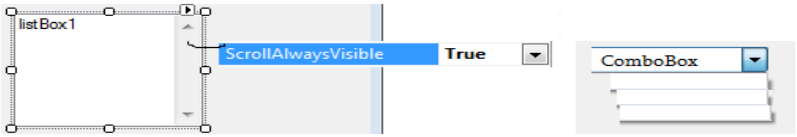
5.4. ააგეთ პროგრამული კოდი ნატურალურ რიცხვთა მწკრივის n-არული წევრის ფაქტორიალის გასათვლელად.

5.5. ააგეთ რეკურსიული ალგორითმი და პროგრამის კოდი ფიბონაჩის მწკრივისათვის.

5.6. როგორ მუშაობს შემთხვევით რიცხვთა გენერატორი და როგორ ხდება მისი პროგრამული რეალიზაცია ?

6. ციკლები და მართვის ვიზუალური ელემენტები: ListBox, CheckedListBox , ComboBox

ციკლების თემასთან, რომელიც ჩვენ წინა თავში განვიხილეთ, უშუალო კავშირშია მართვის ვიზუალური ელემენტები - ListBox და ComboBox, რომლებიც ბევრად ამარტივებს მომხმარებელთა ინტერფეისებს და მოქნილს ხდის მათ. წინამდებარე თავში განვიხილავთ ამ ელემენტებს და მათი საშუალებით პროგრამული პროექტების აგების საილუსტრაციო მაგალითებს. 6.1 ნახაზზე ნაჩვენებია ListBox და ComboBox ელემენტების მაგალითები.



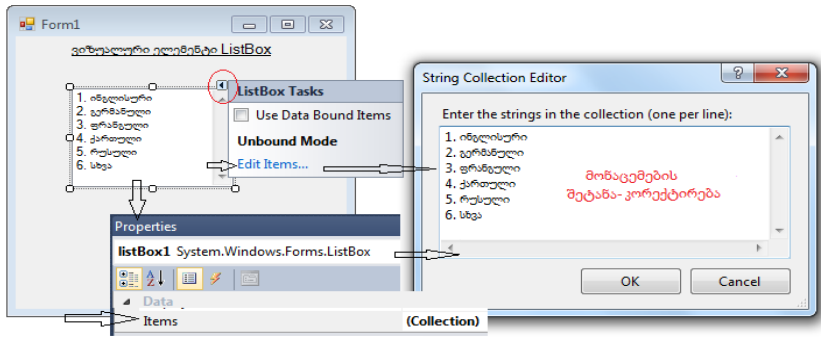
ნახ.6.1

ListBox ახორციელებს მისი ჩანაწერების სიის ასახვას ფორმაზე. მომხმარებელს შეუძლია ამ ჩანაწერების მრავალჯერადი ამორჩევა და გამოყენება. ასეთი ამოცანებისთვის ხშირად ციკლები გამოიყენება. ჩანაწერების სია შეიძლება იყოს დიდი მოცულობის, რომელიც ფანჯარაში ვერ თავსდება, ამიტომაც ListBox-ს გააჩნია ავტომატურად მომუშავე კომპონენტი - ScrollBar, რომელიც - Properties-იდან ყენდება true-მნიშვნელობით (ნახ.6.1). ComboBox არის ჩამოშლადი ველი, საიდანაც აირჩევა სასურველი მნიშვნელობა.

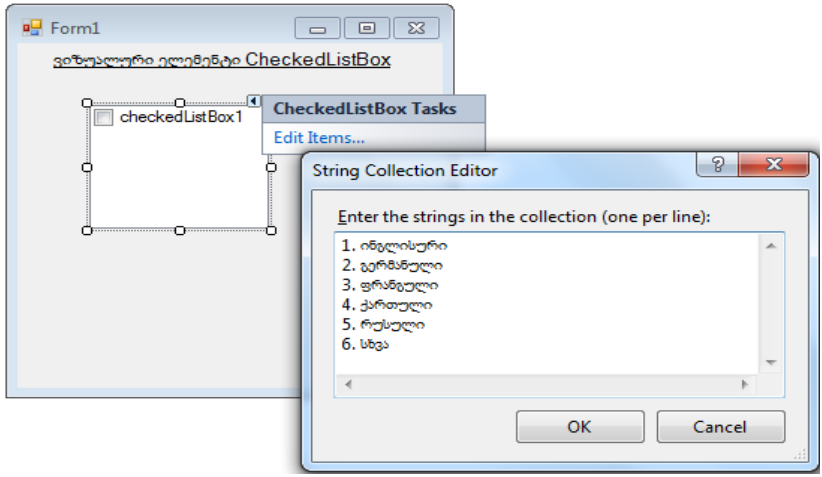
6.1. მართვის ვიზუალური ელემენტები ListBox და CheckedListBox

ListBox-ში ტექსტური სტრიქონების შეტანა ხდება Properties->Items თვისებიდან ან ლისტბოქსის ჩარჩოს ზედა-მარჯვენა კუთხეში ისარზე მაოსის დაწკაპუნებით გამოტანილ Edit Items არჩევით (ნახ.6.2). ორივე შემთხვევაში გამოიტანება String Collection Editor, რომელშიც ჩაიწერება მონაცემები.

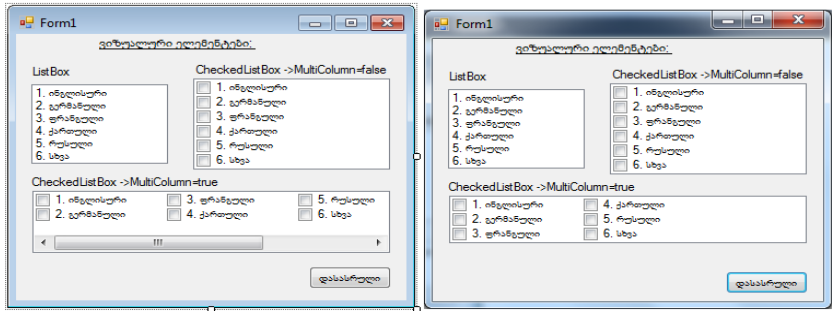
CheckListBox ელემენტი მსგავსია ListBox-ისა, ოღონდაც მის სტრიქონს ემატება წინ chekBox-ელემენტი (ნახ.6.3). სტრიქონების შეტანის შემდეგ Edit Items რედაქტორში მიიღება 6.4 ნახაზზე ნაჩვენები სურათი.



ნახ.6.2



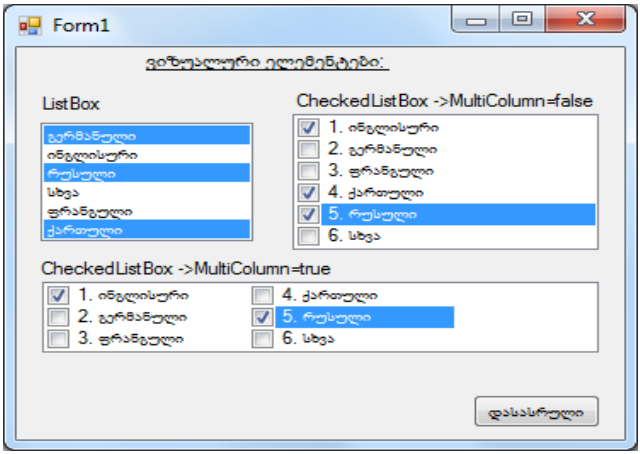
ნახ.6.3



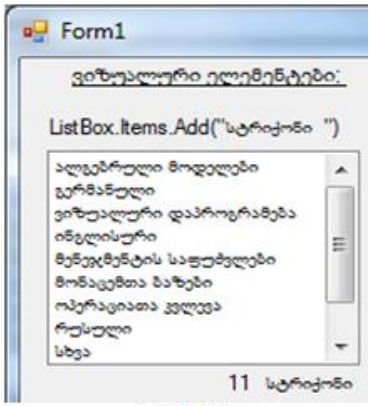
ნახ.6.4

ListBox-ის სტრიქონებს მოვაცილოთ ნომრები და მის Properties-ში თვისება დავაყენოთ: Sorted->true. სტრიქონები მოწესრიგდება ანბანის შესაბამისად (ნახ.6.5).

CheckedListBox-ში შესაძლებელია რამდენიმე სტრიქონის მონიშვნა. ListBox-ში კი საჭიროა ამ მიზნით Properties-ის SelectionMode->MultiExtended არჩევა. ამის შემდეგ შეიძლება Shift და Ctrl კლავიშების დახმარებით რამდენიმე ან ყველა სტრიქონის მონიშვნა (ნახ.6.5).



ნახ.6.5



ნახ.6.6

ახლა განვიხილოთ ტექსტ-ბოქსებთან პროგრამულად მუშაობის ზოგიერთი მეთოდი.

- Add() მეთოდი Item თვისებისთვის. პროგრამული კოდის ფრაგმენტი, რომელშიც ხდება სტრიქონების ჩამატება ListBox-ში (ნახაზი 6.6) მოცემულია 6_1 ლისტინგში. როგორც ვხედავთ, სტრიქონები ჩაწერილია Form1_Load(object ...) მეთოდში.

ფორმის ცარიელ ადგილას მაუსით დაკლიკვით გადავალთ კოდის ამ ფრაგმენტზე.

```
// ლისტინგი_6.1 – ListBox-ის Item-თვისების Add()-მეთოდი----
private void Form1_Load(object sender, EventArgs e)
 {
   int st;
   listBox1.Items.Add("მონაცემთა ბაზები");
   listBox1.Items.Add("ვიზუალური დაპროგრამება");
   listBox1.Items.Add("ოპერაციათა კვლევა");
   listBox1.Items.Add("ალგებრული მოდელები");
   listBox1.Items.Add("მენეჯმენტის საფუძვლები");

   st = listBox1.Items.Count; //სტრიქონების რაოდენობა
   label5.Text = st.ToString();
 }
```

- Items.Count - მეთოდით განისაზღვრება LisBox-ში სტრიქონების რაოდენობა. `label5`-ში გამოიტანება: ჩანაწერი „11 სტრიქონი“ (ნახ.6.6).

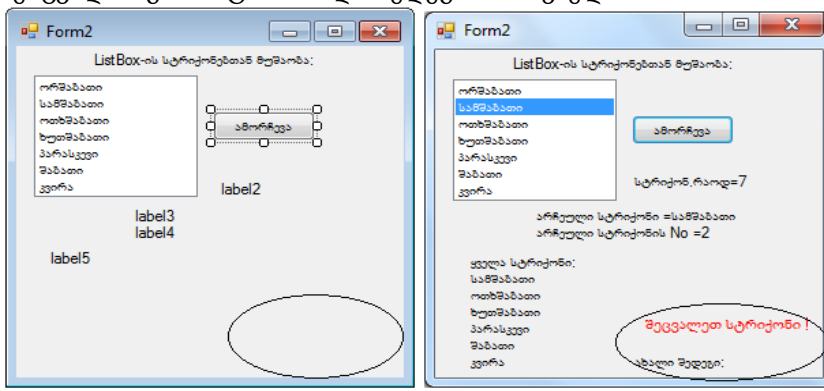
პროგრამულ კოდთან მუშაობისას სტრიქონების იდენტიფიკაციისათვის და ამოსარჩევად გამოიყენება SelectedItem / SelectedItems და SelectedIndex / SelectedIndices თვისებები.

6.2_ლისტინგში მოცემულია ფრაგმენტი ListBox-ის სტრიქონებთან სამუშაოდ. შედეგები „ამორჩევა“ ღილაკის ამოქმედების შემდეგ ასახულია 6.7 ნახაზზე.

```
//ლისტინგი_6.2 – SelectedItem, SelectedIndex, Items[index]--
private void button1_Click(object sender, EventArgs e)
  {
   int st;
   label2.Text = "სტრიქონ.რაოდ=" + listBox1.Items.Count;
   label3.Text = "არჩეული სტრიქონი =" +
                          listBox1.SelectedItem;
   label4.Text = "არჩეული სტრიქონის No =" +
                     (listBox1.SelectedIndex+1).ToString();
   label5.Text = "ყველა სტრიქონი:" + "\n";
   for (st = 1; st < listBox1.Items.Count; st++)
       label5.Text += listBox1.Items[st] + "\n";

   label6.Visible = true;  // ელემენტი გამოჩნდება ეკრანზე
```

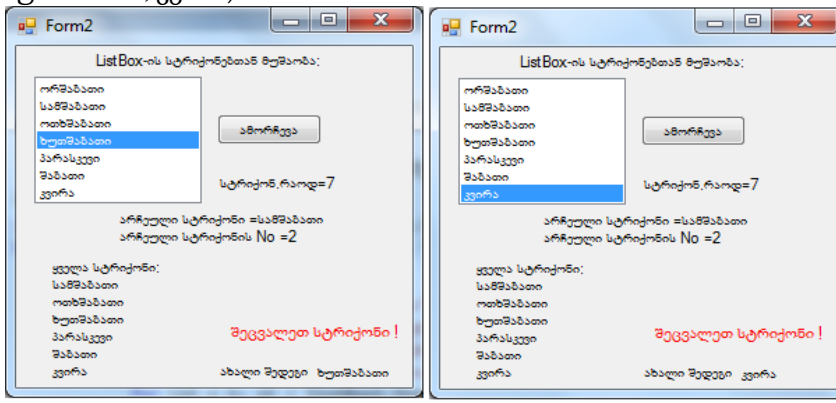
```
    label7.Visible = true;
    label8.Visible = false; // ელემენტი არ ჩანს ეკრანზე
}
```

ListBox-ში სტრიქონის შეცვლისას ავტომატურად უნდა შეიცვალოს გამოსატანი ახალი შედეგის მნიშვნელობა.



ნახ.6.7

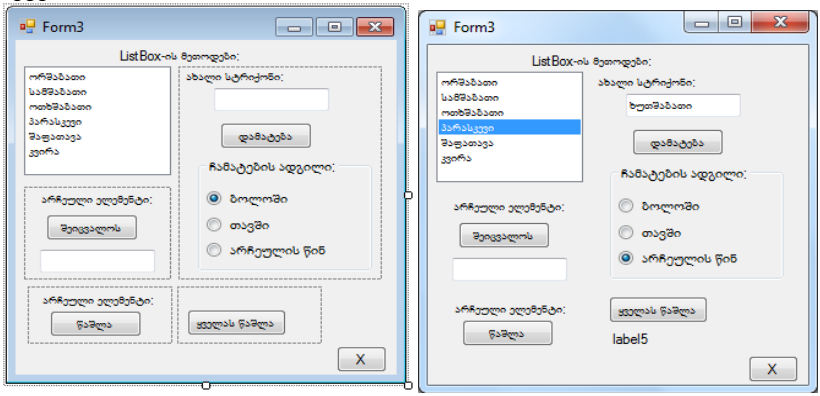
ის გააქტიურდება ღილაკით და გამოჩნდება ფორმაზე „ამორჩევა“. მოვლენა `listBox1_SelectedIndexChanged` განახორციელებს ავტომატურ ცვლილებას. მისი ლისტინგია_6.3, ხოლო შედეგები 6.8 ნახაზზეა ასახული (ახალი სტრიქონი: ხუთშაბათი, კვირა).



ნახ.6.8

```
//ლისტინგი_6.6 --- ListBox-ის სტრიქონის შეცვლის მოვლენის კოდი ------
private void listBox1_SelectedIndexChanged(object sender,
                                                    EventArgs e)
  {
    label8.Visible = true;
    label8.Text = " "+listBox1.SelectedItem;
  }
```

ამოცანა_6.1: დავაპროგრამოთ ListBox-ისთვის სამი მეთოდი: ახალი_სტრიქონის_შეტანის, არასაჭირო_სტრიქონის_ამოშლის და სტრიქონის_შეცვლის მიზნით. შესაძლებელია Insert() და RemoveAt() მეთოდების გამოყენება. შედეგების ასახვისათვის შევქმნათ Form3 (ნახ.6.9).



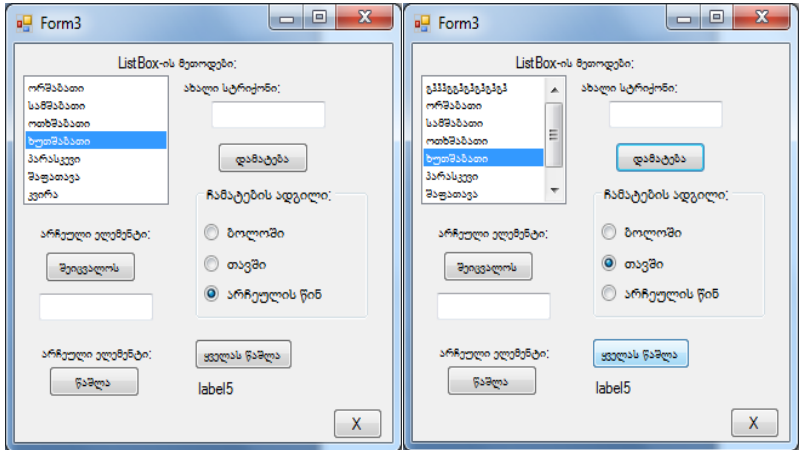
ნახ.6.9

- „დამატება“ ღილაკისთვის, რომელიც ახალ სტრიქონს ამატებს ListBox-ის თავში, ბოლოში ან მითითებული სტრიქონის წინ, კოდს ექნება 6_7 ლისტინგზე ნაჩვენები სახე.

```
//ლისტინგი_6.7 – ListBox-ის Insert() მეთოდი -------
private void button1_Click(object sender, EventArgs e)
 {
  if (textBox1.Text == "")
    return;
  if (radioButton2.Checked)
     listBox1.Items.Insert(0, textBox1.Text);
    else
     if (radioButton3.Checked && listBox1.SelectedIndex != -1)
       listBox1.Items.Insert(listBox1.SelectedIndex,
                                              textBox1.Text);
```

```
        else
         listBox1.Items.Add(textBox1.Text);

           textBox1.Text = "";
}
```

ახალი მონაცემის (მაგალითად, „**ხუთშაბათი**“) დამატების შემდეგ ლისტბოქსი მიიღებს 6.10 ნახაზზე მოცემულ სახეს. თავში დავამატეთ აგრეთვე „**გჰგჰგჰგჰგჰგჰგჰ**“ სტრიქონი.



ნახ.6.10

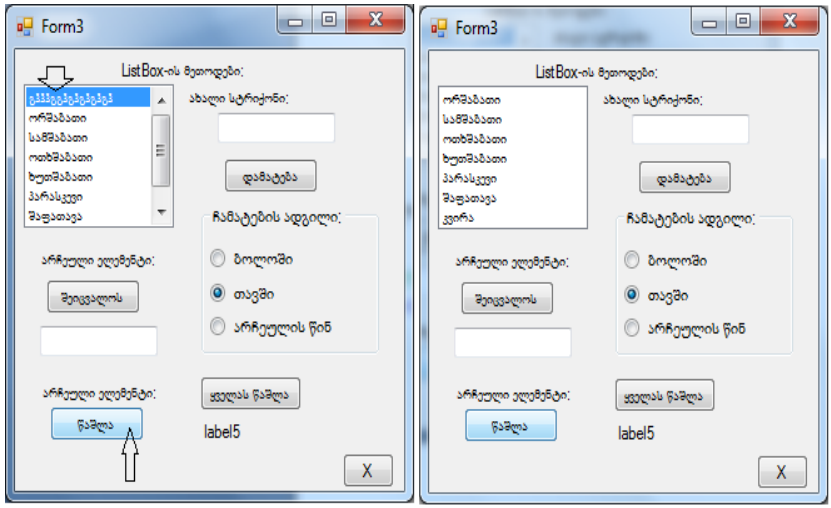
- „წაშლა“ ღილაკისთვის, რომელიც არჩეულ სტრიქონს ამოშლის ListBox-იდან. ListBox-ზე ავირჩიოთ პირველი სტრიქონი („გჰგჰგჰგჰგჰ“) და „წაშლა“-ღილაკი ავამოქმედოთ. შედეგად ლისტბოქსიდან გაქრება ეს სტრიქონი.

აქ იმუშავა `Items.RemoveAt(st)` მეთოდმა, რომელსაც `SelectedIndex`-ით მიეწოდა წასაშლელი სტრიქონის ნომერი. თუ არაა არჩეული, `SelectedIndex` აბრუნებს „-1“ მნიშვნელობას. შედეგი ასახულია 6.11 ნახაზზე.

„წაშლის “ კოდს აქვს 6.8_ ლისტინგზე ნაჩვენები სახე.

```
//ლისტინგი_6.8 – ListBox-ის RemoveAt() მეთოდი -------
private void button2_Click(object sender, EventArgs e)//წაშლა
  {
    int st = listBox1.SelectedIndex;
```

```
    if (st != -1)
      listBox1.Items.RemoveAt(st);
  }
```



ნახ.6.11

- „ყველას_წაშლა“ ღილაკი „სახიფათოა“, რადგან შეიძლება შემთხვევით მონაცემები დავკარგოთ. ეს ღილაკი უნდა შეიცავდეს მომხმარებლის დამატებით გაფრთხილებას. თუ მისგან მიიღებს „დასტურს“ ყველა სტრიქონის წაშლაზე, მხოლოდ მაშინ გაასუფთავებს ლისტბოქსის ჩანაწერებს. 6_9 ლისტინგზე მოცემულია „ყველას_წაშლის“ პროგრამული კოდის ფრაგმენტი.

```
// ლისტინგი_6.9 --- ListBox.Items.Clear() მეთოდი --ყველას წაშლა-----
private void button3_Click(object sender, EventArgs e)
 {
  DialogResult all = MessageBox.Show("ნამდვილად წავშალოთ
    ყველა ?", "გაფრთხილება", MessageBoxButtons.YesNoCancel,
                                      MessageBoxIcon.Question);
    if (all == DialogResult.Yes)
      {
        label5.Text = "ყველაფერი იშლება !";
        listBox1.Items.Clear();
      }
     else
```

```
        if (all == DialogResult.No)
            label5.Text = "არ იშლება ყველა !";
         else
            label5.Text = "Cancel-ია !";
  }
```

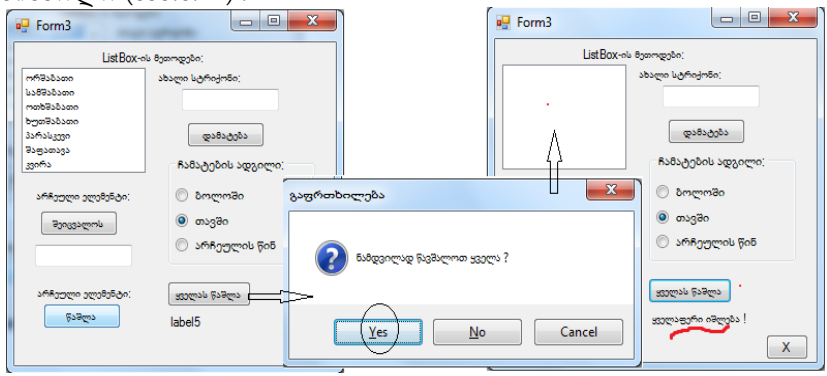
ჩვენს შემთხვევაში კოდში გამოყენებულია MessageBox კლასის Show(პარამეტრები) მეთოდი, “Yes/No/Cancel” ღილაკებით და გამაფრთხილებელი შეტყობინებით (ნახ.6.12). განვიხილოთ კოდის სტრიქონი:

```
DialogResult all = MessageBox.Show("წამდვილად წავშალოთ ყველა ?",
                   "გაფრთხილება", MessageBoxButtons.YesNoCancel,
                    MessageBoxIcon.Question);
```

DialogResult არის System.Windows.Forms სახელსივრცის ჩამონათვლის (enum) ტიპი: `public enum DialogResult`, რომელიც განსაზღვრავს დიალოგური ფანჯრიდან დაბრუნებული იდენტიფიკატორის მნიშვნელობას. all-ს მიენჭება ეს მნიშვნელობა, რომელიც შმდგომ if ბლოკში გამოვიყენეთ.

`MessageBoxButtons` – იძლევა enum ტიპის კონსტანტებს, რომლებიც განსაზღვრავს MessageBox ფანჯარაში გამოსატან ღილაკებს (ჩვენთან: Yes, No, Cancel).

`MessageBoxIcon` - enum ტიპის კონსტანტაა, რომელიც ასახავს შეტყობინებას. მაგალითად, `MessageBoxIcon.Question` ესაა ცისფერ წრეში ჩასმული თეთრი ფერის კითხვის ნიშნის სიმბოლო (ნახ.6.12) .



ნახ.6.12

ListBox-ის რამდენიმე სტრიქონის მონიშვნისათვის, როგორც ზემოთ აღვნიშნეთ, მის Properties-ის SelectionMode-თვისებაში ვაყენებთ MultiExtended-მნიშვნელობას (ნახ.6.5). ახალი ლისტბოქსის გახსნისას, მასში დაყენებულია ერთსტრიქონიანი რეჟიმი: SelectionMode=”One” .

თვისებები SelectedIndices და SelectedItems შეიცავს არჩეული სტრიქონების შესაბამის ნომრებს ან ჩანაწერებს.

ბოლოს, 6.12 ნახაზზე განვიხილოთ ”არჩეული ელემენტის” შეცვლა ახალი სტრიქონით, რომელიც მის ქვემოთ მდებარე ტექსტბოქსშია შეტანილი. კოდს ექნება შემდეგი სახე:

```
// ლისტინგი_6.10 ---შეცვლა---
private void button4_Click(object sender, EventArgs e)
  {
    int st;
    if (textBox2.Text != "" && listBox1.SelectedIndex != -1)
      {
        st = listBox1.SelectedIndex;
        listBox1.Items.RemoveAt(st);
        listBox1.Items.Insert(st, textBox2.Text);
        textBox2.Text = "";
      }
  }
```

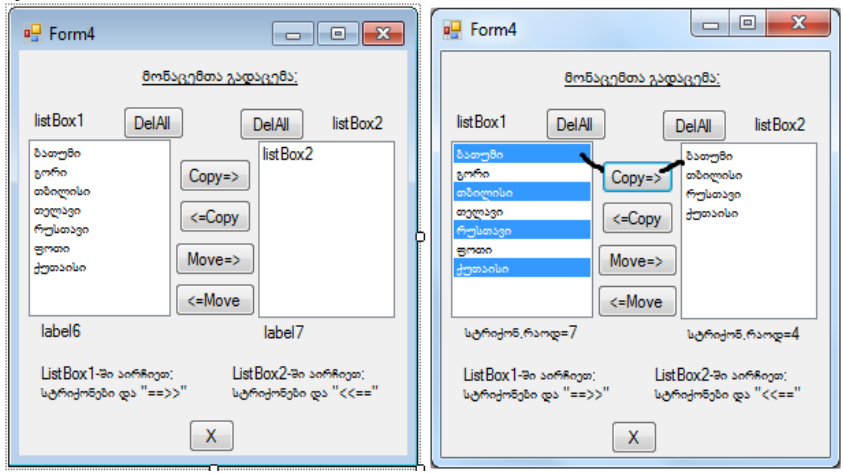
თუ ტექსტბოქსში ჩავწერთ სიტყვას ”შაბათი”, მაშინ ბუტონის ამოქმედებით ”შაფათავა” შეიცვლება სიტყვით ”შაბათი”.

ამოცანა_6.2: ავაგოთ კოდი, რომლითაც შესაძლებელი იქნება ListBox1-ის რამდენიმე სტრიქონის მონიშვნა და მათი ერთდროულად გადატანა (Copy) ListBox2-ში. ამავდროულად, შესაძლებელი უნდა იყოს ListBox2-იდან სტრიქონების უკან დაბრუნება ListBox1-ში. კოდი უნდა შეიცავდეს აგრეთვე ცალკეული სტრიქონების გადატანას (Move) ტექსტბოქსებს შორის და შესაძლებლობას ტქსტბოქსის სტრიქონების მთლიანად წასაშლელად (DeleteAll).

6.13 ნახაზზე ნაჩვენებია ამ ფორმის ერთი ვარიანტი და copy-ღილაკით მიღებული შედეგები.

ListBox-ების ქვეშ გამოიტანება სტრიქონების რაოდენობის მაკონტროლებელი მონაცემები, რომელიც 6_10 ლისტინგზეა მოცემული.

```
// ლისტინგი_6.9 --- Form4-ის label-ში ListBox-ის სტრიქონთა რაოდ. ------
private void Form4_Load(object sender, EventArgs e)
 {
   label6.Text = "სტრიქონ.რაოდ=" + listBox1.Items.Count;
   label7.Text = "სტრიქონ.რაოდ=" + listBox2.Items.Count;
 }
```



ნახ.6.13

6_11 ლისტინგზე ასახულია “copy=>” და “<=copy” ღილაკთა კოდები:

```
// ლისტინგი_6.11 --- ListBox-ში სტრიქონების კოპირება -----
private void button2_Click(object sender, EventArgs e) // =>
 {
      // ჯგუფური რეჟიმი
  listBox1.SelectionMode = SelectionMode.MultiExtended;
   int st;
 for (st = 0; st < listBox1.SelectedItems.Count; st++)
    listBox2.Items.Insert(0, listBox1.SelectedItems[st]);

 label6.Text = "სტრიქონ.რაოდ=" + listBox1.Items.Count;
 label7.Text = "სტრიქონ.რაოდ=" + listBox2.Items.Count;
}
      ...
private void button3_Click(object sender, EventArgs e) //<=
{
```

```
    // ჯგუფური რეჟიმი
 listBox2.SelectionMode = SelectionMode.MultiExtended;
 int st;
 for (st = 0; st < listBox2.SelectedItems.Count; st++)
    listBox1.Items.Insert(0, listBox2.SelectedItems[st]);
 label6.Text = "სტრიქონ.რაოდ=" + listBox1.Items.Count;
 label7.Text = "სტრიქონ.რაოდ=" + listBox2.Items.Count;
}
```

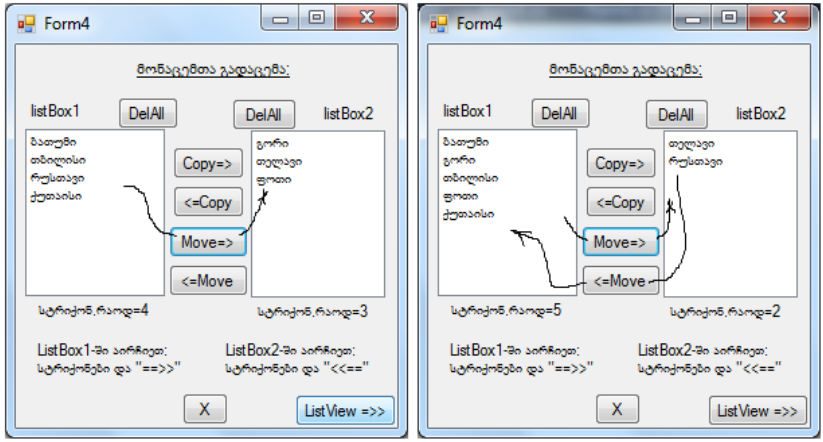
სტრიქონების გადატანა ერთიდან მეორე ტექსტბოქსში ხორციელდება “Move=>” და “<=Move” ღილაკებით, რომელთა კოდები 6_12 ლისტინგზეა მოცემული.

```
// ლისტინგი_6.12 --- ListBox-ებს შორის სტრიქონების გადატანა -----
private void button4_Click(object sender, EventArgs e) // =>
 {    //  ერთსტრიქონიანი რეჟიმი
   listBox1.SelectionMode = SelectionMode.One;
   int st;
   for (st = 0; st < listBox1.SelectedItems.Count; st++)
     {    // სტრიქონის გადატანა მეორეში
       listBox2.Items.Insert(0, listBox1.SelectedItems[st]);
       st=listBox1.SelectedIndex;
       if (st != -1)
         listBox1.Items.RemoveAt(st); // გადატანილი სტრიქონის
                                      // წაშლა პირველში
     }
    label6.Text = "სტრიქონ.რაოდ=" + listBox1.Items.Count;
    label7.Text = "სტრიქონ.რაოდ=" + listBox2.Items.Count;
  }
...
private void button5_Click(object sender, EventArgs e)
{    //  ერთსტრიქონიანი რეჟიმი: გადატანა <=
  listBox2.SelectionMode = SelectionMode.One;
  int st;
  for (st = 0; st < listBox2.SelectedItems.Count; st++)
    {
      listBox1.Items.Insert(0, listBox2.SelectedItems[st]);
      st = listBox2.SelectedIndex;
     if (st != -1)
        listBox2.Items.RemoveAt(st);
     }
   label6.Text = "სტრიქონ.რაოდ=" + listBox1.Items.Count;
```

```
  label7.Text = "სტრიქონ.რაოდ=" + listBox2.Items.Count;
}
```

შედეგი ასახულია 6.14 ნახაზზე.

აქ თავიდან პირველი ლისტბოქსიდან გადატანილ იქნა „გორი“, „თელავი“ და „ფოთი“ Move=> ღილაკით. შემდეგ „გორი“ და „ფოთი“ უკან იქნა დაბრუნებული <=Move ღილაკით და, ბოლოს, „რუსთავი“ გადატანილ იქნა ისევ Mov=> ღილაკით.

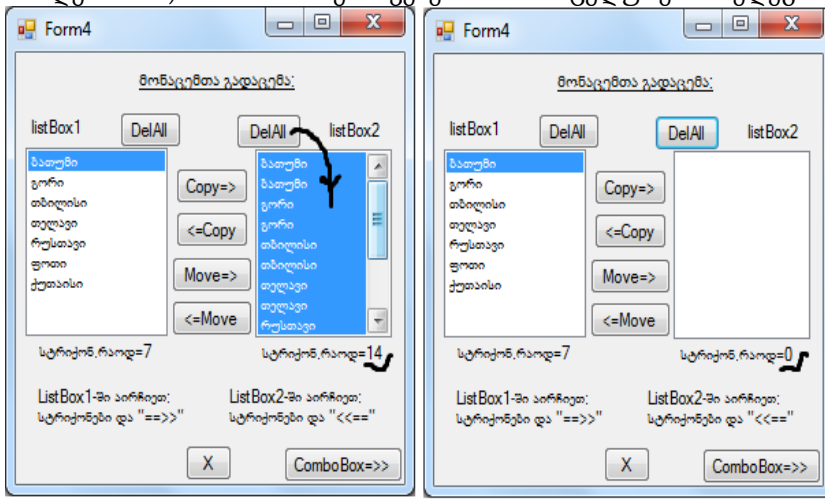


ნახ.6.14

და ბოლოს, ListBox-ების გაწმენდა, ანუ ყველა სტრიქონის წაშლა (გაფრთხილების გარეშე), რომელიც Form4-ზე DellAll-ღილაკებითაა წარმოდგენილი, რეალიზებულია 6_13 ლისტინგში.

```
// ლისტინგი_6.13 --- ListBox-ის გაწმენდა სტრიქონებისგან ----
 private void button6_Click(object sender, EventArgs e)
 {
   listBox1.Items.Clear();
   label6.Text = "სტრიქონ.რაოდ=" + listBox1.Items.Count;
   label7.Text = "სტრიქონ.რაოდ=" + listBox2.Items.Count;
 }

private void button7_Click(object sender, EventArgs e)
 {
   listBox2.Items.Clear();
   label6.Text = "სტრიქონ.რაოდ=" + listBox1.Items.Count;
   label7.Text = "სტრიქონ.რაოდ=" + listBox2.Items.Count; }
```

მარცხენა ფორმაზე ListBox1-დან ორჯერ გადაკოპირდა ყველა სტრიქონი მარჯვენაში, რაც აისახა სტრიქონების რაოდენობაში (14). შემდეგ მოვნიშნეთ ListBox2-ში ყველა სტრიქონი და DelAll ღილაკის ამოქმედებით გავწმინდეთ იგი (სტრიქონების რაოდენობა=0). 6.15 ნახაზზე ნაჩვენებია ამ პროცედურების შედეგი.

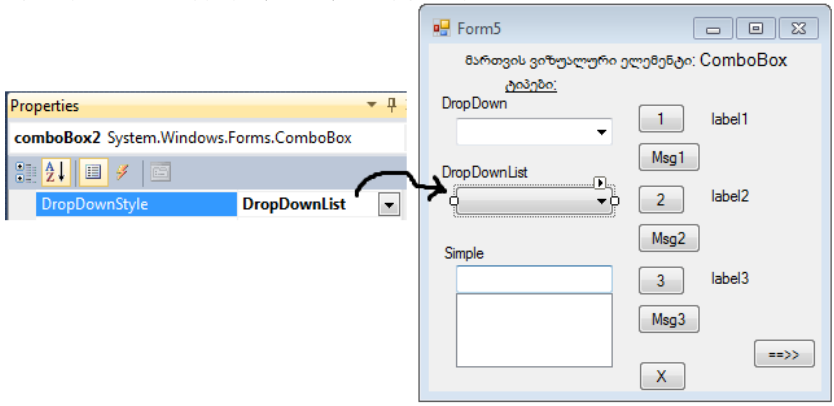


ნახ.6.15

6.2. მართვის ვიზუალური ელემენტი ComboBox და თვისება DropDownStyle

ComboBox ვიზუალური ელემენტი ListBox-ის და TextBox-ის ელემენტთა სიმბიოზია, იგი იყენებს ორივეს მახასიათებლებს, რომლებიც ჩვენ ზემოთ განვიხილეთ. ამავდროულად მისი წარმოდგენის ფორმა განსხვავდება ორივესგან და ეფექტურია (ფორმაზე იკავებს შედარებით მცირე ადგილს და ამორჩევის მექანიზმითაც სარგებლობს). 6.16 ნახაზზე ნაჩვენებია ComboBox-ის სამი ვარიანტი. მისი ტიპი Properties-ში მიეთითება DropDownStyle თვისების ერთ-ერთი მნიშვნელობით.

- DropDown - სტანდარტულად ეს მნიშვნელობაა დაყენებული. სამკუთხა ისრით ჩამოიშლება სტრიქონების სია ListBox-ის მსგავსად, ან ტექსტურ ველში შეიტანება სტრიქონი TextBox-ივით;
- DropDownList - გამორთავს TextBox-ის შესაძლებლობას, ანუ ვეღარ შევტანთ ტექსტს. კომბობოქსი მუშაობს ლისტბოქსის რეჟიმში და არის ტექსტბოქსივით მცირე ზომის;
- Simple - ყოველთვის ღიაა სტრიქონების სია. შეიძლება სტრიქონის არჩევაც და ახლის შეტანაც.



ნახ.6.16

შენიშვნა: ComboBox-ისთვის არ გამოიყენება თვისება SelectionMode.

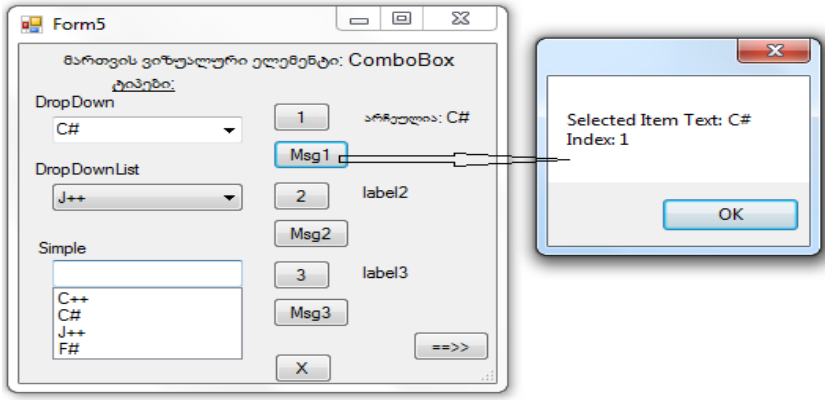
ამოცანა_6.3: ავაგოთ 6.16 ნახაზზე ნაჩვენები ფორმა და ავამოქმედოთ 1,2,3-ღილაკები კომბობოქსის სამი განსხვავებული ტიპის საილუსტრაციოდ. 6_14 ლისტინგზე მოცემულია პროგრამული კოდი. დამატებითი შეტყობინება გამოვიტანოთ MsgBox-ფანჯარაში შესაბამისი კომბობოქსის არჩეული სტრიქონის მნიშვნელობის (`comboBox.SelectedItem`) და მისი ინდექსის (ნომრის) (`comboBox.SelectedIndex`) საილუსტრაციოდ.

თითოეული ღილაკისთვის ფორმაზე Properties-ში დავაყენოთ DropDownStyle თვისების ერთ-ერთი მნიშვნელობა: 1-DropDown, 2- DropDownStyleList ან 3- Simple.

Form5_Load(object…)-ში პროგრამულად ჩაიწერება სამივე ComboBox-ის სტრიქონები, მაგალითად, C++, C#, J++, F# და ა.შ. (ან შეიძლება Edit Items რედაქტორის გამოყენება).

```
// ლისტინგი_6.14 – ComboBox-ის ტიპები და თისება: DropDownStyle ---
using System;
using System.Drawing;
using System.Windows.Forms;
namespace WinListCombo
{
   public partial class Form5 : Form
   {
      public Form5()   {   InitializeComponent();   }
     private void Form5_Load(object sender, EventArgs e)
      {
         comboBox1.Items.Add("C++");
         comboBox1.Items.Add("C#");
         comboBox1.Items.Add("J++");
         comboBox1.Items.Add("F#");
         comboBox2.Items.Add("C++");
         comboBox2.Items.Add("C#");
         comboBox2.Items.Add("J++");
         comboBox2.Items.Add("F#");
         comboBox3.Items.Add("C++");
         comboBox3.Items.Add("C#");
         comboBox3.Items.Add("J++");
         comboBox3.Items.Add("F#");
      }
     private void button1_Click(object sender, EventArgs e)
      {
         label1.Text="არჩეულია: "+ comboBox1.Text;
      }
     private void button2_Click(object sender, EventArgs e)
      {
         label2.Text = "არჩეულია: " + comboBox2.SelectedItem;
      }
   private void button3_Click(object sender, EventArgs e)
      {
         label3.Text = "არჩეულია: " + comboBox3.Text;
      }
  private void button6_Click(object sender, EventArgs e)
      {
         int selectedIndex = comboBox1.SelectedIndex;
```

```
            Object selectedItem = comboBox1.SelectedItem;

            MessageBox.Show("Selected Item Text: " + selectedItem + "\n" +
                        "Index: " + selectedIndex.ToString());
        }
    private void button7_Click(object sender, EventArgs e)
        {
            int selectedIndex = comboBox2.SelectedIndex;
            Object selectedItem = comboBox2.SelectedItem;

            MessageBox.Show("Selected Item Text: " + selectedItem + "\n" +
                        "Index: " + selectedIndex.ToString());
        }
       private void button8_Click(object sender, EventArgs e)
        {
            int selectedIndex = comboBox3.SelectedIndex;
            Object selectedItem = comboBox3.SelectedItem;

            MessageBox.Show("Selected Item Text: " + selectedItem + "\n" +
                        "Index: " + selectedIndex.ToString());
        }
        private void button5_Click(object sender, EventArgs e)
        {
            Close();
        }
}} //პროგრამის მუშაობის შედეგი ნაჩვენებია 6.17 ნახაზზე.
```

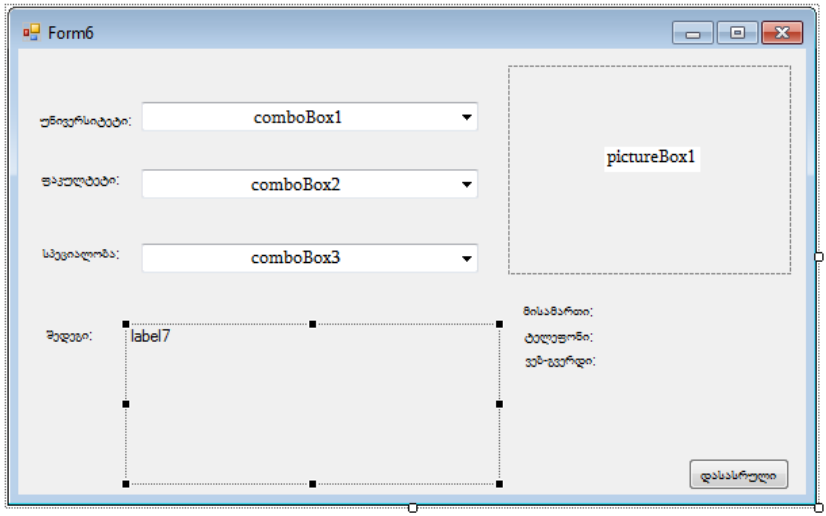


ნახ.6.17

გავლილი მასალის საფუძველზე ავაგოთ ახალი პროგრამული პროექტი, რომელშიც გამოყენებულ იქნება ComboBox კლასის თისებები და მეთოდები.

ამოცანა_6.4: შევქმნათ პროგრამული აპლიკაცია „უნივერსიტეტები“, რომელიც იძლევა ინფორმაციას მათი ფაკულტეტებისა და სპეციალობების შესახებ. მომხმარებელი ირჩევს სამდონიანი ComboBox-ების სისტემაში თავის სასურველ მონაცემებს (ნახ.6.18):

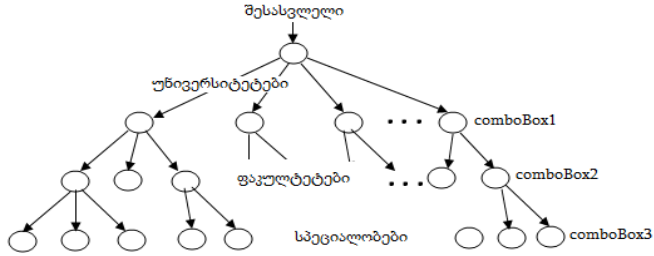
უნივერსიტეტი -> ფაკულტეტი -> სპეციალობა.



ნახ.6.18

შედეგები აისახება label7 ტექსტურ ველში. ცვლილება ახალი მოთხოვნის შესასრულებლად შესაძლებელია სამივე დონეზე. ზოგადი იერარქიული მოდელი, რომლის პროგრამული რეალიზაცია შესაძლებელია ვიზუალური ემენეტებისა და პროგრამული switch() ... case ჩალაგებული გადამრთველების ერთობლიობით, ნაჩვენებია 6.19 ნახაზზე.

ნახ.6.19



ფორმაზე გამოიტანება აგრეთე უნივერსიტეტების თანმხლები გრაფიკული და ტექსტური ინფორმაცია, შესაბამისად pictureBox1 და სამი label-ის საშუალებით (მისამართი, ტელეფონი, ვებ-გვერდი). მათი ცვლილება ხდება comboBox1-ით არჩეული მნიშვნელობის შესაბამისად. 6_15 ლისტინგზე ნაჩვენებია ამ პროგრამის რეალიზაციის კოდის ფრაგმენტი.

```
// ლისტინგი_6.15 --  ComboBox კლასის თვისებები და მეთოდები ----
using System;
using System.Drawing;
using System.Windows.Forms;
namespace WinListCombo
{
 public partial class Form6 : Form
  {
   int fa = 0;
   string itemU, itemF, itemS;
   public Form6() {  InitializeComponent();   }
  private void Form6_Load(object sender, EventArgs e)
  {
    comboBox1.Items.Add("თბილისის სახელმწიფო უნივერსიტეტი");
    comboBox1.Items.Add("საქართველოს ტექნიკური უნივერსიტეტი");
    comboBox1.Items.Add("თბილისის სახელმწიფო სამედიცინო
                                                  უნივერსიტეტი");
    comboBox1.Items.Add("ილიას სახელმწიფო უნივერსიტეტი");
   }
  private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
  {
   int selectedIndexU = comboBox1.SelectedIndex;
   Object selectedItemU = comboBox1.SelectedItem;
   itemU = selectedItemU.ToString();
   switch (selectedIndexU)
```

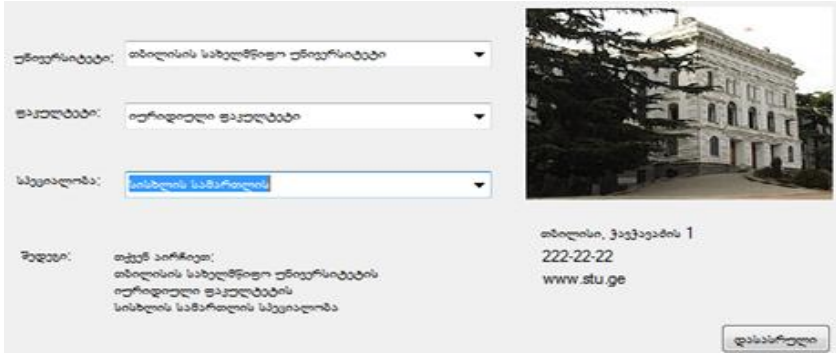
```
    {
     case 0:
        comboBox2.Items.Clear();
        comboBox2.Items.Add("ზუსტ და საბუნებისმეტყველო მეცნ. ფაკ.");
        comboBox2.Items.Add("იურიდიული ფაკულტეტი");
        comboBox2.Items.Add("ჰუმანიტარულ მეცნ. ფაკულტეტი");
        comboBox2.Items.Add("სოციალურ-პოლიტიკურ მეცნ. ფაკ.");
        comboBox2.Items.Add("ეკონომიკისა და ბიზნესის ფაკულტეტი");
        comboBox2.Items.Add("სამედიცინო ფაკულტეტი");
pictureBox1.Image=Image.FromFile("C:\\C#2010\\4_5\\WinListCombo\\tsu.jpg");
        label4.Text = "თბილისი, ჭავჭავაძის 1";
        label5.Text="222-22-22";
        label6.Text="www.stu.ge";
        fa = 100;
        break;
     case 1:
       comboBox2.Items.Clear();
       comboBox2.Items.Add("არქიტექტურის და მშენებლობის ფაკ.");
       comboBox2.Items.Add("ენერგეტიკის და კავშირგაბმულობის ფაკ.");
       comboBox2.Items.Add("ინფორმატიკის და მართვის სისტემების ფაკ.");
       comboBox2.Items.Add("მექანიკური და სატრანსპორტო ფაკ.");
       comboBox2.Items.Add("ბიზნესის ინჟინერიის ფაკულტეტი");
       comboBox2.Items.Add("სამთო-გეოლოგიური ფაკულტეტი");
pictureBox1.Image = Image.FromFile("C:\\C#2010\\4_5\\WinListCombo\\gtu.jpg");
        label4.Text = "თბილისი, კოსტავას 77";
        label5.Text="237-37-37";
        label6.Text="www.gtu.ge";
        fa = 101;
        break;
    case 2:
        comboBox2.Items.Clear();
        comboBox2.Items.Add("მედიცინის ფაკულტეტი");
        comboBox2.Items.Add("სტომატოლოგიის ფაკულტეტი");
       comboBox2.Items.Add("ფარმაციის ფაკულტეტი");
       comboBox2.Items.Add("საზოგადოებრივი ჯანდაცვის ფაკულტეტი");
 comboBox2.Items.Add("სპორტული მედიცინის და რეაბილიტაციის ფაკ.");
 pictureBox1.Image =Image.FromFile("C:\\C#2010\\4_5\\WinListCombo\\meduni.jpg");
        label4.Text = "თბილისი, ვაჟა-ფშაველას 41";
        label5.Text="239-39-39";
        label6.Text="www.tsmu.edu.ge";
```

```
            fa = 102;
            break;
        case 3:
             comboBox2.Items.Clear();
            comboBox2.Items.Add("ბიზნესის ფაკულტეტი");
            comboBox2.Items.Add("საინჟინრო ფაკულტეტი");
            comboBox2.Items.Add("სამართლის ფაკულტეტი");
            comboBox2.Items.Add("მეცნიერებათა და ხელოვნების ფაკ.");
            comboBox2.Items.Add("სპორტის ფაკულტეტი");
            pictureBox1.Image =
Image.FromFile("C:\\C#2010\\4_5\\WinListCombo\\iliauni.jpg");
            label4.Text = "თბილისი, ჭავჭავაძის 101";
            label5.Text="239-39-39";
            label6.Text="www.iliauni.edu.ge";
            fa = 103;
            break;
              // case 4:   და ა.შ. -------
        default: break;
           }
        }
    private void comboBox2_SelectedIndexChanged(object sender, EventArgs e)
        {
         int selectedIndexF = comboBox2.SelectedIndex;
         Object selectedItemF = comboBox2.SelectedItem;
         itemF = selectedItemF.ToString();
          // MessageBox.Show("SelectedIndexF: " + selectedIndexF.ToString());
         switch (fa)
          {
           case 100:    // თსუ
              switch (selectedIndexF)
               {
                case 0:
                     comboBox3.Items.Clear();
                    comboBox3.Items.Add("ფიზიკა-მათემატიკის");
                    comboBox3.Items.Add("ბიოლოგიის");
                    comboBox3.Items.Add("ქიმიის");
                    comboBox3.Items.Add("ინფორმატიკის");
                    break;
                 case 1:
                    comboBox3.Items.Clear();
                    comboBox3.Items.Add("სისხლის სამართლის");
```

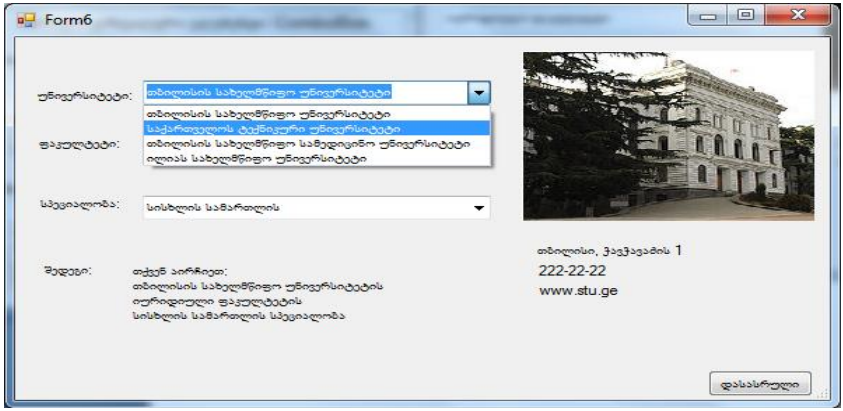
```
        comboBox3.Items.Add("სამოქალაქო სამართლის");
        comboBox3.Items.Add("ადმინისტრაციული სამართლის");
        break;
    case 2:
        comboBox3.Items.Clear();
        comboBox3.Items.Add("საქრთველოს ისტორიის");
        comboBox3.Items.Add("გეოგრაფიის");
        comboBox3.Items.Add("ფილოლოგიისა და ჟურნალისტიკის");
        break;
  case 3:   // და ა.შ. -------
  default: break;
  }
     break;
  case 101:   // სტუ
    switch (selectedIndexF)
     {
      case 0:
          comboBox3.Items.Clear();
          comboBox3.Items.Add("სამოქალაქო მშენებლობის");
          comboBox3.Items.Add("ურბანისტიკის");
          comboBox3.Items.Add("ხუროთმოძღვრების");
          comboBox3.Items.Add("რკინაბეტონის კონსტრუქციების");
          break;
      case 1:
          comboBox3.Items.Clear();
          comboBox3.Items.Add("ჰიდროენერგეტიკის");
          comboBox3.Items.Add("თბოენერგეტიკის");
          comboBox3.Items.Add("ატომური ენერგეტიკის");
          break;
      case 2:
          comboBox3.Items.Clear();
          comboBox3.Items.Add("ქსელების და სისტემების");
          comboBox3.Items.Add("მართვის ავტომატიზებული სისტ.");
          comboBox3.Items.Add("ეკონომიკური ინფორმატიკის");
          break;
      case 3: // და ა.შ. -------
      default: break;
     }
    break;
  case 102:     // სამედიცინო უნივერსიტეტი
    switch (selectedIndexF)
```

```
                {
                 case 0:
                     comboBox3.Items.Clear();
                     comboBox3.Items.Add("ფსიქიატრიის");
                     comboBox3.Items.Add("ქირურგიის");
                     comboBox3.Items.Add("კარდიოლოგიის");
                     comboBox3.Items.Add("თერაპიის");
                     break;
                 case 1:
                     comboBox3.Items.Clear();
                     comboBox3.Items.Add("ყბა-სახის ქირურგიის");
                     comboBox3.Items.Add("თერაპიის");
                     comboBox3.Items.Add("პროტეზირების");
                     break;
                 case 2:
                     comboBox3.Items.Clear();
                     comboBox3.Items.Add("საპროვიზორო");
                     comboBox3.Items.Add("საფარმაცევტო");
                     comboBox3.Items.Add("სადიაგნოსტიკო");
                     break;
                 case 3: // და ა.შ. -------
                 default: break;
                }
                break;
            }
    }
 private void comboBox3_SelectedIndexChanged(object sender, EventArgs e)
    {
       int selectedIndexS = comboBox3.SelectedIndex;
       Object selectedItemS = comboBox3.SelectedItem;
       itemS = selectedItemS.ToString();
       label7.Text = "თქვენ აირჩიეთ: \n" +
            itemU + "ს \n" +
            itemF + "ს \n" +
            itemS +" სპეციალობა";

    }
}}
```

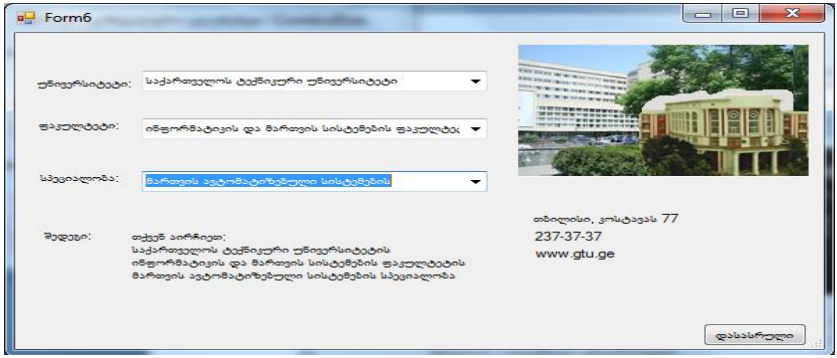
6.20-23 ნახაზებზე ილუსტრირებულია ამ აპლიკაციის მუშაობის ფრაგმენტები სხვადასხვა მოთხოვნების შესრულებისას.



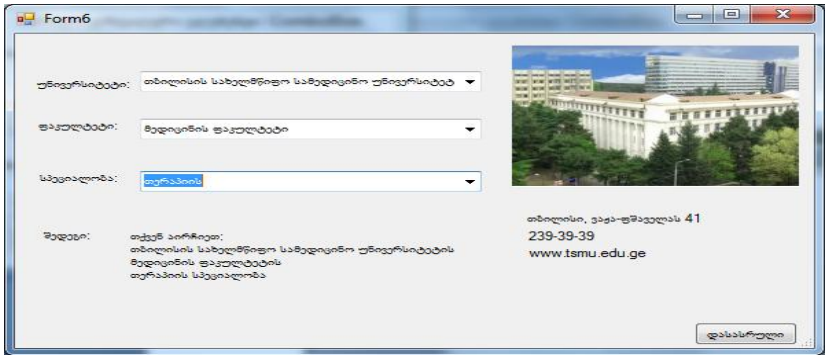
ნახ.6.20



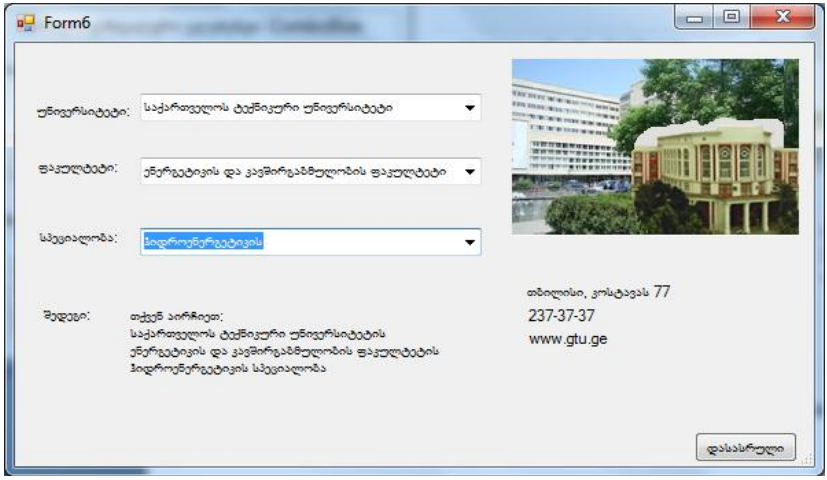
ნახ.6.21



ნახ.6.22



ნახ.6.23



ნახ.6.24

და ა.შ.

სისტემის გაფართოება შესაძლებელია რეალური მონაცემებითაც, თუმცა აპლიკაციის გადასაწყვეტად არსებობს სხვა ხერხები და მეთოდებიც (მაგალითად, მონაცემთა ბაზების გამოყენებით, სადაც მოთავსდება დიდი მოცულობის ინფორმაცია), რომლებიც საგრძნობლად შეამცირებს კოდის სიგრძეს. ჩვენ ამ საკითხებს მომავალში განვიხილავთ.

6.3. კითხვები და სავარჯიშოები:

6.1. როგორ ფუნქციონირებს მართვის ელემენტი ListBox ?

6.2. როდის და როგორ იყენებენ ელემენტს CheckedListBox ?

6.3. როდის და როგორ იყენებენ MultiExtended თვისებას ?

6.4. რა არის Item თვისება და რომელია მისი მეთოდები ?

6.5. ააგეთ კოდი ListBox-ში მონაცემების ჩასამატებლად, შესაცვლელად და წასაშლელად.

6.6. როგორ ფუნქციონირებს მართვის ელემენტი ComboBox ?

6.7. ააგეთ კოდი ორი ListBox-ით და მეთოდებით, რომლებიც უზრუნველყოფენ მონაცემების გაცვლას ამ ლისტბოქსებს შორის, აგრეთვე მონაცემთა წაშლას, აღდგენას და ერთმანეთთან შედარებას.

6.8. ComboBox-ის რომელი ტიპები იცით და როგორ ფუნქციონირებს თითოეული ?

6.9. როგორ ხდება ComboBox-ის სტრიქონების შევსება და კორექტირება ?

6.10. რა განსხვავებაა `SelectedItem` და `SelectedIndex` თვისებებს შორის ?

6.11. როდის გამოიყენება თვისებები `SelectedItems` და `SelectedIndices` ?

6.12. ააგეთ ფორმა „ვალუტის გადაცვლა“: ერთი ComboBox-ით უცხოური ვალუტის ასარჩევად. ორი TextBox-ით: ლარების რაოდენობის და უცხოური ვალუტის დღევანდელი კურსის მნიშვნელობის შესატანად. დაწერეთ ფულის კონვერტაციის კოდი.

6.15. ააგეთ ბიბლიოთეკაში მკითხველის სარეგისტრაციო ანკეტის დიალოგური ფორმა. გამოიყენეთ: ComboBox-ები, TextBox-ები, CheckBox-ები და სხვა ელემენტები, რომელთაც ჩათვლით საჭიროდ თქვენი პროექტისთვის.

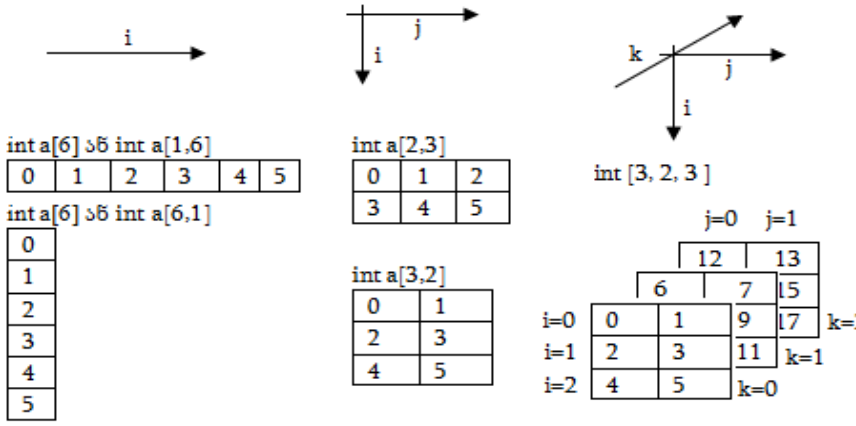
6.16. ააგეთ ბიბლიოთეკაში წიგნების აღრიცხვის ფორმა: ავტორი, დასახელება, ენა, ISBN, გამომცემლობა, ქალაქი, გვერდების_რაოდენობა, გამოცემის წელი, სფერო, ჟანრი.

6.17. ააგეთ პერსონის CV ფორმა.

თავი 7. მასივები: კლასები, მეთოდები და მასივებთან მუშაობის პრაქტიკა

მასივი ერთგვაროვანი ცვლადების მნიშვნელობათა ერთობლიობაა, რომელსაც აქვს უნიკალური სახელი. მასივი შეიძლება იყოს ერთ, ორ, სამ და მეტ განზომილებიანი. მისი ყველა ელემენტი ერთი ტიპისაა. მასივის ელემენტი შეიძლება იყოს ისევ მასივი. მაგალითად, ვექტორი არის ერთგანზომილებიანი მასივი. იგი შეიძლება იყოს მთელრიცხვა (int [] Vect), სტრიქონული (string [] Name) ან ნამდვილრიცხვა (decimal [] Money). ორგანზომილებიანია მატრიცა int [,] Mat, სადაც მძიმით სტრიქონებეის და სვეტების რაოდენობაა გამოყოფილი. სამგანზომილებიანის მაგალითია int [, ,] Cube და ა.შ.

7.1 ნახაზზე ნაჩვენებია 1,2 და 3 განზომილებიანი მასივების საილუსტრაციო მაგალითები.



ნახ.7.1

ერთიდაიგივე რიცხვების (0,1,2,3,4,5) განლაგება მანქანის ოპერატიულ მეხსიერებაში დამოკიდებულია მათ გამოცხადების ფორმაზე. მონაცემთა ასეთი სტრუქტურები კი შემდგომ უნდა

იქნეს გათვალისწინებული პროგრამებში, რომლებიც მათ ამუშავებენ.

7.2 ნახაზზე ნაჩვენებია ორგანზომილებიან მასივში ელემენტთა განლაგების ზოგადი სქემა.

i \ j	0	1	2	j	n
0	a[0,0]	a[0,1]	a[0,2]	a[0,j]	a[0, n]
1	a[1,0]	a[1,1]	a[1,2]	a[1,j]	a[1, n]
2	a[2,0]	a[2,1]	a[2,2]	a[2,j]	a[2, n]
	...	...	...	...	...
i	a[i,0]	a[i,1]	a[i, 2]	a[i, j]	a[i, n]
	...	...	...	...	...
m	a[m,0]	a[m,1]	a[m,2]	a[m,j]	a[m,n]

ნახ.7.2

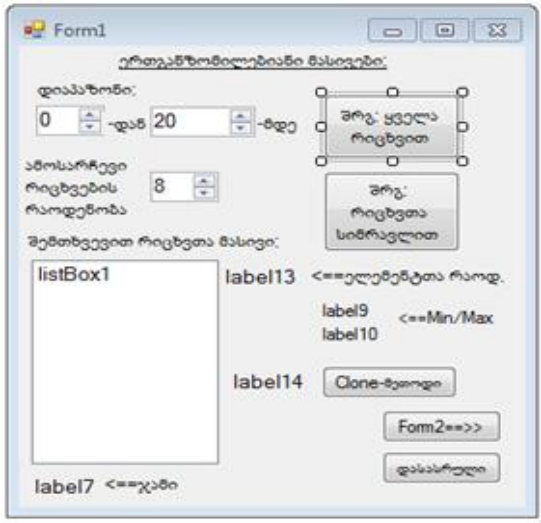
ახლა გადავიდეთ კონკრეტული ამოცანების გადაწყვეტაზე მასივთა კლასების, მეთოდებისა და თვისებების გამოყენების საფუძველზე.

7.1. ერთგანზომილებიანი მასივები

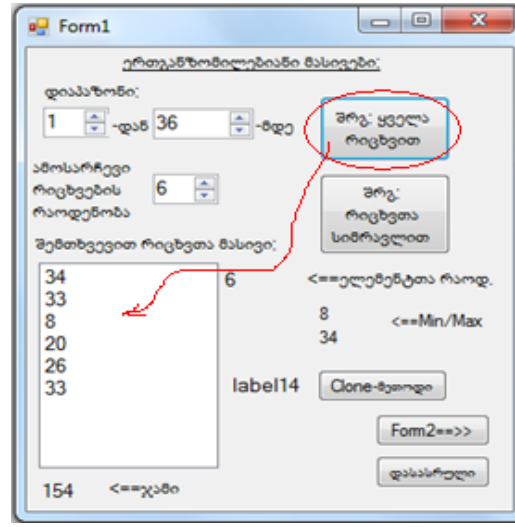
ამოცანა_7.1: საჭიროა დაიწეროს პროგრამა რანდომ-ფუნქციის (შემთხვევით რიცხვთა გენერატორის) გამოყენებით, რომელიც მთელ რიცხვთა მითითებული დიაპაზონიდან ამოარჩევს განსაზღვრული რაოდენობის რიცხვებს. მაგალითად, უნდა შეირჩეს 6 რიცხვი 1-36 დიაპაზონში, შედეგების ფორმაზე გამოტანით. შესაძლებელი უნდა იყოს მრავალჯერადი ექსპერიმენტის ჩატარება. ვარიანტი-1: რიცხვები შეიძლება გამეორდეს (კომპლექტი). ვარიანტი-2: რიცხვები არ უნდა გამეორდეს (სიმრავლე).

7.1 ნახაზზე ნაჩვენებია ფორმის ერთ-ერთი შესაძლო ვარიანტი ვიზუალური ელემენტებით, სადაც საწყისი მონაცემები შეირჩევა numericUpDown (1-3) სამი მთვლელით, ხოლო შედეგები აისახება listBox1-ში.

მაგალითად, რიცხვთა დიაპაზონი [1-20] და ამოსარჩევ რიცხვთა რაოდენობა 10. „შრგ“-გენერატორის ღილაკი ავტომატურად ამოირჩევს შემთხვევით 10 რიცხვს ამ დიაპაზონიდან.



ნახ.7.1-ა



პროგრამის კოდი მოცემულია 7_1 ლისტინგზე.

```
// ლისტინგი_7.1-ა -  Array- მასივები „კომპლექტით“ ---------
namespace WinArray
{
  public partial class Form1 : Form
   {
     Random Ricxvi = new Random();
     int i;
     public Form1() { InitializeComponent(); }
private void button1_Click(object sender, EventArgs e)
{ // რიცხვები მეორდება
  int minNum=(int)numericUpDown1.Value;  // დიაპაზონის
  int maxNum = (int)numericUpDown2.Value; //მნიშვნელობები
  int Raod = (int)numericUpDown3.Value; //დიაპაზ.რიცხ.რაოდ.
 int[] mas = new int[Raod]; // mas-ერთგანზომ.მასივის  გამოცხ.
                                          // Raod-ელემენტით
 int SUM = 0, Min, MinInd, Max, MaxInd;
 listBox1.Items.Clear();
 // შემთხვევითი რიცხვების  გენერაცია ----
 for (i = 0; i < Raod; i++)
    {
     mas[i] = Ricxvi.Next(minNum, maxNum);
     listBox1.Items.Add(mas[i].ToString());
    }
    // ჯამის ანგარიში -----
 for (i = 0; i < mas.Length; i++)  // მასივის ელემენტების
                                   // რაოდენობა (length-1)
  {
   SUM += mas[i];
   label7.Text = SUM.ToString();
  }
  // მინ  და  მაქს  -  რიცხვების  პოვნა მასივში -----
  Min = mas[0]; Max = mas[0];
  MinInd = 0; MaxInd = 0;
  for (i = 1; i < mas.Length; i++)
    {
     if (mas[i] < Min)
       {
         Min = mas[i];
         MinInd = i;
       }
```

```
      if (mas[i] > Max)
        {
          Max = mas[i];
          MaxInd = i;
        }
       }
      label9.Text = Min.ToString();
      label10.Text = Max.ToString();
     }
  }
}
```

კოდში for-ციკლისთვის გამოიყენება Length თვისება, რომელიც მასივის ელემენტთა რაოდენობაა. მისი მაქსიმალური მნიშვნელობაა Length-1, რადგანაც ელემენტთა ინდექსირება [index] იწყება 0-ით.

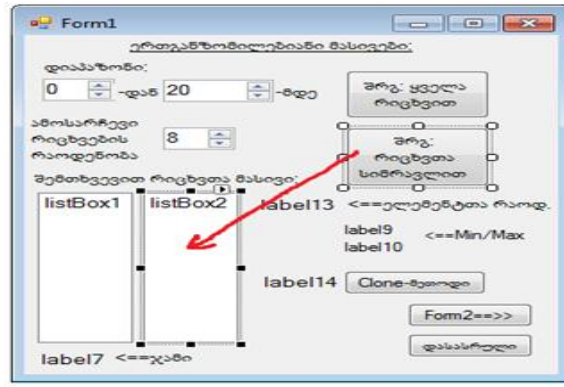
ამოცანის მეორე ნაწილი ეხება შემთხვევით რიცხვთა გენერატორის მიერ შექმნილი მასივიდან სიმრავლის ფორმირებას, ანუ მასივში ერთნაირი ელემენტები არ მეორდება (ნახ.7.1-ბ). ამისათვის გამოიყენება Distinct(T) მეთოდი და IEnumerable<T> ჩამოთვლითი ტიპის ინტერფეისი.

IEnumerable<T> ხელს უწყობს მითითებული ტიპის კოლექციის ელემენტების მარტივ გადარჩევას (იტერაციის პროცედურას). ის მოთავსებულია System.Collections.Generic სახელსივრცეში.

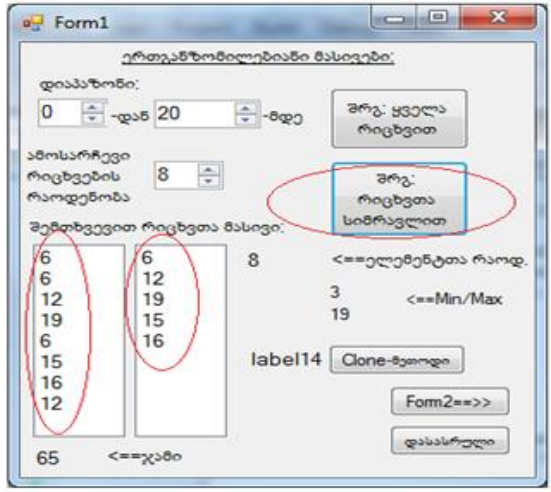
Distinct(T) მეთოდი უზრუნველყოფს მითითებული ელემენტების ერთობლიობიდან დააფორმიროს განსხვავებული მნიშვნელობის ელემენტების ერთობლიობა (ანუ სიმრავლე). ელემენტების ერთმანეთთან შედარება ხორციელდება ავტომატურად. ის მოთავსებულია System.Linq სახელსივრცეში. ღილაკის პროგრამული კოდი მოცემულია 7.1-ბ ლისტინგზე.

```
// ლისტინგი_7.1-ბ -  Array- მასივები „სიმრავლით“ ---------
using System.Collections.Generic;
using System.Linq;     // იხ. დამატება 7.5
...
namespace WinArray
{
  public partial class Form1 : Form
```

```
    {
     Random Ricxvi = new Random();
     int i;
     public Form1() { InitializeComponent(); }
...
private void button2_Click(object sender, EventArgs e)     {
// რიცხვები არ მეორდება
 int minNum = (int)numericUpDown1.Value;
 int maxNum = (int)numericUpDown2.Value; //დიაპაზონები
 int Raod = (int)numericUpDown3.Value; // რიცხვების რაოდ.
 int i;
 int[] mas = new int[Raod];
 listBox1.Items.Clear();
 for (i = 0; i < Raod; i++)
  { // საწყისი მასივის ფორმირება
    mas[i] = Ricxvi.Next(minNum, maxNum);
   listBox1.Items.Add(mas[i].ToString());
  }
       // სიმრავლის ფორმირება
  IEnumerable<int> distinctMas = mas.Distinct();

  listBox2.Items.Clear();
  foreach (int i_mas in distinctMas)
    {  // სიმრავლის ელემენტები
       listBox2.Items.Add(i_mas).ToString();
    }
     listBox1.Refresh();
  }
```



ნახ.7.1-ბ1



ნახ.7.1-ბ2

Array კლასის Clone() მეთოდი:

Array კლასის გამოყენებით ენის შესაძლებლობები საგრძნობლად ფართოვდება. მაგალითისათვის განვიხილოთ ამ კლასის Clone() მეთოდი. მას შეუძლია მთლიანი მასივის კოპირება, ანუ მთლიანი ობიექტის ასლის შექმნა.

ამოცანა_7.2: პროგრამა ქმნის მასივის ასლს Clone()-მეთოდით, ახდენს მის მოწესრიგებას და ძებნის პროცედურის განხორციელებას მითითებული მნიშვნელობისათვის. ილუსტრაციისთვის გამოვიყენოთ 7.1 ნახაზე, რომელზეც მოთავსებულია „Clone“-ღილაკი.

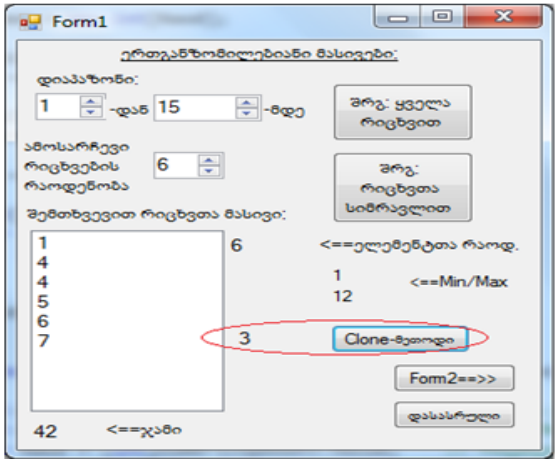
7_2 ლისტინგზე მოცემულია Clone() მეთოდის შესაბამისი ღილაკის კოდი.

```
// ლისტინგი_7.2 – Array კლასის Clone() მეთოდი ----------
private void button5_Click(object sender, EventArgs e)
{ // Clone() მეთოდი
  int minNum = (int)numericUpDown1.Value;
  int maxNum = (int)numericUpDown2.Value; // დიაპაზონები
  int Raod = (int)numericUpDown3.Value; // რიცხვების რაოდ.
  int[] a = new int[Raod], b = new int[Raod];
  int i, Ind;
```

```
  for (i = 0; i < a.Length; i++)
     a[i] = Ricxvi.Next(minNum, maxNum);

    b = (int[])a.Clone();
    Array.Sort(b);          // მოწესრიგება ზრდადობით
    listBox1.Items.Clear();
    for (i = 0; i < a.Length; i++)
       listBox1.Items.Add(b[i]);

   Ind=Array.IndexOf(b,5); //პოულობს რიცხვი 5-ის ინდექსს
   label14.Text = Ind.ToString();
}
```

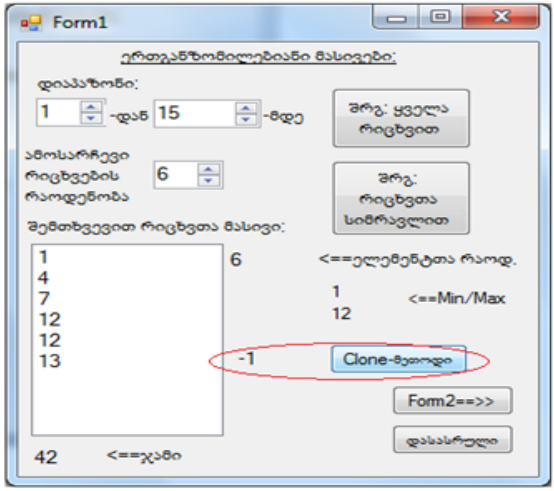
შედეგები ასახულია 7.2 ნახაზზე. ამგვარად, Clone მეთოდი, თავიდან ქმნის მიმთითებელს ზოგადი object-კლასის ობიექტზე. შემდეგ ეს მიმთითებელი (int[])-კასტით გარდაიქმნება int ტიპის ერთგანზომილებიან მასივზე მიმთითებლად. ბოლოს b მასივში თავსდება a[] მასივის შესაბამისი ელემენტები.

Array კლასის Sort() მეთოდი ალაგებს b მასივს ზრდადობით. შემდეგ კოდს გამოაქვს listBox-ში ეს რიცხვები. IndexOf მეთოდით განისაზღვრება მითითებული “5“-ანის მდებარეობის ინდექსი Ind = Array.IndexOf(b,5). მაგალითად, Ind=3 პირველ შემთხვევაში (პირველი ელემენტის ინდექსია 0) და Ind=-1, მეორე შემთხვევაში, როცა 5-ანი არ არის რიცხვთა მწკრივში.



ნახ. 7.2-ა

ნახ. 7.2-ბ



Clone() მეთოდით მასივის ასლის შექმნა პრაქტიკული ამოცანებისათვის მეტად მნიშვნელოვანია, რამეთუ აქ ფიზიკურად ორ სხვადასხვა მასივთან გვაქვს საქმე და ერთში ცვლილებების განხორციელება არ იწვევს მეორის ცვლილებებს. ე.ი. შენარჩუნებულია საწყის მონაცემთა მნიშვნელობები, რაც ხშირად საჭირო და სასურველია.

საჭიროა აღვნიშნოთ, რომ **b = a** მარტივი მინიჭების ოპერატორი ვერ ასრულებს a მასივის კოპირებას b მასივში. ამ დროს იქმნება მხოლოდ მეორე მიმთითებელი a მასივზე b-დან. ე.ი. „ორივე მასივიდან“ მიმართვა ხდება ერთიდაიმავე ობიექტზე. ცვლილება b-მასივში ნიშნავს ცვლილებას a-ში, რაც პრაქტიკულად ხშირად მიუღებელია.

7.2. მრავალგანზომილებიანი მასივები

საინჟინრო და სხვა სფეროს პრაქტიკულ ამოცანებში ხშირად საჭიროა ორ-, სამ- და მეტგანზომილებიანი მონაცემთა მასივების გამოყენება. განვიხილოთ რამდენიმე მაგალითი.

ამოცანა_7.2: დავუშვათ, სამი ობიექტისათვის უნდა განისაზღვროს მითითებული რაოდენობის შემთხვევითი რიცხვების ერთობლიობა და მათი ჯამები.

ამ შემთხვევაში გვექნება მთელრიცხვა ტიპის ორგანზომილებიანი მასივი:

int[,] mas= new int[m,n],

სადაც პირველი ინდექსი m რიცხვების რაოდენობაა, მეორე n ობიექტების (ანუ სვეტების) რაოდენობა. 7.3 ნახაზზე ნაჩვენებია ფორმა, რომელზეც სამ listBox-ში უნდა აისახოს „შემთხვევით რიცხვთა გენერატორის“ ღილაკის ამოქმედებით ამორჩეული მნიშვნელობები.

ნახ.7.3

ამ ღილაკის შესაბამისი კოდი მოცემულია 7.3_ლისტინგში.

```
// ლისტინგი_7.3 -- ორგანზომილებიანი მასივის მაგალითი ----
namespace WinArray
{
 public partial class Form2 : Form
  {
```

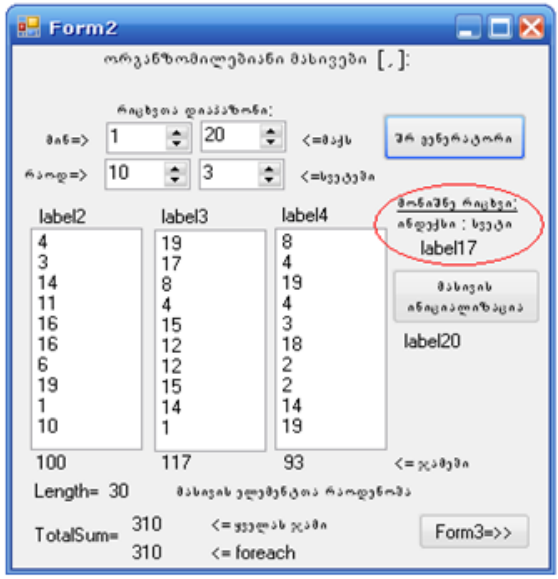
```
    Random Ricxvi = new Random();  // შემთხვ.რიცხვ.გენ.
    int i,s;
    public Form2() {InitializeComponent(); }
private void button1_Click(object sender, EventArgs e)
 {
   int minNum = (int)numericUpDown1.Value;
   int maxNum = (int)numericUpDown2.Value; //დიაპაზონები
   int Raod = (int)numericUpDown3.Value; //რიცხვების რაოდ.
   int SvetRaod = (int)numericUpDown4.Value; // სვეტ.რაოდ.

   int[,] mas;
   mas = new int[Raod, SvetRaod];
   int sum1 = 0, sum2 = 0, sum3 = 0, TotalSum=0;
   listBox1.Items.Clear();
   listBox2.Items.Clear();
   listBox3.Items.Clear();
// GetUpperBound(0)-მეთოდი მასივის 1-ელი ინდექსისთვის, ანუ 0
   for (i = 0; i <= mas.GetUpperBound(0); i++)
    {
            // მასივის მე-2 ინდექსისთვის, ანუ 1
      for (s = 0; s <= mas.GetUpperBound(1); s++)
             mas[i, s] = Ricxvi.Next(minNum, maxNum);

     listBox1.Items.Add(mas[i,0]); //შემთხვევითი რიცხვები
     listBox2.Items.Add(mas[i,1]);
     listBox3.Items.Add(mas[i,2]);
    }
    for (i = 0; i <= mas.GetUpperBound(0); i++)
      {
       sum1 += mas[i, 0];
       sum2 += mas[i, 1];
       sum3 += mas[i, 2];
      }
    TotalSum = sum1 + sum2 + sum3;
    label11.Text = sum1.ToString(); //სვეტში რიცხვთა ჯამი
    label12.Text = sum2.ToString();
    label13.Text = sum3.ToString();
    label14.Text = mas.Length.ToString(); //მასივის სიგრძე
    label25.Text = TotalSum.ToString();
  //TotalSum -- მთლიანი ჯამის ანგარიში for ციკლით ---
    label29.Text = "";
     label30.Text = "";
```

```
    TotalSum = 0;
    for (i = 0; i < Raod; i++) // გარე ციკლი
      for (int j = 0; j < SvetRaod; j++) // შიგა ციკლი
        {
          label29.Text += " " + mas[i, j];
          TotalSum += mas[i, j];
          label30.Text += " "+TotalSum.ToString();
        }
  // TotalSum მთლიანი ჯამის ანგარიში foreach ციკლით -----
    TotalSum = 0;
    foreach (int i in mas)
      {
        TotalSum += i;
        label27.Text = TotalSum.ToString();
      }
     }
      ...
    }
}
```

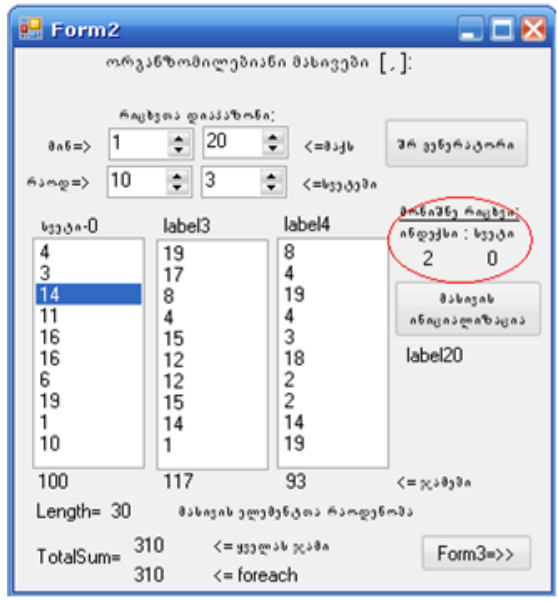
პროგრამაში mas[m,n] მასივის სიგრძე განისაზღვრება მისი თვისებით `Length`, რომელიც გაითვლება m*n ნამრავლით და `label15`-ში თავსდება (ნახ.7.3 Length = 8x3 =24).

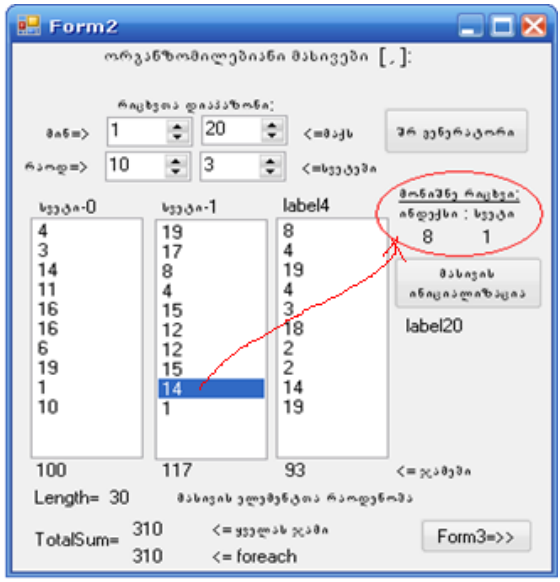
ორგანზომილებიანი mas[m,n] მასივის ელემენტებთან სამუშაოდ პროგრამაში ორი for() ციკლი გამოიყენება, რომლებშიც `GetUpperBound()` მეთოდით განისაზღვრება მასივის ელემენტთა „ზედა საზღვრები“, ანუ m-მასივის სტრიქონების მაქსიმალური რაოდენობა და n-მასივის სვეტების მაქსიმალური რაოდენობა (ჩვენს შემთხვევაში [8,3] ნახ.7.3).

ამოცანა_7.3: საჭიროა მიღებულ რიცხვთა მასივის ელემენტების მონიშვნა და შესაბამისი ინდექსების განსაზღვრა. მაგალითად, 7.4 ნახაზზე ნაჩვენებია [10,3] მასივის 10-10 რიცხვი სამ სვეტში, რომლის ერთი მნიშვნელობის მონიშვის შემდეგ (მაგ., 14) label17-ში ჩაიწერება 2-რიცხვის ინდექსი და 0-სვეტის ნომერი. აქ მოყვანილია ოთხი შემთხვევა. სვეტის ნომრებია: 0,1,2;

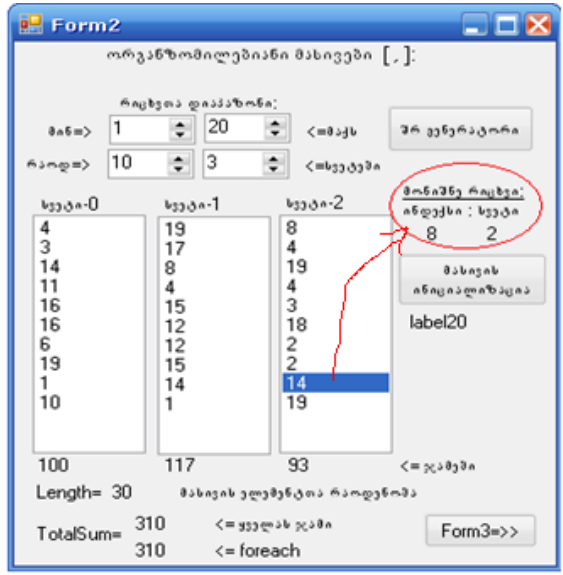

Form2
ორგანზომილებიანი მასივები [,]:
რაცხვთა დიაპაზონი:
მინ=>
1
20
<=მაქს
ჟრ ვუნრაკორა
რაოდ=>
10
3
<=სვეტები
label2
label3
label4
4
3
14
11
16
16
6
19
1
10
19
17
8
4
15
12
12
15
14
1
8
4
19
4
3
18
2
2
14
19
მონიშნე რაცხვა:
ინდექსი : სვეტი
label17
მასივის ინიციალიზაცია
label20
100
117
93
<= ჯამები
Length= 30
მასივის ელემენტთა რაოდენობა
TotalSum=
310
<= ყველას ჯამი
310
<= foreach
Form3=>>

ნახ.7.4-ა


Form2
ორგანზომილებიანი მასივები [,]:
რაცხვთა დიაპაზონი:
მინ=>
1
20
<=მაქს
ჟრ ვუნრაკორა
რაოდ=>
10
3
<=სვეტები
სვეტი-0
label3
label4
4
3
14
11
16
16
6
19
1
10
19
17
8
4
15
12
12
15
14
1
8
4
19
4
3
18
2
2
14
19
მონიშნე რაცხვა:
ინდექსი : სვეტი
2
0
მასივის ინიციალიზაცია
label20
100
117
93
<= ჯამები
Length= 30
მასივის ელემენტთა რაოდენობა
TotalSum=
310
<= ყველას ჯამი
310
<= foreach
Form3=>>


Form2
ორგანზომილებიანი მასივები [,]:
რაცხვთა დიაპაზონი:
მინ=>
1
20
<=მაქს
შრ გენერატორი
რაოდ=>
10
3
<=სვეტები
სვეტი-0
სვეტი-1
label4
მონიშნე რაცხვა:
ინდექსი : სვეტი
8 1
მასივის ინიციალიზაცია
label20
4
3
14
11
16
16
6
19
1
10
19
17
8
4
15
12
12
15
14
1
8
4
19
4
3
18
2
2
14
19
100
117
93
<=ჯამები
Length= 30
მასივის ელემენტთა რაოდენობა
TotalSum=
310
<= ყველას ჯამი
310
<= foreach
Form3=>>

ნახ.7.4-ბ


Form2
ორგანზომილებიანი მასივები [,]:
რაცხვთა დიაპაზონი:
მინ=>
1
20
<=მაქს
შრ გენერატორი
რაოდ=>
10
3
<=სვეტები
სვეტი-0
სვეტი-1
სვეტი-2
მონიშნე რაცხვა:
ინდექსი : სვეტი
8 2
მასივის ინიციალიზაცია
label20
4
3
14
11
16
16
6
19
1
10
19
17
8
4
15
12
12
15
14
1
8
4
19
4
3
18
2
2
14
19
100
117
93
<=ჯამები
Length= 30
მასივის ელემენტთა რაოდენობა
TotalSum=
310
<= ყველას ჯამი
310
<= foreach
Form3=>>

ამოცანის გადაწყვეტა პროგრამულად ხორციელდება listBox-ისთვის სპეციალური მოვლენის (event) კოდის შექმნით, რომელიც 7_5 ლისტინგზეა მოცემული. listBox-ში კურსორით ახალი რიცხვის მონიშვნა გააქტიურებს listBox1_SelectedIndexChanged () მოვლენას.

```
// ლისტინგი_7.5 --- მასივის ინდექსების გამოთვლა ---
private void listBox1_SelectedIndexChanged(object
                                    sender, EventArgs e)
{
 label2.Text = "სვეტი-0";
 listBox2.SelectedIndex = -1;
 listBox3.SelectedIndex = -1;
 label17.Text= listBox1.SelectedIndex + " 0";
}

private void listBox2_SelectedIndexChanged(object
                                    sender, EventArgs e)
{
 label3.Text = "სვეტი-1";
 listBox1.SelectedIndex = -1;
 listBox3.SelectedIndex = -1;
 label17.Text = listBox2.SelectedIndex + "  1";
}

private void listBox3_SelectedIndexChanged(object
                                   sender, EventArgs e)
{
 label4.Text = "სვეტი-2";
 listBox1.SelectedIndex = -1;
 listBox2.SelectedIndex = -1;
 label17.Text = listBox3.SelectedIndex + " 2";
}
```

მასივში მაუსით ელემენტის ამორჩევისას `SelectedIndex` თვისების მნიშვნელობა შეესაბამება მის ინდექსებს. თუ რიცხვი 0-სვეტშია, მაშინ `SelectedIndex = -1` ორი სხვა სვეტისტის (1 და 2);

ახლა განვიხილოთ მრავალგანზომილებიანი მასივების ინიციალიზაციის ამოცანა, ანუ მასივის ელემენტებისთვის საწყისი მნიშვნელობების მინიჭების მექანიზმი.

ამოცანა_7.4: ავაგოთ პროგრამის კოდი „მასივის ინიციალიზაცია“-ღილაკისთვის, რომელიც ერთ-, ორ- და სამგანზომილებიანი წინასწარგანსაზღვრული მასივებიდან (იხ. ლისტინგი_7.6) ამოარჩევს ელემენტების მნიშვნელობებს მითითებული ინდექსების საფუძველზე.

```
// ლისტინგი_7.6 -- მასივის ინიციალიზაცია {....} ----
private void button2_Click(object sender, EventArgs e)
 {
  int[] mas1 = { 9, 27, 37, 74 };
  double[,] mas2 = {{ 0.5, 2.75, 9.99 }, { 1.0, 3.8, 4.4 }};
  int[, ,] mas3 = {{{  5,31,60,51}, {9,27,37,74}},
                   {{11,25,30,80}, { 2, 1,26,85}} };
  label20.Text = "mas1[1] ="  +      mas1[1].ToString() +
                 "\nmas2[0,2]=" +   mas2[0,2].ToString() +
                 "\nmas3[0,0,0]=" + mas3[0,0,0].ToString() +
                 "\nmas3[1,0,0]=" + mas3[1,0,0].ToString() +
                 "\nmas3[0,1,0]=" + mas3[0,1,0].ToString() +
                 "\nmas3[0,0,1]=" + mas3[0,0,1].ToString() +
                 "\nmas3[0,1,3]=" + mas3[0,1,3].ToString() ;
 }
```

ფიგურულ ფრჩხილებში მოთავსებულია მასივის ელემენტთა მნიშვნელობები. როგორც ლისტინგიდან ჩანს, მასივის განზომილებას შეესაბამება {...}, {{...}} და {{{...}}} ფრჩხილთა რაოდენობა.

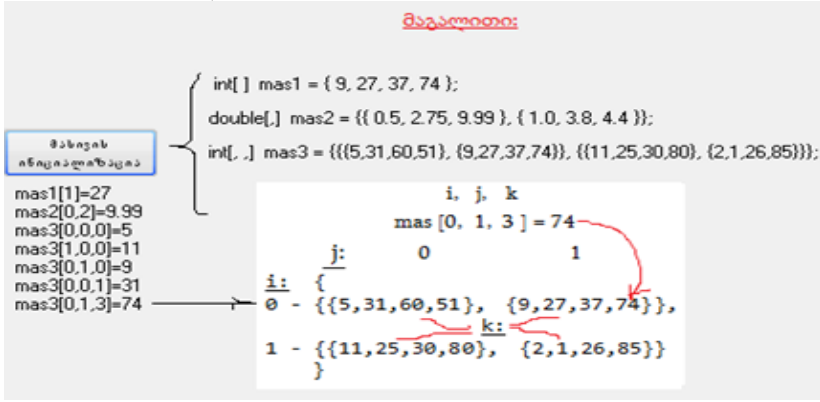
7.5 ნახაზზე მოცემულია საილუსტრაციო მაგალითი აღწერილ mas1, mas2 და mas3 მასივებში ფრჩხილებში მოთავსებულ ელემენტთა იდენტიფიკაციის შესახებ მათი ინდექსების მიხედვით.

ერთგანზომილებიანი მასივის {.,.,.} -ში მოთავსებული პირველი ელემენტის ინდექსი არის 0, ანუ mas1[0] = 9; mas1[2] = 37 და ა.შ.

ორგანზომილებიან მასივში მოთავსებულია 2 სტრიქონი და 3 სვეტი: mas2={{...}, {...}}. მაგალითად, mas2[0,0] = 0.5; mas2[0,2] = 9.99; mas2[1,0] = 1.0; mas2[1,2] = 4.4 და ა.შ.

სამგანზომილებიან მასივში mas3(i,j,k) ინდექსები ასე იცვლება: i=0..1; j=0..1; k=0..4. მაგალითად, mas3[0,0,0] = 5; mas3[0,0,1]

= 31; mas3[0,0,2] = 60; mas3[0,0,3] = 51; mas3[0,1,0] = 9; mas3[0,1,1] = 27; mas3[1,0,0] = 11 და ა.შ.



ნახ.7.5

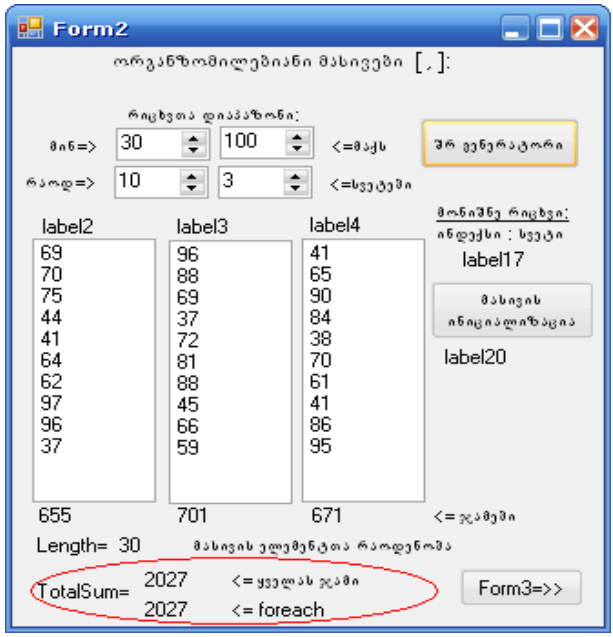
პროგრამულად შესაძლებელია მრავალგანზომილებიანი მასივის ყველა ელემენტის მნიშვნელობის გამოტანა ეკრანზე მითითებული ინდექსების შესაბამისად. აქ საჭიროა ჩადგმული ციკლების ორგანიზება. ერთგანზომილებიან მასივს ერთი, ორგანზომილებიანს ორი, სამგანზომილებიანს - სამი ციკლი დასჭირდება. მაგალითად, სამგანზომილებიანი რიცხვთა მასივისთვის ზოგადად გვექნება:

```
for (int i=1; i<M; i++)
    for(int j=1; j<N; j++)
        for(int k=1; k<K;  k++)
            label1.Text = mas[i,j,k].ToString();
```

7.6 ნახაზზე ფორმის ქვედა ნაწილში ნაჩვენებია მასივის ელემენტთა მთლიანი ჯამის ანგარიშის შედეგები (მაგ., TotalSum=2027).

ერთი შედეგი მიღებულია ჩალაგებული for…for ციკლებით (ლისტინგი_7_7), ხოლო მეორე foreach ციკლით (ლისტინგი_7_8). ლისტინგების შედარებით კარგად ჩანს, თუ რამდენად ეფექტურია მეორე მექანიზმის გამოყენება.

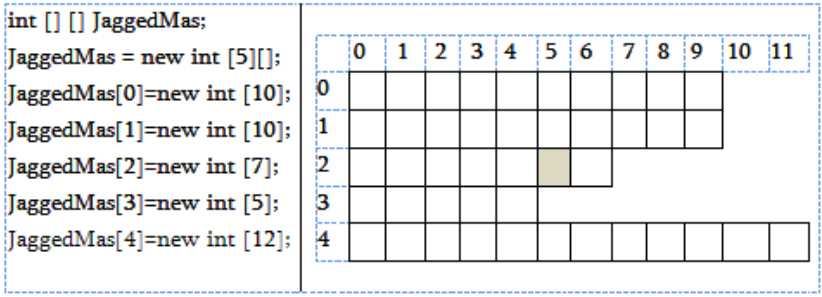
```
//ლისტინგი_7.7: TotalSum -მთლიანი ჯამის ანგარიში for ციკლით--
  label29.Text = "";
  label30.Text = "";
  TotalSum = 0;
  for (i = 0; i < Raod; i++)   // გარე ციკლი
    for (int j = 0; j < SvetRaod; j++)   // შიგა ციკლი
      {
       label29.Text += " " + mas[i, j];
       TotalSum += mas[i, j];
       label30.Text += " "+TotalSum.ToString();
      }

// ლისტინგი_7.8: TotalSum ჯამის ანგარიში foreach ციკლით --
   TotalSum = 0;
   foreach (int i in mas)
      {
        TotalSum += i;
        label27.Text = TotalSum.ToString();
      }
```



ნახ.7.6

7.3. თავისუფალი მასივები

თავისუფალი ანუ ”დაკბილული” მასივები განსხვავდება ჩვენს მიერ აქამდე განხილულ ”მართკუთხა” მასივებისგან, რომლებშიც სტრიქონებს ერთნაირი სიგრძის სვეტები ჰქონდა (ნახ.7.7).



ნახ.7.7

ასეთი ტიპის მასივებს ხშირად უწოდებენ „მასივების მასივს“. ანუ, ჩვენ შემთხვევაში JaggedMas [5][] მასივი შედგება 5 ერთგანზომილებიანი მასივისგან, რომელთაც სხვადასხვა სიგრძე ექნება. მათ ეს მნიშვნელობა მიენიჭება ცალკ-ცალკე, როგორც ეს 7.7 ნახაზზეა ნაჩვენები.

„დაკბილული“ მასივებისთვის დამახასიათებელია მის ელემენტებზე წვდომის განსხვავებული ხერხი. მაგალითად, თუ გვინდა მე-2 სტრიქონის მე-5 სვეტის ელემენტის მნიშვნელობის ამოღება, მაშინ სწორკუთხა მასივში გვაქვს: JaggedMas[2,5], თავისუფალში - JaggedMas[2][5].

გასათვალისწინებელია აგრეთვე ჩადგმული for ციკლების მუშაობის განსხვავებული პრინციპი დაკბილული მასივებისთვის. თუ სწორკუთხა მასივებისთვის გვქონდა, მაგალითად:

```
for (int i=0; i<Jagged.Mas.Length; i++)  // გარე ციკლი
  for(int j=0; j<JaggedMas.Length; j++)  // შიგა ციკლი
      sum  +=  JaggedMas[i,j];  // სწორკუთხა მასივის
                                // ელემენტთა ჯამი
```

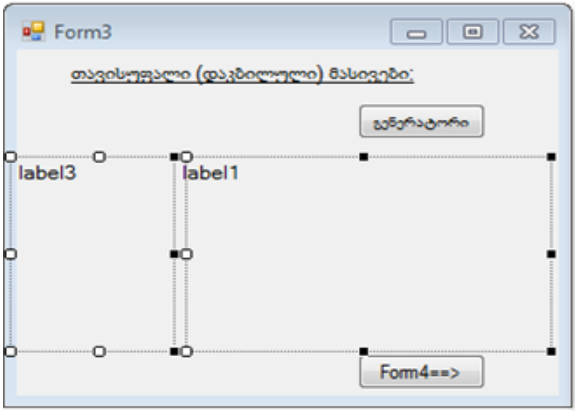
დაკბილული მასივებისთვის საჭიროა გათვალისწინებულ იქნას (შიგა ციკლისთვის) ჩადგმული ერთგანზომილებიანი მასივების (სტრიქონების) განსხვავებული სიგრძეები, რომლებიც i-გარე ციკლის ინდექსის ფუნქციაა:

```
for (int i=0; i<Jagged.Mas.Length; i++)   // გარე ციკლი
    for(int  j=0;  j<JaggedMas[i].Length;  j++)     // შიგა ციკლში გაჩნდა
                                        // [i] ინდექსი !!!
      sum += JaggedMas[i][j];   // დაკბილული მასივის ელემენტთა ჯამი
```

ახლა განვიხილოთ პროგრამული კოდის აგების ამოცანა თავისუფალი (დაკბილული) მასივის გამოყენებით.

ამოცანა_7.5: ავაგოთ პროგრამა, რომელშიც 5-სტრიქონიანი დაკბილული მასივის ელემენტები შეივსება შემთხვევითი რიცხვების გენერატორიდან. მასივის ტიპი ამჯერად იყოს ნამდვილრიცხვა, მაგალითად, double. ჩადგმულ მასივთა (სტრიქონთა) სიგრძეები იყოს: 3, 7, 2, 2, 5. შედეგები გამოტანილ უნდა იქნეს label-ში.

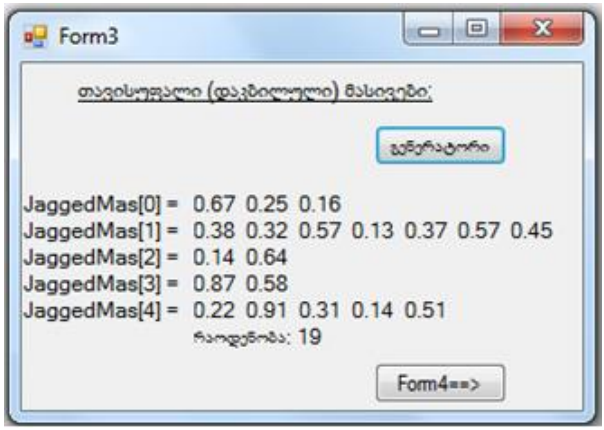
7.8-ა ნახაზზე ნაჩვენებია ფორმა ერთი ღილაკთ, რომელიც აამოქმედებს შემთხვევით რიცხვთა გენერატორს. დაკბილული მასივების დამუშავების პროგრამის ტექსტი მოთავსებულია ამ ღილაკის კოდში, რომელიც 7_8 ლისტინგზეა წარმოდგენილი.



ნახ.7.8-ა

შედეგები ასახულია 7.8-ბ ნახაზზე.

ნახ.7.8-ბ




```
// ლისტინგი_7.8 --- თავისუფალი მასივები ----
using System;
using System.Drawing;
using System.Windows.Forms;
namespace WinArray
{
  public partial class Form3 : Form
   {
    Random Ricxvi = new Random();
    public Form3() { InitializeComponent(); }
private void button1_Click(object sender, EventArgs e)
 {
  double[][] JaggedMas=new double[5][];
  int i, j, Raod = 0;
  JaggedMas[0] = new double[3];
  JaggedMas[1] = new double[7];
  JaggedMas[2] = new double[2];
  JaggedMas[3] = new double[2];
  JaggedMas[4] = new double[5];
  label1.Text = "";
  label3.Text = "";
  for (i = 0; i < JaggedMas.Length; i++)
    {
     label3.Text += "JaggedMas[" + i + "]= ";
    for (j = 0; j < JaggedMas[i].Length; j++)
      {
       JaggedMas[i][j] = Ricxvi.NextDouble();
  label1.Text += JaggedMas[i][j].ToString("F") + "  ";
```

```
        }
       Raod += JaggedMas[i].Length;
       label1.Text += "\n";
       label3.Text += "\n";
      }
      label1.Text += "რაოდენობა: " + Raod;
    }
  }
}
```

კოდში გამოყენებულია შემთხვევით რიცხვთა გენერატორის მეთოდი `NextDouble()`, რომელიც ავტომატურად ირჩევს რიცხვებს [0.0 – 1.0] დიაპაზონში.

label1-ში `ToString("F")`-შაბლონის გამოყენებით `JaggedMas[i][j]`-ის double ტიპის რიცხვი იბეჭდება #.## ფორმატით. “F”-ის გარეშე იქნებოდა #.################ (მაგალითად, 0.123456789123456).

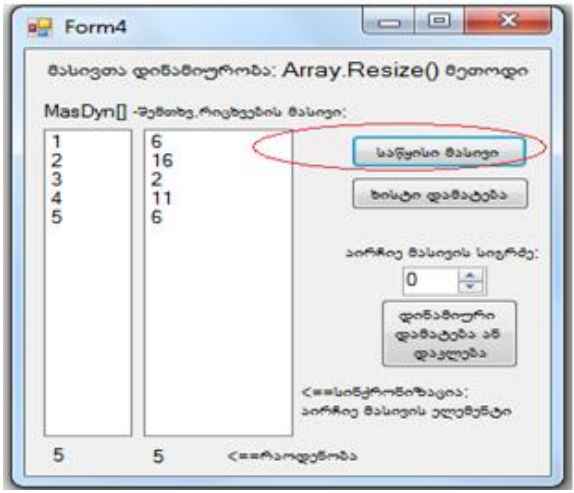
7.4. მასივების დინამიურობა

თუ პროგრამის ამოქმედების დასაწყისში არაა ცნობილი, რამდენი ელემენტი ექნება მასივს, შესაძლებელია ეს რაოდენობა განისაზღვროს კოდის უშუალოდ მუშაობის პროცესში, დინამიკურად.

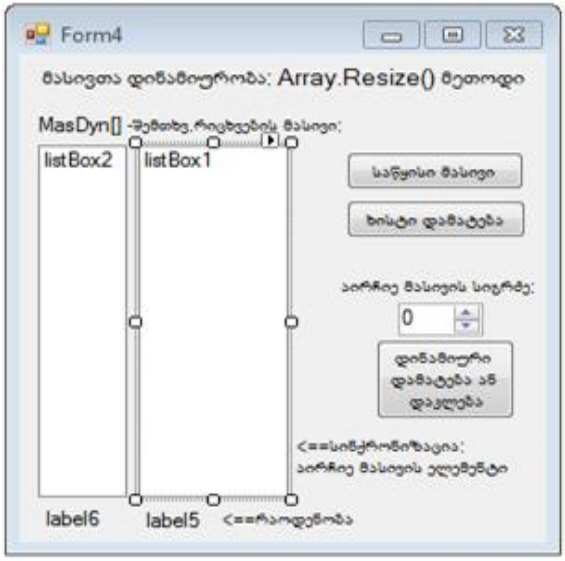
მასივის განზომილების განსაზღვრა ან ცვლილება ხორციელდება Array კლასის Resize() მეთოდით: Array.Resize() კონსტრუქციით. თუ მასივის ზომა იზრდება, მაშინ ძველი ელემენტები ინახება. თუ მცირდება - მაშინ ზედმეტი ელემენტები იკარგება. განვიხილოთ რამდენიმე მაგალითი მასივების დინამიური ცვლილების შესახებ.

ამოცანა_7.6: ავაგოთ 7.9-ა ნახაზზე ნაჩვენები ფორმა მართვის ელემენტებით: listBox1,2 - მთელრიცხვა ტიპის მასივის (MasDyn[]) ელემენტების მნიშვნელობების და მისი ინდექსების გამოსატანად, რომლებსაც აფორმირებს შემთხვევით რიცხვთა გენერატორი (შრგ). მასივის საწყისი ზომა კოდში ცნობილია (მაგალითად, 5). ღილაკით „საწყისი მასივი“ ამოქმედდება შრგ და

მასივს მიანიჭებს ელემენტთა მნიშვნელობებს ციკლურად. შედეგები აისახება listBox-ებში (ნახ.7.9-ბ).

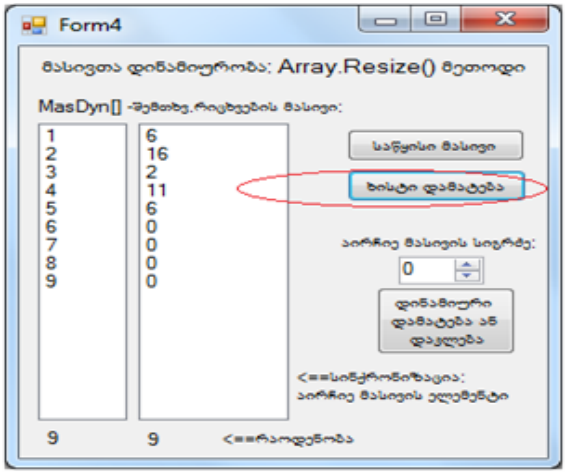


ნახ.7.9-ა

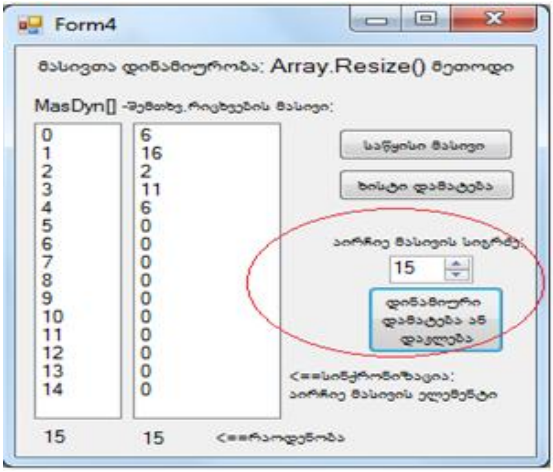


ნახ. 7.9-ბ

ღილაკით „ხისტი დამატება“ ხდება მასივის ზომის პროგრამული ცვლილება, კერძოდ, ჩვენ შემთხვევაში Array.Resize(ref MasDyn, 9) ოპერატორით ამოქმედდება Arrai კლასის Resize მეთოდი, რომელიც **ref**-საკვანძო სიტყვით განახორციელებს მასივზე მითითებას, რომლის ზომაც უნდა გახდეს 9, ანუ საწყის მასივს ამჯერად დაემატება 4 ელემენტი (5+4=9). 7.10 მარცხენა ნახაზზე ნაჩენებია ეს შემთხვევა, ოთხი 0-ელემენტით მასივის ბოლოში.

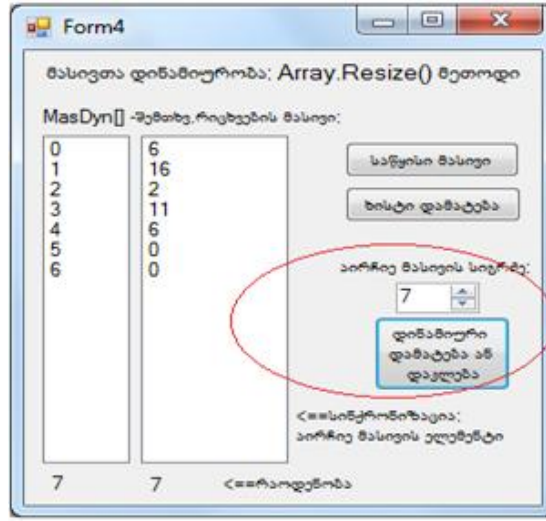


ნახ.7.10-ა



ნახ.7.10-ბ

მასივის ზომის „ხისტი“ ცვლილება შეზღუდული შესაძლებლობაა, ვინაიდან ხშირად მასივის ზომა წინასაწარ უცნობია და მხოლოდ პროგრამის მუშაობის პროცესში ხდება ცნობილი, თუ რა ზომის ცვლილებებია საჭირო. ასეთი დინამიკური რეჟიმი ნაჩვენებია 7.10-ბ და 7.11 ნახაზებზე. პირველში მასივის ზომა 15-ია, მეორეში 7 (ანუ მცირედება, რაც იწვევს ელემენტების დაკარგვას) და მესამეში - 50, ანუ მასივის ზომა იზრდება და მას ემატება, ჩვენ შემთხვევაში 0-ელემენტები.



ნახ. 7.11

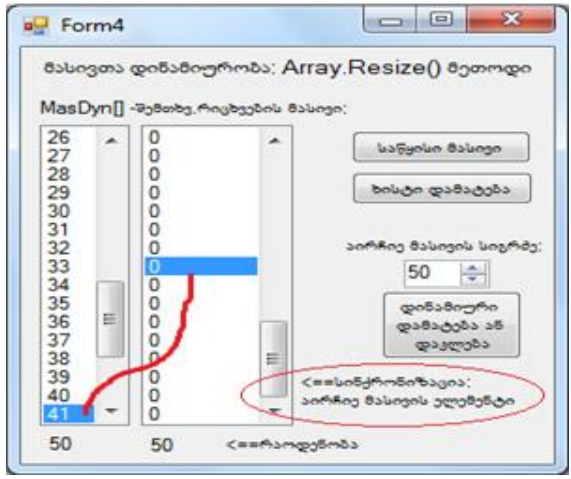
მასივის ზომის დინამიკური ცვლილების (გაზრდის) შემდეგ შესაძლებელია მისი ელემენტების გარკვეული მნიშვნელობების დამატება, მაგალითად ისევ შრგ-ით, ან ხელით, ან სხვა მასივიდან და ა.შ. 7.9_ლისტინგზე მოცემულია აღწერილი ამოცანის გადაწყვეტის ერთ-ერთი ვარიანტის პროგრამული კოდი.

```
// ლისტინგი_7.9 --- მასივების დინამიკა:  Arrai.Resize() მეთოდი ----
using System;
using System.Drawing;
using System.Windows.Forms;
namespace WinArray
{
```

```
  public partial class Form4 : Form
   {
    Random Ricxvi = new Random(); // შრგ
    int[] MasDyn;  // ერთგანზომილებიანი მასივის გამოცხადება
    public Form4() { InitializeComponent(); }
private void button1_Click(object sender, EventArgs e)
  {  // „საწყისი მასივი“
    MasDyn = new int[5];   // მასივის საწყისი სიგრძე = 5
    int i;
    label5.Text = "";
    listBox1.Items.Clear();
    listBox2.Items.Clear();
    for (i = 0; i < MasDyn.Length; i++)
      { // მასივისთვის რიცხვების არჩევა შრგ-ით
        MasDyn[i] = Ricxvi.Next(1, 20);
        // listBox1-ში მასივის რიცხვების გამოტანა
        listBox1.Items.Add(MasDyn[i]);
        // listBox2-ში მასივის ინდექსების გამოტანა
        listBox2.Items.Add(i+1);                }
        label5.Text = (listBox1.Items.Count).ToString();
        label6.Text = (listBox2.Items.Count).ToString();
     }
private void button2_Click(object sender, EventArgs e)
 {  // „ხისტი დამატება“
   Array.Resize(ref MasDyn, 9); // მასივის ზომის შეცვლა
                                // (გახდა  9 ელემენტიანი)
  int i;
  listBox1.Items.Clear();
  listBox2.Items.Clear();
   for (i = 0; i < MasDyn.Length; i++)
    {
     listBox1.Items.Add(MasDyn[i]); // ემატება ბოლოში 0-ები
     listBox2.Items.Add(i + 1);  // მასივის ინდექსების ველი
    }
  label5.Text = (listBox1.Items.Count).ToString();
  label6.Text = (listBox2.Items.Count).ToString();
 }
private void button3_Click(object sender, EventArgs e)
 { //„დინამიური დამატება/დაკლება“
                                // ხელით აირჩევა მასივის სიგრძე
   Array.Resize(ref MasDyn, (int)numericUpDown1.Value);
```

```
    int i;
    listBox1.Items.Clear();
    listBox2.Items.Clear();
    for (i = 0; i < MasDyn.Length; i++)
      {
       listBox1.Items.Add(MasDyn[i]);
       listBox2.Items.Add(i);
      }
    label5.Text = (listBox1.Items.Count).ToString();
    label6.Text = (listBox2.Items.Count).ToString();
   }
  // მოვლენა: მასივის ელემენტების და ინდექსების სინქრონიზაცია
private void listBox1_SelectedIndexChanged(object
                                    sender, EventArgs e)
  {
   listBox2.SetSelected(listBox1.SelectedIndex, true);
  }
}}
```

როდესაც მასივის ზომა, ანუ ელემენტთა რაოდენობა დიდია, და სხვადასხვა listBox-ებში მოთავსებული მონაცემები (მაგალითად, მასივის ინდექსი და მასივის ელემენტი) „წაძრულია“ ერთმანეთისგან (არ ჩანს მასივის ელემენტი მერამდენეა), საჭიროა „სინქრონიზაციის“ პროცედურის შექმნა, ანუ შესაბამისი „მოვლენის“ დაპროგრამება (ნახ.7.12).

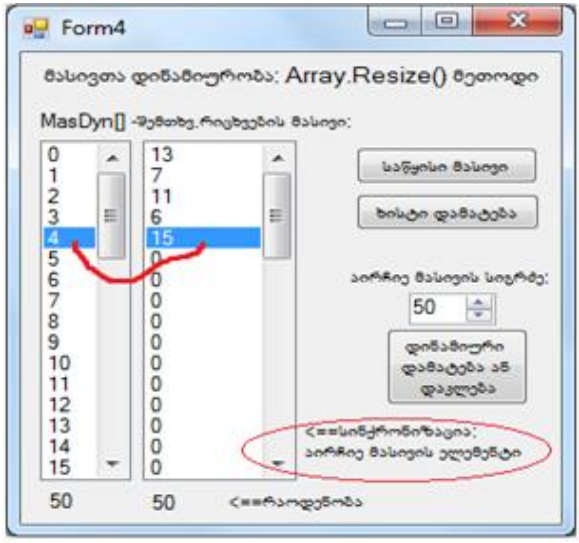


ნახ. 7.12-ა

ეს მოვლენა იქმნება 7.9_ლისტინგის ბოლო სტრიქონით:

```
listBox2.SetSelected(listBox1.SelectedIndex, true)
```

შესაძლებელია მასივის ელემენტებში მოძრაობა ამორჩევის მიზნით, რომელსაც ავტომატურად მოჰყვება შესაბამის ინდექსზე გადაადგილება. ამას ვუწოდებთ ჩვენ სინქრონიზაციას.



ნახ. 7.12-ბ

ასეთი ტიპის მოვლენების დაპროგრამება ხელსაყრელია დიდი მასივების შემთხვევაში, რათა უზრუნველყოფილ იქნას მისი ელემენტების ზუსტი კონტროლი და სხვა მიზნებისთვის გამოყენება.

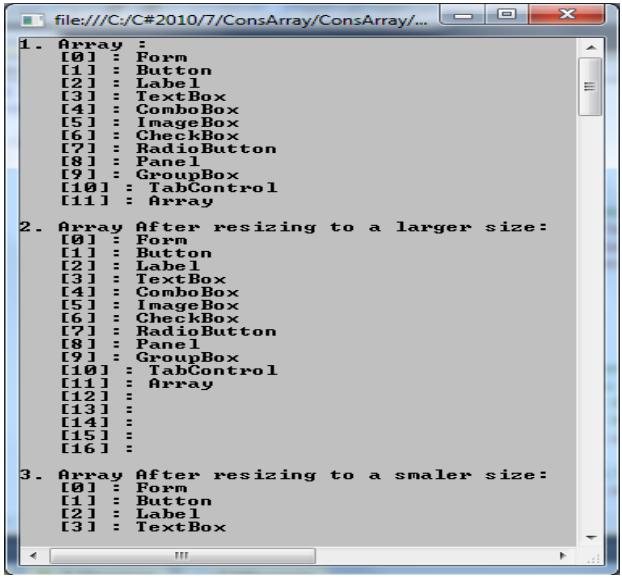
ამოცანა_7.7: ავაგოთ პროგრამის კოდები, რომლებიც კონსოლის რეჟიმში და ვინდოუსის ფორმის რეჟიმში შეასრულებს სტრიქონული ტიპის მასივებთან მუშაობას: მასივის გამოცხადება, ინიციალიზაცია, მასივის ზომების შეცვლა (ელემენტების დამატებით ან გამოკლებით), და ბოლოს ეკრანზე მათი ასახვით.

ა) კონსოლის რეჟიმში პროგრამის საწყისი ტექსტი მოცემულია 7_10 ლისტინგზე:

```
// ლისტინგი_7.10 -- სტრიქონულ მასივებთან მუშაობა: კონსოლის რეჟიმი -
using System;
namespace ConsArray
{
  class Program
    {
     public static void Main()
      {  //სტრიქონული მასივის შექმნა და ინიციალიზაცია
       String[] myArr = {"ბუტონი", "ლებელი", "ტექსტბოქსი",
                         "კომბობოქსი", "ჩეკბოქსი",
                         "რადიობუტონი", "პანელი",
                          "ტაბკონტროლი", "მასივი", "ფორმა"};
       Console.BackgroundColor = ConsoleColor.Gray; // ფონი
       Console.Clear(); // ფორმის გაწმენდა Gray-ფერით
       Console.ForegroundColor = ConsoleColor.Black; // ტექსტი
       // მასივის ელემენტების გამოტანა ეკრანზე
       Console.WriteLine("მასივის ელემენტთა მნიშვნელობები: ");
      PrintIndexAndValues(myArr);  // გამოცხადებულია მეთოდი,
                        // რომელიც  რეალიზებულია კოდის ბოლოში
      Array.Resize(ref myArr, myArr.Length + 5); // მასივის
                                          // გაზრდა 5 ელემენტით
      // Resize მეთოდით
       Console.WriteLine("After resizing to a larger size, ");

      // მასივის გამოტანა ეკრანზე ახალი მნიშვნელობებით
       Console.WriteLine("გაზრდილი მასივი: ");
       PrintIndexAndValues(myArr);
       Array.Resize(ref myArr, 4);  // მასივის ზომის შემცირება
       Console.WriteLine("შემცირებული მასივი: ");
       PrintIndexAndValues(myArr);

      // კონსოლის საწყისი ფერის აღდგენის მეთოდი
       Console.ResetColor();
       Console.ReadLine();
    }
  public static void PrintIndexAndValues(String[] myArr)
```

```
      { // მეთოდის აღწერა
        for (int i = 0; i < myArr.Length; i++)
          {
            Console.WriteLine("   [{0}] : {1}", i, myArr[i]);
          }
        Console.WriteLine();
      }
  }
}
```

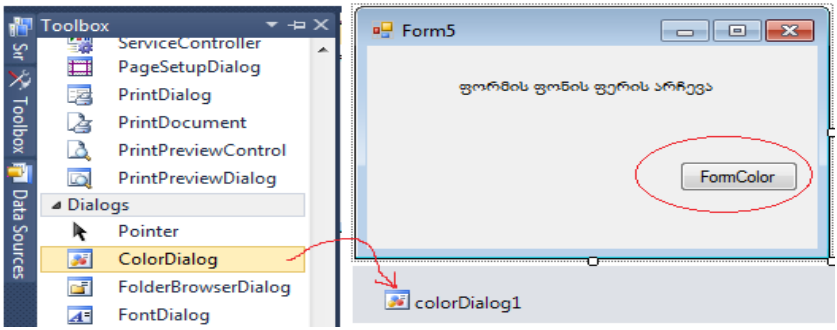
შედეგი მოცემულია 7.13 ნახაზზე.



ნახ. 7.13

ბ) WinForm რეჟიმში 7.7_ამოცანის პროგრამის კოდის აგება.

ფორმის ფონის ფერის დასაყენებლად ვიყენებთ Toolbox-ის Dialogs-ჯგუფიდან ColorDialog-კომპონენტს, რომელიც ფორმაზე გადმოტანის შემდეგ თავსდება Form5-ის ქვემოთ, კომპონენტების პანელზე: colorDialog1 (ნახ.7.14).

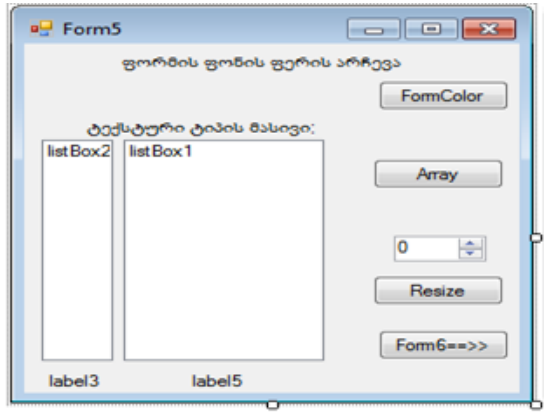


ნახ.7.14

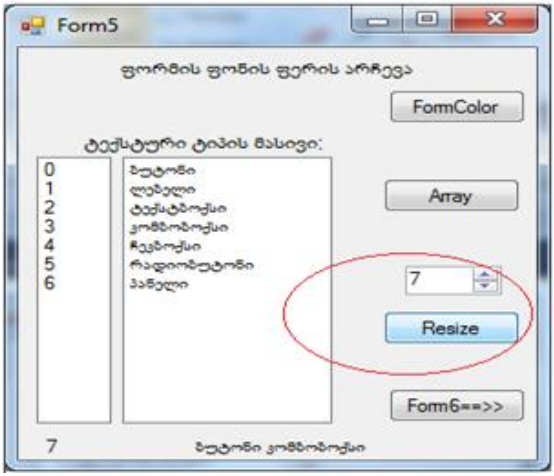
ფორმაზე ვდებთ ბუტონს - FormColor, რომლისთვისაც უნდა დაიწეროს მოვლენის დამუშავების კოდი (ლისტინგი_7.11).

```
// ლისტინგი_7.11 ---  ფორმის ფონის ფერის არჩევის დიალოგი -----
private void button1_Click(object sender, EventArgs e)
 { // ღილაკი FormColor
   if (colorDialog1.ShowDialog() == DialogResult.OK)
     {
       this.BackColor = colorDialog1.Color;
     }
 }
```

ავაწყოთ ახალი ფორმა ჩვენი ამოცანისათვის, რომლის დიზაინი ნაჩვენებია 7.15 ნახაზზე.

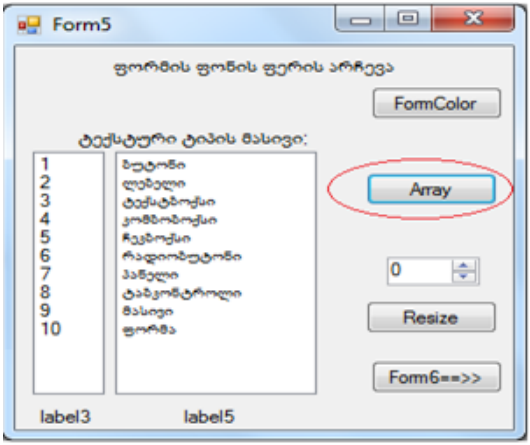


ნახ.7.15-ა

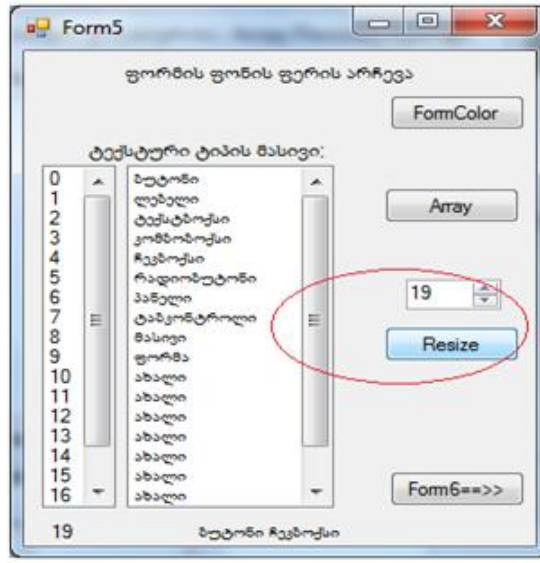


ნახ.7.15-ბ

თავიდან Array-ღილაკით listBox1-ში გამოიტანება ტექსტური myArray[] მასივის ელემენტები. listBox2-ში კი - მათი ინდექსები. შემდეგ numericUpDown1-მრიცხველში უნდა ავირჩიოთ რიცხვი: მასივის ახალი სიგრძე და Resize-ღილაკით ავამოქმედოთ მოვლენის დამმუშავებელი კოდი. შედეგად მიიღებთ 7.16 ნახაზზე ნაჩვენებ სურათს.



ნახ.7.16-ა



ნახ.7.16-ბ

7.16-ა ფორმაზე არჩეულია რიცხვი 19. ან მასივის ზომა 10-დან გაიზარდა +9 ელემენტით, ხოლო 7.16 ბ-ზე მითითებულია რიცხვი 7. მასივის ზომა შემცირდა 7-მდე. ასეთი დინამიკური ცვლილებისთვის პროგრამაში გამოიყენება კონსტრუქცია:

```
Array.Resize(ref myArray, (int)numericUpDown1.Value);
```

თუ მასივის ზომა იზრდება, მაშინ ცარიელ ელემენტებში ვწერთ სიტყვას „ახალი“, რათა სტრიქონული ტიპის ელემენტი არ იყოს ცარიელი და ჩვენც შევძლოთ სიტაუციის კონტროლი. ინდექსების ველის ქვეშ label3-ში გამოგვაქვს საკონტროლო ციფრი (მასივის ახალი სიგრძე). ხოლო label5-ში იბეჭდება მასივის ორი ელემენტის კონკატენაცია (0-ოვანი ელემენტი + რანდომფუნქციით არჩეული ელემენტი). კოდის ფრაგმენტი ასეთია:

```
if (myArray.Length != 0)
   label5.Text = myArray[0] + " " +
         myArray[a.Next(0, myArray.Length)];
```

რეალურ სიტუაციაში საჭიროა სიფრთხილე, ვინაიდან მასივის ზომის შემცირება იწვევს ელემენტების დაკარგვას აღდგენის გარეშე. ამ პრობლემის გადასაწყვეტად შემუშავებულ უნდა იქნას სპეციალური გამაფრთხილებელი შეტყობინებები ან ალგორითმი დაკარგული სტრიქონების აღდგენის მექანიზმით (იდეა: სანამ წაიშლება მასივის ზედმეტი ელემენტები, ისინი უნდა მოთავსდეს ახალ დროებით მასივში, რომელიც საჭიროების შემთხვევაში გამოყენებულ იქნება დაკარგული ელემენტების აღსადგენად).

ჩვენი ამოცანის სრული პროგრამული კოდი მოცემულია 7_12 ლისტინგში.

```
// ლისტინგი_7.12 --  string[] მასივი --------
using System;
using System.Drawing;
using System.Windows.Forms;
namespace WinArray
{
  public partial class Form5 : Form
   {
    Random a = new Random();
    String [] myArray = {"ბუტონი", "ლებელი",
                         "ტექსტბოქსი", "კომბობოქსი",
                         "ჩეკბოქსი", "რადიობუტონი",
                         "პანელი",  "ტაბკონტროლი",
                         "მასივი", "ფორმა"  };
     public Form5() { InitializeComponent();  }
     private void button1_Click(object sender, EventArgs e)
      {
       if (colorDialog1.ShowDialog() == DialogResult.OK)
        {
          this.BackColor = colorDialog1.Color;
        }
      }
    private void button2_Click(object sender, EventArgs e)
     {
      listBox1.Items.Clear();
```

```
        listBox2.Items.Clear();
        // მასივის ელემენტების გამოტანა ეკრანზე
        int i;
        for (i = 0; i < myArray.Length; i++)
         {
          if (myArray.Length != 0)
           {
            if (myArray[i] != null)
              listBox1.Items.Add(myArray[i]);
             else
              listBox1.Items.Add("ახალი");

              listBox2.Items.Add(i + 1);
             }
            }
          }

private void button3_Click(object sender, EventArgs e)
{  // მასივის ზომის ცვლილება
   Array.Resize(ref myArray, (int)numericUpDown1.Value);
   int i;
   label3.Text = "";
   label3.Text = myArray.Length.ToString();
   listBox1.Items.Clear();
   listBox2.Items.Clear();
   for (i = 0; i < myArray.Length; i++)
     {
       if (myArray[i] != null)
         listBox1.Items.Add(myArray[i]);
        else
         listBox1.Items.Add("ახალი");

       listBox2.Items.Add(i);
      }
    if (myArray.Length != 0)
    label5.Text = myArray[0] + " " + myArray[a.Next(0, myArray.Length)];
  }
}}
```

7.5. დამატება: ინტეგრირებული მოთხოვნების ენა LINQ

ჩვენ 7.1 პარაგრაფში დეტალური განხილვის გარეშე პროგრამაში შემოვიტანეთ სახელსივრცე “using System.Linq”. როგორც ახალი პროგრამული პროექტების აგებისას შევნიშნეთ, სისტემა საწყის კოდში ყოველთვის ათავსებს ამ სტრიქონს, მისი საყურადღებო მნიშვნელობის გამო. ამჯერად გავეცნოთ ამ ინსტრუმენტს უფრო დეტალურად.

LINQ (Language-Integrated Query) არის ინტეგრირებული მოთხოვნების ენა, რომელიც Ms Visual Studio .NET Framework_3.0-ის C# -ში (ვერსია 2008) შემოიტანეს და დიდი წარმატებითაც გამოიყენებენ, იგი, თავისი {from...wehre…select} კონსტრუქციით მოგვაგონებს SQL ენას. შეიძლება ითქვას, რომ იგი არის „ენაში ჩადგმული ენა“, საკმაოდ განვითარებული ინსტრუმენტული საშუალება. აქ ჩვენ შევეხებით მის ძირითად ასპექტებს და პრაქტიკულ მაგალითებს, რამეთუ სრულად ამ ენის შესაძლებლობების აღწერა ერთ პარტაგრფში წარმოუდგენელია, მისი დიდი მოცულობის გამო.

ინტეგრირებული მოთხოვნების ენის ფუნქციაა ინფორმაციის ამოღება მონაცემთა წყაროებიდან, პირველ რიგში მონაცემთა ბაზებიდან. მაგალითად, პროგრამული აპლიკაციისათვის ყოველთვის მნიშვნელოვანია მისგან დამოუკიდებლად არსებულ მონაცემთა ბაზებთან მუშაობა და ინფორმაციის წვდომა. ეს ბაზები შეიძლება იყოს უნივერსიტეტის სტუდენტების, ლექტორების და ლექციების შესახებ ინფორმაცია, ან პროდუქციის დიდი კატალოგები, მათი მწარმოებლების, ხარისხისა და ფასების, სავაჭრო ფირმების შესახებ და ა.შ. ასეთი ინფორმაცია დღეს უმეტესწილად განთავსებულია მონაცემთა რელაციურ ბაზებში და მათზე წვდომა, LINQ ენის გარეშე, შეიძლებოდა მხოლოდ SQL ენით ან XML ფორმატით. LINQ ენის

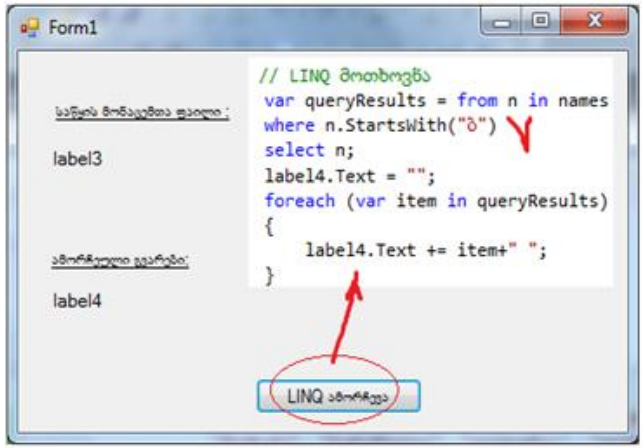
შემოტანამ C# ენაში ერთგვარად გაამარტივა და ეფექტური გახადა პროგრამული აპლიკაციებისა და მონაცემთა ბაზების ურთიერთობა.

განვიხილოთ მარტივი მაგალითი და მისი პროგრამული რეალიზაცია.

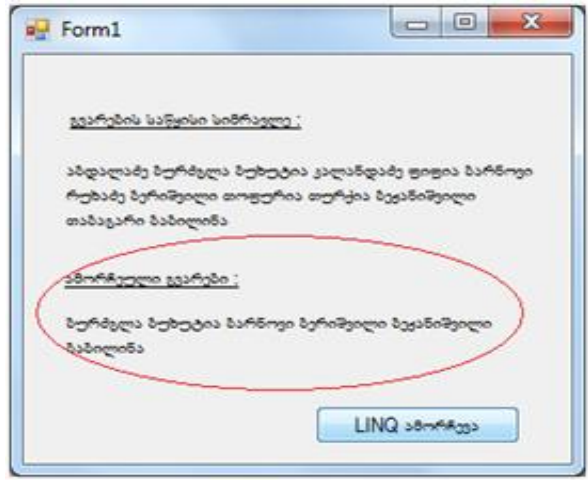
ამოცანა_7.8: მოცემულია გვარების მასივი. საჭიროა განხორციელდეს წვდომა პროგრამულად ამ მასივზე და ამორჩეულ იქნას გვარები მითითებული კრიტერიუმით. მაგალითად, შედეგში ამოვიღოთ ყველა გვარი, რომელიც იწყება ასოთი „ბ“ . პროგრამის ლისტინგი_7.13 აღწერს ამ კოდს, შედეგები კი მოცემულია 7.17 ნახაზზე.

```
// ლისტინგი_7.13 --- LINQ ------------------
using System;
using System.Drawing;
using System.Linq;    // !!!
using System.Windows.Forms;
namespace WinLinq
{
  public partial class Form1 : Form
   {
    public Form1() { InitializeComponent();  }
   private void button1_Click(object sender, EventArgs e)
    {  // საწყის მონაცემთა მასივი ----
     string[] names = {"აბდალაძე","ბურძგლა","ბუხუტია",
                       "კალანდაძე","ფიფია","ბარნოვი",
                       "რუხაძე", "ბერიშვილი",
                        "თოფურია", "თურქია",
                        "ბეჟანიშვილი", "თაბაგარი",
                                          "ბაბილინა" };
     label1.Text = "გვარების საწყისი სიმრავლე :";
     label3.Text = "";
    foreach (var item in names) // საწყისი მონაცემების გამოტანა
     {
      label3.Text+=item+" "; }
    var queryResults = from n in names  // LINQ მოთხოვნა
    where n.StartsWith("ბ")
```

```
select n;
label2.Text = "ამორჩეული გვარები :";
label4.Text = "";
// LINQ მოთხოვნის შესაბამისი მონაცემები ----
foreach (var item in queryResults)
  {
    label4.Text += item+" ";
  }
}}} // შდეგები ასახულია 7.17-ა,ბ ნახაზებზე.
```



ნახ.7.17-ა

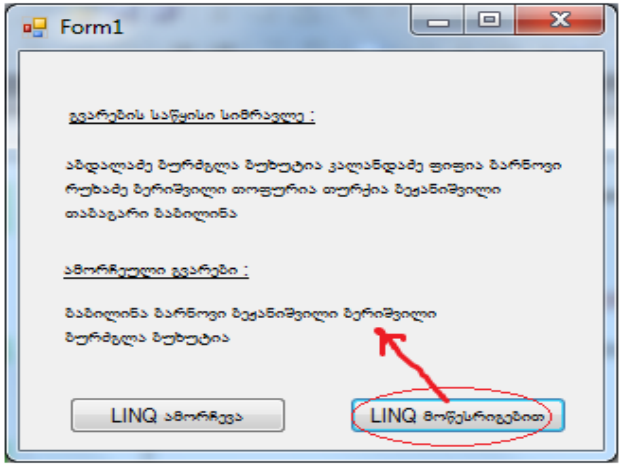


ნახ.7.17-ბ

იმისათვის, რომ შედეგში გვარები დალაგებული იყოს მოწესრიგებულად, საჭიროა LINQ მოთხოვნის სტრიქონის შეცვლა:

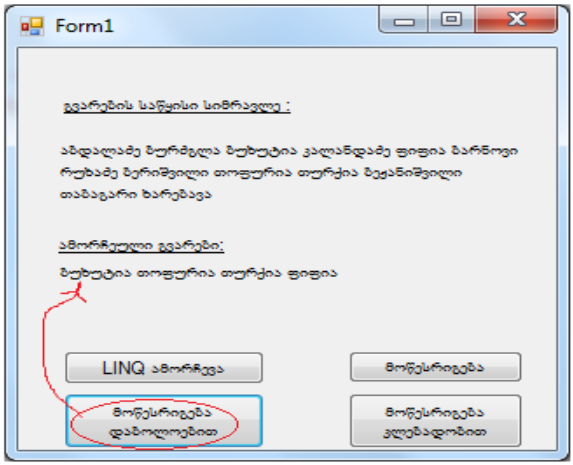
```
var queryResults = names.OrderBy(n => n).Where(n => n.StartsWith("ბ"));
```

შედეგს ექნება 7.18-ა ნახაზზე მოცემული სახე.



ნახ.7.18-ა

ახალი მოთხოვნა: ვიპოვოთ „ია“-ზე დაბოლოებული გვარები და დავალაგოთ მოწასრიგებით (ნახ.7.18-ბ).



ნახ.7.18-ბ

Linq მოთხოვნის შესაბამისი კოდი მოცემულია ქვემოთ.

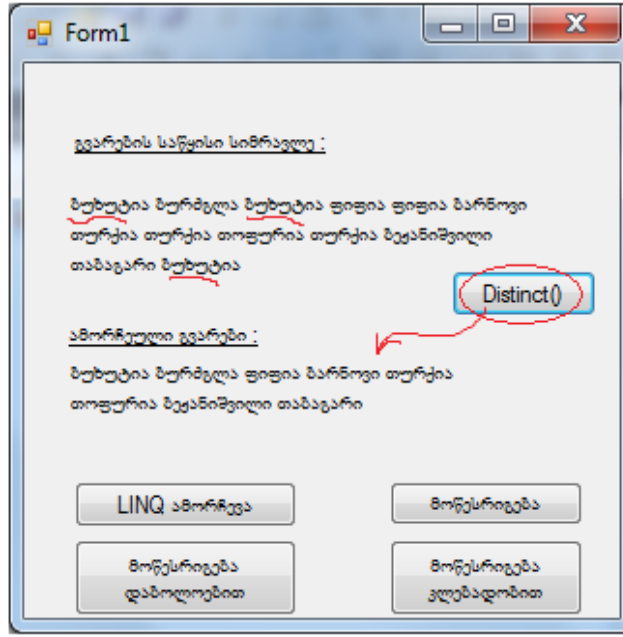
```
// ლისტინგი_7.14 ---
private void button4_Click(object sender, EventArgs e)
 {
  string[] names = { "აბდალაძე", "ბურძგლა", "ბუხუტია",
                     "კალანდაძე", "ფიფია", "ბარნოვი",
                     "რუხაძე", "ბერიშვილი", "თოფურია",
                     "თურქია", "ბეჟანიშვილი",
                     "თაბაგარი", "ხარებავა" };

  label1.Text = "გვარების საწყისი სიმრავლე :";
  label3.Text = "";

foreach (var item in names)
   {
     label3.Text += item + " ";
   }

 var queryResults = from n in names   // LINQ მოთხოვნა
                    where n.EndsWith("ია")
                    orderby n
                    select n;
 label4.Text = "";
 foreach (var item in queryResults)
      { label4.Text += item + " ";  }
  }
```

Select Distinct() მეთოდი გამოიყენება Linq-ში, რათა შესაძლებელი იყოს მონაცემთა კომპლექტიდან (განმეორებადი ელემენტებით) მარტივად მივიღოთ სიმრავლე (ნახ.7.19).

ამ შემთხვევისთვის Linq-მოთხოვნას ექნება შემდეგი სახე:

```
...
var queryResults =
  (from n in names select n).Distinct();
```

. . .



ნახ.7.19

7.6. კითხვები და სავარჯიშოები:

7.1. როგორ აღიწერება ერთ-, ორ- და სამგანზომილებიანი მასივები ?

7.2. როგორ ხდება მასივის საწყისი ელემენტების მინიჭება ?

7.3. რას ემსახურება Length თვისება, როგორ გამოიყენება იგი ?

7.4. რას წარმოადგენს Array კლასის Clone() მეთოდი და როგორ გამოიყენება იგი ?

7.5. რა არის შემთხვევით რიცხვთა გენერატორი და როგორ ხდება მისი ფუნქციონირება ?

7.6. დაწერეთ კოდი, რომელიც უზრუნველყოფს რიცხვთა სიმრავლის ფორმირებას.

7.7. დაწერეთ კოდი, რომელიც უზრუნველყოფს რიცხვთა ორი სიმრავლის შედარებას ტოლობაზე.

7.8. დაწერეთ კოდი, რომელიც უზრუნველყოფს რიცხვთა ორი სიმრავლის გაერთიანებას.

7.9. დაწერეთ კოდი, რომელიც უზრუნველყოფს რიცხვთა ორი სიმრავლის გადაკვეთის ქვესიმრავლის მიღებას.

7.10. დაწერეთ კოდი, რომელიც უზრუნველყოფს რიცხვთა ორი სიმრავლის სხვაობის ქვესიმრავლის მიღებას.

7.11. როგორ ხდება მასივში ელემენტების მოწესრიგებული დალაგება ?

7.12. როგორ ხდება მასივის ინდექსების გამოთვლა ?

7.13. ააგეთ კოდი, რომელიც (M x M) ორგანზომილებიან მასივის ელემენტებს შეავსებს ავტომატურად შემთხვევით რიცხვთა გენერატორით. გამოთვალე ამ მატრიცის სტრიქონების ელემენტთა ჯამი (ჰორიზონტალურად), სვეტების ელემენტთა ჯამი (ვერტიკალურად). მთავარ დიაგონალთა ელემენტების ნამრალი.

7.14. რას წარმოადგენს თავისუფალი (დაკბილული) მასივი, რა შემთხვევაში ხდება მათი გამოყენება ?

7.15. ავაგოთ კოდი, რომელშიც 5-სტრიქონიანი დაკბილული ნამდვილრიცხვა მასივის ელემენტები შეივსება შემთხვევითი რიცხვების გენერატორიდან. მაგალითად, double. ჩადგმულ მასივთა (სტრიქონთა) სიგრძეები იყოს: 10, 5, 2, 7,1. შედეგები გამოტანილ უნდა იქნეს label-ში.

7.16. ააგეთ კოდი, რომელიც (M x N) ორგანზომილებიან მასივის ელემენტებს შეავსებს მთელი რიცხვებით ავტომატურად შემთხვევით რიცხვთა გენერატორით და შემდეგ დაალაგებს სტრიქონებს პირველი სვეტის რიცხვთა კლებადობით.

8. კლასების შექმნა კონსოლის და ვინდოუს_ფორმების რეჟიმებში: პროცესების შედარება

ობიექტ-ორიენტირებული დაპროგრამების ერთ-ერთი ძირითადი თვისებაა ინკაფსულაცია, რომელიც კლასს განსაზღვრავს, როგორც მონაცემებისა და მეთოდების ერთობლიობას. როგორ შევქმნათ კლასები, მათი ობიექტები, მეთოდები და კლასთაშორის კავშირები ? როგორ შეიძლება მათი პროგრამული რეალიზაცია კონსოლისა და ვინდოუს_ფორმების რეჟიმებში ?

განვიხილოთ აღნიშნული საკითხები კონკრეტული ამოცანის საფუძველზე, მათი შემდგომი განზოგადების მიზნით.

ამოცანა_8.1: ავაგოთ ობიექტ-ორიენტირებული პროგრამის კოდი (კლასების და მეთოდების გამოყენებით) მაღლივ შენობაში ლიფტის გადაადგილების მოდელირებისათვის.

მაგალითად, შენობა N სართულიანია. მას აქვს ლიფტი. პერსონა, რომელიც შედის ლიფტში (ხდება მგზავრი), ირჩევს მისთვის საჭირო სართულს და მიემგზავრება (ზევით ან ქვევით). საჭიროა ამ პროცესის მოდელირება და დაპროგრამება ისე, რომ დაფიქსირდეს ლიფტის საწყისი მდგომარეობა, ამუშავების შემდეგ მისი საბოლოო მდგომარეობა, გავლილი სართულების რაოდენობა (ერთი ამუშავებისას). საბოლოოდ გამოიცეს რეპორტი, თუ სულ რამდენი სართული გაიარა ლიფტმა ერთი სეანსის (გარკვეული პერიოდის) განმავლობაში.

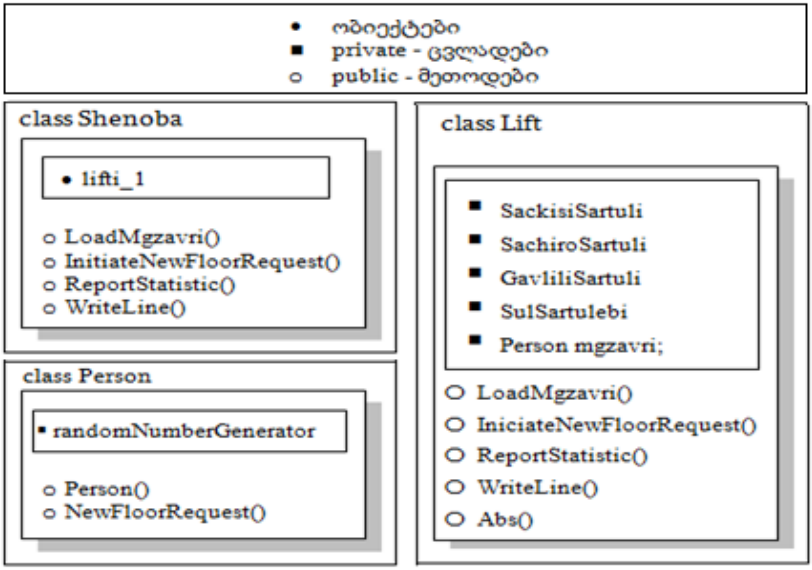
8.1. კლასთა მოდელი და მათი აგების პირობები:

ინკაფსულაციის სქემა მოცემულია 8.1 ნახაზზე.

- პროგრამაში სამი კლასია: `Shenoba`, `Lift` და `Person`;
- `Shenoba` კლასს აქვს `Lift` კლასის ერთი ობიექტი, სახელით `lifti_1` ;

- Person კლასის ერთი ობიექტი იმყოფება mgzavri - ცვლადის შიგნით, რომელიც იყენებს `lifti_1`;

- `Lift` კლასის ობიექტს შეუძლია გადაადგილება ნებისმიერ სართულზე, ინტერვალში [1,N=60].



ნახ.8.1

- პროგრამაში სართულის არჩევა ხდება მგზავრის მიერ (გამოიყენება „შემთხვევით რიცხვთა გენერატორი“ Random-ფუნქციით);

- `lifti_1` ლიფტი მოძრაობს საჭირო სართულისკენ, რაც აისახება კონსოლზე;

- მგზავრ(ებ)ის ლიფტით მოძრაობის სეანსი შედგება რამდენიმე ეტაპისგან, რომლბზეც შეირჩევა ახალი მიზნობრივი სართულები;

- სეანსის ბოლოს გამოიცემა ანგარიშის ტექსტი (რეპორტი), თუ ჯამში რამდენი სართული გაიარა ლიფტმა;

- შუალედური და საბოლოო შედეგები გამოიტანება ეკრანზე.

8.2. კლასთა კავშირების აღწერა UML ტექნოლოგიით:

მომხმარებლის სამი კლასი: Shenoba, Lift, Person და ერთი სისტემური კლასი System.Random ამოდელირებს ლიფტის მუშაობის პროცესს:

- Shenoba-კლასი შეიცავს Lift-ობიექტს და იძახებს მის მეთოდებს;

- Lift-ობიექტი იყენებს Person-ობიექტს, რათა მართოს ლიფტის მოძრაობა;

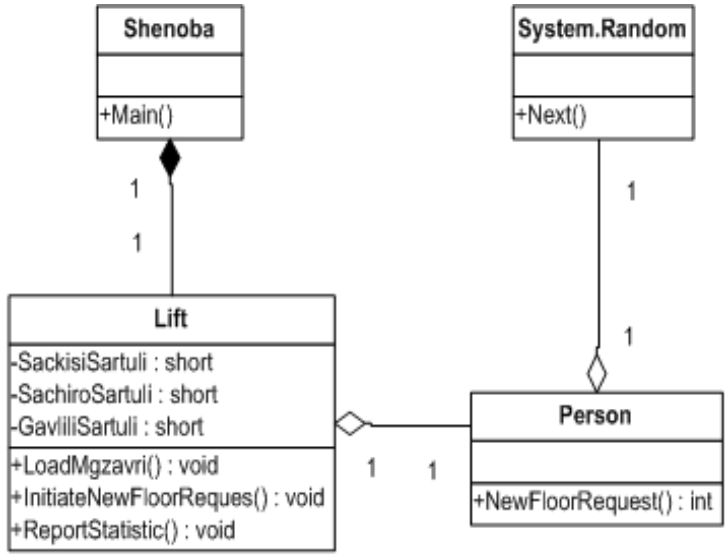
- Person-ობიექტი კი იყენებს System.Random-ობიექტს, რათა აირჩიოს საჭირო სართული შემდგომი გადაადგილებისთვის.

ობიექტ-ორიენტირებული პროგრამული სისტემების დაპროექტებისას კლასებმა მსგავსი ფუნქციები უნდა შეასრულოს. თითოეულს უნდა ჰქონდეს თავისი უნიკალური ფუნქციონალობა, და ყველა ერთად უნდა უზრუნველყოფდეს მთლიანი პროგრამის მუშაობას.

ნებისმიერი მაღლივი შენობა, როგორც „მთელი“ შედგება სხვადასხვა ნაწილებისგან: იატაკები, კედლები, ფანჯრები, ლიფტები და ა.შ. ამგვარად, შენობასა და ლიფტს შორის არსებობს დამოკიდებულება „მთელი-ნაწილი“ (აგრეგატული კავშირი UML ენაზე). ამიტომაცაა, რომ Lift-კლასის ობიექტის ცვლადი გამოცხადებულია Shenoba-კლასის შიგნით (ნახ.8.1, lifti_1). პროგრამულად იგი რეალიზებულია სტატიკური ცვლადის სახით (47-ე სტრიქონი, ლისტინგი_8.1): private static Lift lifti_1;

იგი ინახავს Lift კლასის ობიექტს და შეუძლია მისი მეთოდების გამოძახება. გარდა ამისა, Shenoba-კლასს შეიძლება ჰქონდეს აგრეთვე სხვა ცვლადებიც, რომლებიც აგრეგატულად დაქვემდებარებული კლასების ობიექტების ცვლადები იქნება.

8.2 ნახაზზე მოცემულია ჩვენი მაგალითის კლასებს შორის კავშირების UML დიაგრამა, აგებული Ms Visio პაკეტის გარემოში.

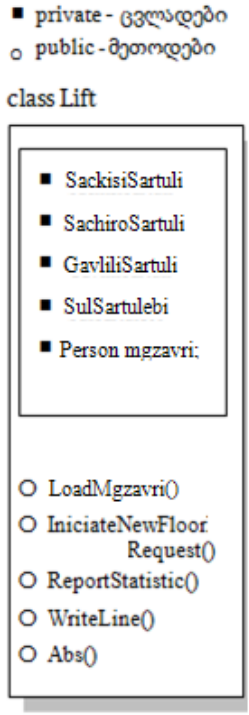


ნახ.8.2

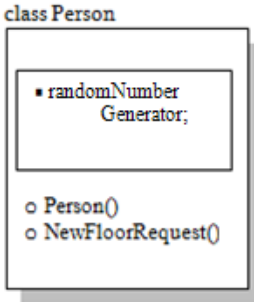
Shenoba და Lift კლასებს შორის აგრეგაციას უწოდებენ კომპოზიციას და იგი შავი რომბიკით გამოისახება. „1“-ები ნიშნავს, რომ 1 შენობაში არის 1 ლიფტი (ჩვენს შემთხვევაში). Lift და Person, აგრეთვე Person და System.Random კლასებს შორის აგრეგატული დამოკიდებულებაა, მაგრამ არა-კომპოზიციური. ის თეთრი რომბიკითაა ნაჩვენები. განსხვავება ისაა, რომ შენობას ლიფტი ყოველთვის აქვს. ლიფტში კი პერსონა შეიძლება იყოს, ან არ იყოს, ლიფტი ისეც მუშაობს.

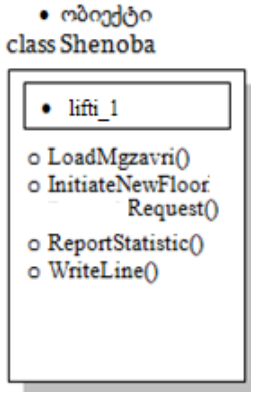
8.3. პროგრამული რეალიზაცია:

ცხრილში მოცემულია განხილული კლასების პროგრამული რეალიზაციის ლისტინგი.




```
1  using System;
2  namespace ConsoleApp_Lift
3  {
4   class Lift
5   {
6    private int SackisiSartuli = 1;
7    private int SachiroSartuli = 0;
8    private int GavliliSartuli = 0;
9    private int SulSartulebi = 0;
10   public  int i = 1;
11   private Person mgzavri;

12   public void LoadMgzavri()
13    {
14      mgzavri = new Person();
15    }
16  public void InitiateNewFloorRequest()
17  {
18    SachiroSartuli = mgzavri.NewFloorRequest();
19    GavliliSartuli = Math.Abs(SackisiSartuli -
                                  SachiroSartuli);
20    Console.WriteLine("{0,2}.| Sackisi: {1,2}|
        Sachiro: {2,2}| Gavlili: {3,2} ",  i.ToString(),
        SackisiSartuli.ToString(),
        SachiroSartuli.ToString(),
        GavliliSartuli.ToString());

         // Math.Abs(SackisiSartuli - SachiroSartuli);
21    SulSartulebi += GavliliSartuli;
22     SackisiSartuli = SachiroSartuli;
23     i++;
24   }
25
26   public void ReportStatistic()
27   {
28     Console.WriteLine("\n===>>> Sul gavlili
                          sartulebi: " + SulSartulebi);
29    }
```






```
30      }
31
32  class Person
33    {
        private System.Random
                              randomNumberGenerator;
34      public Person()   // კონსტრუქტორი
35      {
36        randomNumberGenerator = new
                                    System.Random();
37        }
38
39      public int NewFloorRequest()
40      {
41        // აბრუნებს არჩეულ შემთხვევით
          //  რიცხვს 1-60 დიაპაზონში
42        return randomNumberGenerator.Next(1,60);
43        }
44    }
45
46  class Shenoba   // კლასი შენობა
47    {
48    private static Lift lifti_1;
      private static void Main()
49    {
50      lifti_1 = new Lift();
51      lifti_1.LoadMgzavri();
        Console.WriteLine("      samgzavro sartulebi \n
52      ============================\n");
53      lifti_1.IniciateNewFloorRequest();
54      lifti_1.IniciateNewFloorRequest();
55      lifti_1.IniciateNewFloorRequest();
56      lifti_1.IniciateNewFloorRequest();
57      lifti_1.IniciateNewFloorRequest();
58      lifti_1.ReportStatistic();
59    }
60  }
}
```

ლისტინგი_8.1: კონსოლის რეჟიმი

8.4. პროგრამის ლისტინგის ანალიზი

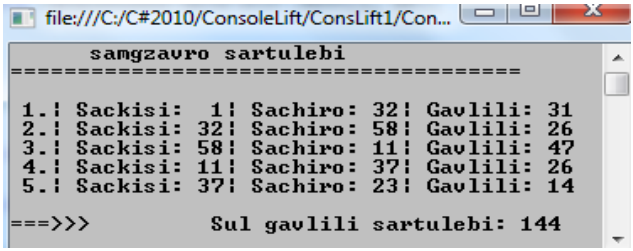
N	დანიშნულება
4	Lift - კლასის განსაზღვრების დასაწყისი
6	SackisiSartuli - ცვლადის გამოცხადება **int**-ტიპით, private-წვდომის სპეციფიკატორით და საწყისი მნიშვნელობით=1
11	Mgzavri - ცვლადის გამოცხადება, რომელიც შეიცავს Person-კლასის ობიექტს. Person- კლასი ასრულებს მგზავრის როლს Lift-კლასთან მიმართებაში
12	LoadMgzavri() - მეთოდის განსაზღვრების დასაწყისი. ის არის Lift-კლასის ინტერფეისის ნაწილი და გამოცხადებულია როგორც public
14	Person-კლასის ახალი (**new**) ობიექტის შექმნა. ეს ობიექტი მიენიჭება ცვლადს - mgzavri
16	InitiateNewFloorRequest() - მეთოდის განსაზღვრების დასაწყისი. ის არის Lift-კლასის ინტერფეისის ნაწილი და გამოცხადებულია როგორც public
18	mgzavri-ობიექტისთვის NewFloorRequest() მეთოდის გამოძახება. ამ მეთოდით დაბრუნებული მნიშვნელობა მიენიჭება ცვლადს SachiroSartuli
19	ერთი მგზავრობის შემდეგ იანგარიშება გავლილი სართულების რაოდენობა
20	ეკრანზე გამოიტანება: საწყისი_სართული, საჭირო_სართული, გავლილი_სართულების_რაოდენობა
21	იანგარიშება სულ გავლილი სართულების რაოდენობა ლიფტის მუშაობის მთელი სეანსის დროს
22	ლიფტის საწყისი სართულის მნიშვნელობას ენიჭება მისი ბოლო გაჩერების სართულის მნიშვნელობა
23	ლიფტის ამუშავების მომდევნო ბიჯის ინკრემენტი
26	ReportStatistic() მეთოდის გამოძახებით ეკრანზე გამოიტანება სტატისტიკა SulSartulebi ცვლადით
31	Person კლასის განსაზღვრების დასაწყისი

33	randomNumberGenerator ცვლადის გამოცხადება System.Random კლასის ობიექტის შესანახად
34	სპეციალური მეთოდის (კონსტრუქტორის !) განსაზღვრების დასაწყისი, რომელიც გამოიძახება ავტომატურად Person კლასის ობიექტის შექმნის დროს
36	System.Random-კლასის ახალი ობიექტის შექმნა და მისი მინიჭება randomNumberGenerator - ცვლადზე
39	int ტიპის NewFloorRequest() მეთოდის განსაზღვრა. ის Person-კლასის ინტერფეისის ნაწილია
42	პერსონა (ვირტუალური მგზავრი) ლიფტში ირჩევს საჭირო სართულს (შემთხვევით რიცხვთა გენერატორი ასრულებს ამ ფუნქციას) დიაპაზონში [1-60]
45	Shenoba კლასის აღწერა
47	Shenoba კლასში გამოცხადებულია Lifti ტიპის ცვლადი Lifti_1. Shenoba კლასი კომპოზიციურ კავშირშია Lift კლასთან (ნახ.8.2)
48	Main( ) მეთოდის აღწერის დასაწყისი
49	Lifti კლასის ახალი ობიექტის შექმნა და მისი მინიჭება lifti_1 ცვლადზე
50	lifti_1 ობიექტისთვის LoadMgzavri( ) მეთოდის გამოძახება
51-56	lifti_1 ობიექტისთვის .IniciateNewFloorRequest( ) მეთოდის გამოძახება 5-ჯერ
57	lifti_1 ობიექტისთვის . ReportStatistic() მეთოდის გამოძახება (შედეგების გამოსაცემად ეკრანზე)

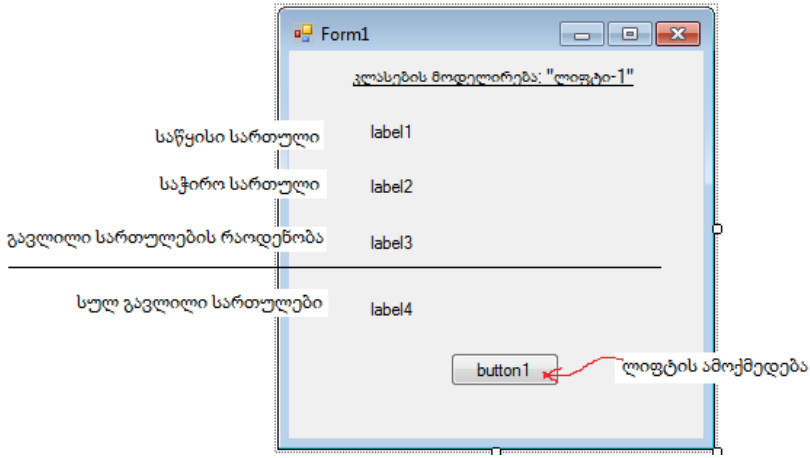
8.5. პროგრამის მუშაობის შედეგები:

8.3 ნახაზზე ნაჩვენებია განხილული პროგრამის შესრულების შედეგები კონსოლის რეჟიმში.

ნახ.8.3



ამოცანა_8.2: განხილული ამოცანისათვის ავაგოთ პროგრამული კოდი კლასების საფუძველზე ვინდოუსის ფორმის რეჟიმში. 8.4 ნახაზზე ნაჩვენებია სამუშაო ფანჯარა, რომელიც Form კლასის Form1 ობიექტია. მასზე განთავსებულია ოთხი label (1,2,3,4) და ერთი button1, რომლითაც მოდელირდება ლიფტის ამოქმედება (ლიფტის გამომახება, როცა პერსონა გარეთაა ან სართულის არჩევა ლიფტის ღილაკით, როცა პერსონა ლიფტშია, ანუ მგზავრია).

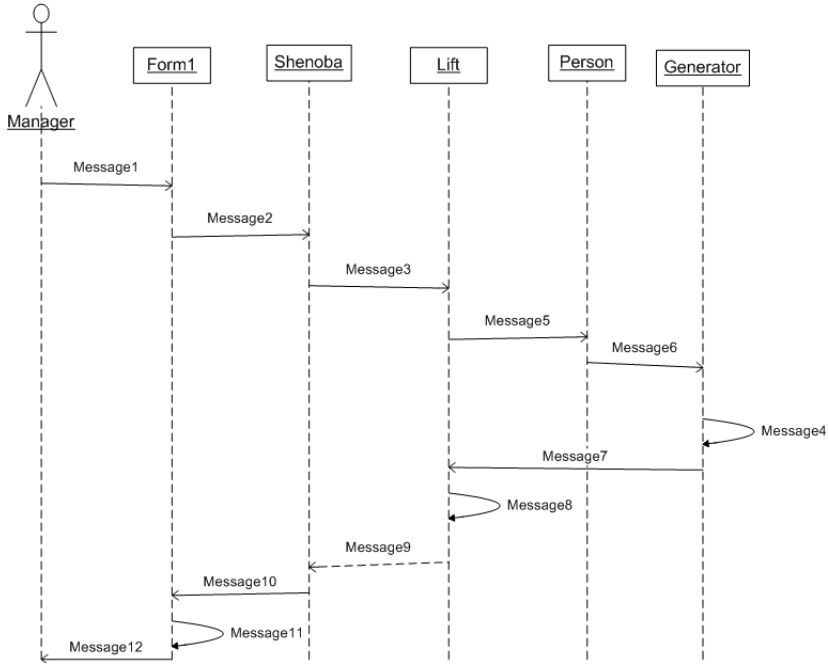


ნახ.8.4

```
// ლისტინგი_8.2--Form1  ელემენტებით და  button1-ის კოდი ---
using System;
using System.Drawing;
using System.Windows.Forms;
namespace WinFormLift
{
   public partial class Form1 : Form
```

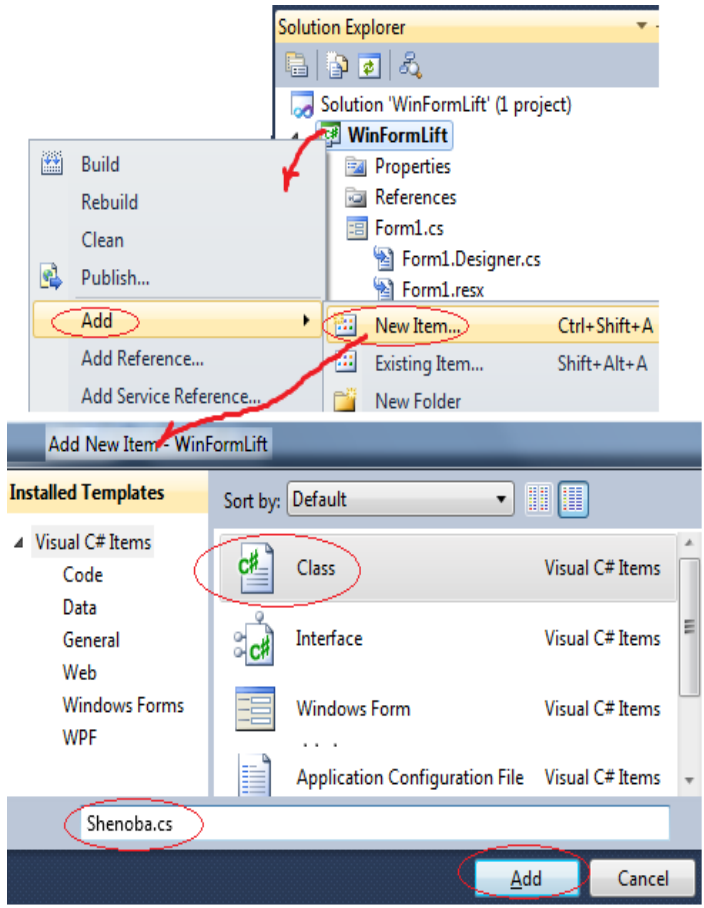
```
{
    Shenoba shenoba_1;   // კლასი Shenoba  უნდა შეიქმნას
    public Form1() { InitializeComponent();  }
    private void button1_Click(object sender, EventArgs e)
    {   // shenoba_1  ობიექტის შექმნა
        shenoba_1 = new Shenoba(label1, label2, label3, label4);
    }
}}
```

გარდა Form კლასისა (Form1 ობიექტით), რომელიც ასრულებს პროგრამის მომხმარებლის ინტერფეისის ფუნქციას, ლიფტის მუშაობის პროცესის ობიექტ-ორიენტირებული მოდელის ასაგებად საჭიროა კლასები: Shenoba, Lift და Person. ამ კლასების თვისებები და ფუნქციონალობა ჩვენ ზემოთ აღვწერეთ. ახლა საჭიროა ავაგოთ სცენარი „ლიფტის მუშაობა“, რომლისთვისაც გამოვიყენებთ UML-ის მიმდევრობითობის (Sequence) დიაგრამას.



ნახ.8.5

ახლა შევქმნათ დანარჩენი კლასები და აღვწეროთ მათი ფუნქციონალობები. Solution Explorer-ში დავამატოთ (Add new Items) ახალი კლასები, როგორც ეს 8.6 ნახაზზეა ნაჩვენები.



ნახ.8.6

Shenoba კლასის კოდი მოცემულია 8.3_ ლისტინგში.

```
// ლისტინგი_8.3 --- კლასი Shenoba --------
using System;
using System.Windows.Forms;
namespace WinFormLift
{
  public class Shenoba
   {
     static Lift lift_1 = new Lift();

   public Shenoba(Label label1, Label label2, Label
                                    label3, Label label4)
 {
  lift_1.LoadMgzavri();
  lift_1.InitiateNewFloorRequest(label1,label2, label3);
  lift_1.ReportStatistic(label4);
 }
}
}
```

Lift კლასის პროგრამული კოდი მოცემულია 8.4_ლისტინგში.

```
// ლისტინგი_8.4 --- კლასი Lift  --------
using System;
using System.Windows.Forms;

namespace WinFormLift
{
  public class Lift
   {
     private int SackisiSartuli = 1;
     private int SachiroSartuli = 0;
     private int GavlilSartulebi = 0;
     private int SulSartulebi = 0;
     public int i;
     private Person mgzavri;

   public void LoadMgzavri()
     {
       mgzavri = new Person();
     }
```

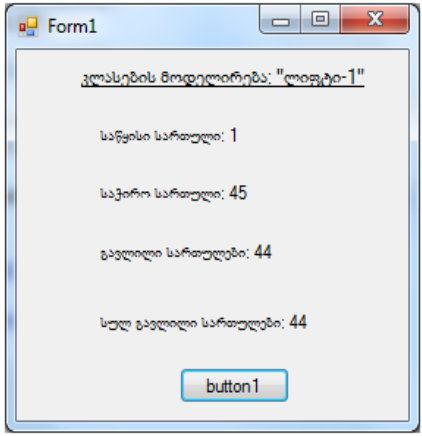
```
    public void InitiateNewFloorRequest(Label label1,
                              Label label2, Label label3)
     {
      SachiroSartuli = mgzavri.NewFloorRequest();
      GavlilSartulebi = Math.Abs(SackisiSartuli -
                                        SachiroSartuli);
      label1.Text = "საწყისი სართული: " +
                              SackisiSartuli.ToString();
      label2.Text = "საჭირო სართული: " +
                              SachiroSartuli.ToString();
     label3.Text = "გავლილი სართულები: " +
                             GavlilSartulebi.ToString();
      SulSartulebi += GavlilSartulebi;
      SackisiSartuli = SachiroSartuli;
      i++;
     }

  public void ReportStatistic(Label label4)
   {
    label4.Text = "სულ გავლილი სართულები: " +
                                              SulSartulebi;
   }
}}
```

Person კლასის პროგრამა მოცემულია 8.5_ლისტინგში.

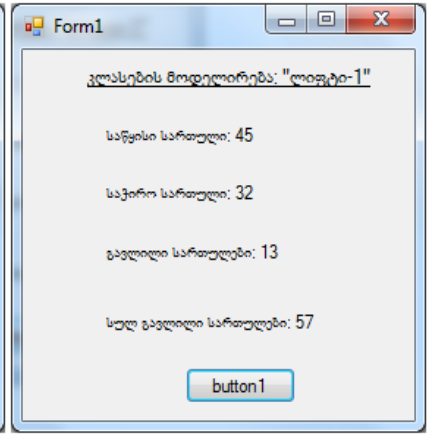
```
// ლისტინგი_8.5 --- კლასი Person  --------
using System;
using System.Windows.Forms;
namespace WinFormLift
{
  public class Person
  {
   private System.Random randomNumberGenerator;
   public Person()     // კონსტრუქტორი
    {
     randomNumberGenerator = new System.Random();
    }
  public int NewFloorRequest()
```

```
    { return randomNumberGenerator.Next(1, 60);
    }
}}
```

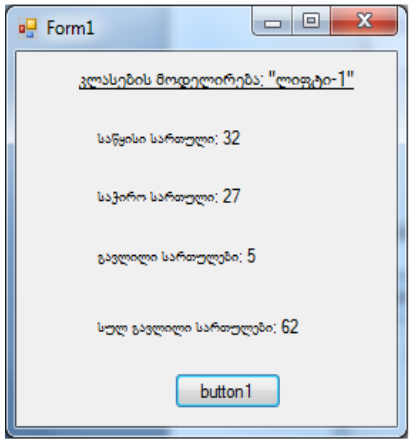
პროგრამის ამუშავების შემდეგ მიიღება 8.7 ნახაზზე ნაჩვენები შედეგები.



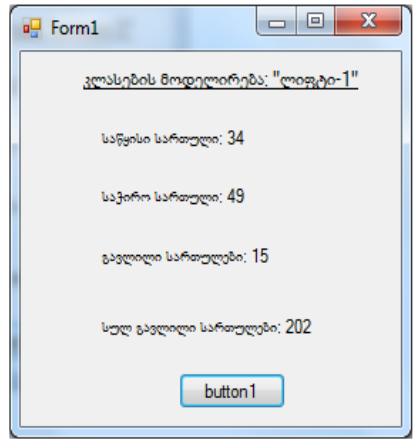
ნახ.8.7-ა. 1-ელი ბიჯი



ნახ.8.7-ბ. მე-2 ბიჯი



ნახ.8.7-გ. მე-3 ბიჯი

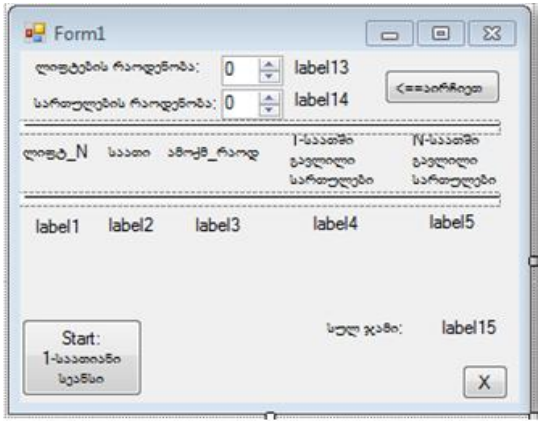


ნახ.8.7-დ. მე-10 ბიჯი,

და ა.შ.

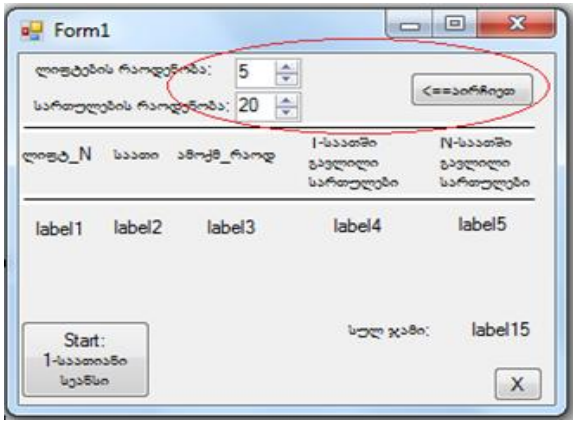
ამოცანა_8.3: ავაგოთ ლიფტების მომსახურების მენეჯერის C# პროგრამის კოდი, რომლითაც შესაძლებელი იქნება მრავალსადარბაზოიანი შენობის ლიფტების მუშაობის პროცესის მოდელირება. პროგრამაში წინასწარ დინამიკურად განისაზღვრება ლიფტების (Raod) და შენობის სართულების (Raod2) რაოდენობები.

პირობითად მივიღოთ, რომ ლიფტის მუშაობის ერთი სეანსი არის 1 საათი. 8.8 ნახაზზე ნაჩვენებია ფორმის მაკეტი, რომლის ცენტრში label(1-:-5) ელემენტებში შესაბამისად ჩაიწერება ლიფტის_N, მერამდენე საათია, რამდენჯერ ამოქმედდა ლიფტი 1-საათში, რამდენი სართული გაიარა ლიფტმა 1-სეანსში და, ბოლოს, სულ რამდენიმე სეანსის შემდეგ თითოეულმა ლიფტმა რამდენი სართული გაიარა ჯამში.

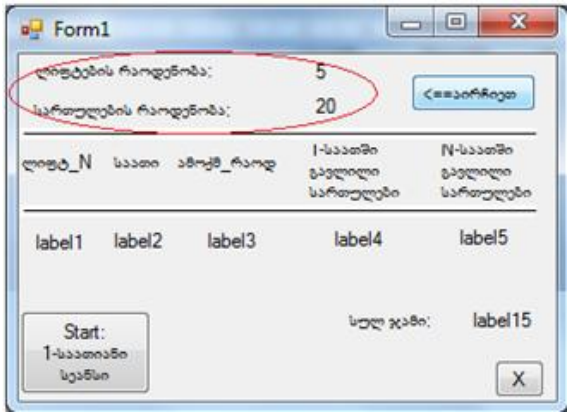


labe15-ში გამოიტანება სართულების ჯამური რაოდენობა ყველა ლიფტისთვის.

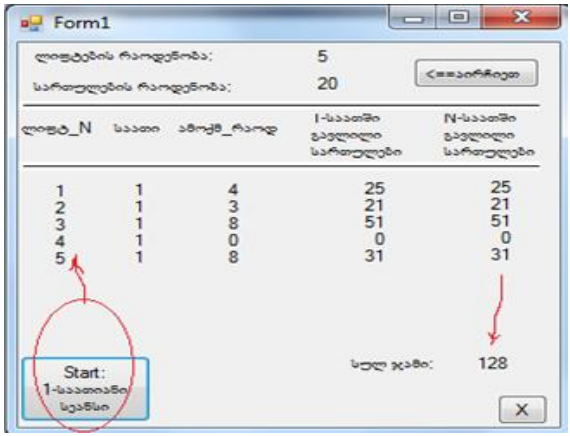
ნახ.8.8



თავიდან, ლიფტების და სართულების რაოდენობის არჩევის და „←აირჩიეთ“ ბუტონით მათი დაფიქსირების შემდეგ (ნახ.8.9-ა) მზადდება ცხრილის საწყისი მდგომარეობა. აქვე “Start”-ბუტონით ამოქმედდება პროგრამა და პირველ ბიჯზე (ნახ.8.9-ბ) მივიღებთ 1-ელ საათში ლიფტების ამოქმედების რაოდენობას (შემთხვევით რიცხვთა გენერატორით). ყველა ლიფტს ექნება განსხვავებული (შიძლება ერთნაირიც) ამოქმედების_რაოდენობა.

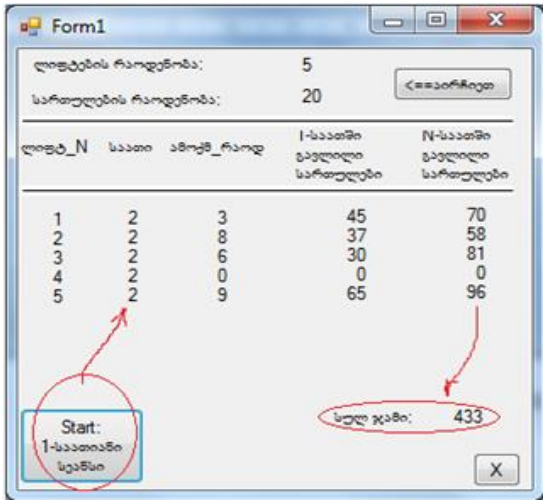


ნახ.8.9-ა. საწყისი მდგომარეობა

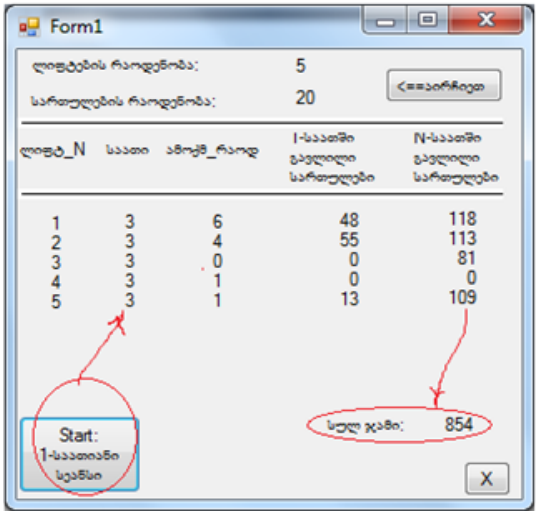


ნახ.8.9-ბ. 1-ელი ბიჯი

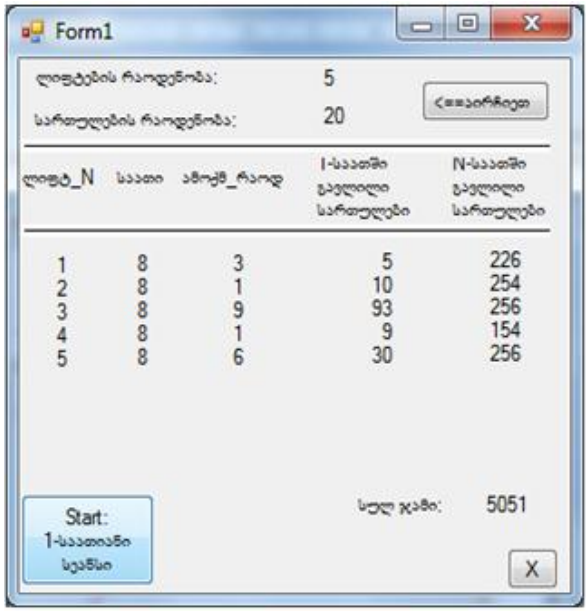
8.9-გ,დ,ე,ვ ნახაზებზე ნაჩვენებია ორ, სამ, რვა და 12 საათში თითოეული ლიფტის მიერ გავლილი სართულების ჯამური რაოდენობა, და აგრეთვე სულ ჯამი, ანუ ყველა ლიფტების მიერ გავლილი სართულების ჯამური მნიშვნელობა.



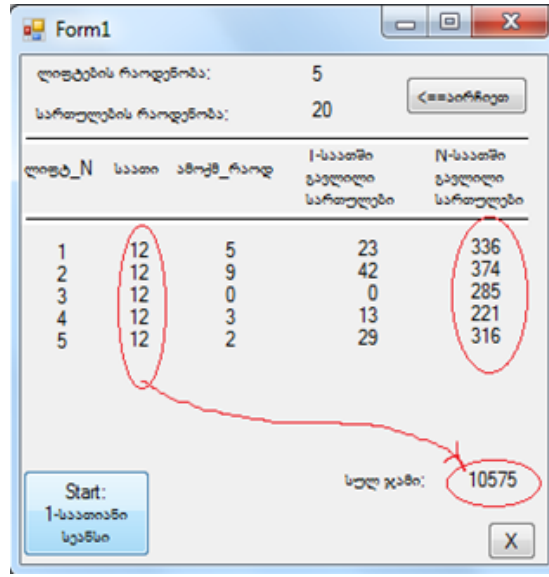
ნახ.8.9-გ. მე-2 ბიჯი



ნახ.8.9-დ. მე-3 ბიჯი



ნახ.8.9- ე.
მე-8 ბიჯი



ნახ.8.9-ვ. მე-12 ბიჯი

გარდა სტატისტიკური მონაცემების შეგროვებისა, ამოცანა შეიძლება გაფართოვდეს გარკვეული ანალიტიკური პროცედურების დამატებით (როგორია ლიფტების დატვირთვა საათების მიხედვით, სადარბაზოების მიხედვით და ა.შ.), ან ეკონომიკური ანაგარიშებით, მაგალითად, როგორია თითოეული ლიფტის დღიური, სადღეღამისო, ან თვიური ხარჯი და ა.შ.

ამოცანის გადაწყვეტის ერთ-ერთი ვარიანტის პროგრამული კოდი მოცემულია 8.6_ლისტინგში.

```
// ლისტინგი_8.6 ---
using System;
using System.Drawing;
using System.Windows.Forms;
namespace WinLiftN
{
  public partial class Form1 : Form
   {
    Random r1 = new Random();
    int i, j, AmoqmRaod=0, s=0; // s-სეანსების რაოდენობა
    int Raod, Raod2, ss=0; // ss- გავლილი სართულების
                           // ჯამური რაოდენობა
    int[] JamiSulSartulebi = new int[10];
    public Form1()
     {
      InitializeComponent();
      label13.Visible=false; // ეს label-ები თავიდან არ ჩანს
      label14.Visible = false;
     }
 private void button3_Click(object sender, EventArgs e)
  {  // სადარბაზოები=ლიფტ.რაოდ. არჩევა
    Raod = (int)numericUpDown1.Value;
    Raod2 = (int)numericUpDown2.Value; // შენობის
                                       // სართულების არჩევა
    numericUpDown1.Visible = false; //  არჩევის შემდეგ ეს
                                    //  მთვლელები ითიშება
    numericUpDown2.Visible = false; //  (რომ არ მოხდეს
                                    // სეანსის დროს მათი შეცვლა) და
    label13.Visible = true;  //  ჩაირთვება label-ები ლიფტებისა
                             // და სართულების
     label14.Visible = true;    //  არჩეული რაოდენობებით
```

```
    label13.Text = Raod.ToString();
    label14.Text = Raod2.ToString();
   }
private void button1_Click(object sender, EventArgs e)
 { // Start ბუტონის ამოქმედება
   int SackisiSartuli = 1;
   int SachiroSartuli = 0;
   int GavliliSartulebi = 0;
   int[] SulSartulebi = new int[Raod]; // ერთი სეანსის
                                       // დროს გავლილი სართულები
   s++; // სეანსის ნომერი
   label1.Text = "";
   label2.Text = "";
   label3.Text = "";
   label4.Text = "";
   label5.Text = "";
   for (i = 0; i < Raod; i++) // i-ური ლიფტი
    { // 1 საათში ლიფტის ამოქმედების რაოდენობა
      AmoqmRaod = r1.Next(0, 10);
      label1.Text += (i+1).ToString() + "\n";
      label2.Text += s.ToString() + "\n";
      label3.Text += AmoqmRaod.ToString() + "\n";
      for (j = 0; j < AmoqmRaod; j++)
       { // i-ური ლიფტის მიერ 1-საათში გავლილი
                    //სართულების რაოდენობის ანგარიში
         SachiroSartuli = r1.Next(1, Raod2);  // საჭირო
                              // სართულის არჩევა გენერატორით
         GavliliSartulebi = Math.Abs(SackisiSartuli -
                                     SachiroSartuli);
         SulSartulebi[i] += GavliliSartulebi;
         SackisiSartuli = SachiroSartuli;
       }
     // ერთი სეანსის დროს ყველა ლიფტის მიერ გავლილი
      // სართულების ჯამური რაოდენობა
    JamiSulSartulebi[i] += SulSartulebi[i];
    label4.Text += SulSartulebi[i] + "\n";
    label5.Text += JamiSulSartulebi[i] + "\n";
  }
//  ყველა სეანსის დროს ყველა ლიფტის მიერ გავლილი
// სართულების ჯამური რაოდენობა
   for (i = 0; i < Raod; i++)
```

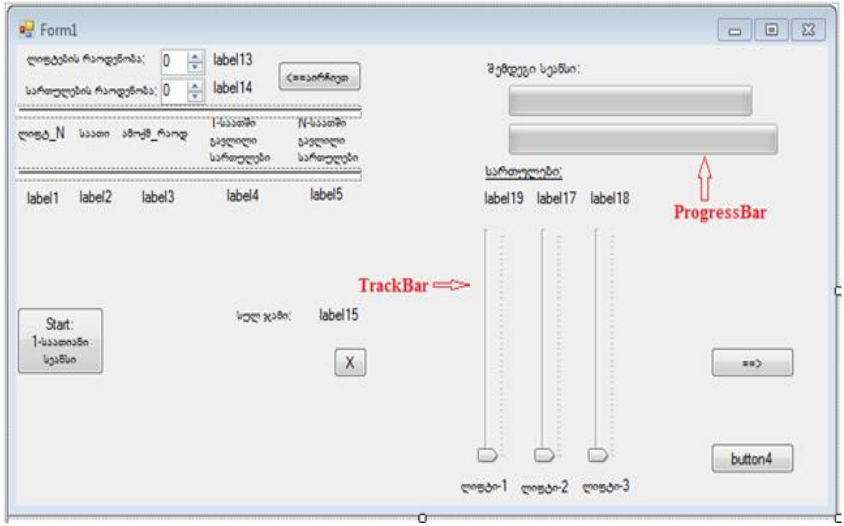
```
      { ss += JamiSulSartulebi[i];   }
     label14.Text = ss.ToString();
    }
private void button2_Click(object sender, EventArgs e)
  { Close(); }
 }
}
```

8.6. ვიზუალური კომპონენტები: TrackBar და ProgressBar

გავეცნოთ TrackBar და ProgressBar ელემენტების ფუნქციონალობას და მუშაობის პრინციპებს.

TrackBar ელემენტი იძლევა სტანდარტულ ზოლს (ბილიკს), რომლის გასწვრივაც სრიალებს პატარა ”სანიშნე” (ნახ.8.10). მისი დახმარებით შეიძლება სხვადასხვა ამოცანების გადაწყვეტა, მაგალითად, ზომების დაყენება და ა.შ.

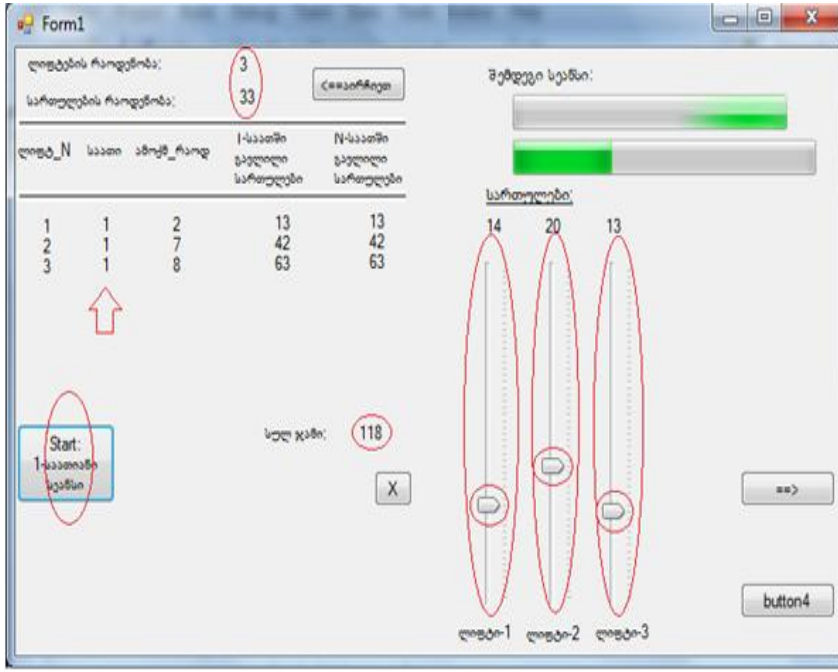
ProgressBar ელემენტის დანიშნულებაა რაიმე პროცესის ხანგრძლიობის მოდელირება, ან თვით პროგრამის ან მასში ციკლის მუშაობის ხანგრძლივობის განსაზღვრა.



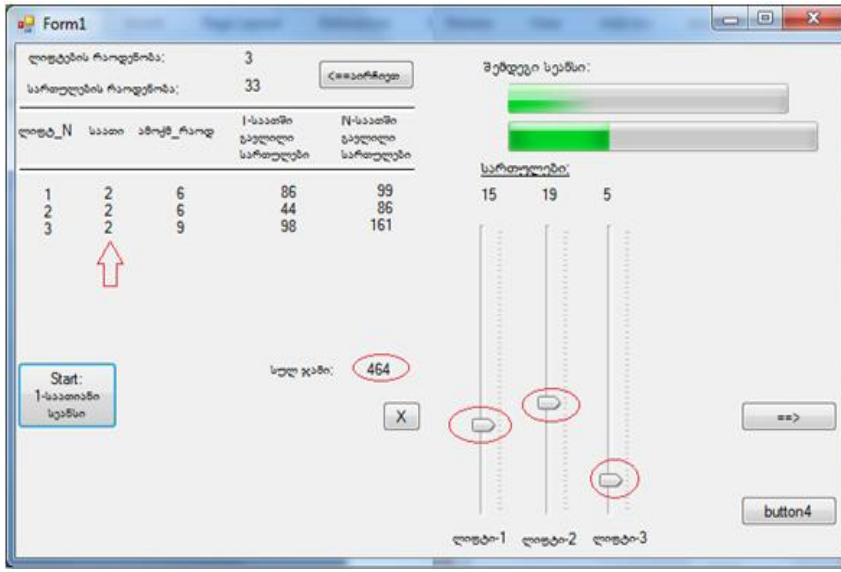
ნახ.8.10

ამოცანა_8.4: ჩვენ ზემოთ განვიხილეთ ამოცანა ლიფტების შესახებ. ახლა გამოვიყენოთ TrackBar და ProgressBar ელემენტები და ავაგოთ პროგრამა, რომელშიც TrackBar განასახიერებს ლიფტს, რომელიც მოძრაობს სართულებს შორის და ჩერდება რომელიმე სართულზე. ProgressBar ასახავს ლიფტების მუშაობის პროცესის ხანგრძლივობას 1 საათის განმავლობაში.

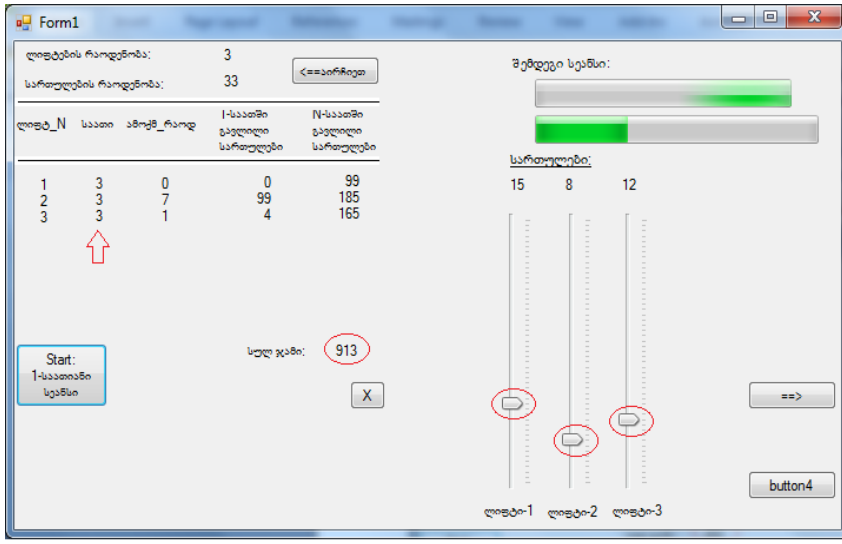
8.11 ნახაზზე მოცემულია მაგალითი 3 ლიფტის და 33 სართულის შემთხვევაში (ეს რაოდენობები პროგრამაში შეიძლება აირჩიოს მომხმარებელმა. მაშინ უნდა შეიცვალოს შესაბამისად TrackBar-ების რაოდენობა).



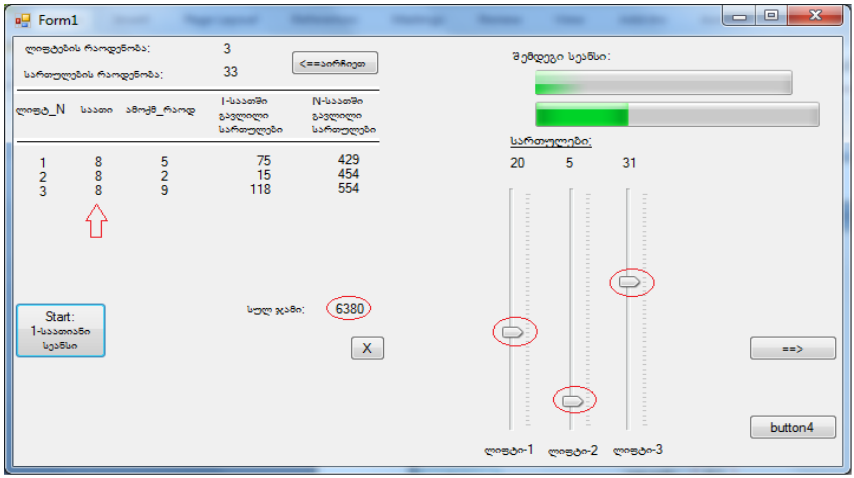
ნახ.8.11-ა. მდგომარეობა პირველი საათის შემდეგ



ნახ.8.11-ბ. მდგომარეობა მეორე საათის შემდეგ



ნახ.8.11-გ. მდგომარეობა მესამე საათის შემდეგ



ნახ.8.11-დ. მდგომარეობა მერვე საათის შემდეგ

პროგრამის კოდი ლიფტების მუშაობის პროცესის მოდელირების შესახებ მოცემულია 8_7 ლისტინგში.

```
// ლისტინგი_8.7 --- TrackBar, ProgressBar-----
using System;
using System.Drawing;
using System.Windows.Forms;
using System.Threading;
namespace WinLiftN
{
  public partial class Form1 : Form
   {
     Random r1 = new Random();
     int i, j, AmoqmRaod=0, s=0,ss=0;
     int Raod, Raod2;
    int[] JamiSulSartulebi = new int[10];
    public Form1()
     {
       InitializeComponent();
       label13.Visible=false;
       label14.Visible = false;
       trackBar1.Value = 10;
       label17.Text = trackBar2.Value.ToString();
       label18.Text = trackBar3.Value.ToString();
```

```
        label19.Text = trackBar4.Value.ToString();
       }
private void button3_Click(object sender, EventArgs e)
 { // სადარბაზოები=ლიფტ.რაოდ.
  Raod = (int)numericUpDown1.Value;
  Raod2 = (int)numericUpDown2.Value; //შენობის სართულები
  numericUpDown1.Visible = false;
  numericUpDown2.Visible = false;
  label13.Visible = true;
  label14.Visible = true;
  label13.Text = Raod.ToString();
  label14.Text = Raod2.ToString();
 }
private void button1_Click(object sender, EventArgs e)
 {
  int SackisiSartuli = 1;
  int SachiroSartuli = 0;
  int GavliliSartulebi = 0;
  int[] SulSartulebi = new int[Raod];
  s++;
  label1.Text = ""; label2.Text = ""; label3.Text = "";
  label4.Text = ""; label5.Text = "";
  progressBar2.Value = progressBar2.Minimum;
  progressBar2.PerformStep();
  for (i = 0; i < Raod; i++)
   { // 1 საათში ლიფტის ამოქმედების რაოდენობა
    AmoqmRaod = r1.Next(0, 10);
    label1.Text += (i+1).ToString() + "\n";
    label2.Text += s.ToString() + "\n";
    label3.Text += AmoqmRaod.ToString() + "\n";
    for (j = 0; j < AmoqmRaod; j++)
     { // საჭირო სართულის არჩევა გენერატორით
       SachiroSartuli = r1.Next(1, Raod2);
       GavliliSartulebi = Math.Abs(SackisiSartuli -
                                   SachiroSartuli);
       progressBar1.Enabled = true;
       progressBar1.Minimum = 0;
       progressBar1.Maximum = 100;
       progressBar1.Style = ProgressBarStyle.Marquee;
       progressBar1.MarqueeAnimationSpeed = i+1;
       // graf
       if (i == 1)
```

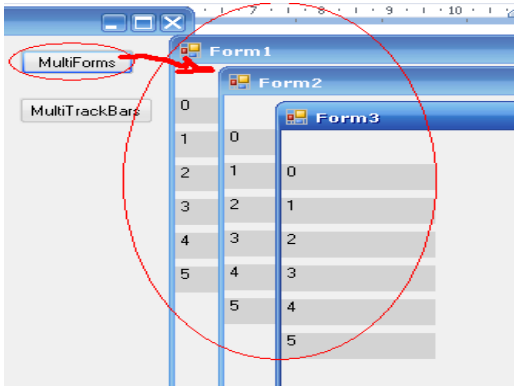
```
        {
         trackBar2.Value = SachiroSartuli;
         Thread.Sleep(500);
         label17.Text = trackBar2.Value.ToString();
        }
       else
        if (i == 2)
        {
         trackBar3.Value = SachiroSartuli;
         Thread.Sleep(500);
         label18.Text = trackBar3.Value.ToString();
        }
       else
        {
         trackBar4.Value = SachiroSartuli;
         Thread.Sleep(500);
         label19.Text = trackBar4.Value.ToString();
        }
       // -----
       SulSartulebi[i] += GavliliSartulebi;
       SackisiSartuli = SachiroSartuli;
      }
     JamiSulSartulebi[i] += SulSartulebi[i];
     label4.Text += SulSartulebi[i] + "\n";
     label5.Text += JamiSulSartulebi[i] + "\n";
    }
    for (i = 0; i < Raod; i++)
      {
       ss += JamiSulSartulebi[i];
       progressBar2.Minimum = 0;
       progressBar2.Maximum = 100;
       progressBar2.Value = progressBar2.Minimum;
progressBar2.Value = progressBar2.Value + 100 / Raod;
       System.Threading.Thread.Sleep(1000);
      }
     progressBar1.Enabled = false;
     label14.Text = ss.ToString();
   }
 private void button2_Click(object sender, EventArgs e)
  {   Close();  }
 private void button5_Click(object sender, EventArgs e)
  {
```

```
            Form2 f2 = new Form2();
            f2.Show();
        }
    }
}
```

8.7. ვიზუალური ელემენტების დინამიკური გამოყენება

პროგრამული აპლიკაციების შექმნისას ხშირად არაა წინასწარ ცნობილი, თუ რამდენი ესა თუ ის ვიზუალური ელემენტი იქნება საჭირო ფორმაზე განსათავსებლად. ეს საკითხი წყდება პროგრამის მუშაობის დროს და მომხმარებელი (ან შემთხვევით რიცხვთა გენერატორი) ირჩევს თვითონ ამ მნიშვნელობას. მაგალითად, ჩვენი „ლიფტების“ ამოცანაში ვიხილავდით სამ ცალ TrackBar ელემენტს (ანუ სახლი სამი ლიფტით). განვაზოგადოთ ამოცანა.

ამოცანა_8.4: საჭიროა აიგოს კოდი, რომელის ამუშავების შემდეგ განისაზღვრება ვიზუალური ელემენტების რაოდენობა და პროგრამულად მოხდება ამ ელემენტების განთავსება ფორმაზე. მაგალითად, საჭიროა 5, 7,10 ან სხვა რაოდენობა ლიფტებისა.



ნახ.8.12

აქ MultiForms ღილაკით მუშავდება მოვლენა, როცა საჭიროა დინამიკურად რამდენიმე Form-ის შექმნა, გახსნა და ამ ფორმაზე

ციკლურად რამდენე label ელემენტის განთავსება. ყოველი ახალი ფორმის Name ავტომატურად იქმნება „Form”+i.ToString() კონკატენაციით. 10 ლებელი კი ციკლურად იქმნება ყოველ ფორმაზე. პროგრამის ტექსტი მოცემულია 8.8_ლისტინგში.

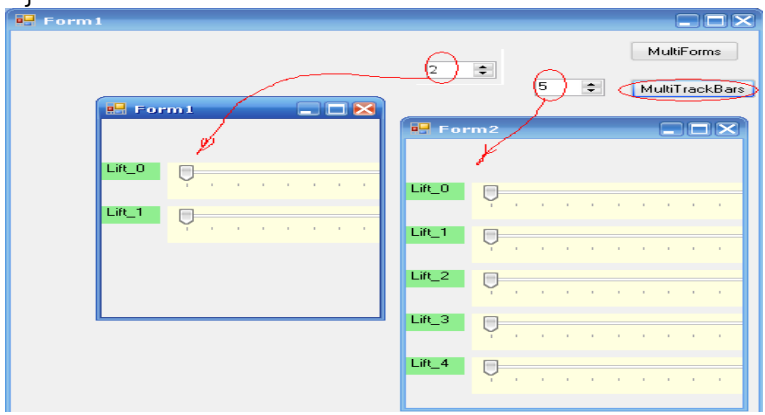
```
// ლისტინგი_8.8 – MultiForms ------
private void button2_Click_1(object sender, EventArgs e)
 {
   Form frm = new Form();
   frm.Show();
   frm.Height = 600; frm.Width = 500;
   array.Add(frm);
   frm.Name = "Form" + array.Count.ToString();
   frm.Text = frm.Name;
    int top = 0;
    for (int i = 0; i < 10; i++)
      {
       Label lbl = new Label();
        lbl.BackColor = Color.LightGray;
        lbl.ForeColor = Color.Black;
        lbl.Text = i.ToString();
        top = top + 30;
        lbl.Top = top;
        frm.Controls.Add(lbl);
       }
      frm.Text = frm.Name;
   }
```

MultiTrackBar ბუტონით უნდა დამუშავდეს მოვლენა, რომელიც ფორმაზე გამოიტანს TrackBar -ების საჭირო რაოდენობას.

8.9_ლისტინგზე მოცემულია კოდის ტექსტი, ხოლო შედეგები ასახულია 8.13 ნახაზზე.

```
  // ლისტინგი_8.9 –MultiTrackBar ------------------
  private void button3_Click(object sender, EventArgs e)
   {
     int Raod;
     Raod = (int)numericUpDown1.Value; // Raod-დინამიკურად
                                       //  განსაზღვრა
     Form frm = new Form();
     frm.Show();
     frm.Height = 500;
     frm.Width = 400;
```

```
array.Add(frm);
frm.Name = "Form" + array.Count.ToString();
// Form-ებისთვის
frm.Text = frm.Name;
int top = 0;
for (int i = 0; i < Raod; i++)
  {
    Label lbl = new Label();
   lbl.BackColor = Color.Green;
   lbl.ForeColor = Color.White;
   lbl.Size = new Size(40,20);
   lbl.Location = new Point(0, 200);
   lbl.Text = "Lift_" + i.ToString(); // Label-ებისთვის
   top += 50;
   lbl.Top = top;
   frm.Controls.Add(lbl);
   TrackBar trb = new TrackBar();  // TrackBar-ებისთვის
     trb.BackColor = Color.Red;
     trb.Size = new Size(200, 45);
     trb.Location = new Point(45, 200);
     trb.Text = i.ToString();
     //top += 50;
     trb.Top = top;
     frm.Controls.Add(trb);
   }
  frm.Text = frm.Name;
}
```



ნახ.8.13

8.8 კითხვები და სავარჯიშოები:

8.1. როგორ გესმით კლასის ინკაფსულაციის თვისება ?

8.2. როგორ ხდება გამოყენებით კვლევის სფეროსთვის კლასთა მოდელის აგება უნიფიცირებული მოდელირების ენის საფუძველზე ?

8.3. ააგეთ უნივერსიტეტის სასწავლო პროცესის (”მეცადინეობა”, ”გამოცდა” ან სხვ.) კლასთა მოდელი. განსაზღვრეთ ობიექტები, თვისებები, მეთოდები და მოვლენები.

8.4. რას წარმოადგენს კლასთა კავშირები და როგორ ხდება მათი მოდელირება ?

8.5. კლასთა როგორ კავშირებს უწოდებენ ”აგრეგაციას” და ”კომპოზიციას”, რა განსხვავებაა მათ შორის ?

8.6. ააგეთ უნივერსიტეტისთვის ”მეცადინეობა” პროცესის სცენარი, რომელიც UML-ის მიმდევრობითობის დიაგრამით გამოისახება.

8.7. ააგეთ უნივერსიტეტისთვის ”გამოცდა” პროცესის სცენარი, რომელიც UML-ის მიმდევრობითობის დიაგრამით გამოისახება.

8.8. როგორ ხდება კლასის პროგრამული რეალიზაცია C# კოდის მაგალითზე ?

8.9. როგორ ხდება პროგრამის ლისტინგის (საწყისი ტექსტის) ანალიზი ?

8.10. როგორ ხდება პროგრამაში ახალი კლასის შექმნა ვიზუალური საშუალებების გამოყენებით ?

8.11. რას წარმოადგენს TrackBar ელემენტი და როგორ გამოიყენება იგი პროგრამაში პროცესების მოდელირებისათვის ?

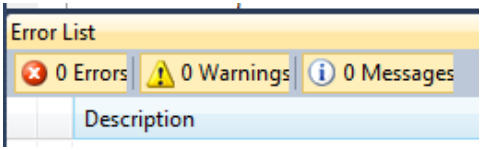
8.12. რას წარმოადგენს ProgressBar ელემენტი და როგორ გამოიყენება იგი პროგრამაში პროცესების მოდელირებისათვის ?

8.13. როგორ ხდება პროგრამულად პროცესის დაყოვნების რეალიზაცია ?

9. პროგრამული შეცდომების აღმოჩენის და გამორიცხვის ვიზუალური საშუალებები

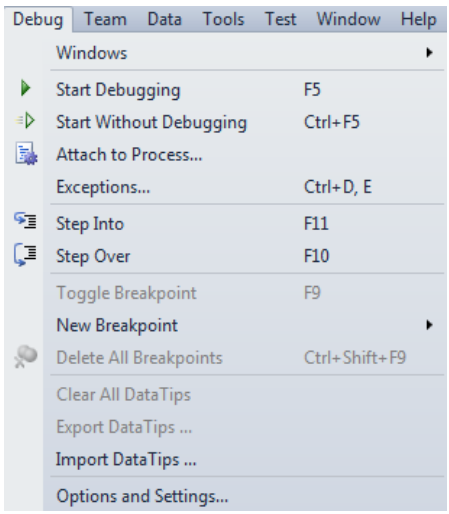
როგორც ცნობილია, პროგრამის გამართვის და ტესტირების (შესრულების) პროცესში ადგილი აქვს პროგრამული შეცდომების გამოვლენას. ეს შცდომები სამი სახისაა: სინტაქსური, პროცედურული და ლოგიკური.

სინტაქსური შეცდომების აღმოჩენა ხდება C#-ენის კომპილატორის საშუალებით და გამოიტანება პროგრამული ტექსტების რედაქტორის ქვედა ნაწილში, Error List ფანჯარაში (ნახ.9.1). მათი პოვნა და შესწორება შედარებით ადვილია.



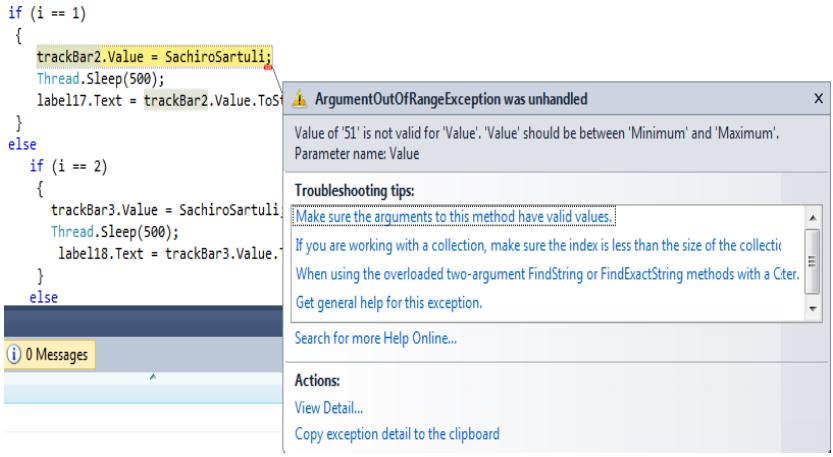
ნახ.9.1

პროცედურული შეცდომები ვლინდება პროგრამის შესრულების პროცესში. როცა პროგრამაში აღარაა სინტაქსური შეცდომები და ხდება მისი ამუშავება: Start Debugging ან F5 (ნახ.9.2), შესაძლებელია ისეთი შეცდომის გამოვლენა, რომელიც წყვეტს პროგრამის შესრულების პროცესს (ანუ სრულდება ავარიულად).



ნახ.9.2

დებაგერს გამოაქვს ამ დროს გარკვეული შეტყობინება (ნახ.9.3), რომელიც მოითხოვს პროგრამისტის მხრიდან ანალიზს და შეცდომის გამორიცხვას (Exception Handling).

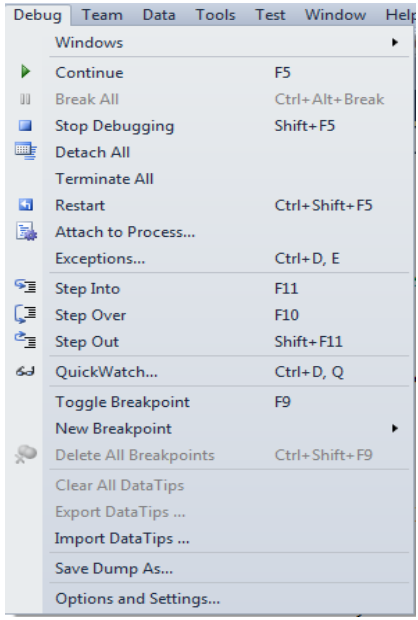


ნახ.9.3

ლოგიკური შეცდომების აღმოჩენა შედარებით რთულია. პროგრამა ამ დროს მუშაობს და სრულდება ნორმალურად (არაავარიულად), მაგრამ შედეგები „საეჭვოა“.

ტესტირების პროცესში, რომელიც აუცილებლად მოსდევს პროგრამის გამართვას, საჭიროა ასეთი „კვლევის“ ჩატარება, რათა გამოვლენილ იქნას მოსალოდნელი, ფარული ლოგიკური შეცდომები.

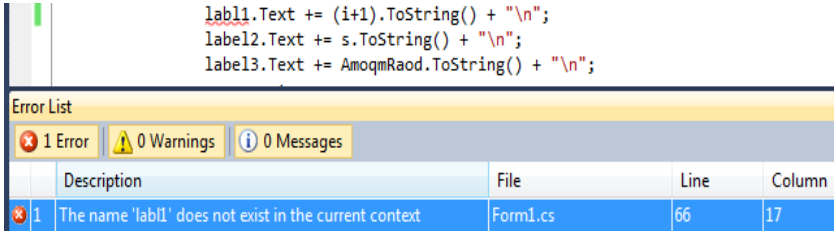
C# ენის რედაქტორს აქვს კარგი ინსტრუმენტული საშუალებები (Debugger) ამ ტიპის შეცდომების მოსაძებნად და აღმოსაფხვრელად (ნახ.9.4).



ნახ.9.4

9.1. სინტაქსური შეცდომების აღმოფხვრის საშუალებანი

თუ პროგრამის არაკორექტულ კოდში შეცდომითაა ჩაწერილი ენის ოპერატორი (მაგალითად, “Whail” ნაცვლად while - ისა) ან კონსტრუქცია (მაგალითად, “case: “, რომელსაც არ უძღვის წინ switch() {…}) და ა.შ. როგორც ზემოთ აღვნიშნეთ, ამ დროს კომპილატორი მიუთითებს არსებულ შეცდომას და მის ადგილმდებარეობას (ნახ.9.5).



ნახ.9.5

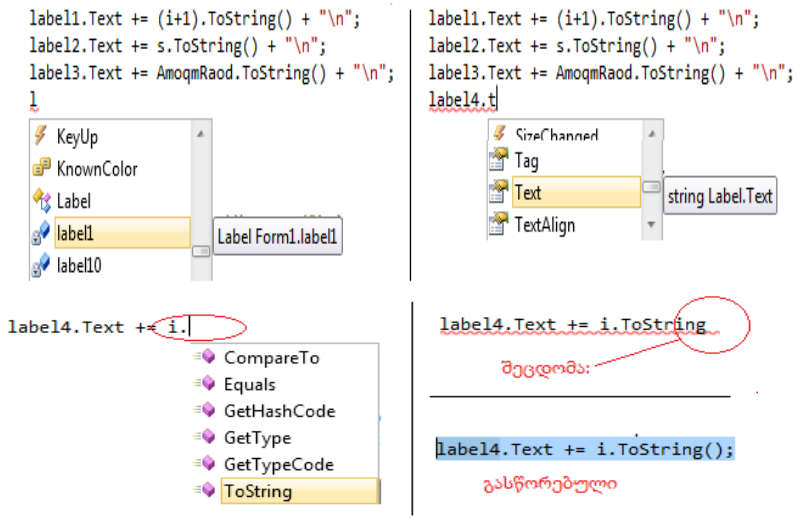
სინტაქსური შეცდომები თუ არ გასწორდა, მაშინ ვერ მოხერხდება პროგრამის ობიექტური (.obj) და შესრულებადი (.exe) კოდების ფორმირება.

სინტაქსური შეცდომების თავიდან ასაცილებლად C# ენის რედაქტორს აქვს სხვადასხვა ვიზუალური დამხმარე საშუალებები. მაგალითად, პროგრამაში ოპერატორების ტექსტის შეტანისას, ან ობიექტის მეთოდის და მოვლენის არჩევისას (წერტილის „.“ დასმისას) ხდება ვიზუალური ბლოკის (Intellisense - ავტომდამატება) შემოთავაზება (ნახ.9.6), საიდანაც ამოირჩევა საჭირო სიტყვა და Enter-კლავიშით სწრაფად ჩაჯდება მითითებულ ადგილას.

ეს გამორიცხავს როგორც ოპერატორის (ობიექტის თვისების, მეთოდის და ა.შ.) არასწორ სინტაქსურ ჩაწერას, ასევე არარელევანტური სიტყვის მითითებას (სიტყვა, რომელიც აქ „უადგილოა“).

შესაძლებლია აგრეთვე კოდში „გახსნილ-დასახურ“ ფრჩხილების რაოდენობის კონტროლი, რაც ძალზე ხშირი შეცდომების წყაროა. მთლიანად, შეიძლება ითქვას, რომ ენის

ასეთი ვიზუალური კონტროლის და დამხმარე საშუალებები ეფექტურს ხდის პროგრამისტის მუშაობას.

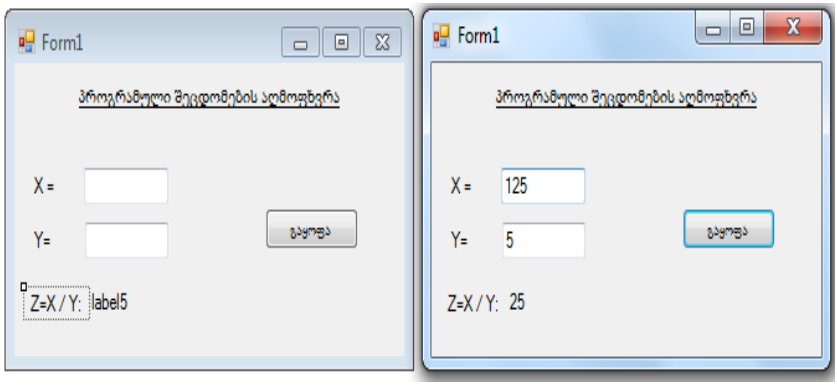


ნახ.9.6

9.2 განსაკუთრებული შემთხვევები: შეცდომები პროგრამის შესრულებისას და მათი აღმოფხვრა

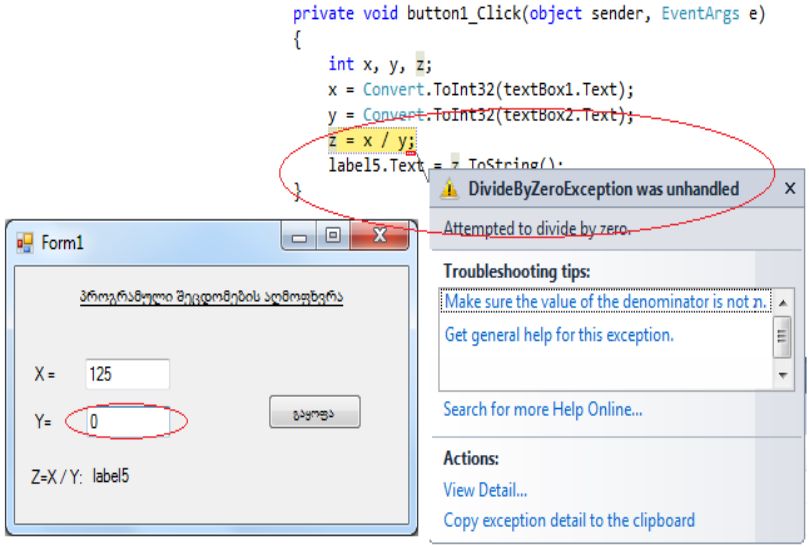
თუ პროგრამის ტექსტი სინტაქსური შეცდომებისგან თავისუფალია, ის შეიძლება ამუშავდეს შესრულებაზე. ამ დროს შესაძლებელია ისეთი შეცდომების გამოვლენა, რომლებიც პროგრამას ავარიულად დაასრულებს, ან საერთოდ არ დაასრულებს („გაჭედავს“). მაგალითად, უსასრულო ციკლი ან სხვ. განსაკუთრებული შემთხვევა. განვიხილოთ ასეთი მაგალითები:

ამოცანა_9.1: ავაგოთ პროგრამის კოდი, რომელიც შეასრულებს მთელი რიცხვების შეტანას და გაყოფის ოპერაციას. 9.7 ნახაზზე ნაჩვენებია ფორმა ორი ტექსტბოქსით (რიცხვების შესატანად), label5 შედეგის გამოსატანად და ღილაკი „გაყოფა“ პროგრამული კოდით (ნახ.9.8).



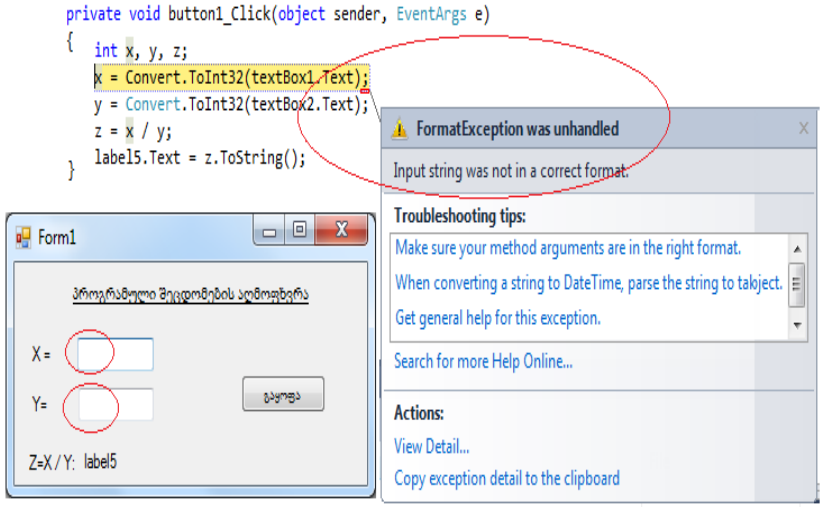
ნახ.9.7

როგორც ვხედავთ, პროგრამა მუშაობს თითქოს ნორმალურად, ასრულებს გაყოფას. ტესტირების პროცესში, რომელიც გულისხმობს კოდის ფუნქციონალობის გამოკვლევას საწყისი მონაცემების სხვადსხვა მნიშვნელობისათვის, ვღებულობთ „ნულზე გაყოფის“ შეცდომას (ნახ.9.8).



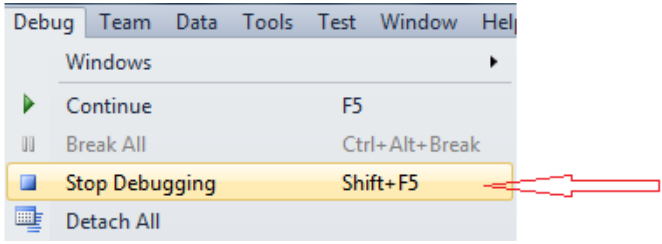
ნახ.9.8

პროგრამაში საჭიროა ამ სიტუაციის გათვალისწინება (მომხმარებელმა ყოველთვის შეიძლება შეიტანოს შემთხვევით ან „არცოდნის“ გამო 0 !). ანუ თუ იქნება შეტანილი „0“, მაშინ პროგრამამ „გვერდი აუაროს“ (გამორიცხოს, აღმოფხვრას) ასეთი ტიპის შეცდომა და თან შეატყობინოს მომხმარებელს, რომ შეიტანოს კორექტული რიცხვი (0-სგან განსხვავებული).



ნახ.9.9

რედაქტირების რეჟიმში გადასასვლელად საჭიროა მენიუდან „დებაგერის შეჩერება“ (ნახ.9.10).

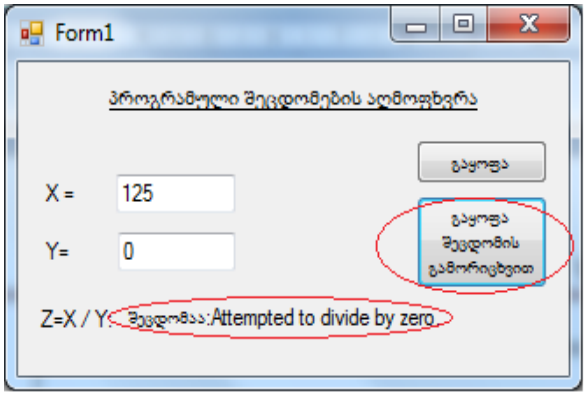


ნახ.9.10

ახლა შესაძლებელია ზემოაღწერილი ტიპის შეცდომებისათვის გამორიცხვის პროცედურის კოდის ფორმირება. მაგალითად, 9_1 ლისტინგზე მოცემულია ასეთი კოდი.

```
//ლისტინგი_9.1 --- Exception -----
private void button2_Click(object sender, EventArgs e)
 {
   int x, y, z;
   try
    {
      x = Convert.ToInt32(textBox1.Text);
      y = Convert.ToInt32(textBox2.Text);
      z = x / y;
      label5.Text = z.ToString();
    }
   catch (Exception nul_Div)  // ობიექტი nul_Div
    {
     label5.Text = "შეცდომაა:" + nul_Div.Message;
    }
 }
```

9.11 ნახაზზე ნაჩვენებია ამ კოდის მუშაობის შედეგი, რომელიც მოთავსებულია ღილაკზე „გაყოფა შეცდომის გამორიცხვით“.

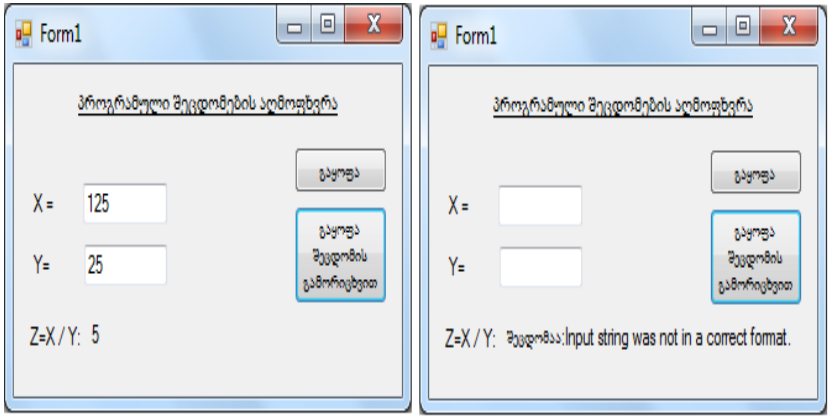


ნახ.9.11

საყურადღებო: გამოყენებულია კონსტრუქცია try { }… catch{ }. საკვანძო სიტყვა try ინიცერებას უკეთბს გამორიცხვის მექანიზმს. ამ წერტილიდან პროგრამა იწყებს {..} ბლოკის შესრულებას. ამ ბლოკის ოპერატორების შესრულებისას თუ გაჩნდა გამორიცხვის (Exception) შემთხვევა, მაშინ მას „დაიჭერს“ catch და შეცვლის გამორიცხვის სიტუაციას თავის {...} ბლოკით.

ჩვენ შემთხვევაში catch ბლოკში მოთავსებულია Exception კლასის ობიექტი (`nul_Div`), რომელიც შეიცავს ინფორმაციას აღმოცენებული შეცდომის შესახებ. Message თვისებით ხდება შეტყობინების გამოტანა ეკრანზე. პროგრამა ბოლომდე სრულდება არა-ავარიულად.

9.12 ნახაზზე მოცემულია try…catch - გამორიცხვის მექანიზმით შესრულებული პროგრამული კოდის შედეგები: როცა რიცხვები შეტანილია ნორმალურად გაყოფის ოპერაციისთვის (ამ დროს არ ხდება „შეცდომის“ დაფიქსირება try-ში), და მეორე, როცა არასწორადაა შეტანილი საწყისი მონაცემები (აქ იმუშავებს შეცდომების გამორიცხვის ბლოკი).



ნახ.9.12

განსაკუთრებულ შემთხვევათა დამუშავების try…catch მექანიზმი შეიძლება გაფართოვდეს ბიბლიოთეკაში არსებული

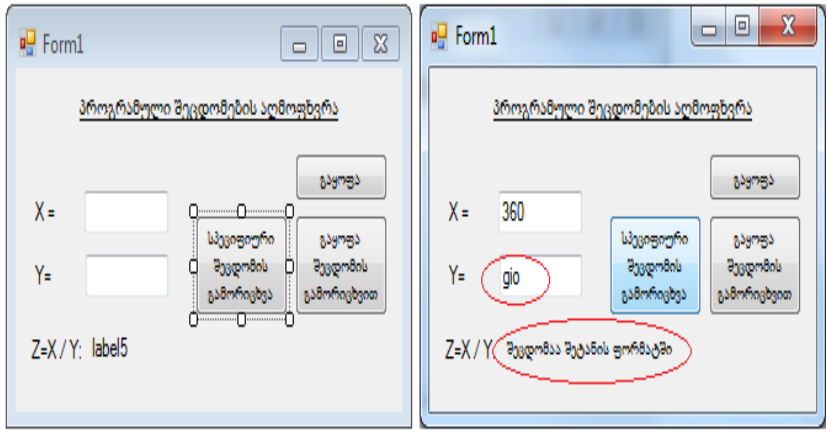
Exception-კლასის საფუძველზე. აქ იგულისხმება სპეციფიური შეცდომების აღმოჩენის შესაძლებლობა, მაგალითად, ტიპების გარდაქმნისას, ნულზე გაყოფისას და ა.შ. თუ შეცდომის სახე წინასწარ არაა განსაზღვრული, მაშინ გამოიყენება ზოგადი Exception კლასის ობიექტი.

9.2_ლისტინგში მოცემულია ღილაკის „შეცდომის გამორიცხვა“ შესაბამისი კოდის ფრაგმენტი.

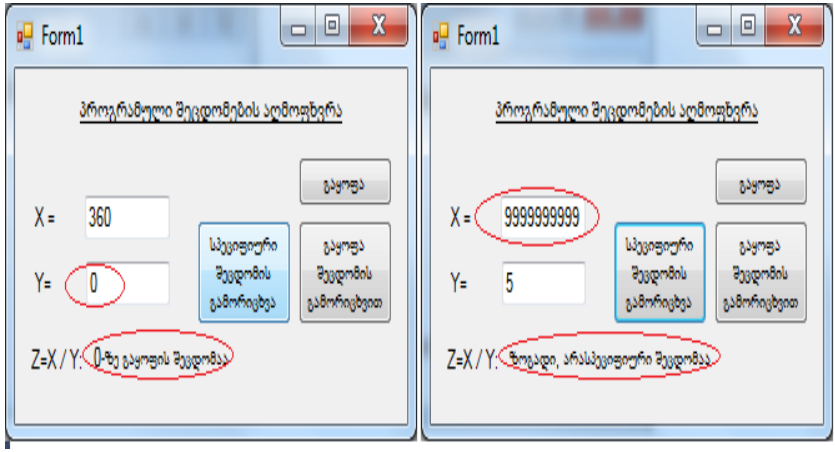
```
  // ლისტინგი_9.2 ---- - სპეციფიური და ზოგადი შეცდომები ---------
private void button3_Click(object sender, EventArgs e)
 {
  int x, y, z;

  try
    {
     x = Convert.ToInt32(textBox1.Text);
     y = Convert.ToInt32(textBox2.Text);
     z = x / y;
     label5.Text = z.ToString();
    }

catch(FormatException nul_Div) //ტიპის გარდაქმნის შეცდომა
    {
     label5.Text = "შეცდომაა შეტანის ფორმატში\n" +
                                        nul_Div.Message;
    }

catch(DivideByZeroException nul_Div) // 0-ზე გაყოფის შეცდ.
    {
     label5.Text = "0-ზე გაყოფის შეცდომაა\n" +
                                      nul_Div.Message; ;
    }
catch (Exception nul_Div)  // ზოგადი შეცდომა
    {
     label5.Text = "ზოგადი, არასპეციფიური შეცდომაა\n"+
                                        nul_Div.Message;
    }
 }
```

ლისტინგში catch{…} ბლოკების მიმდევრობას აქვს მნიშვნელობა (თუ სრულდება პირველი, წყდება პროცესი. თუ არა, გადადის შემდეგზე).

შედეგები ასახულია 9.13-ა,ბ ნახაზებზე.



ნახ.9.13-ა



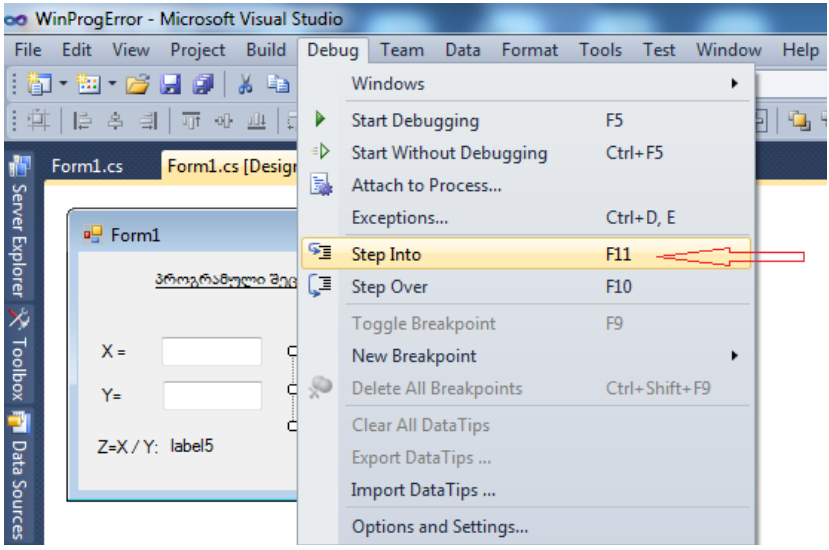
ნახ.9.13-ბ

9.3. ლოგიკური შეცდომები და პროგრამის გამართვა (Debugging) ბიჯური შესრულების რეჟიმში

როგორც აღვნიშნეთ, ლოგიკური შეცდომები მაშინ აღმოჩნდება, როდესაც სინტაქსური და შესრულების პროცესის შეცდომები აღარაა, მაგრამ სასურველ (დაგეგმილ სავარაუდო) შედეგს პროგრამა არ გვაძლევს.

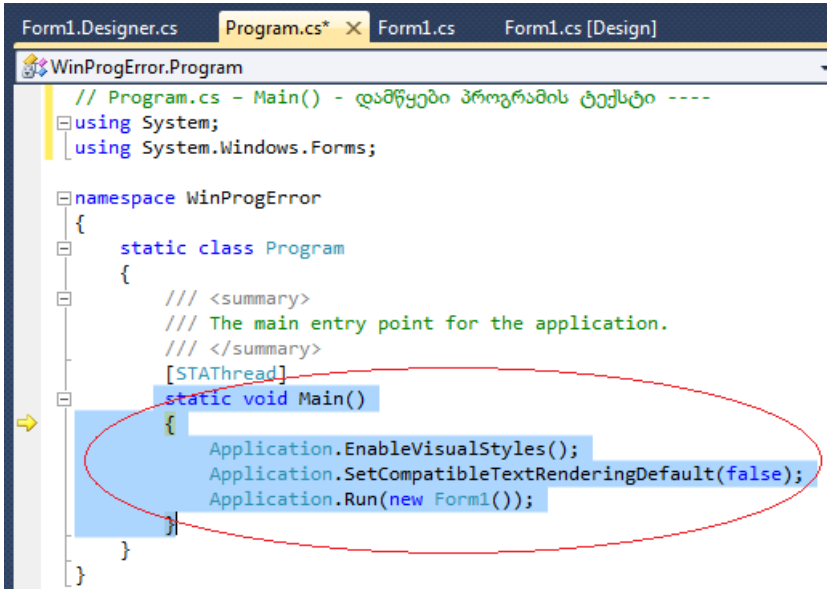
ეს ნიშნავს, რომ პროგრამის აგების ლოგიკა არასწორია !

ასეთი შეცდომების აღმოჩენა საკმაოდ რთულია და მოითხოვს ტესტირების პროცესის და პროგრამის შესრულების მიმდევრობის შედეგების ანალიზს. ვიზუალური C# ენა ფლობს პროგრამის გამართვის (Debugging) კარგ დამხმარე საშუალებებს. განვიხილოთ ისინი ჩვენი WinProgError პროექტის მაგალითზე (ნახ.9.14). გავხსნათ პროექტი, მოვამზადოთ Form1 ფორმა და მენიუს Debug-ში ავირჩიოთ Step Into (ან F11 ღილაკი კლავიატურის ზედა რიგში).



ნახ.9.14

ჩაირთვება პროგრამის გამართვის ბიჯური (Step Into) რეჟიმი. ეკრანზე დიზაინის ფორმა შეიცვლება (იხ. Solution Explorer) Program.cs დამწყები პროგრამის ტექსტით, რომელშიც მოთავსებულია Main() მთავარი ფუნქცია (ნახ.9.15). მარცხნივ ჩანს ყვითელი ისარი, რომელიც F11-ით ბიჯურად გადაადგილდება იმ სტრიქონზე, რომელიც სრულდება მოცემულ მომენტში.



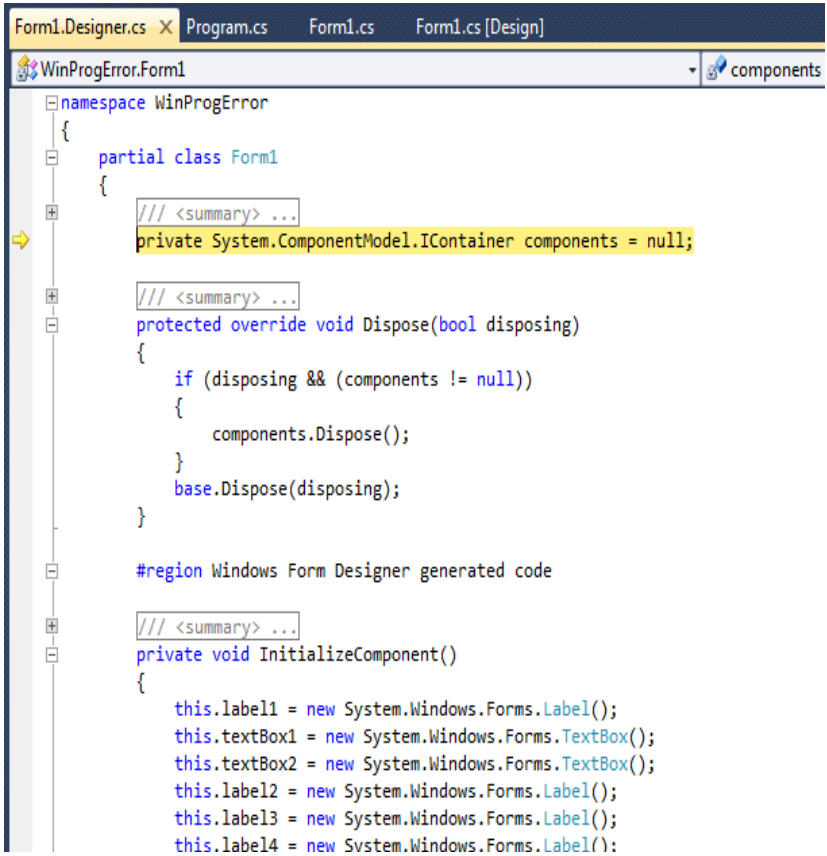
ნახ.9.15

Application.Run (new Form1()); სტრიქონის ამუშავებით ისარი გადადის (იხ. Solution Explorer) Form1.Designer.cs პროგრამაში (ნახ.9.16). შემდეგ InitializeComponent()-ში გაივლის ფორმაზე დალაგებულ ყველა ელემენტს.

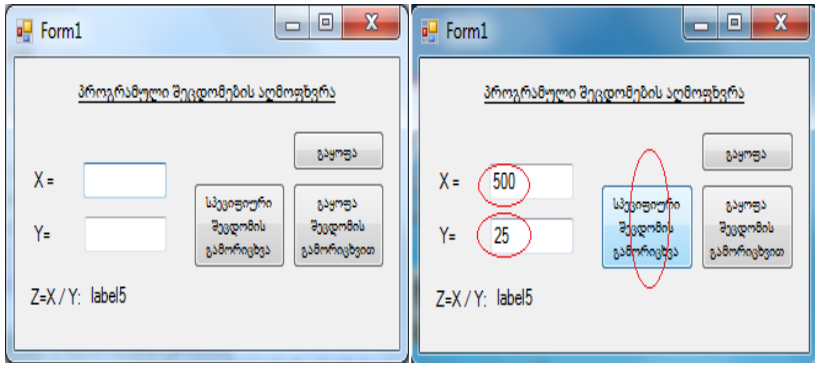
Form1.Designer.cs პროგრამის ბიჯურად გავლის შემდეგ Main()-იდან ამუშავდება Run და ეკრანზე გამოვა Form1 (ნახ.9.17),

სადაც უნდა შევიტანოთ X და Y მნიშვნელობები და ავამოქმედოთ ღილაკი „სპეციფიური შეცდომის გამორიცხვა“ (ნახ.9.18).

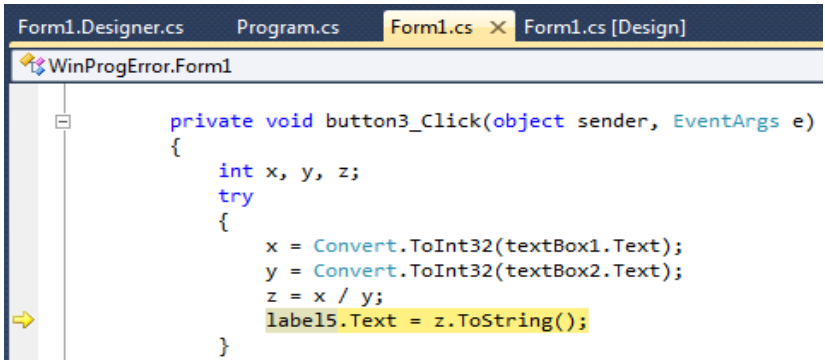
ბიჯის ისარი გადადის Form1.cs პროგრამის ტექსტზე (ნახ.9.18), სადაც button3_Click მოვლენის შესაბამის try {…} ბლოკში შედის.



ნახ.9.16

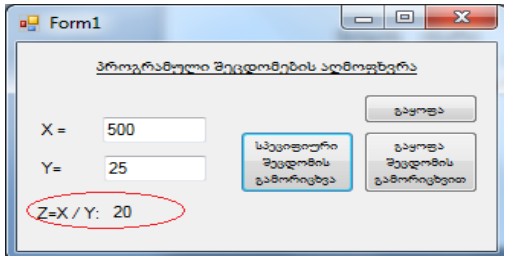


ნახ.9.17



ნახ.9.18

ვინაიდან X და Y-ის შეტანილი მნიშვნელო-ბები დასაშვებია, შედეგში აისახება label5.Text=z.ToString() მნიშვნელობა, ანუ 20 (ნახ.9.19).



ნახ.9.19

თუ ახლა განვიხილავთ Y=0 შემთხვევას, და ავამოქმედებთ იგივე ბუტონს, მაშინ try ბლოკში მომზადდება განსაკუთრებული შემთხვევა, როცა გამყოფი 0-ია.

```
private void button3_Click(object sender, EventArgs e)
{
    int x, y, z;
    try
    {
        x = Convert.ToInt32(textBox1.Text);
        y = Convert.ToInt32(textBox2.Text);
        z = x / y;
        label5.Text = z.ToString();
    }
```

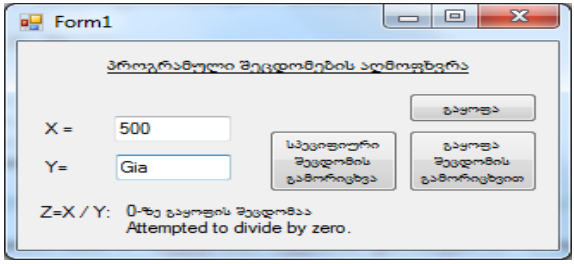
ნახ.9.20

მოქმედება გადაეცემა catch ბლოკს:

```
catch (DivideByZeroException nul_Div)  // 0-ზე გაყოფის შეცდომა
{
    label5.Text = "0-ზე გაყოფის შეცდომაა\n" + nul_Div.Message;
}
```

ნახ.9.21

დავუშვათ, რომ X ან Y არარიცხვითი სიმბოლო ან სტრიქონია, მაგალითად, “G, Gia,…” (ნახ.9.22).



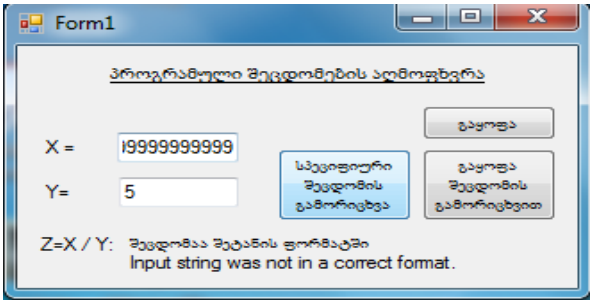
ნახ.9.22

F11-ით გადაადგილების შემდეგ პროგრამის ტექსტის აქტიური ფრაგმენტი იქნება შემდეგი (ნახ.9.23).

```
private void button3_Click(object sender, EventArgs e)
{
    int x, y, z;
    try
    {
        x = Convert.ToInt32(textBox1.Text);
        y = Convert.ToInt32(textBox2.Text);
        z = x / y;
        label5.Text = z.ToString();
    }
    catch (FormatException nul_Div)  // ტიპის გარდაქმნის შეცდომა
    {
        label5.Text = "შეცდომაა შეტანის ფორმატში\n" + nul_Div.Message;
    }
```

ნახ.9.23

მესამე, ზოგადი ანუ არასპეციფიური შემთხვევაა, ამ დროს სისიტემა თვითონ აღმოაჩენს, თუ რა სახის შეცდომასთან გვაქვს საქმე. მაგალითად, თუ X ან Y - ში შევიტანთ „დიდ რიცხვს“ (ნახ.9.24), მაშინ კოდის ფრაგმენტი ასე გამოიყურება (ნახ.9.25).



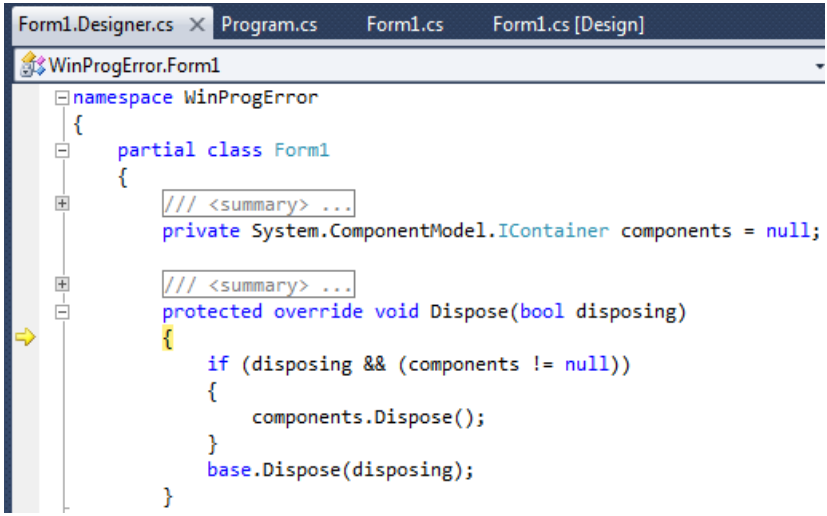
ნახ.9.24

```
catch (Exception nul_Div)  // ზოგადი შეცდომა
{
    label5.Text = "ზოგადი, არასპეციფიური შეცდომაა\n" + nul_Div.Message;
}
```

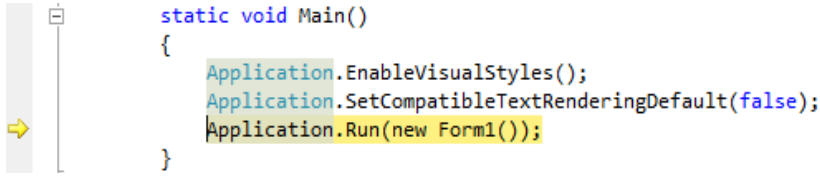
ნახ.9.25

პროგრამასთან მუშაობის დასამთავრებლად დავხუროთ Form1. ამ დროს მართვა (ისარი) გადაეცემა Form1.Designers.cs

(ნახ.9.26) და ბოლოს პროგრამას Form1.cs, რომელშიც არის Main(), და რომლითაც დაიწყო თავიდან ამ პროგრამის მუშაობა (ნახ.9.27).



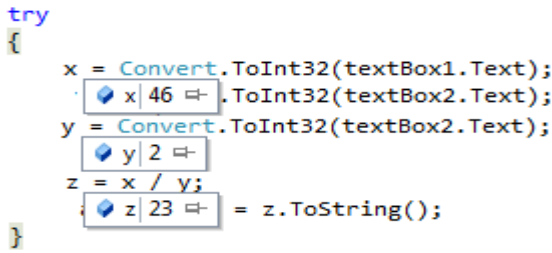
ნახ.9.26



ნახ.9.27

ამით დასრულდება დებაგერის მუშაობის პროცესი.

პროგრამის ანალიზის პროცესში შესაძლებელია ცვლადების მნიშვნელობათა ვიზუალური შემოწმება. ამისათვის მაუსის კურსორი უნდა მივიტანოთ ცვლადთან. 9.28 ნახაზზე ნაჩვენებია პროგრამაში X, Y და Z -ის მნიშვნელობები.



ნახ.9.28

დიდი პროგრამების ანალიზის დროს ბიჯურ რეჟიმში მუშაობა არაეფექტურია, სჭირდება ხანგრძლივი დრო. უფრო მოსახერხებელია ვიზუალური კონტროლის ორგანიზების მეორე ხერხი, რომელიც წყვეტის წერტილების კონცეფციითაა ცნობილი.

წყვეტის წერტილი არის პროგრამის კოდის ის ადგილი (სტრიქონი), სადაც წყდება პროგრამის შესრულება. ამ სტრიქონში მოთავსებული გამოსახულება (ოპერატორი ან მეთოდი) არ შესრულდება და მართვა მომხმარებელს გადაეცემა.

წყვეტის წერტილის შესაქმნელად კოდის საჭირო სტრიქონის გასწვრივ მარცხენა ველში მოვათავსოთ კურსორი და დავაჭიროთ თაგვის მარცხენა კლავიშს (ან F9 კლავიშს). გამოჩნდება შინდისფერი წრე. პროგრამის კოდში შეგვიძლია შევქმნათ წყვეტის რამდენიმე წერტილი (ნახ.9.29).

მენიუდან Debug→Windows→Autos არჩევით რედაქტორის ქვედა ნაწილში გამოიტანება ცვლადების მონიტორინგის ფანჯარა, რომელშიც ერთდროულად ჩანს რამდენიმე ცვლადი მათი აქტუალური მნიშვნელობებით (ნახ.9.30).

მენიუდან Debug→Windows→Locals პუნქტის არჩევით მონიტორინგის ფანჯარაში გამოიტანება მხოლოდ ლოკალური ცვლადები და მათი მნიშვნელობები.

```
            try
            {
                x = Convert.ToInt32(textBox1.Text);
                y = Convert.ToInt32(textBox2.Text);
                z = x / y;
                label5.Text = z.ToString();
            }
            catch (FormatException nul_Div)  // ტიპის გარდაქმნის შეცდომა
            {
                label5.Text = "შეცდომაა შეტანის ფორმატში\n" + nul_Div.Message;
            }
            catch (DivideByZeroException nul_Div)  // 0-ზე გაყოფის შეცდომა
            {
                label5.Text = "0-ზე გაყოფის შეცდომაა\n" + nul_Div.Message;
            }
            catch (Exception nul_Div)  // ზოგადი შეცდომა
            {
                label5.Text = "ზოგადი, არასპეციფიური შეცდომაა\n" + nul_Div.Message;
            }
```

ნახ.9.29

```
        private void button3_Click(object sender, EventArgs e)
        {
            int x, y, z;
            try
            {
                x = Convert.ToInt32(textBox1.Text);
                y = Convert.ToInt32(textBox2.Text);
                z = x / y;
                label5.Text = z.ToString();
            }
            catch (FormatException nul_Div)  // ტიპის გარდაქმნის შეცდომა
            {
                label5.Text = "შეცდომაა შეტანის ფორმატში\n" + nul_Div.Message;
            }
            catch (DivideByZeroException nul_Div)  // 0-ზე გაყოფის შეცდომა
            {
                label5.Text = "0-ზე გაყოფის შეცდომაა\n" + nul_Div.Message;
            }
            catch (Exception nul_Div)  // ზოგადი შეცდომა
            {
                label5.Text = "ზოგადი, არასპეციფიური შეცდომაა\n" + nul_Div.Message;
            }
        }
```

utos

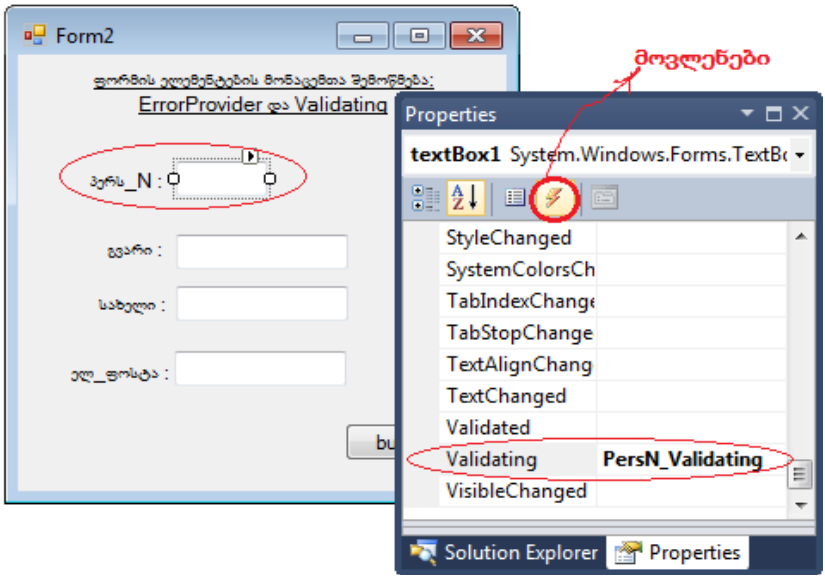
Name	Value	Type
label5	{System.Windows.Forms.Label, Text: 0-ზე გაყოფის შეცდო	System.Windows.Forms.Label
label5.Text	"0-ზე გაყოფის შეცდომაა\nAttempted to divide by zer	string
this	{WinProgError.Form1, Text: Form1}	WinProgError.Form1
x	400	int
y	40	int
z	10	int

ნახ.9.30

9.4. შესატან მონაცემთა კონტროლი

ვინდოუს-ფორმის ელემენტების შევსების პროცესში, როცა მომხმარებელს უხდება მათი ხელით შეტანა, შესაძლებელია შეცდომების არსებობა. მაგალითად, ამას ხშირად აქვს ადგილი ტქსტბოქსების შევსების დროს.

იმისათვის, რომ შესაძლებელი იყოს შესატან მონაცემთა კონტროლი, საჭიროა ErrorProvider ვიზუალური კომპონენტის გადმოტანა ინსტრუმენტების პანელიდან და საკონტროლო ელემენტების Properties-ში CausesValidation თვისებაში “True” მნიშვნელობის არსებობა (ნახ.9.31).



ნახ.9.31

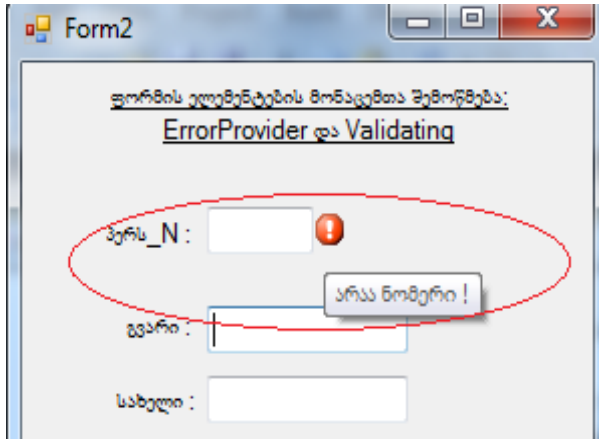
ტექსტბოქსის შევსების შემდეგ, როცა მომხმარებელი გადადის სხვა ელემენტზე, ხდება ამ ტექსტბოქსში შეტანილი მნიშვნელობის შემოწმება. თუ რა ლოგიკით შემოწმდება

ტექსტბოქსში შეტანილი მონაცემი, დამოკიდებულია ჩვენ მიერ განსაზღვრულ მეთოდზე, რომელიც მიებმება მოვლენის დამმუშავებელს (Event Handling).

ამგვარად ახლა საჭიროა მოვლენის დამმუშავებელის შექმნა. მაგალითად, ვდგებით „პერს_N“ ტექსტბოქსზე და Properties-ში Validating თვისებას ვამლევთ სახელს: PersN_Validating. დამმუშავებლის მეთოდის კოდი მოცემულია 9_3 ლისტინგში.

```
// ლისტინგი_9.3 ---  მოვლენების დამმუშავებელი ------
private void PersN_Validating(object sender,
                                        CancelEventArgs e)
 {
   if (textBox1.Text.Length == 0)
     {
      errorProvider1.SetError(textBox1, "არაა ნომერი !");
     }
    else
      errorProvider1.SetError(textBox1, "");
 }
```

შედეგი მოცემულია 9.32 ნახაზზე. ველში „პერს_N“ არ ჩაწერეს მონაცემი, ისე გადავიდნენ სხვა ტექსტბოქსზე. ამ დროს მოხდა მოვლენის შემოწმება და შეცდომის აღმოჩენა, რომ ველში არაა შეტანილი მონაცემი (`textBox1.Text.Length` არის `0`). ჩაირთვება `errorProvider1.SetError` და „პერს_N“-ის გვერდით გამოიტანს წითელი ფერის ძახილის ნიშანს (გაფრთხილება). მაუსის კურსორის მიტანისას ამ ველზე ჩნდება ქართული წარწერა „არაა ნომერი“ (`SetError`-ის მეორე პარამეტრი).

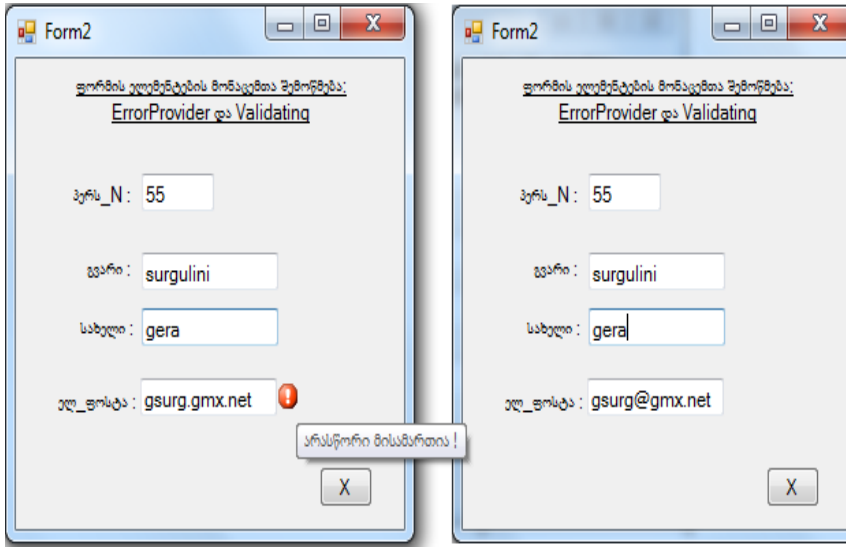


ნახ.9.32

მეოთხე ტქსტბოქსში უნდა ჩაიწეროს ელ_ფოსტის მისამართი. იგი სტრიქონული მონაცემია, რომელიც აუცილებლად უნდა შეიცავდეს ‘@’ და ‘ . ’ სიმბოლოებს. თუ რომელიმე აკლია, მაშინ შეცდომაა და უნდა ამუშავდეს მოვლენის დამმუშავებელი ამ ველისთვის (9.4_ლისტინგი).

```
   // ლისტინგი_9.4 ---  eMail_Validating მეთოდი --------
 private void eMail_Validating(object sender,
                                      CancelEventArgs e)
{
  string email = textBox4.Text;
    // კონტროლის ლოგიკა
  if(email.IndexOf('@')==-1 || email.IndexOf('.')==-1)
    {
     errorProvider1.SetError(textBox4,"არასწორი
                                          მისამართია !");
    }
   else
     errorProvider1.SetError(textBox4, "");
}
```

შედეგი ნაჩვენებია 9.33 ნახაზზე.



ნახ.9.33

9.5. კითხვები და სავარჯიშოები

9.1. რა ტიპის პროგრამული შეცდომები არსებობს და რა განსხვავებაა მათ შორის ?

9.2. როგორ ვპოულობთ სინტაქსურ შეცდომებს ?

9.3. როგორ ვპოულობთ პროცედურულ შეცდომებს ?

9.4. რას წარმოადგენს Debugging პროცესი და როგორ მუშაობს იგი ?

9.5. დებაგერის შეცდომის აღმოჩენის შემდეგ როგორ ვბრუნდებით შეცდომების გასასწორებლად რედაქტორში ?

9.6. როგორ ვპოულობთ ლოგიკურ შეცდომებს ?

9.7. რა არის განსაკუთრებული შემთხვევა და როგორ ხდება ამ დროს შეცდომის აღმოფხვრა ?

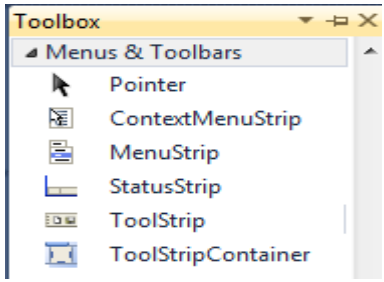
10. მენიუების და დიალოგების აგების ვიზუალური საშუალებები

ვინდოუს-პროგრამებში განსაკუთრებული ადგილი უჭირავს დიალოგური პროცესების აგებას, რომელთა შორის მნიშვნელოვანია კომპიუტერული სისტემის მთავარი მენიუს და კონტექსტური მენიუს ვიზუალური მართვის ელემენტები. წინამდებარე თავი ეთმობა სწორედ ამ კომპონენტების შესწავლას.

მთავარი მენიუს (MainMenu) დანიშნულებაა დიდი ინფორმაციული მასივების იერარქიულად ორგანიზება და მათი მოხერხებული წვდომის რეალიზება მომხმარებლებისთვის. მენიუს შეუძლია აამუშაოს სხვა დამოუკიდებელი პროგრამა ან შეასრულოს ბრძანება, გამოიძახოს დიალოგის პროცედურა, რომელშიც მომხმარებელი შეიტანს ან ამოარჩევს მონაცემებს. მთავარი მენიუ შეიძლება შედგებოდეს ქვემენიუებისაგან.

კონტექსტური მენიუ ასრულებს დამხმარე ინსტრუმენტის ფუნქციას, იგი გამოიძახება მაუსის მარჯვენა ღილაკით, როდესაც მის კურსორს მივიტანთ ჩვენთვის საჭირო ელემენტზე. გამოჩნდება ის ფუნქციები და პუნქტები, რომელიც ამ ელემენტთანაა დაკავშირებული.

C# ენის ვიზუალური ელემენტები, რომლებიც ინსტრუმენტების პანელზეა განთავსებული, ნაჩვენებია 10.1 ნახაზზე.

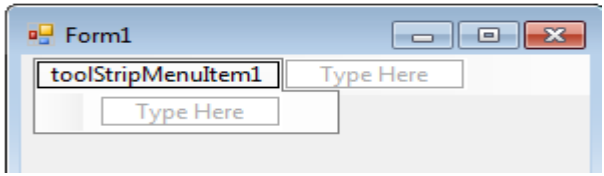


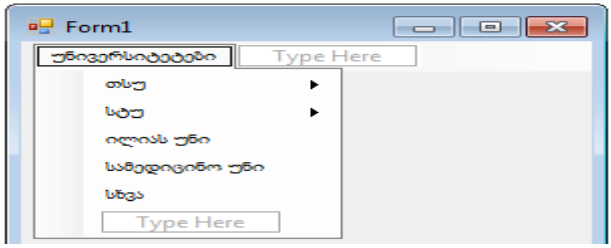
ნახ.10.1

10.1. მთავარი მენიუს ვიზუალური დაპროგრამება

ამოცანა_10.1: ავაგოთ მთავარი მენიუ „უნივერსიტეტები“ და ქვემენიუ - პუნქტებით „ფაკულტეტები“ და „კათედრები“.

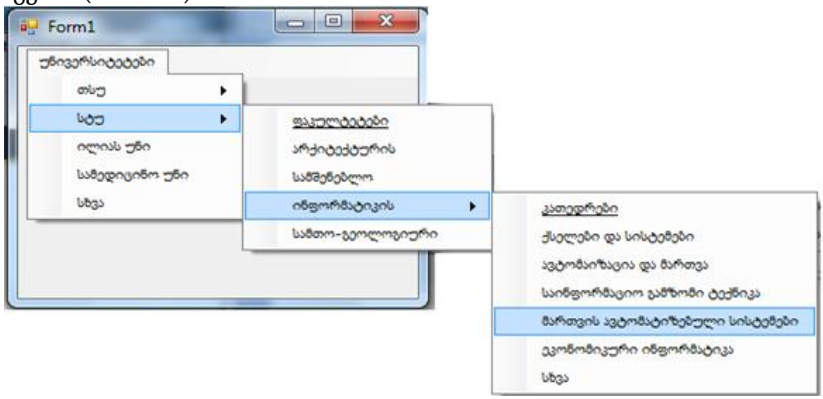
ფორმაზე მთავარი მენიუს შესაქმნელად ToolBox-იდან გადმოვიტანოთ MenuStrip ელემენტი. ფორმაზე გაჩნდება 10.2 ნახაზზე ნაჩვენები გამოსახულება. შევიტანთ მენიუს პუნქტებს.





ნახ.10.2

ეს ხორციელდება ვერტიკალურად და/ან ჰორიზონტალურად. თითოეული სტრიქონისთვის შეიძლება ქვემენიუს შექმნა (ნახ.10.3).

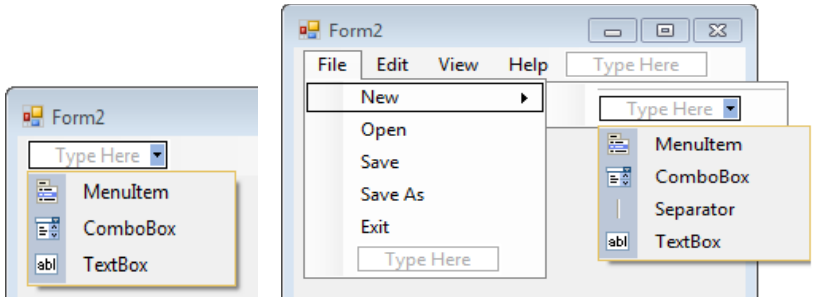


ნახ.10.3

მენიუს პუნქტების (სტრიქონების) გადაადგილება შეიძლება Form1[Design] რეჟიმში მაუსის მარცხენა ღილაკით Drag&Drop (გადატანა) საშუალებით.

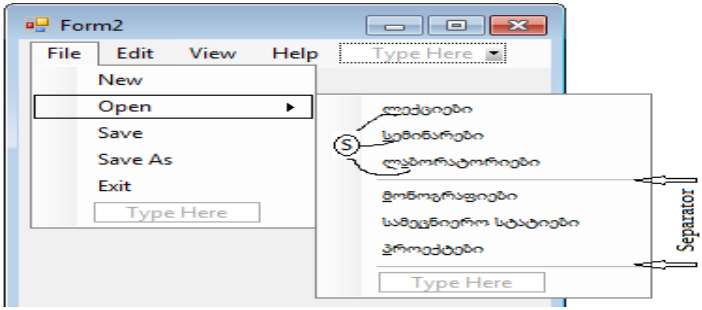
მთავარი მენიუს პუნქტები შიძლება სამი სახისა იყოს (ნახ.10.4):

- ჩვეულებრივი შესატანი სტრიქონი;
- კომბობოქსი, რომელშიც მოხდება ამორჩევა ან შეტანა;
- ტექსტბოქსი.



ნახ.10.4

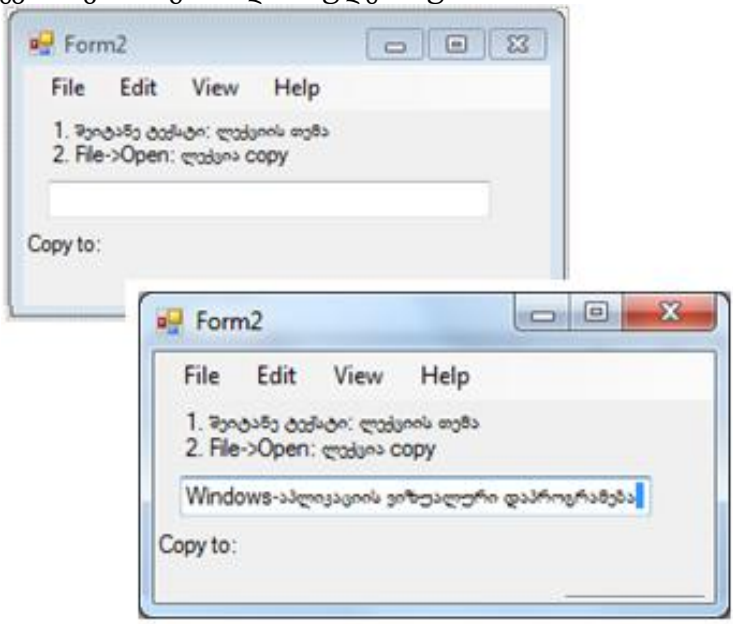
ქვემენიუსთვის დასაშვებია Separator პუნქტიც. რომლის დანიშნულებაა მენიუს პუნქტების ერთმანეთისაგან გამოყოფა ხაზით, რაც ვიზუალურ კომფორტს უქმნის მომხმარებელს (ნახ.10.5). მენიუში ხშირად იყენებენ სტრიქონის ერთი ასოს გამოყოფას (ქვეშგახაზვა), რომლითაც ამოქმედდება ეს პუნქტი. ნახაზზე იგი S სიმბოლოთია მითითებული.



ნახ.10.5

მენიუს პუნქტებით უნდა მოხდეს გარკვეული დავალების შესრულება (საჭირო ინფორმაციისკენ გზის განსაზღვრა და ბოლოს ამორჩევა). ამიტომ ეს პუნქტები დაკავშირებულია მოვლენებითან და მეთოდებთან. მოვლენის მთავარი სახეა Click, რომელზეც მიბმულია შესასრულებელი პროცედურის კოდი. განვიხილოთ ეს საკითხი.

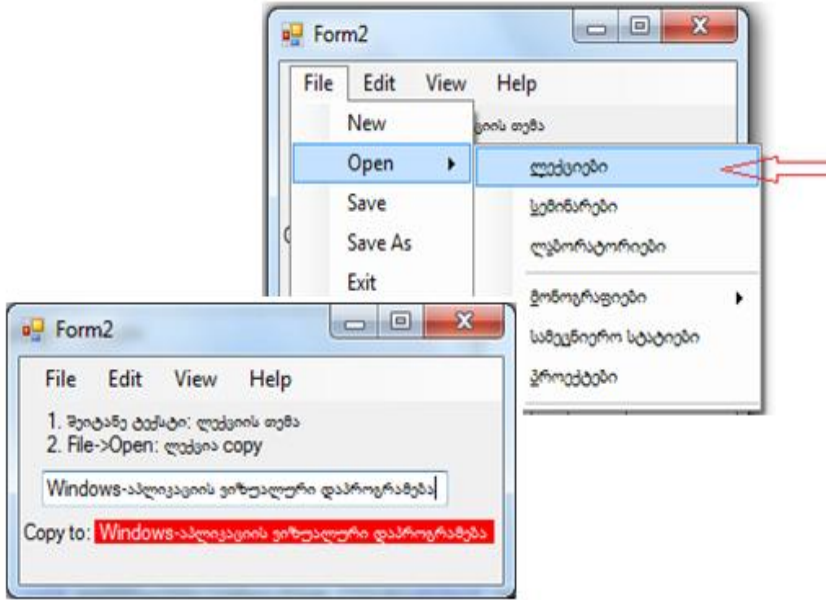
ამოცანა_10.2: ფორმაზე ავაგოთ მთავარი მენიუ (ნახ.10.6), რომლის პუნქტი „ლექციები“ გადააკოპირებს textBox1-ში ჩაწერილ ლექციის თემის დასახელებას label1-ში. მენიუს Exit პუნქტის არჩევისას კი პროგრამა დაასრულებს მუშაობას.



ნახ.10.6

`textBox1`-ში ჩაწერილი სტრიქონი „Windows-აპლიკაციის ვიზუალური დაპროგრამება“ მთავარი მენიუს „File->Open->ლექციები“ პუნქტის არჩევით ამ სტრიქონს გადააკოპირებს `label9` ველში, რომელ;იც „`Copy to`“-ლებელის მარჯვნივაა

მოთავსებული (ის არ ჩანს). ამ მოვლენის დამმუშავებელს აქვს 10.1 ლისტინგში მოცემული კოდის ფორმა. შედეგი მოცემულია 10.7 ნახაზზე, წითელი ფონით და თეთრი ტექსტით.



ნახ.10.7

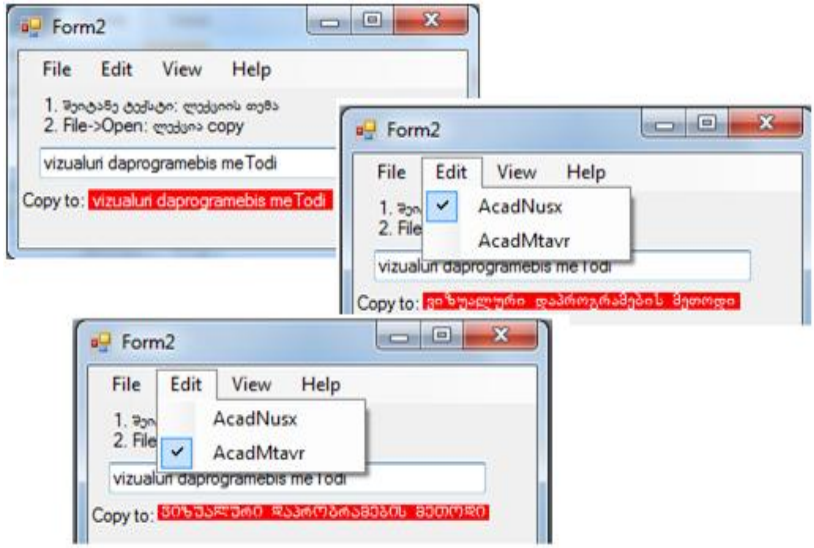
```
// ლისტინგი_10.1 ---- მთავარი მენიუ: File -> Open -> ლეციები --------
private void ToolStripMenuItem_Click(object sender,
                                                EventArgs e)
 {
   label9.BackColor = Color.Red;
   label9.ForeColor = Color.White;
   label9.Text = textBox1.Text;
 }

private void exitToolStripMenuItem_Click(object sender,
                                                  EventArgs e)
 {  Close();
 }
```

10.2. მთავარი მენიუდან შრიფტების და ფერების მართვა

ამოცანა_10.3: Form2-ზე მთავარი მენიუს Edit პუნქტში ჩავდოთ კომბობოქსი შრიფტების არჩევის მიზნით, კერძოდ AcadNusx და AcadMtavr მნიშვნელობებით. ამ ფონტების არჩევით უნდა შეიცვალოს label9-ში ჩაწერილი სტრიქონის შრიფტი.

საილუსტარციო მაგალითის შდეგები მოცემულია 10.8 ნახაზზე, პროგრამული კოდის ფრაგმენტი კი - 10.2_ლისტინგში.

```
// ლისტინგი_10.2 --  ფონტის შეცვლა მთავარი მენიუდან ------
private void acadNusxToolStripMenuItem_Click(object sender, EventArgs e)
 {
    label9.Font = new Font("AcadNusx", label9.Font.Size, label9.Font.Style);
    acadNusxToolStripMenuItem.Checked = true;
    acadMtavrToolStripMenuItem.Checked = false;
 }
private void acadMtavrToolStripMenuItem_Click(object sender,EventArgs e)
 {
   label9.Font = new Font("AcadMtavr", label9.Font.Size, label9.Font.Style);
   acadMtavrToolStripMenuItem.Checked = true;
   acadNusxToolStripMenuItem.Checked = false;
  }
```



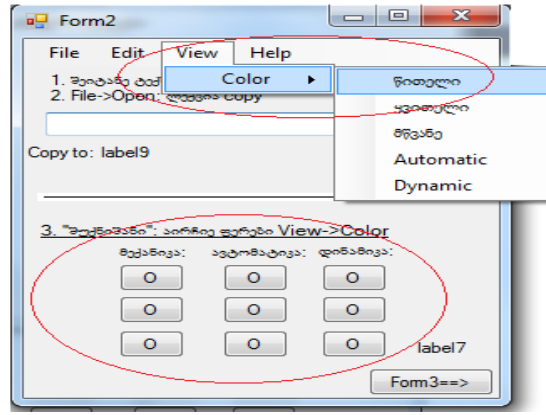
ნახ.10.8

ამოცანა_10.4: ფორმაზე დალაგებულია მართვის ელემენტები (მაგალითად, ბუტონები), რომლებიც განასახიერებს „შუქნიშანს“ სამი ფერით: წითელი, ყვითელი და მწვანე. საჭიროა მთავარი მენიუს View -> Color პუნქტის გასწვრივ (ნახ.10.9-ა), ავირჩიოთ რომელიმე ფერი, რომელიც შეაფერადებს შესაბამის O-ბუტონს, ან ავირჩიოთ ამ ქვემენიუს სტრიქონი Automatic ან Dynamic. „ავტომატიკას“ გამოაქვს ერთბაშად სამივე ფერი, ხოლო „დინამიკას“ - ციკლში ფერთა მონაცვლეობით (ნახ.10.9-ბ).

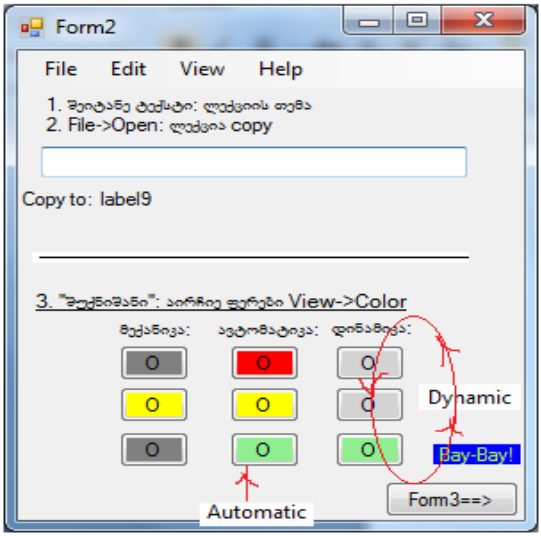
ამოცანის გადაწყვეტის კოდი მოცემულია 10.3_ლისტინგში.

```
// ლისტინგი_10.3 ---- მთავარი მენიუ->View->Color -------
private void წითელიToolStripMenuItem_Click(object
                                    sender, EventArgs e)
 {
   button1.BackColor = Color.Red;
   button2.BackColor = Color.Gray;
   button3.BackColor = Color.Gray;
 }
private void ყვითელიToolStripMenuItem_Click(object
                                    sender, EventArgs e)
 {
   button1.BackColor = Color.Gray;
   button2.BackColor = Color.Yellow;
   button3.BackColor = Color.Gray;
 }
private void მწვანეToolStripMenuItem_Click(object
                                    sender, EventArgs e)
 {
   button1.BackColor = Color.Gray;
   button2.BackColor = Color.Gray;
   button3.BackColor = Color.LightGreen;
 }
private void automaticToolStripMenuItem_Click(object
                                    sender, EventArgs e)
 {
   button4.BackColor = Color.Red;
   button5.BackColor = Color.Yellow;
   button6.BackColor = Color.LightGreen;
 }
private void dynamicToolStripMenuItem_Click(object
                                    sender, EventArgs e)
```

```
{
 for (int j = 1; j < 5; j++)
 {
  for (int i = 1; i < 4; i++)
  {
  switch (i)
   {
    case 1: button7.BackColor = Color.Red;
            button8.BackColor = Color.LightGray;
            button9.BackColor = Color.LightGray;
            break;
    case 2: button7.BackColor = Color.LightGray;
            button8.BackColor = Color.Yellow;
            button9.BackColor = Color.LightGray;
            break;
    case 3: button7.BackColor = Color.LightGray;
            button8.BackColor = Color.LightGray;
            button9.BackColor = Color.LightGreen;
            break;
    default: break;
   }
  button7.Refresh();  button8.Refresh(); button9.Refresh();
  Thread.Sleep(1500);
 }
}
label7.BackColor = Color.Blue;
label7.ForeColor = Color.GreenYellow;
label7.Text = "Bay-Bay!";
}
```

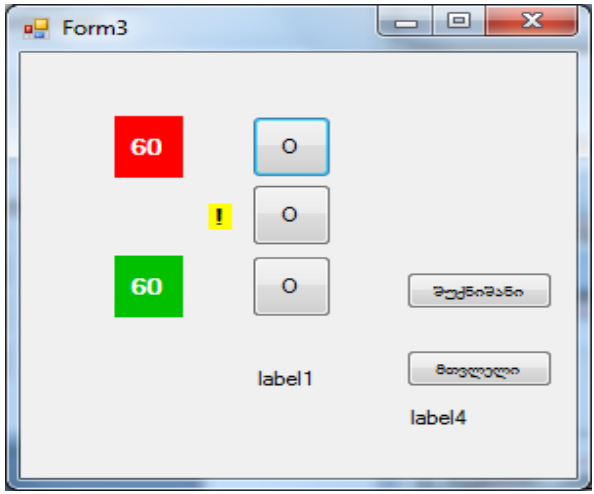


ნახ.10.9-ა



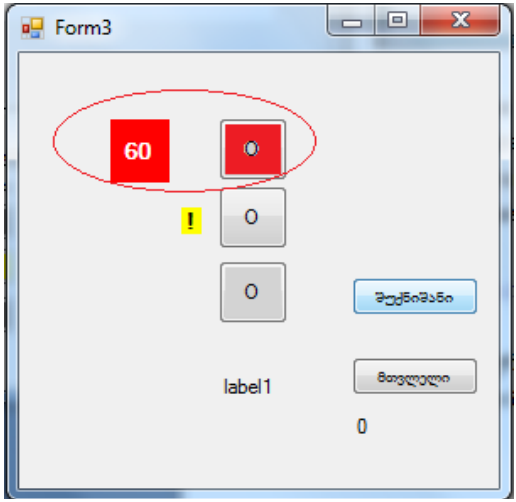
ნახ.10.9-ბ

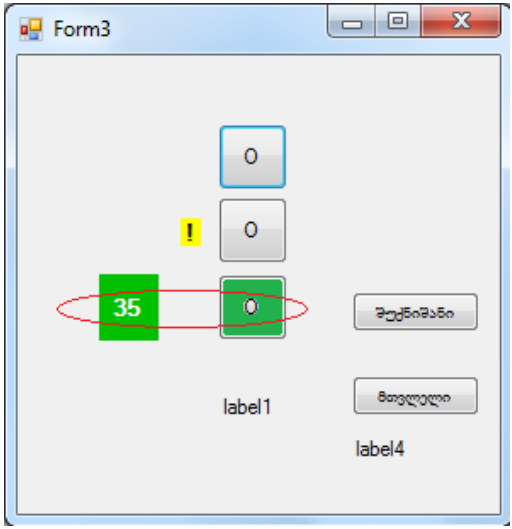
ამოცანა_10.5: ფორმაზე Form3 ავაგოთ მოდელი „შუქნიშანი“, რომელიც იმუშავებს რეალური შუქნიშნის მსგავსად ფერთა მონაცვლეობის პრინციპით დროის გარკვეული ინტერვალებით. 10.10 ნახაზზე ნაჩვენებია ასეთი პროექტის ერთ-ერთი ვარიანტი.



ნახ.10.10

ღილაკი „შუქნიშანი“ რეალიზებულია ფერთამონაცვლეობის დინამიკით და დროითი დაყოვნებების გათვალისწინებით. ეს ფუნქციონალობა ასახულია 10.3_ლისტინგში, ხოლო შედეგები 10.11 ნახაზზე.





ნახ.10.11

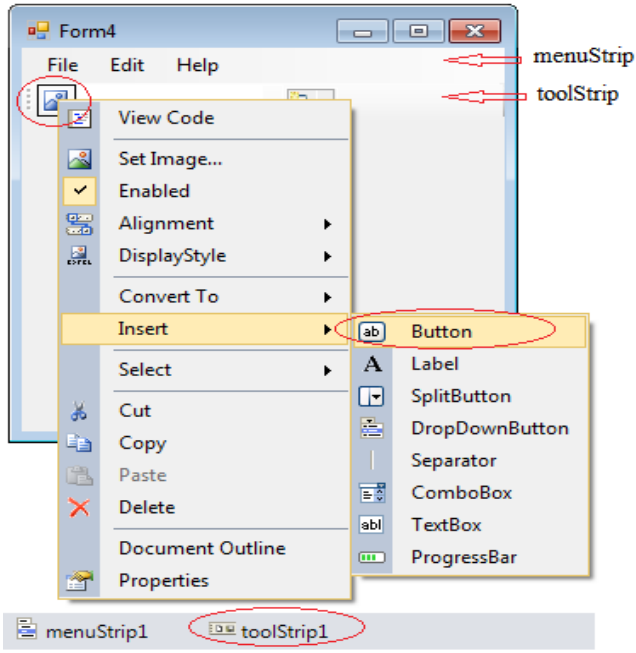
```
// ლისტინგი_10.4 --- „შუქნიშანი“ --------
using System;
using System.Drawing;
using System.Windows.Forms;
using System.Threading;
namespace WinMenus
{
  public partial class Form3 : Form
   {
     public Form3() { InitializeComponent(); }

private void Form3_Load(object sender, EventArgs e)
  {
    System.Windows.Forms.Timer tm = new
                          System.Windows.Forms.Timer();
    tm.Interval = 500;
    tm.Tick += new EventHandler(tm_tick);
    tm.Enabled = false; // true;
  }
private void button4_Click(object sender, EventArgs e)
 {
  for (int j = 1; j < 3; j++)
   {
    for (int i = 1; i < 3; i++)
     {
     switch (i)
      {
      case 1: label2.Visible = true;
              label3.Visible = false;
              button1.BackColor = Color.Red;
              button2.BackColor = Color.LightGray;
              button3.BackColor = Color.LightGray;
              button1.Refresh();
              button2.Refresh();
              button3.Refresh();
              for (int k = 15; k >= 0; k--)
               {
                label2.Text = k.ToString();
                label2.Refresh();
                Thread.Sleep(500);
               }
```

```
            label2.Visible = false;
             label3.Visible = false;
             button1.BackColor = Color.LightGray;
             button2.BackColor = Color.Yellow;
             button3.BackColor = Color.LightGray;
             button1.Refresh();
             button2.Refresh();
             button3.Refresh();
             Thread.Sleep(800);
             break;

        case 2: label2.Visible = false;
            label3.Visible = true;
            button1.BackColor = Color.LightGray;
            button2.BackColor = Color.LightGray;
            button3.BackColor = Color.LightGreen;
            button1.Refresh();
            button2.Refresh();
            button3.Refresh();
            for (int k = 15; k >= 0; k--)
             {
              label2.Refresh();
              label3.Text = k.ToString();
              label3.Refresh();
              Thread.Sleep(500);
             }
            label2.Visible = false;
            label3.Visible = false;
            button1.BackColor = Color.LightGray;
            button2.BackColor = Color.Yellow;
            button3.BackColor = Color.LightGray;
            button1.Refresh();
            button2.Refresh();
            button3.Refresh();
            Thread.Sleep(250);
            // 3 secondiani  ----------------
            label2.Visible = false;
```

```
                label3.Visible = true;
                button2.BackColor = Color.Yellow;
                button3.BackColor = Color.LightGreen;
                button2.Refresh();
                button3.Refresh();

              for (int k = 3; k >= 0; k--)
                 {
                  label3.Text = k.ToString();
                  label3.Refresh();
                  Thread.Sleep(500);
                 }
                Thread.Sleep(250);
                break;
                default: break;
               }
              Thread.Sleep(500);
             }
            }
            button2.BackColor = Color.LightGray;
            button3.BackColor = Color.LightGray;
            button2.Refresh(); button3.Refresh();
            label3.Visible = false;
            label1.BackColor = Color.Blue;
            label1.ForeColor = Color.GreenYellow;
            label1.Text = "Bay-Bay!";
          }
private void button5_Click(object sender, EventArgs e)
 {
  for (int i = 10; i >= 0; i--)
   {
     label4.Text = i.ToString();
     label4.Refresh();
     Thread.Sleep(500);
   }
 }
private void tm_tick(object sender, EventArgs e)
  {
   label5.Visible = !label5.Visible;
  }
}}
```

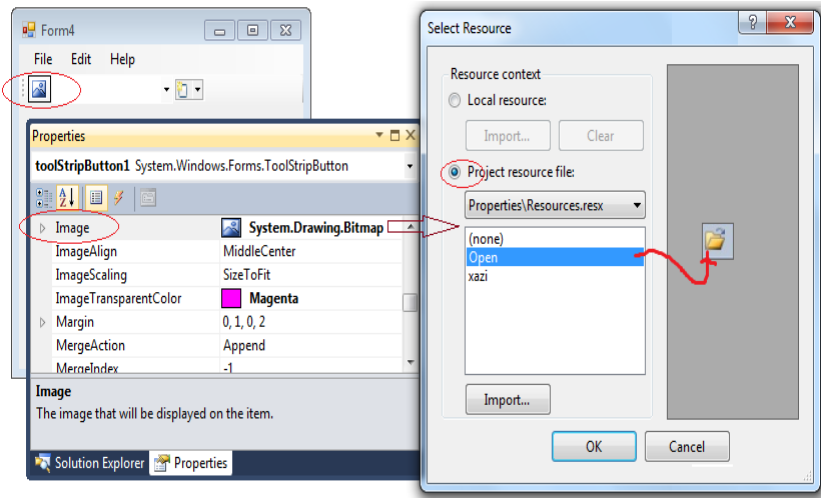
10.3. გრაფიკული მენიუს აგება - ToolStrip

კომპიუტერული სისტემების აგების დროს ინტერფეისში მთავარი მენიუს გარდა ხშირად იყენებენ გრაფიკულ პიქტოგრამების (icons), რაც უფრო ეფექტურს და მოქნილს ხდის მის გამოყენებას. C# ენაში ასეთი ვიზუალური ელემენტია Menus&Toolbars პანელის toolStrip ელემენტი. მისი გადატანით ფორმაზე მივიღებთ 10.12 ნახაზზე ნაჩვენებ სურათს. საჭიროა ავირჩიოთ სახე: button, Label, ComboBox და ა.შ.



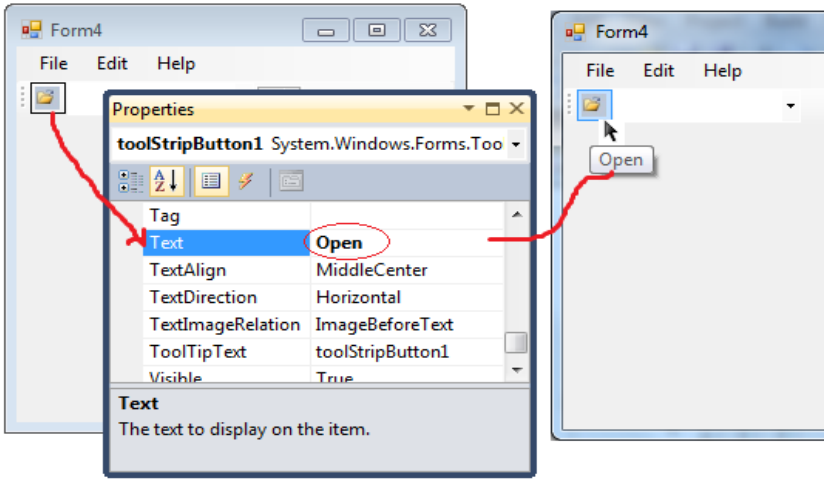
ნახ.10.12

აქ შესაძლებელია მაუსის მარჯვენა ღილაკით სტანდარტულად მიღებული პიქტოგრამის შეცვლა, ჯერ Properties-ში Image თვისების არჩევით და შემდეგ საჭირო პიქტოგრამის Import-ირებით დიალოგური ფანჯრიდან.



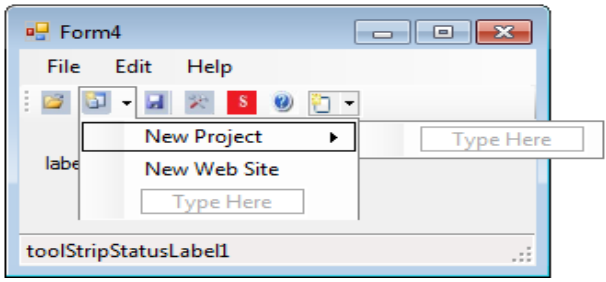
ნახ.10.13

ახალ პიქტოგრამას თვისებაში Text ჩაუწერეთ მისი ფუნქციის შესაბამისი სიტყვა, მაგალითად, Open. 10.14 ნახაზზე ჩანს მიღებული შედეგი.



ნახ.10.14

საბოლოო შედეგები რამდენიმე ახალი პიქტოგრამის ჩასმის შემდეგ, რომელთაგანაც ერთ-ერთი comboBox ტიპისაა, მოცემულია 10.15 ნახაზზე.



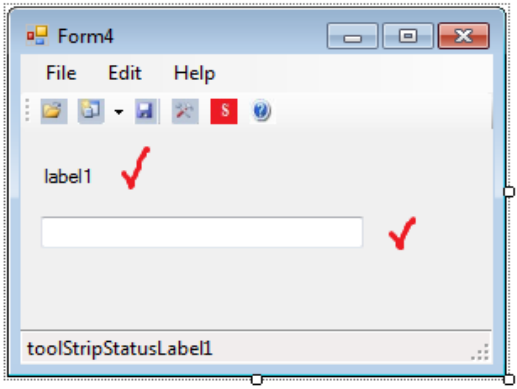
ნახ.10.15

10.4. კონტექსტური მენიუს ვიზუალური დაპროგრამება - ContextMenuStrip

როგორც ამ თავის შესავალში აღვნიშნეთ, კონტექსტური მენიუ გამოიტანება ფორმაზე ან ფორმის ელემენტზე კურსორის მიტანის შემდეგ და მაუსის მარჯვენა ღილაკის დაჭერით.

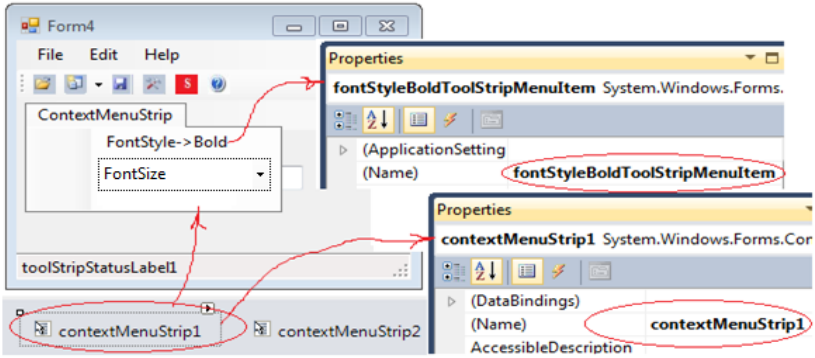
ToolBox-პანელიდან გამოიყენება ელემენტი ContextMenuStrip. მისი პუნქტების შევსების მექანიზმი ჰგავს მთავარი მენიუსას. საყურადღებოა ის ფაქტი, რომ კონტექსტური მენიუს სტრიქონთა მნიშვნელობები უნდა შეესაბამებოდეს ფორმის ან ფორმაზე დადებული ელემენტებისთვის საჭირო ფუნქციებს.

ამოცანა_10.6: ავაგოთ ფორმა, მაგალითად, 10.16 ნახაზზე მოცემულია სახით, რომელზეც label1 და textBox1 ელემენტებია დადებული. ამ ელემენტებისთვის უნდა შეიქმნას ორი კონტექსტური მენიუ. textBox1-ში ჩაწერილი სტრიქონი კონტექსტური მენიუთი უნდა გადაკოპირდეს label1-ში. შემდეგ label1-ში გადატანილი სტრიქონის ფორმა (სტილი, ზომა) უნდა შეიცვალოს კონტექსტური მენიუდან.

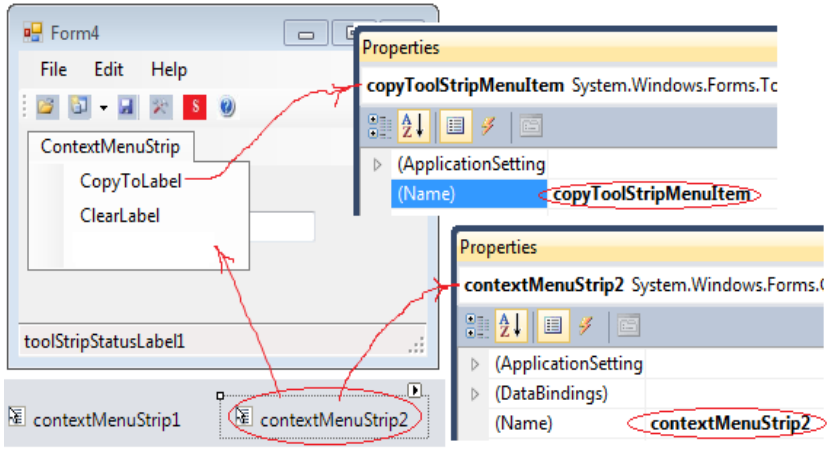


ნახ.10.16

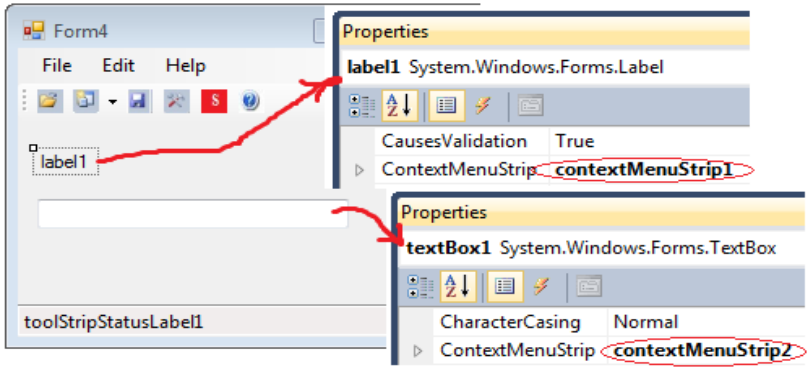
10.17-ა,ბ ნახაზებზე ნაჩვენებია Form4-ზე მოთავსებული label1 და textBox1 ელემენტებისთვის ჩვენ მიერ შექმნილი კონტექსტური მენიუები, შესაბამისად ContextMenuStrip1 და ContextMenuStrip2. კონტექსტური მენიუები აგებულია. ახლა ისინი უნდა „მივაბათ“ Form4-ის textBox1 და label1 ელემენტებს, რათა მაუსის მარჯვენა ღილაკმა იმუშაოს და ამ ელემენტებზე კურსორის მიტანისას გამოჩნდეს კონკრეტულად მისი შესაბამისი კონტექსტური მენიუ. ამისათვის მოვნიშნოთ label1 და მის Properties-ის ContextMenuStrip-თვისებაში ჩავწეროთ contextMenuStrip1 მნიშვნელობა (ნახ.10.18). ასევე textBox1-სთვის ჩავწეროთ contextMenuStrip2.



ნახ.10.17-ა



ნახ.10.17-ბ



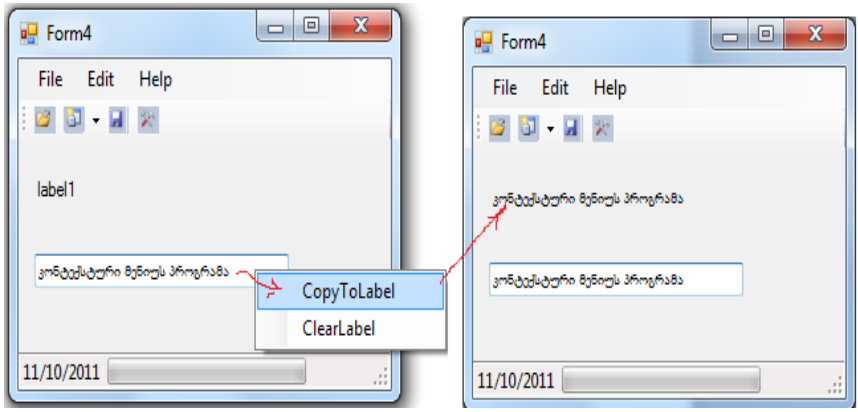
ნახ.10.18

პროგრამული კოდი მოცემულია 10.4 ლისტიგში, შედეგები კი 10.19 ნახაზზე.

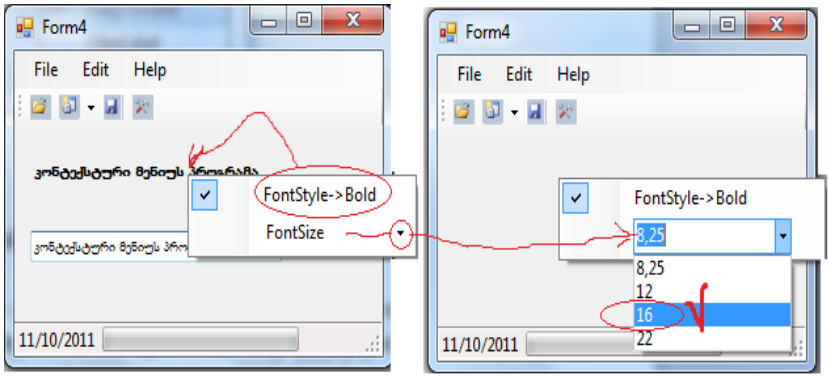
```
// ლისტინგი_10.5 -- შრიფტის შეცვლა ---
using System;
using System.Drawing;
using System.Windows.Forms;
using System.Threading;
namespace WinMenus
```

```
{
 public partial class Form4 : Form
 {
  double DamtavrDro;
  public Form4() { InitializeComponent();  }
private void Form4_Load(object sender, EventArgs e)
  {
   toolStripStatusLabel1.Text =
                    DateTime.Today.ToShortDateString();
  }
private void Stop_Click(object sender, EventArgs e)
  {
    DamtavrDro = 0;
     timer1.Enabled = true;
   }
private void timer_Tick(object sender, EventArgs e)
   {
     DamtavrDro += 0.1;
     if (DamtavrDro >= 5)
        Close();
       else
         toolStripProgressBar1.Value = (int)DamtavrDro;

     textBox1.Text = DamtavrDro.ToString();
   }
private void copyToolStripMenuItem_Click(object sender,
                                          EventArgs e)
   {
     label1.Text = textBox1.Text;
     if (label1.Text == "")
       label1.Text = "(leer)";
   }
private void clearLabelToolStripMenuItem_Click(object
                                    sender, EventArgs e)
   {
     label1.Text = "";
   }
private void fontStyleBoldToolStripMenuItem_Click(object
                                           sender, EventArgs e)
 {
   label1.Font = new Font(label1.Font.FontFamily,
        label1.Font.Size, label1.Font.Style ^ FontStyle.Bold);
```

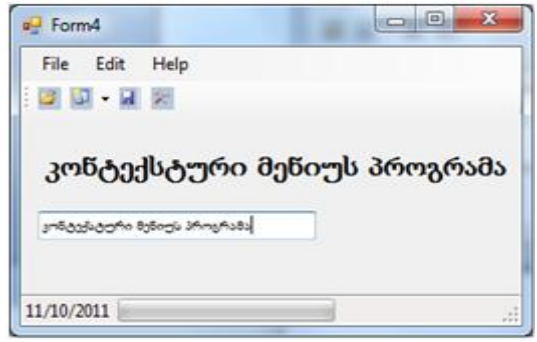
```
    fontStyleBoldToolStripMenuItem.Checked =!
                fontStyleBoldToolStripMenuItem.Checked;
  }
private void fontSize16ToolStripMenuItem_Click(object
                                    sender, EventArgs e)
  {
   label1.Font = new Font(label1.Font.FontFamily,
                  label1.Font.Size, label1.Font.Style ^
                  FontStyle.Italic);
   fontStyleBoldToolStripMenuItem.Checked =!
                fontStyleBoldToolStripMenuItem.Checked;
  }
private void toolStripComboBox1_TextChanged(object
                                    sender, EventArgs e)
{
  double FontSize;
 try
 {
    FontSize = Convert.ToDouble(toolStripComboBox1.Text);
 }
catch
   {
    FontSize = 8.25;
   }

 label1.Font = new Font(label1.Font.FontFamily,
                   (float)FontSize, label1.Font.Style);
}
private void toolStripComboBox1_Click(object sender,
                                          EventArgs e)
   {
    toolStripComboBox1.Items.Clear();
    toolStripComboBox1.Items.Add("8,25");
    toolStripComboBox1.Items.Add("12");
    toolStripComboBox1.Items.Add("16");
    toolStripComboBox1.Items.Add("22");
    toolStripComboBox1.SelectedIndex = 0;
   }
  }
} //  შედეგები მოცემულია 10.19 – 21 ნახაზებზე.
```



ნახ.10.19



ნახ.10.20

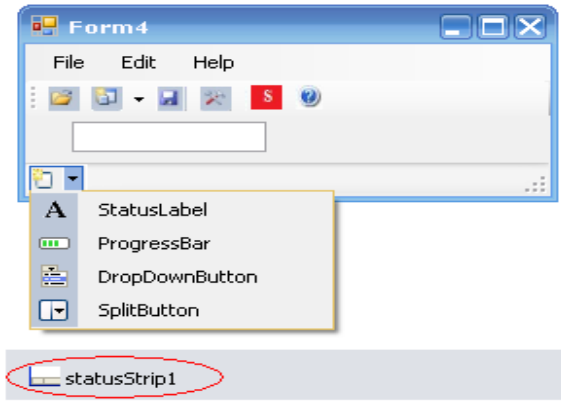


ნახ.10.21-ა

ნახ.10.21-ბ

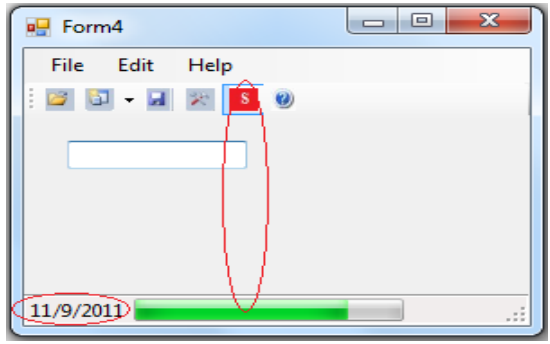
10.5. სტატუსის პანელის შექმნა - StatusStrip

სტატუსის პანელის დანიშნულებაა პროგრამის მუშაობის პროცესში საჭირო ინფორმაციის ასახვა ეკრანზე. იგი იქმნება Toolbox-იდან StatusStrip ელემენტის გადმოტანით ფორმაზე და თავსდება ფორმის ქვედა ნაწილში (ნახ.10.18). მისთვის ძირითადად გამოიყენება label - ტიპი.



ნახ.10.22-ა

ნახ.10.22-ბ



ავირჩიოთ ჯერ StatusLabel, რომელშიც გამოვიტანთ სისტემურ თარიღს (ანუ დღევანდელს) და მეორე, ProgressBar, რომელიც იმუშავებს გრაფიკულ მენიუდან „S“ (stop) ღილაკის ამოქმედებით. იგი გვაძლევს პროგრამული სისტემის დამთავრების პროცესის ხანგრძლიობას (წამებში). 10.6 ლისტინგში მოცემულია აღწერილი პროცესის პროგრამული კოდი.

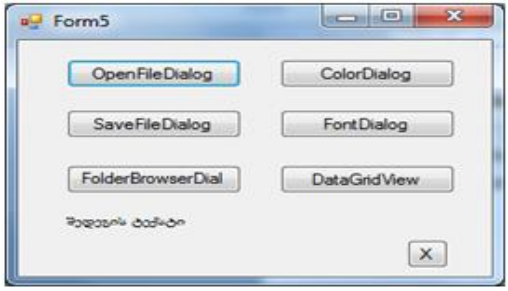
```
// ლისტინგი_10.6 -- შრიფტის შეცვლა ---
using System;
using System.Drawing;
using System.Windows.Forms;
using System.Threading;
namespace WinMenus
{
public partial class Form4 : Form
  {
   double DamtavrDro;  // პროგრამის დამთავრების დროის ცვლადი
   public Form4() { InitializeComponent(); }
  private void Form4_Load(object sender, EventArgs e)
   {    // სტატუს-ველი მიმდინარე თარიღის გამოსატანად ----
    toolStripStatusLabel1.Text =
                  DateTime.Today.ToShortDateString();
   }
  private void Stop_Click(object sender, EventArgs e)
   {  // მენიუდან “S” ამოქმედება
    DamtavrDro = 0;
```

```
    timer1.Enabled = true; // საათის ჩართვა სტატუს-ველის
                           // ProgressBar-სთვის
   }
  private void timer_Tick(object sender, EventArgs e)
   {   // პროგრამის დამთავრების ხანგრძლივობის დათვლა --------
    DamtavrDro += 0.1;
    if (DamtavrDro >= 5)
      Close();
     else
      toolStripProgressBar1.Value = (int)DamtavrDro;

    textBox1.Text = DamtavrDro.ToString();
   }
  }
}
```

10.6. C# ენის ვიზუალური სტანდარტული დიალოგური საშუალებანი

დაპროგრამების ვიზუალურ C# ენაში არსებობს ხუთი სახის დიალოგური კლასი, რომლებიც ხშირად გამოიყენება პროგრამული პროექტების აგების პროცესში: OpenFileDialog, SaveFileDialog, FolderBrowserDialog, ColorDialog და FontDialog.

განვიხილოთ ეს დიალოგები დეტალურად (ნახ.10.23).

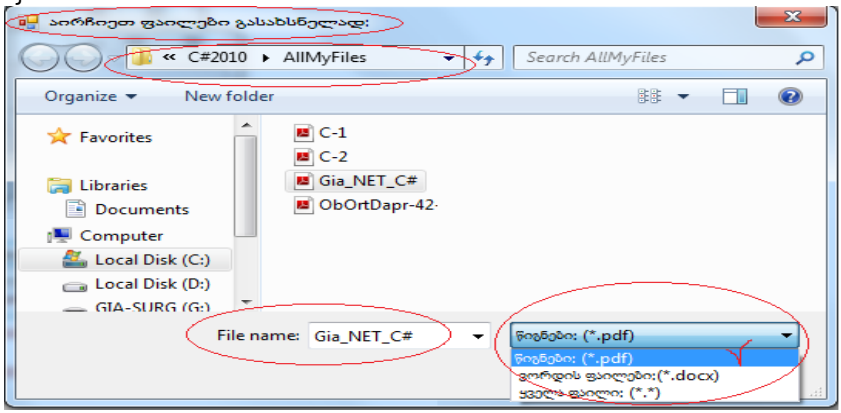


ნახ.10.23

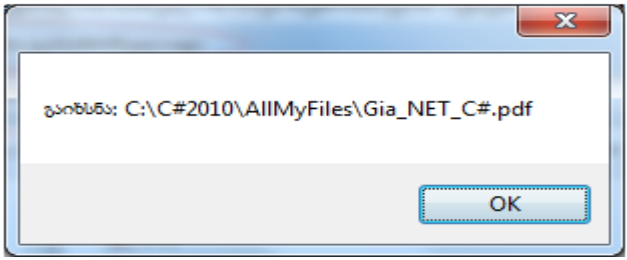
➢ **OpenFileDialog** კლასის ობიექტის დანიშნულებაა გასახსნელი ფაილის ამორჩევა, მითითებული ფოლდერის (InitialDirectory), ფილტრის (Filter - მაგალითად, ფაილის ტიპით და დიალოგური ველის სათაურის (Title) მიხედვით (ნახ.10.24,25).

დიალოგის შედეგი ფაილის სახელის თვისებისთვის იქნება - FileName.

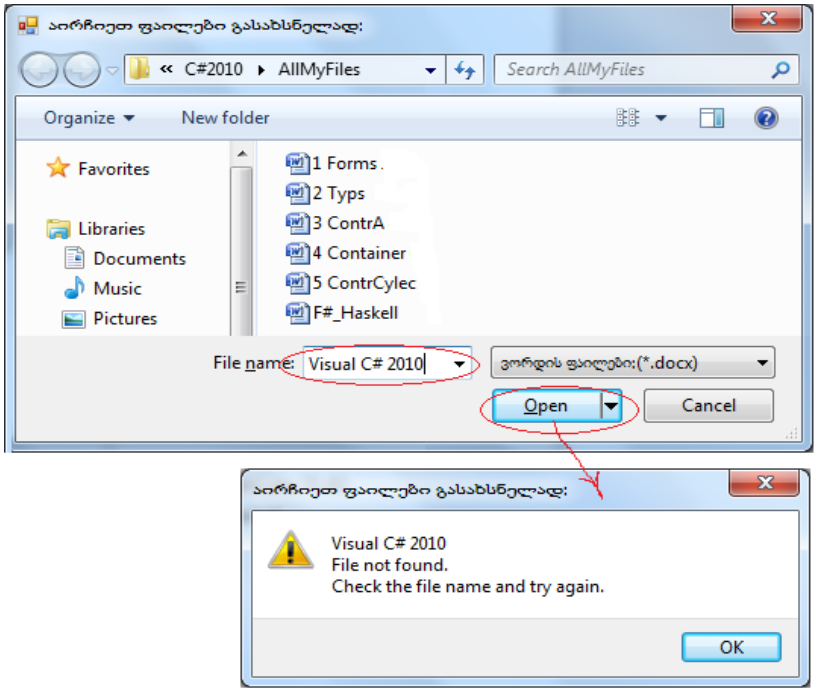
```
// ლისტინგი_10.7 --- OpenFileDialog -------
private void button1_Click(object sender, EventArgs e)
{
  OpenFileDialog f = new OpenFileDialog();
  f.InitialDirectory = "c:\\C#2010\\AllMyFiles";
  f.Filter = "წიგნები: (*.pdf)|*.pdf|"+
             "ვორდის ფაილები:(*.docx)|*.docx|" +
             "ყველა ფაილი: (*.*)|*.*";
  f.Title = "აირჩიეთ ფაილები გასახსნელად:";
  if (f.ShowDialog() == DialogResult.OK)
     MessageBox.Show("გაიხსნა: " + f.FileName);
   else
     MessageBox.Show("უშედეგო დასასრული !");
 }
```



ნახ.10.24-ა



ნახ.10.24-ბ. დადებითი შედეგით



ნახ.10.25. უშედეგოდ დასრულება

➢ **SaveFileDialog** კლასის ობიექტის დანიშნულებაა იმ ფაილის შეტანა ან ამორჩევა, რომელიც შენახულ უნდა იქნას. შენახვის და დიალოგის განსახორციელებლად მითითებულ უნდა იქნას ფოლდერის (InitialDirectory), ფილტრის (Filter - მაგალითად, ფაილის ტიპით და დიალოგური ველის სათაური (Title).

```
// ლისტინგი_10.8 --- SaveFileDialog -------
private void button2_Click(object sender, EventArgs e)
 {
   SaveFileDialog fs = new SaveFileDialog();
  fs.InitialDirectory = "c:\\C#2010\\AllMyFiles";
  fs.Filter = "წიგნები: (*.pdf)|*.pdf|" +
```

```
                "ვორდის ფაილები:(*.docx)|*.docx|" +
              "ყველა ფაილი: (*.*)|*.*";
    fs.Title = "ფაილების არჩევა შესანახად:";
    if (fs.ShowDialog() == DialogResult.OK)
       MessageBox.Show("შენახვა: " + fs.FileName);
      else
        MessageBox.Show("უშედეგო დასასრული !");
  }
```

➢ **FolderBrowserDialog** კლასის ობიექტის დანიშნულებაა კატალოგის (ფოლდერის) ამორჩევა, რომელიც იქნება მომდევნო პროგრამული პროცედურების საბაზო წერტილი. შესაძლებელი ასევე ახალი კატალოგის შექმნა. დიალოგური ფორმის გახსნის წინ საჭიროა შემდეგი პარამეტრების მიწოდება: RootFolder: კატალოგი, რომელიც უნდა გამოჩნდეს დიალოგის ველში.

ShowNewFolderButton: ახალი კატალოგის შექმნისათვის საჭირო ბუტონის მითითება. Description: დიალოგური ველის სათაური.

```
// ლისტინგი_10.9 --- FolderBrowserDialog -------
private void button3_Click(object sender, EventArgs e)
{
 FolderBrowserDialog fb = new FolderBrowserDialog();
 fb.RootFolder = Environment.SpecialFolder.MyDocuments;
 fb.ShowNewFolderButton = false;
 fb.Description = "კატალოგის არჩევა";
 if (fb.ShowDialog() == DialogResult.OK)
   MessageBox.Show("წვდომა კლატალოგზე: " +
                                    fb.SelectedPath);
  else
    MessageBox.Show("უშედეგო დასასრული !");
 }
```

სტრიქონით:

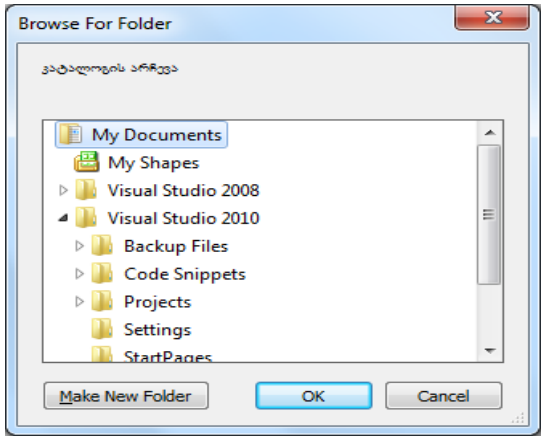
```
fb.RootFolder=Environment.SpecialFolder.MyDocuments;
```

- ფესვურ კატალოგად, ჩვენ შემთხვევაში აიღება „MyDocuments“.

სტრიქონით:

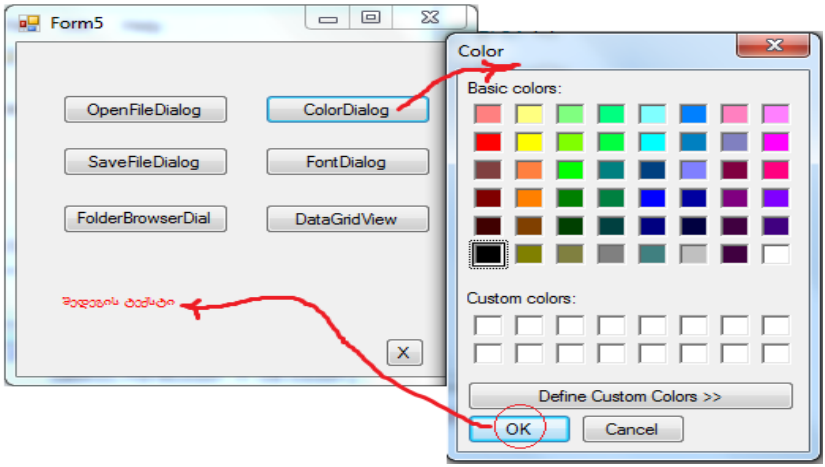
```
fb.ShowNewFolderButton = false;
```

- ახალი კატალოგი არ იქმნება, ხოლო თუ არის “true” , მაშინ ფორმაზე ჩნდება ბუტონი “Make New Folder” (ნახ.10.26).



ნახ.10.26

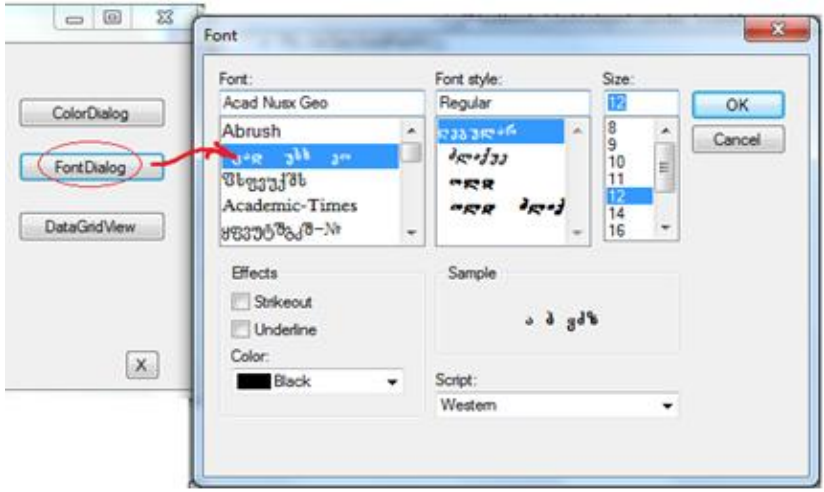
➢ ColorDialog კლასის ობიექტის დანიშნულებაა ფერის არჩევა ფორმაზე მოთავსებული მონიშნული ელემენტისთვის (ლისტინგი_10.10 და ნახ.10.27).



ნახ.10.27

```
// ლისტინგი_10.10 --- ColorDialog -------
private void button4_Click(object sender, EventArgs e)
 {
   ColorDialog cd = new ColorDialog();
   if (cd.ShowDialog() == DialogResult.OK)
      label1.ForeColor = cd.Color;
    else
      MessageBox.Show("უშედეგო დასასრული");
 }
```

➢ **FontDialog** კლასის ობიექტის დანიშნულებაა შრიფტის (ფონტის) არჩევა ფორმაზე მოთავსებული მონიშნული ელემენტისთვის (ლისტინგი_10.11 და ნახ.10.28).



ნახ.10.28

```
// ლისტინგი_10.11 --- FontDalog -------
private void button5_Click(object sender, EventArgs e)
 {
   FontDialog fd = new FontDialog();
   fd.ShowColor = true;
   fd.MaxSize = 22;
   fd.MinSize = 8;
   if (fd.ShowDialog() == DialogResult.OK)
```

```
        {
          label1.Font = fd.Font;
          label1.ForeColor = fd.Color;
        }
       else
          MessageBox.Show("უშედეგო  დასასრული");
   }
```

10.8. კითხვები და სავარჯიშოები:

10.1. რა სახის მენიუებს და დიალოგურ საშუალებებს იცნობთ ?

10.2. როგორ ვაპროგრამებთ მთავარ მენიუს ვიზუალური ელემენტების გამოყენებით ?

10.3. როგორ ხდება იერარქიული მენიუ-ქვემენიუების აგება ?

10.4. რა სახის შეიძლება იყოს მენიუს პუნქტი ?

10.5. როგორ ხდება მენიუს პუნქტებისთვის ”მოვლენის დამმუშავებლის” (მეთოდების) შექმნა ?

10.6. როგორ ხდება მენიუს პუნქტებიდან შრიფტების და ფერის არჩევა ?

10.7. როგორ ხდება ”შუქნიშნის” მუშაობის პროცესის მოდელირება სტატიკური და დინამიკური პრინციპებით ?

10.8. ააგეთ მთავარი მენიუ ვინდოუს აპლიკაციისთვის „ფაკულტეტი“: ჰორიზონტალური პუნქტებით - „ჯგუფები“, „საგნები“, „გამოცდები“, „Help“; აგრეთვე 3-4 ვერტიკალური პუნქტით თითოეულ ჰორიზონტალურში. მაგალითად, „ჯგუფები: -> სპეციალობები -> კურსები -> ჯგ_ნომრები“. ჯგუფის არჩევის შემდეგ გამოვა ეკრანზე ამ ჯგუფის სტუდენტთა გვარები და სხვა ინფორმაცია.

10.9. როგორ ხდება ToolStrip ვიზუალური ელემენტით პიქტოგრამის გამოყენება გრაფიკული მენიუს შესაქმნელად ?

10.10. რას არის კონტექსტური მენიუ და როგორ იქმნება იგი ?

10.11. როგორ იქმნება სტატუს-პანელი და როგორ უნდა გამოვიტანოთ მასზედ სისტემური თარიღი და პროგრამის მუშაობის ხანგრძლივობის ამსახველი პროგრეს-ველი ?

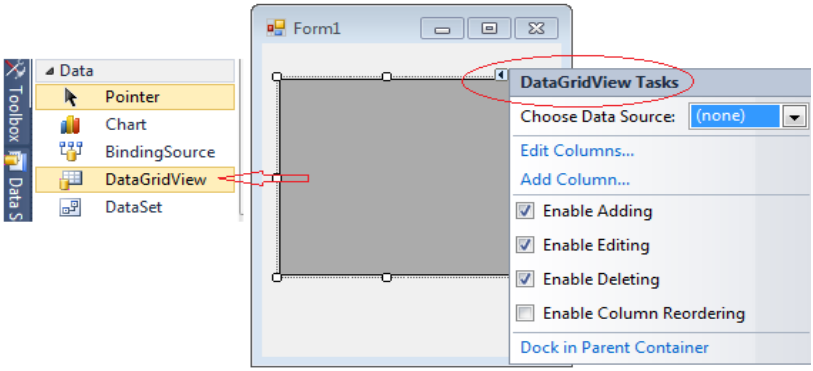
10.12. ჩამოთვალეთ და ახსენით მუშაობის პრინციპები ვიზუალური სტანდარტული დიალოგების საშუალებებისთვის.

11. ცხრილების წარმოდგენის მართვის ელემენტი - DataGridWiew

მარტივი სიებისა და მონაცემთა ერთგანზომილებიანი ველების ასახვის მიზნით გამოიყენება ListBox- და ComboBox-ები (იხ. თავი 6).

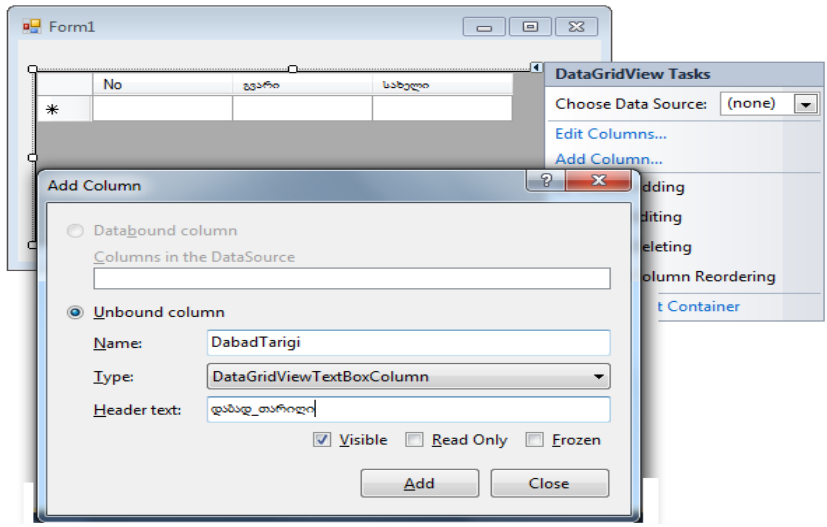
ცხრილები (Tables), რომლებიც სტრიქონებისა და სვეტებისგან შდგება, საუკეთესო საშუალებაა მონაცემთა ორგანზომილებიანი ველების, მასივების წარმოსადგენად. C# ენაში ასეთი ობიექტების ასახვის მიზნით გამოიყენება მართვის ელემენტი, ტიპით DataGridView. ამ ელემენტს, პრაგმატული თვალსაზრისით, დიდი გამოყენება აქვს მონაცემთა ბაზების სისტემებში (ADO.NET), რასაც ჩვენ მომდევნო თავში დავუბრუნდებით.

ახლა განვიხილოთ ცხრილებთან მუშაობის ვიზუალური საშუალებანი. 11.1 ნახაზზე მოცემულია ToolBox-დან Form1-ფორმაზე გადმოტანილი DataGridView ელემენტი და საწყისი სიტუაცია.



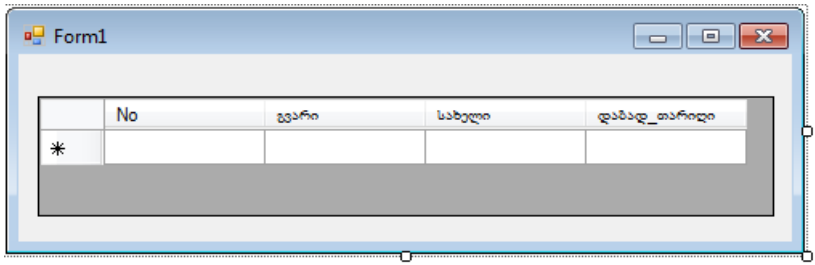
ნახ.11.1

დიალოგურ რეჟიმში ცხრილის ველების (სვეტების) შესატანად ვირჩევთ Add Columns და გადავდივართ 11.2 ნახაზზე მოცემულ ფანჯარაში. შვიტანოთ მიმდევრობით „სტუდენტები“-ს ატრიბუტები, მაგალითად, No, First_Name (გვარი), Last_Name (სახელი),Birth_data (დაბად_თარიღი) და ა.შ.



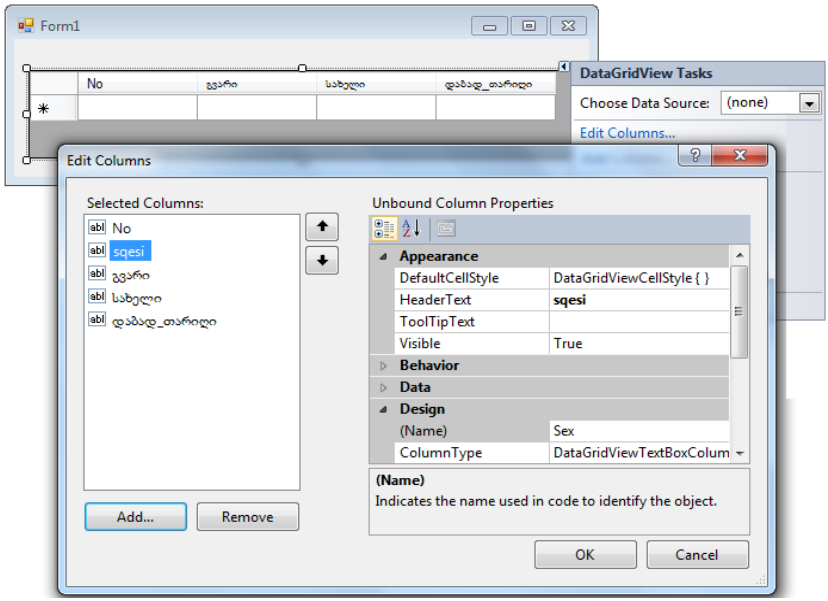
ნახ.11.2

პროგრამის ამუშავების შემდეგ მივიღებთ 11.3 ნახაზზე მოცემულ ცხრილს.

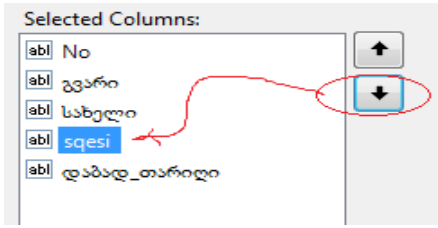


ნახ.11.3

შეტანილი ველების კორექტირებისათვის ვიყენებთ Edit Columns პუნქტს. ჩავამატოთ ველები Sex (სქესი) ან შვასწოროთ უკვე ჩაწერილი დასახელებები. მაგალითად, 11.3 ნახაზზე, ედიტორის ფანჯარაში გვინდა ველი sqesi შევცვალოთ ქართული შრიფტით და ამასთანავე იგი გადავიტანოთ ველი „სახელი“-ს შემდეგ. 11.4 და 5 ნახაზებზე ნაჩვენებია ეს შემთხვევები.

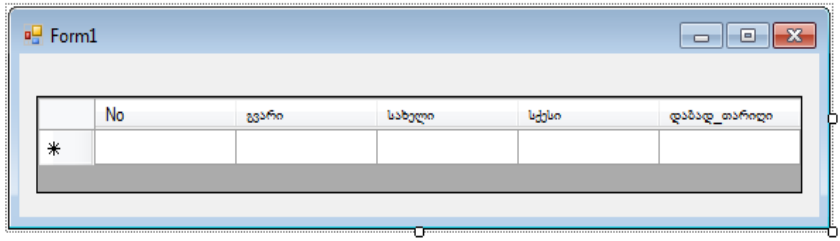


ნახ.11.4



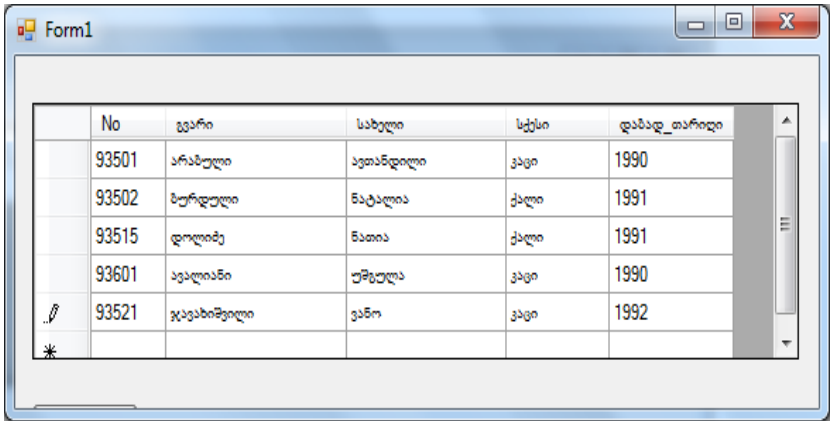
ნახ.11.5

კოდის მუშაობის ახალი შედეგი მოცემულია 11.6 ნახაზზე.



ნახ.11.6

აქვე შეიძლება მონაცემთა სტრიქონების (Rows) შეტანა ველების ქვეშ. მაგალითი ნაჩვენებია 11.7 ნახაზზე.



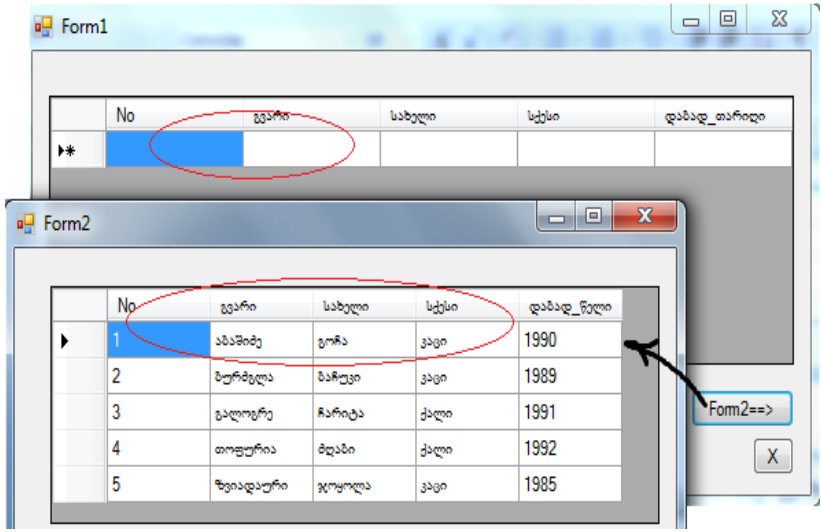
ნახ.11.7

პროგრამის დამთავრებისა და ხელახალი ამუშავების შემდეგ შეტანილი მონაცემები იკარგება, ანუ არაა შენახული მეხსიერებაში. თუ გვინდა, რომ პროექტის გაშვებისას მონაცემები ჩაიტვირთოს პროგრამულად, მაშინ ან უნდა გამოვიყენოთ მონაცემთა ბაზასათან კავშირი (იხ. მომდევნო თავი), ან კოდის Form2_Load მეთოდში უნდა ჩავწეროთ შემდეგი სტრიქონები (ლისტინგი 11_1).

```
   // ლისტინგი_11.1 --- DataGridView -------
private void Form2_Load(object sender, EventArgs e)
{
  int i;
  // ველების (სვეტების) შევსება ------
  dataGridView1.Columns.Add("No", "No");
  dataGridView1.Columns.Add("FirstName", "გვარი");
  dataGridView1.Columns.Add("LastName", "სახელი");
  dataGridView1.Columns.Add("Sex", "სქესი");
  dataGridView1.Columns.Add("Dab_Celi", "დაბად_წელი");
 // ველების სიგანის დაყენება -----
 for (i = 0; i < dataGridView1.Columns.Count; i++)
    dataGridView1.Columns[i].Width = 75;
```

```
  // სტრიქონების შევსება ----------
 dataGridView1.Rows.Add("1","აბაშიძე","გოჩა","კაცი","1990");
 dataGridView1.Rows.Add("2","ბურძგლა","ბაჩუკი","კაცი","1989");
 dataGridView1.Rows.Add("3","გალოგრე","ჩარიტა","ქალი","1991");
 dataGridView1.Rows.Add("4","თოფურია","ძღაბი","ქალი","1992");
 ataGridView1.Rows.Add("5","ზვიადაური","ჯოყოლა","კაცი","1985");
}
```

პროგრამის ამუშავებით მიიღება 11.8 ნახაზზე მოცემული სურათი.



ნახ.11.8

როგორც აღვნიშნეთ, კოდში ხისტადაა შეტანილი კონკრეტული მონაცემები სტუდენტების შესახებ. ნებისმიერი დამატება ან ცვლილება მოითხოვს პროგრამის გადაკეთებას, რაც არაა რეკომენდებული. ამის თავიდან აცილება შესაძლებელია მონაცემთა ბაზის გამოყენებით. შედეგიდან ჩანს, რომ Form1 ცარიელია, ხოლო Form2 შევსებულია საწყისი მონაცემებით.

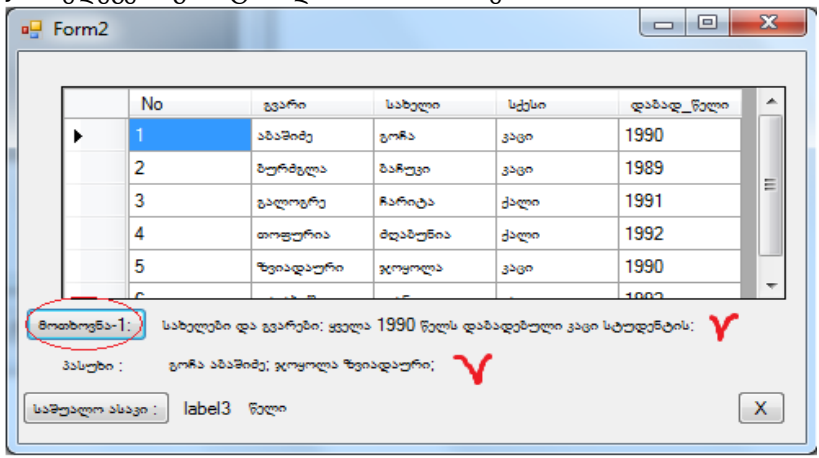
ახლა განვიხილოთ ჩვენი კოდის მაგალითით `DataGridView` ცხრილში შეტანილი მონაცემების საფუძველზე მომხმარებლის მოთხოვნების პროგრამული დამუშავების შესაძლებლობანი.

ამოცანა_11.1: ვიპოვოთ სახელები და გვარები ყველა 1990 წელს დაბადებული მამაკაცი სტუდენტის.

მოთხოვნის ფორმალური მხარე მდგომარეობს „გვარი“ ველის მნიშვნელობების გამობეჭდვაში, ველი „სქესი“=“კაცი“ მნიშვნელობისთვის.

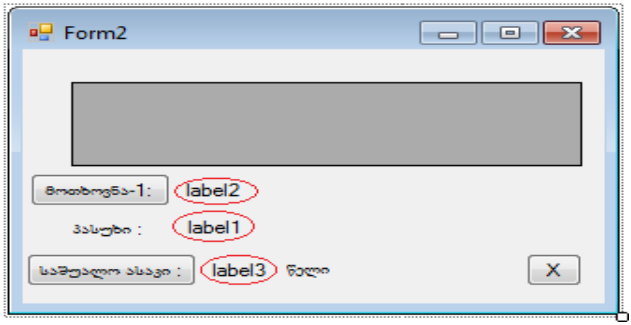
საჭიროა Form2-ზე დავდოთ ბუტონი და მივაბათ მას 11.2_ლისტინგის პროგრამული კოდი.

```
//ლისტინგი_11.2---DataGridView-ში სტრიქონების ამორჩევა პირობით ----
 private void button1_Click(object sender, EventArgs e)
 {
   label1.Text = " ";
   label2.Text = "სახელები და გვარები: ყველა 1990 წელს
                  დაბადებული კაცი სტუდენტის:";
for (int i = 0; i < dataGridView1.Rows.Count; i++)
   {
  if (dataGridView1.Rows[i].Cells[3].Value == "კაცი" &&
      dataGridView1.Rows[i].Cells[4].Value == "1990")
   {
    label1.Text += dataGridView1.Rows[i].Cells[2].Value
    + " " +dataGridView1.Rows[i].Cells[1].Value + "; ";
   }
 }
} // შედეგები გამოტანილია 11.9 ნახაზზე.
```



ნახ.11.9

ამოცანა_11.2: შევადგინოთ კოდი ღილაკისთვის „საშუალო ასაკი“, რომელიც გამოიტანს label3-ში ყველა სტუდენტის საშუალო ასაკის მნიშვნელობას (ნახ. 11.10). 11_3 ლისტინგზე მოცემულია ეს კოდი.

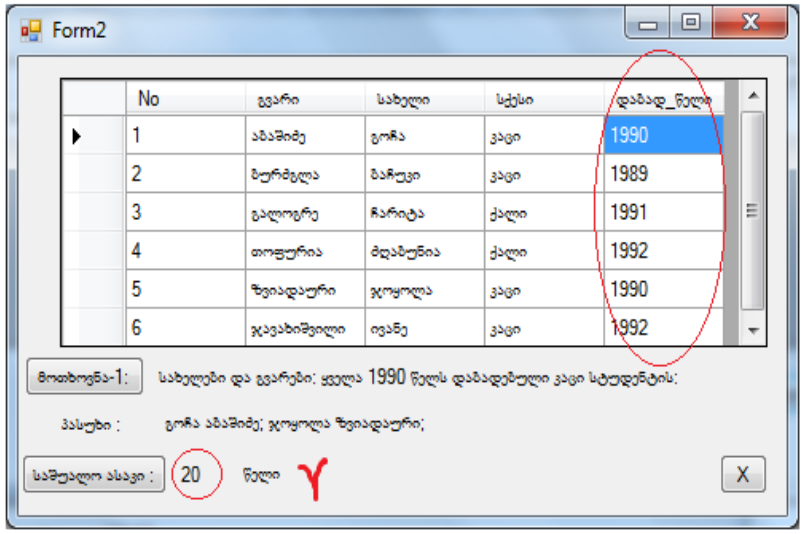


ნახ.11.10

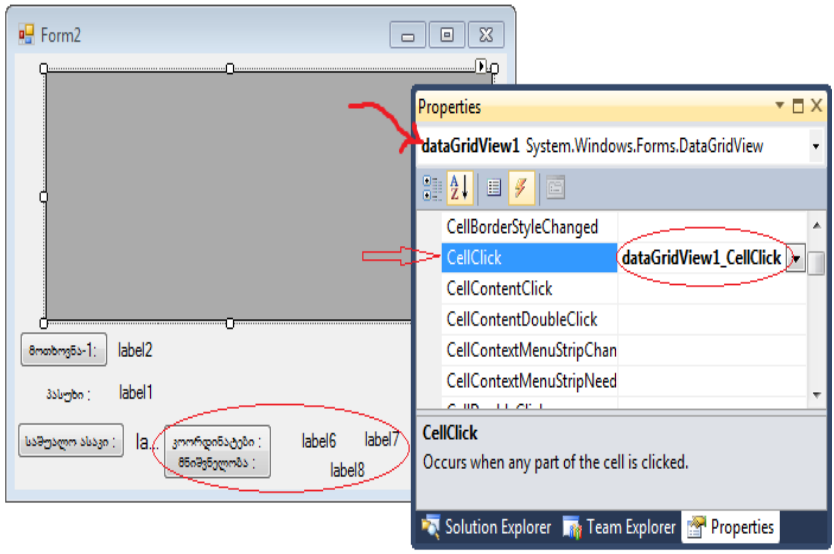
```
// ლისტინგი_11.3 --- DataGridView საშუალო ასაკის ანგარიში -------
private void button2_Click(object sender, EventArgs e)
 {
   int Birth_Year, Averag_Age, Sum=0;
   DateTime now = DateTime.Now; // მიმდინარე თარიღი
   int age,a,b;
   for (int i = 0; i < dataGridView1.Rows.Count; i++)
    {
      Birth_Year=Convert.ToInt32(dataGridView1.
                                 Rows[i].Cells[4].Value);
      a = now.Year; b = Birth_Year;  age = a - b;
      Sum += age;
    }
   Averag_Age = Sum / dataGridView1.Rows.Count;
   label3.Text = Averag_Age.ToString();
 }
```

შედეგები ასახულია 11.11 ნახაზზე.

ამოცანა_11.3: ავაგოთ კოდი, რომელიც იმუშავებს ცხრილთან. კერძოდ, მაუსის კურსორის ცხრილის რომელიმე უჯრაზე დაწკაპუნებით, ეკრანზე გამოიტანს ამ უჯრის სვეტის და სტრიქონის კოორდინატებს და შიგ მოთავსებულ მნიშვნელობას. მოვლენის შექმნა მოცემულია 11.12 ნახაზზე.



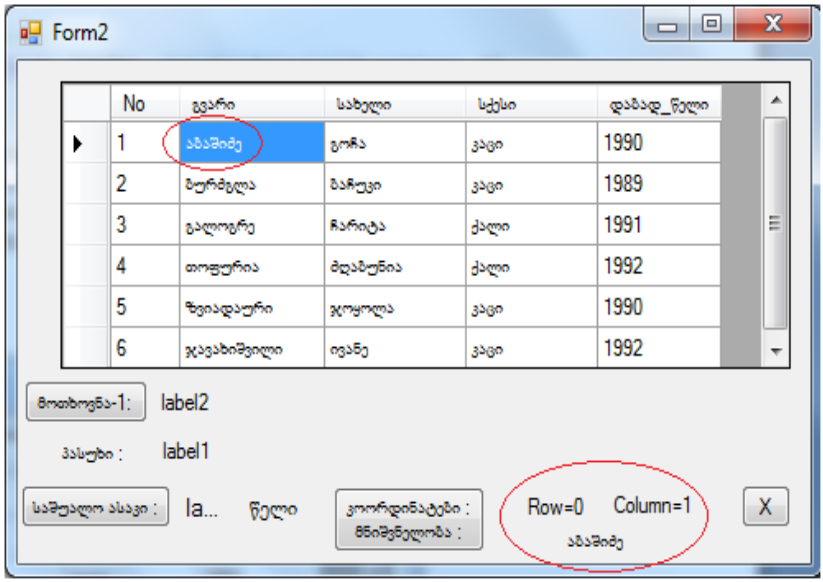
ნახ.11.11



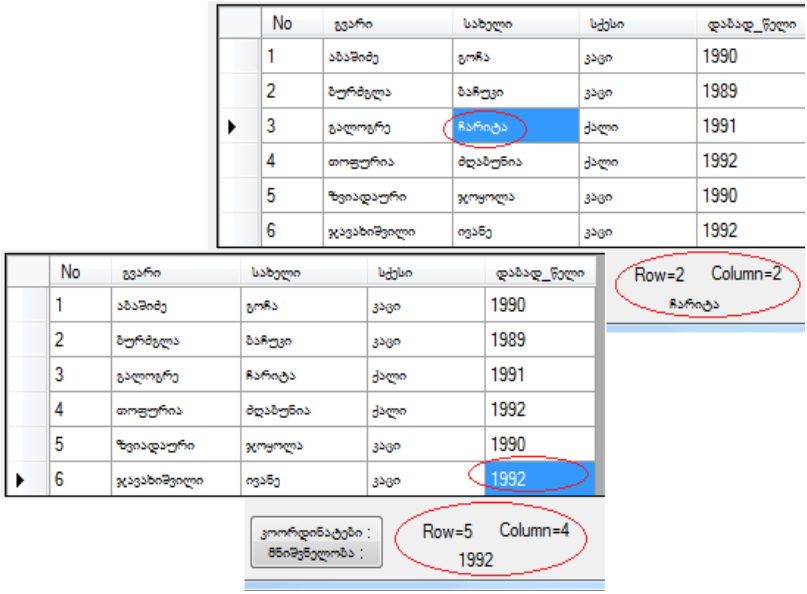
ნახ.11.12

```
// ლისტინგი_11.4  ---- ცხრილის კოორდინატები -----
private void dataGridView1_CellClick(object sender,
                                     DataGridViewCellEventArgs e)
 {
   label8.Text = "";
   label6.Text = "Row="+e.RowIndex.ToString();
   label7.Text = "Column="+e.ColumnIndex.ToString();
   if (e.RowIndex>=0 && e.ColumnIndex >=0)
     label8.Text += dataGridView1.Rows[e.RowIndex].
                                     Cells[e.ColumnIndex].Value;
 }
```

შდეგები ასახული 11.13-ა,ბ ნახაზზე.



ნახ.11.13-ა



ნახ.11.13-ბ

11. კითხვები და სავარჯიშოები:

11.1. როგორ იქმნება ცხრილი DataGridView ელემენტით ?

11.2. როგორ ხდება ცხრილში სვეტების და სტრიქონების ჩამატება ?

11.3. როგორ ხდება ცხრილში სვეტების და სტრიქონების წაშლა ?

11.4. როგორ ხდება ცხრილში სვეტების და სტრიქონების კორექტირება ?

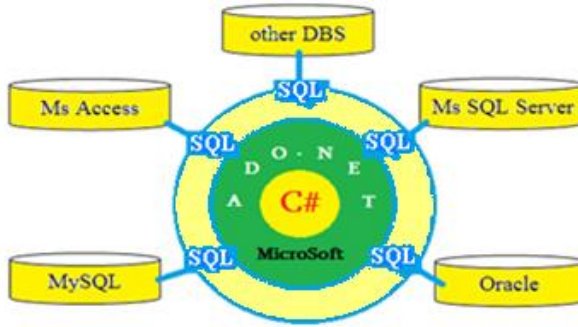
11.5. ააგეთ ცხრილი „წიგნები“, რომელსაც ექნება შემდეგი ველები: შიფრი, დასახელება, ავტორი, გამოცემის_წელი, გამომცემლობა, ქალაქი, ჟანრი, ენა, გვერდების_რაოდ., ფასი. შეიტანეთ 10 სტრიქონი და განახორციელეთ მასში წიგნის ძებნა ავტორით, წიგნების ძებნა ჟანრით, დაალაგეთ წიგნები გამოცემის წლის კლებადობით.

II ნაწილი

Visual C# და ADO.NET: მონაცემთა ბაზებთან კავშირი

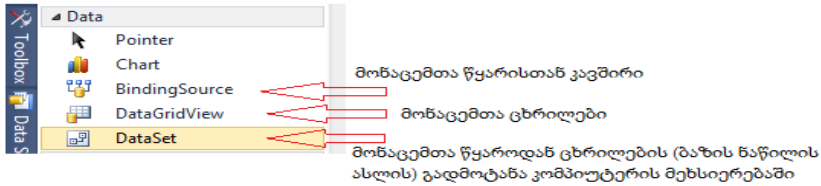
მომხმარებელთა პროგრამულ აპლიკაციებს (დანართებს) აუცილებლად ესაჭიროება ურთიერთქმედება მონაცემთა ცენტრალიზებულ ბაზებთან, მონაცემთა საცავებთან XML–ფორმატით (Extensible Markup Language), ან მონაცემთა ლოკალურ ბაზებთან, როდესაც ისინი მუშაობენ კლიენტის მანქანებზე.

Microsoft Visual Studio .NET Framework 4.0/4.5 პლატფორმა, რომელსაც ჩვენ ვიხილავთ C#-ენის საფუძველზე, მონაცემთა ბაზებთან სამუშაოდ იყენებს ADO.NET ტექნოლოგიას და SQL ენას (ნახ.12.1). პირველი დამაკავშირებელი დრაივერია C#-ენასა და ბაზებს შორის, მეორე კი - მომხმარებლის საკონტაქტო ენაა ბაზებთან, ე.წ. სტრუქტურირებულ მოთხოვნათა ენა [1,13].



ნახ.12.1. C# <-> ADO.NET <-> SQL <–> DBS

C# ენა .NET გარემოში მონაცემთა ბაზებთან სამუშაოდ გვთავაზობს შემდეგ კომპონენტებს (ნახ.12.2), რომელთა საფუძველია ADO.NET.



ნახ.12.2

ADO.NET - მაიკროსოფთის ტექნოლოგიაა (ActiveX Data Object), რომელიც გვთავაზობს მონაცემებთან მიმართვისათვის გამოსაყენებლად მარტივ, მაგრამ მეტად მძლავრ საშუალებებს. იგი უზრუნველყოფს სისტემის რესურსების მაქსიმალურად სრულ ურთიერთქმედებას [23-25]. წინამდებარე პარაგრაფში გავეცნობით:

- ADO.NET დრაივერის მონაცემებთან მიმართვის ძირითად კომპონენტებს;
- თითოეული კომპონენტის როლს ფუნქციონალობას;
- ADO.NET-ის მონაცემებთან მიმართვის ორგანიზაციის სცენარის აღწერას.
- C# პროგრამული აპლიკაციის კავშირს მონაცემთა რელაციური ბაზების მართვის სისტემებთან: Ms Access, MySQL, Ms SQL Server.

სხვადასხვა დანართები მონაცემებთან მიმართვის ორგანიზაციისათვის აყენებს სხვადასხვა მოთხოვნებს. მნიშვნელობა არა აქვს იმას, თუ რას აკეთებს დანართი: ასახავს ცხრილების შინაარსს, თუ გადაამუშავებს და განაახლებს მონაცემებს ცენტრალურ SQL-სერვერზე. ADO.NET აძლევს მომხმარებელს მონაცემებთან მიმართვის მარტივ და ეფექტურ საშუალებებს რეალიზაციის სხვადასხვა სცენარით.

12.1. გამოყოფილ მონაცემებთან მიმართვა

მონაცემებთან მიმართვის ტრადიციული ტექნოლოგიები, ჩვეულებრივად, ახორციელებდა მონაცემების წვდომას წყაროსთან მუდმივი მიერთების გზით.

ასეთი მოდელის გამოყენებისას პროგრამული აპლიკაცია გახსნის მონაცემთა ბაზასთან მიერთებას და არ დახურავს მას მუშაობის დამთავრებამდე. დანართის სირთულის ზრდასთან ერთად იზრდება მონაცემთა ბაზის კლიენტების რაოდენობაც, რაც არაეფექტურს ხდის ბაზასთან მუდმივი მიერთების ტექნოლოგიას, კერძოდ:

- სისტემური რესურსები გამოიყენება არაეფექტურად. რაც უფრო მეტი მუდმივად მიერთებული (გახსნილი) ფაილია, მით უფრო დაბალია სისტემის მწარმოებლურობა;

- დანართები, რომლებიც იყენებს მონაცემებთან მიმართვას მუდმივი მიერთების საშუალებით, ცუდად მასშტაბირებადია. ასეთი დანართები კარგად ემსახურება ორ მიერთებულ კლიენტს, 10-თან უკვე უჭირს მუშაობა და 100-თან საერთოდ ვერ ფუნქციონირებს.

ADO.NET სისტემაში ეს პრობლემები წყდება მონაცემებთან მიმართვის ისეთი მოდელის გამოყენებით, როგორიცაა **გამოყოფილი** მონაცემები. ასეთი მოდელის შემთხვევაში მონაცემთა წყაროსთან მიერთება გახსნილია მხოლოდ გარკვეული პროცედურების შესასრულებლად.

მაგალითად, თუ აპლიკაციას დასჭირდა მონაცემები ბაზიდან, იგი მიუერთდება მას ამ მონაცემების გადმოტვირთვამდე, შემდეგ კი მიერთება დაიხურება.

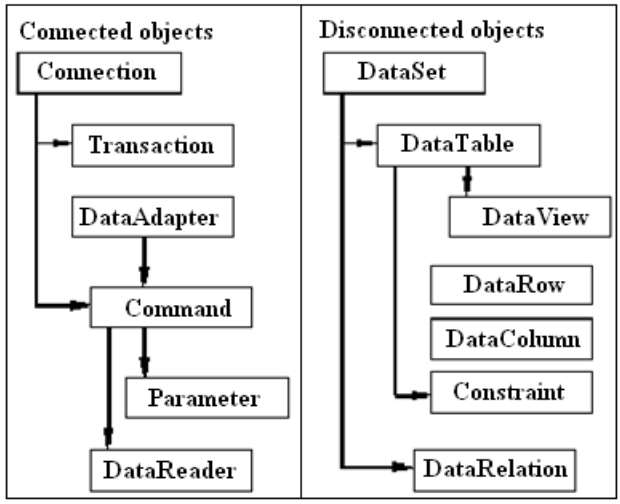
ასევე, როდესაც ხორციელდება მონაცემთა განახლება ბაზაში, მიერთება წყაროსთან განხორციელდება UPDATE-ბრძანების შესრულების დამთავრებამდე, შემდეგ იგი დაიხურება. ამგვარად, მონაცემებთან მიერთების დროის (გახსნა-დახურვის პერიოდი) შემცირებით, ADO.NET უზრუნველყოფს სისტემური რესურსების ეკონომიურ გამოყენებას და მონაცემთა წვდომის ინფრასტრუქტურის მასშტაბირებას, მწარმოებლურობის შემცირების გარეშე.

12.2. ADO.NET მონაცემთა არქიტექტურა

სისტემის ობიექტური მოდელი ორი ნაწილისგან შედგება: მარცხენა - მიერთებადი ობიექტები (Connected Objects) და მარჯვენა - განცალკევებადი ობიექტები (Disconnected Objects). 12.3 ნახაზზე ნაჩვენებია ADO.NET ობიექტური მოდელის შემადგენელი კლასები, რომელთა დანიშნულებასაც მოკლედ შევეხებით ამ პარაგრაფში.

მონაცემებთან მიმართვა ADO.NET-ში ხორციელდება ორი კომპონენტით:

- მონაცემთა ერთობლიობით (DataSet ობიექტით), რომელშიც მონაცემები ინახება ლოკალურ კომპიუტერში;
- მონაცემთა მიმწოდებლით (DataProvider პროვაიდერით), რომელიც ასრულებს შუამავლის ფუნქციას პროგრამასა და მონაცემთა ბაზას შორის.



ნახ.12.3

ობიექტი DataSet: ესაა მონაცემთა წარმოდგენა კომპიუტერის მეხსიერებაში მონაცემთა წყაროსგან იზოლირებულად. ეს ობიექტი შეიძლება განვიხილოთ, როგორც მონაცემთა ბაზის ფრაგმენტის ლოკალური ასლი (კოპიო).

DataSet-ში მონაცემთა ჩატვირთვა შესაძლებელია ნებისმიერი დასაშვები წყაროდან, მაგალითად, SQL Server, Ms Access ბაზებიდან ან XML–ფაილიდან. დასაშვებია მეხსიერებაში ამ მონაცემებით მანიპულირება, აგრეთვე მათი განახლება მთავარი წყაროსაგან დამოუკიდებლად.

ობიექტი DataSet შედგება DataTable ობიექტთა ერთობლიობისგან (ის შეიძლება ცარიელიც იყოს, ანუ არ შეიცავდეს არც ერთ DataTable-ს).

ყოველი DataTable ობიექტი კომპიუტერის მეხსიერებაში ასახავს ერთ ცხრილს. მისი სტრუქტურა შეიცავს ორ ერთობლიობას: DataColumns, რომელშიც თავსდება ცხრილის სვეტები, და ცხრილის შეზღუდვათა ერთობლიობა. ეს ორი ერთობლიობა ქმნის ცხრილის სქემას.

DataTable ობიექტი შეიცავს აგრეთვე DataRows ერთობლიობას, რომელშიც ინახება DataSet-ობიექტის მონაცემები.

გარდა ამისა, DataSet ობიექტი შეიცავს DataRelations ერთობლიობას, რომელიც უზრუნველყოფს კავშირების შექმნას სხვადასხვა ცხრილის სტრიქონებს შორის. DataRelations შეიცავს DataRelation ობიექტთა ერთობლიობას, რომლებიც განსაზღვრავს ცხრილთაშორის კავშირებს (მაგალითად, 1:M კავშირის სარეალიზაციოდ).

და ბოლოს, DataSet ობიექტი შეიცავს ExtendedProperties ერთობლიობას, რომელშიც შეინახება დამატებითი მონაცემები.

მონაცემთა პროვაიდერი: ესაა ურთიერთდაკავშირებულ კომპონენტთა ერთობლიობა, რომელიც უზრუნველყოფს ეფექტურ მაღალმწარმოებლურ კავშირს მონაცემთა ბაზასთან.

.NET Framework-ს აქვს ორი პროვაიდერი: SQL Server .NET Data Provider, რომელიც შექმნილია SQL Server 7.0 ან უფრო მაღალ ვერსიებთან სამუშაოდ, და OleDb .NET Data Provider - სხვა ტიპის მონაცემთა ბაზებთან დასაკავშირებლად.

მონაცემთა ნებისმიერი პროვაიდერი შედგება მსგავსი უნივერსალური კლასების კომპონენტებისგან:

- Connection, რომელიც უზრუნველყოფს მონაცემთა ბაზასთან მიერთებას;

- Command, რომელიც გამოიყენება მონაცემთა წყაროს სამართავად. იგი გამოიყენებს ბრძანებებს, რომლებიც არ აბრუნებს მონაცემებს, მაგალითად, INSERT, UPDATE და DELETE, ან

ბრძანებებს, რომლებიც აბრუნებს DataReader ობიექტს (მაგალითად, SELECT);

- DataReader გამოიყენება მხოლოდ ჩანაწერთა ერთობლიობის წასაკითხად მიერთებული მონაცემთა წყაროდან;
- DataAdapter შეავსებს გამოყოფილ DataSet ან DataTable ობიექტს და განაახლებს მათ შედგენილობას.

მონაცემებთან მიმართვა ხორციელდება შემდეგნაირად: ობიექტი Connection აყენებს დანართის (აპლიკაციის) მონაცემთა ბაზასთან მიერთებას, რომელიც პირდაპირ მისაწვდომია Command და DataAdapter ობიექტებისთვის. Command ობიექტი უზრუნველყოფს ბრძანებათა შესრულებას უშუალოდ მონაცემთა ბაზაში. თუ შესასრულებელი ბრძანება აბრუნებს რამდენიმე მნიშვნელობას, მაშინ Command ხსნის მათთან მიმართვას DataReader ობიექტის საშუალებით. მიღებული შედეგები შესაძლებელია დამუშავდეს უშუალოდ დანართის კოდით, ან DataSet ობიექტით, რომელიც შეივსება DataAdapter ობიექტის დახმარებით. მონაცემთა ბაზის განახლებისთვის ასევე გამოიყენება Command და DataAdapter ობიექტები.

ობიექტი Connection გვთავაზობს მიერთებას მონაცემთა ბაზასთან. Visual Studio .NET –ს აქვს Connection-ის ორი კლასი: SqlConnection (MsSQL_Server-თან შესაერთებლად) და OleDbConnection (სხვა ტიპის მონაცემთა ბაზებთან დასაკავშირებლად). მონაცემთა ბაზასთან კავშირის არხის გასახსნელი აუცილებელი მონაცემები ინახება Connection ობიექტის ConnectionString თვისებაში. ეს ობიექტი ინახავს აგრეთვე რიგ მეთოდებს, რომლებიც საჭიროა მონაცემთა დასამუშავებლად ტრანზაქციების გამოყენებით.

ობიექტს Command აქვს ორი კლასი: SqlCommand და OleDbCommand. იგი უზრუნველყოფს ბრძანებათა გამოყენებას მონაცემთა ბაზაზე, რომელთანაც დამყარებულია კავშირი (მიერთება). აქ შეიძლება გამოყენებულ იქნას შენახვადი პროცედურები (Stored Procedures), SQL-ენის ბრძანებები, აგრეთვე ოპერატორები მთლიანი ცხრილების მისაღებად. Command ობიექტს აქვს სამი მეთოდი :

- Execute Non Query: იყენებს ბრძანებებს, რომლებიც არ აბრუნებს მონაცემებს, მაგალითად, INSERT, UPDATE და DELETE;

- Execute Scalar: იყენებს მოთხოვნებს მონაცემთა ბაზისადმი, რომლებიც აბრუნებს მხოლოდ ერთ მნიშვნელობას;

- Execute Reader: აბრუნებს საშედეგო ერთობლიობას DataReader ობიექტის საშუალებით.

ობიექტი DataReader გვთავაზობს ნაკადს მონაცემთა ბაზის ჩანაწერების ერთობლიობით, ოღონდ მხოლოდ ერთი მიმართულებით წასაკითხად. მონაცემთა პროვაიდერის სხვა კომპონენტებისგან განსხვავებით DataReader-ის ეგზემპლარების შექმნა პირდაპირ არაა დასაშვები. მისი მიღება შეიძლება Command ობიექტის ExecuteReader მეთოდებით:

- SqlCommand.ExecuteReader მეთოდი აბრუნებს SqlDataReader ობიექტს;

- OleDbCommand.ExecuteReader მეთოდი კი - OleDbDataReader ობიექტს.

თუ DataReader ობიექტის შემცველი მონაცემების ჩაწერა დისკზე არაა საჭირო, მაშინ ეს სტრიქონები შეიძლება პირდაპირ გადაეგზავნოს დანართს. ვინაიდან დროის ნებისმიერ მომენტში მეხსიერებაში იმყოფება მხოლოდ ერთი სტრიქონი, DataReader ობიექტის გამოყენება თითქმის არ ამცირებს სისტემის მწარმოებლურობას, ოღონდ მოითხოვს მონოპოლურ მიმართვას გახსნილ Connection-ობიექტზე DataReader ობიექტის სასიცოცხლო დროის განმავლობაში.

ობიექტი DataAdapter არის ADO.NET–ის ძირითადი კლასი, რომელიც უზრუნველყოფს გამოყოფილ მონაცემებთან მიმართვას. არსებითად, იგი ასრულებს შუამავლის ფუნქციებს მონაცემთა ბაზისა და DataSet-ობიექტის ურთიერთ-ქმედებისთვის.

Fill მეთოდის გამოძახებისას DataAdapter ობიექტი შეავსებს მონაცემებით DataTable-ს ან DataSet-ს მონაცემთა ბაზიდან. მონაცემების დამუშავების შემდეგ, რომლებიც ჩატვირთულია მეხსიერებაში, შესაძლებელია მოდიფიცირებული ჩანაწერების

მოთავსება მონაცემთა ბაზაში, DataAdapter ობიექტის Update მეთოდის გამოძახებით. DataAdapter-ს აქვს ოთხი თვისება, რომლებიც წარმოადგენს მონაცემთა ბაზის ბრძანებებს:

- SelectCommand შეიცავს ტექსტს ან ბრძანების ობიექტს, რომელიც ახორციელებს მონაცემთა ბაზიდან ამორჩევას (მაგალითად, მეთოდი Fill);
- InsertCommand შეიცავს ტექსტს ან ბრძანების ობიექტს, რომელიც ახორციელებს სტრიქონის ჩასმას ცხრილში;
- DeleteCommand შეიცავს ტექსტს ან ბრძანების ობიექტს, რომელიც ახორციელებს სტრიქონის წაშლას ცხრილიდან;
- UpdateCommand შეიცავს ტექსტს ან ბრძანების ობიექტს, რომელიც ახორციელებს მნიშვნელობათა განახლებას მონაცემთა ბაზაში;

Update მეთოდის გამოძახებისას ყველა შეცვლილი მონაცემი კოპირდება DataSet ობიექტიდან მონაცემთა ბაზაში, შესაბამისი ბრძანებების InsertCommand, DeleteCommand ან UpdateCommand გამოყენებით.

12.3. მონაცემთა ბაზასთან მიერთება

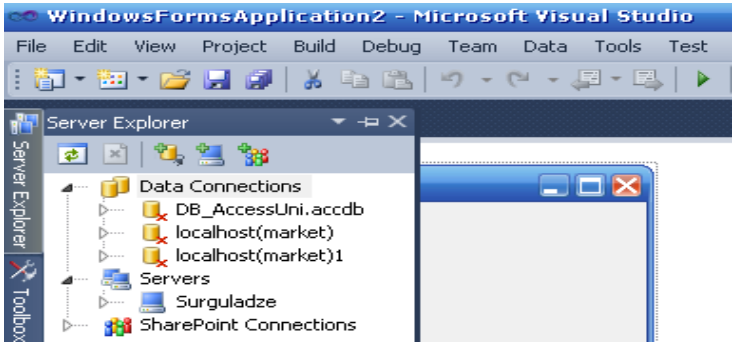
Visual Studio .NET სისტემას აქვს სტანდარტული ოსტატი პროგრამებისა და დიზაინერების სიმრავლე, რომელთა საშუალებითაც ადვილად და ეფექტურად ხორციელდება მონაცემებთან წვდომის არქიტექტურა დანართების დამუშავების პროცესში. ამასთანავე ADO.NET ობიექტური მოდელის ყველა შესაძლებლობა მისაწვდომია პროგრამულად, რაც უზრუნველ-ყოფს არასტანდარტული ფუნქციების რეალიზაციის ან დანართების აგების შესაძლებლობას, რომლებიც მომხმარებელთა მოთხოვნილებებზეა ორიენტირებული.

აქ ჩვენ გავეცნობით, თუ როგორ დავუკავშირდეთ მონაცემთა ბაზას ADO.NET–ის გამოყენებით, როგორ ამოვიღოთ საჭირო მონაცემები და გადავცეთ ისინი პროგრამულ აპლიკაციას. ეს საკითხები შეიძლება შესრულდეს Visual Studio .NET-ის გრაფიკული ინსტრუმენტებითაც და პროგრამულადაც.

C# პროგრამულ აპლიკაციაში არსებობს მონაცემთა ბაზასთან მიერთების რამდენიმე ხერხი. ყველაზე მარტივია ამის განხორციელება Visual Studio .NET-ის გრაფიკული ინსტრუმენტით. მონაცემთა წყაროსთან (DataSource) მიერთებისა და მისი მართვისათვის გამოიყენება ფანჯარა Server Explorer.

ძირითადი ამოცანა, რომელსაც ჩვენ აქ განვიხილავთ, არის ADO.NET პროგრამული პაკეტის გამოყენებით მომხმარებელთა სამუშაო ინტერფეისის დამუშავების სადემონსტრაციო მაგალითის აგება. ამასთანავე, მონაცემთა ბაზების სახით უნდა გამოვიყენოთ Ms_Access, MySQL და Ms_SQL_Serever პაკეტებით აგებული ცხრილები.

დავუშვათ, რომ ზემოაღნიშნული მონაცემთა ბაზების მართვის სისტემები დაინსტალირებულია ჩვენს კომპიუტერზე. თვით Visual Studio .NET -ის დაინსტალირებისას ავტომატურად ყენდება Ms SQL Server Expression, რომელზეც ასევე შესაძლებელია ექსპერიმენტის ჩატარება. .NET სამუშაო გარემოს ჩატვირთვის შემდეგ საჭიროა Server Explorer-ის გახსნა და ბაზებთან კავშირის შემოწმება (მაგალითად, ნახ.12.4).

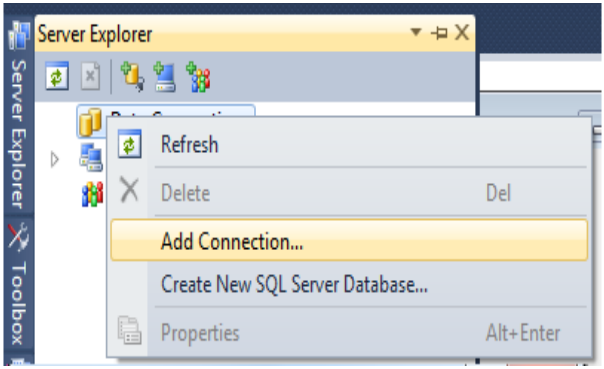


ნახ.12.4

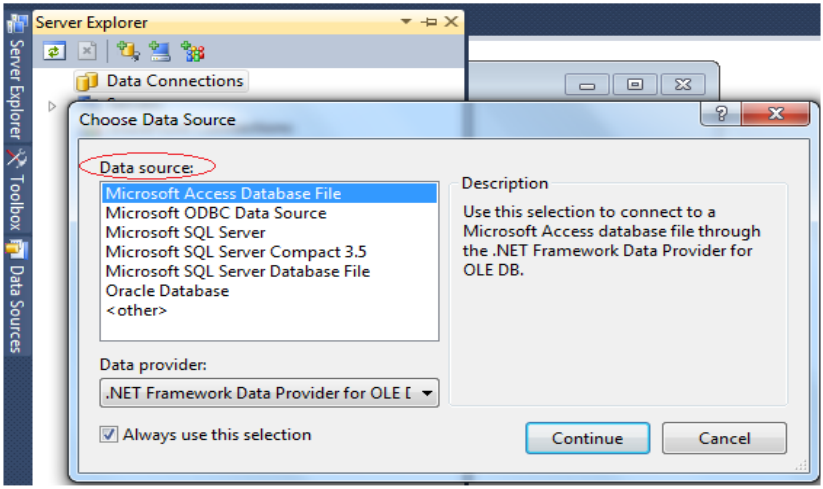
განვიხილოთ მონაცემთა ბაზებთან მუშაობის საკითხები.

12.5. C# და Ms Access

სისტემის მენიუდან View | Server Explorer-ით გამოვიტანოთ ფანჯარა (ნახ.12.5) და ავირჩიოთ Add Connection.



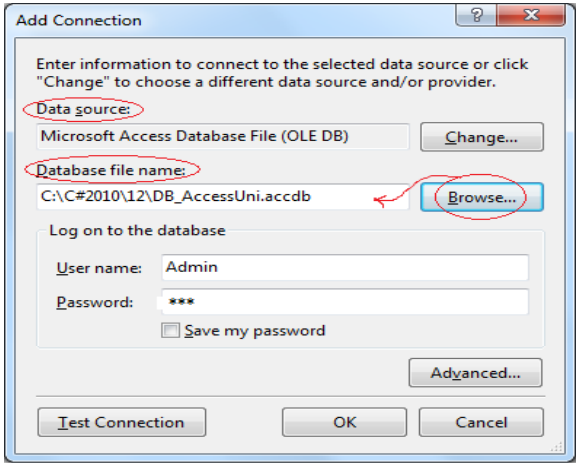
ნახ.12.5



ნახ.12.6

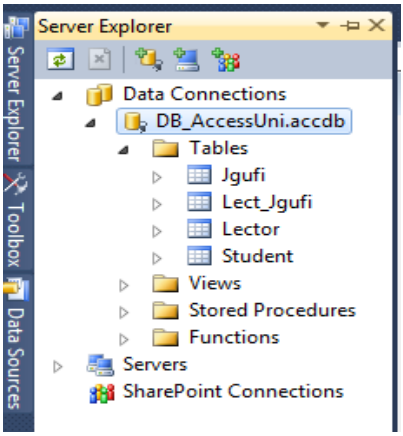
12.6 ნახაზზე Data Source ველში ავირჩიოთ სტრიქონი Microsoft Access Database File და Continue. მივიღებთ 12.7 ფანჯარას, რომელშიც Browse ღილაკით გამოვიძახებთ კატალოგის მართვის

დიალოგის ფანჯარას და მივუთითებთ ჩვენს მიერ წინასწარ მომზადებულ Ms Access-ის ბაზის ფაილს. მაგალითად, როგორც ეს Database file name ველშია ჩაწერილი.



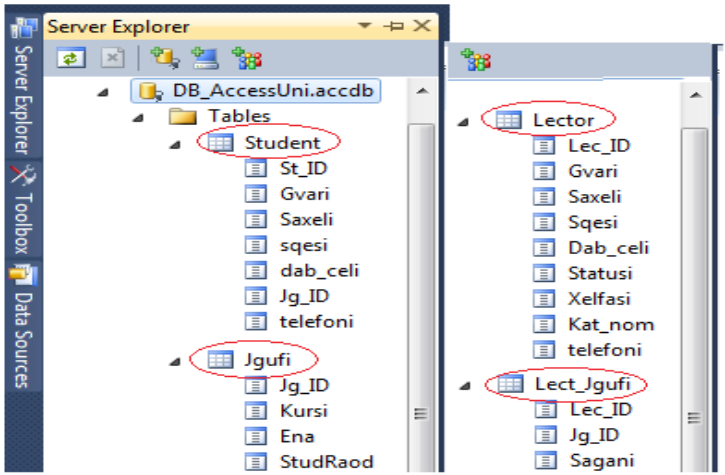
ნახ.12.7

აქვე, საჭიროების შემთხვევაში, მიეთითება User name და Password. ამის შემდეგ Server Explorer-ში გამოჩნდება 12.8 ნახაზზე მოცემული სურათი.



ნახ.12.8

როგორც ვხედავთ, Data Connection-ში უნივერსიტეტის მონაცემთა ბაზის ფაილი - **DB_AccessUni.accdb** გამოჩნდა, რომელიც შედგება ოთხი ცხრილისგან: Jgufi, Lect_Jgufi, Lector და Student. ცხრილები შედგება ველებისგან, რომელთაგან ერთ-ერთი გასაღებურია (ინდექსი): Lec_ID, St_ID, Jg_ID და ერთიც შედგენილი გასაღებია ორი ატრიბუტით: Lec_ID+Jg_ID (ნახ.12.9).

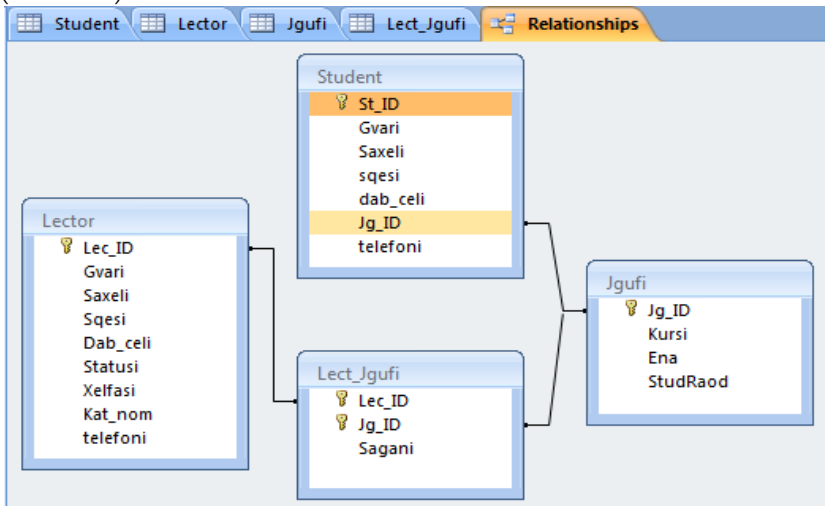


ნახ.12.9

რელაციური კავშირები მათ საფუძველზეა აგებული (ნახ.12.10).

ცხრილში **სტუდენტი** პირველადი გასაღებური ატრიბუტია St_ID ინდექსი, ხოლო მისი მეორადი გასაღებია Jg_ID, რომლითაც იგი უკავშირდება ცხრილს **ჯგუფი,** პირველადი ინდექსით Jg_ID. ესაა კავშირი 1:N, რომელიც ასახავს ბიზნეს-წესს (არსებულ კანონზომიერებას), რომ ერთი სტუდენტი შეიძლება იყოს მხოლოდ ერთ ჯგუფში და ერთ ჯგუფში შეიძლება იყოს რამდენიმე (N) სტუდენტი. ასევე, **ლექტორი** ასწავლის რამდენიმე (N) ჯგუფს, მაგრამ ჯგუფსაც ჰყავს რამდენიმე (M) ლექტორი. ესაა M:N კავშირი. მისი რეალიზაცია არაა შესაძლებელი **ლექტორი**-ს და **ჯგუფი**-ს პირდაპირი კავშირით (განმეორებადი ველების პრობლემა !).

ამისათვის შემოტანილია დამატებითი ცხრილი (რელაცია) **ლექტორი_ჯგუფი.** მასში შედგენილი ინდექსი იქმნება Lec_ID+Jg_ID, რომლებიც უკავშირდება ცალკ-ცალკე **ლექტორს** და **ჯგუფს** (ნახ.12.10).



ნახ.12.10

Student | Lector | Jgufi | Lect_Jgufi | Relationships

St_ID	Gvari	Saxeli	sqesi	dab_cel	Jg_ID	telefoni
8	აკოფოვი	რობერტინო	კაცი	1989	108936	297-11-11-11
1	ალავიძე	ალეკო	კაცი	1990	108935	222-22-20
2	ბურდული	ნინო	ქალი	1991	108935	137-33-33
3	ბურძგლა	დიტო	კაცი	1989	108935	599-10-20-20
4	გაბედავა	ვახტანგ	კაცი	1992	108935	333-67-89
5	გაბელია	ცუცა	ქალი	1992	108935	222-45-67
13	გალოგრე	ხვიჩა	კაცი	1989	108937	270-44-44
6	დაბელია	მიმოზა	ქალი	1990	108935	577-44-44-45
9	დოლიძე	რიჩარდი	კაცი	1990	108936	597-34-56-78
14	დუნდუა	გოჩა	კაცი	1991	108937	599-22-33-44
15	ვასაძე	სოკრატე	კაცი	1985	108937	577-33-67-55
10	ზარანდია	მუშნი	კაცი	1980	108936	597-12-23-34
11	თოფურია	ძღაბი	ქალი	1991	108936	577-10-10-10
12	კეკელია	კეკელა	ქალი	1992	108936	579-30-30-30
7	ხვიტია	ძუკუ	კაცი	1992	108935	593-45-67-89
16	ჯალაღონია	მაცი	კაცი	1992	108937	577-99-00-00

ნახ.12.11-ა

Student | Lector | Jgufi | Lect_Jgufi | Relationships

Lec_ID	Gvari	Saxeli	Sqesi	Dab_cel	Statusi	Xelfasi	Kat_nom	telefoni
6	ბარათელი	ანი	ქალი	1970	ასოც.პროფესორი	600	94	599-23-23-23
7	კუცია	თეა	ქალი	1987	ლაბორანტი	280	94	577-12-13-14
8	გაბედავა	ომიკო	კაცი	1950	სრ.პროფესორი	900	94	577-33-55-22
9	დვალი	დავითი	კაცი	1950	სრ.პროფესორი	900	86	599-99-90-90
10	ფიფია	კოჩი	კაცი	1960	ასოც.პროფესორი	650	86	233-33-46
11	მეფარია	თვარისა	ქალი	1980	ას.პროფესორი	450	94	593-55-55-55
12	ნიწიძე	ლია	ქალი	1980	ასოც.პროფესორი	600	94	577-78-89-90
13	ოდიშარია	ოდიშა	კაცი	1956	ას.პროფესორი	450	86	236-37-38
14	სამხარაძე	ვახშამა	კაცი	1970	ასოც.პროფესორი	690	51	577-88-99-00

ნახ.12.11-ბ

Student | Lector | Jgufi | Lect_Jgufi | Relatio

Jg_ID	Kursi	Ena	StudRaod
108050	2	ქართული	29
108051	2	ქართული	30
108059	2	რუსული	12
108835	4	ქართული	29
108836	4	ქართული	25
108935	3	ქართული	28
108936	3	ქართული	30
108937	3	ქართული	17
108940	3	ინგლისური	20

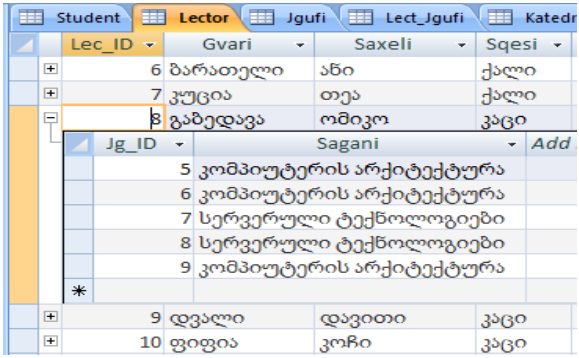
ნახ.12.11-გ

Student | Lector | Jgufi | Lect_Jgufi | Relatio

Lec_ID	Jg_ID	Sagani
5	5	კომპიუტერის არქიტექტურა
8	6	კომპიუტერის არქიტექტურა
8	7	სერვერული ტექნოლოგიები
8	8	სერვერული ტექნოლოგიები
8	9	კომპიუტერის არქიტექტურა
9	7	მათემატიკა
9	8	მათემატიკა
10	12	მონაცემთა ბაზები
10	13	მონაცემთა ბაზები

ნახ.12.11-დ

ჩანაწერის წინ „+” სიმბოლო ხსნის კავშირს მეორე, იერარქიულად დაქვემდებარებულ ცხრილთან (ნახ.12.12).



ნახ.12.12

ამგვარად, მონაცემთა ბაზა DB_AccessUni.acce მზადაა. ახლა განვიხილოთ C# პროგრამიდან მონაცემთა ბაზასთან წვდომის საკითხი. ეს პროცესი შედგება ოთხი ბიჯისგან:

- მონაცემთა ბაზასთან მიერთება (რაც ზემოთ განვიხილეთ Server Explorer->Data Connection-ში);
- SQL-ბრძანების გადაცემა მონაცემთა ბაზაზე;
- SQL-ბრძანების შეფასება (და შესრულება);
- მონაცემთა ბაზასთან კავშირის დახურვა.

SQL-ბრძანება წარმოადგენს სტრუქტურირებული მოთხოვნების ენაზე დაწერილ სკრიპტს, რომელიც გასაგებია მონაცემთა ბაზების მართვის სისტემისთვის და ასრულებს მას. ძირითადად, არსებობს ორი ტიპის მოთხოვნა:

- select : მონაცემთა ამორჩევის SQL-ბრძანება;
- insert, delete, update : მონაცემთა ბაზაში ცვლილებების განსახორციელებელი SQL-ბრძანებები.

C# პროგრამულ პროექტს მონაცემთა ბაზასთან სამუშაოდ სჭირდება სახელსივრცე OleDb, რომელიც using.System.Data.OleDB ბრძანებითაა რეალიზებული.

SQL-ბრძანებები Ms_Access ბაზაში გადაიგზავნება OleDb სახელსივრცის OleDBCommand კლასის ობიექტით. ამ კლასის ორი მნიშვნელოვანი თვისებაა: Connection (დავალება ბაზასთან დასაკავშირებლად, საითაც გაიგზავნება SQL-მოთხოვნა) და CommandText (თვით SQL ბრძანების ტექსტი).

მოთხოვნის ტიპებისგან დამოკიდებულებით (არჩევითი, ცვლილებების), OleDBCommand კლასს გააჩნია შემდეგი მეთოდები:

ExecuteReader() - აქვს select მოთხოვნის გაგზავნის და შედეგების მიღების ფუნქცია;

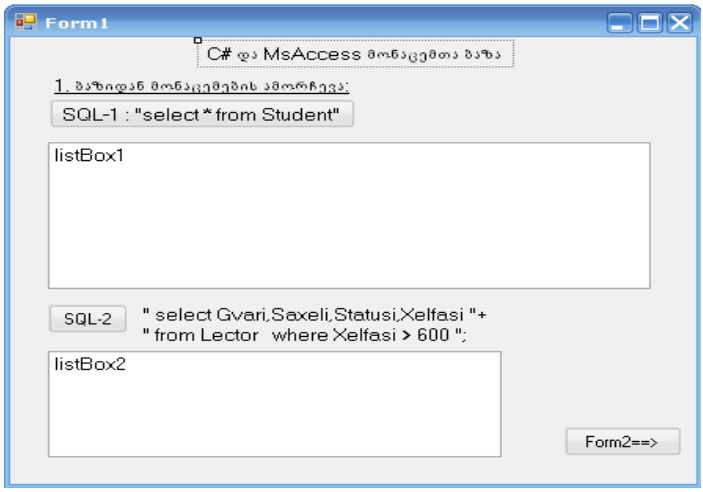
ExecuteNonQuery() - ემსახურება ცვლილების (insert, delete, update) აქციის ჩატარებას და რიცხვის მიღებას, რომელიც გვიჩვენებს, თუ მონაცემთა ბაზის რამდენ ჩანაწერს შეეხო ეს პროცედურა.

მონაცემთა ბაზიდან select-მოთხოვნით ამორჩეული ჩანაწერები (ველებით და მნიშვნელობებით) ინახება OleDb სახელსივრცის OleDBReader კლასის ობიექტში. განვიხილოთ კონკრეტული მაგალითი.

ამოცანა_12.1: DB_AccessUni.acce უნივერსიტეტის მონაცემთა ბაზაში: 1 - ვნახოთ ყველა სტუდენტის ყველა ატრიბუტის მნიშვნელობა (ანუ მთლიანი ინფორმაცია); 2 - ვიპოვოთ იმ ლექტორთა გვარი, სახელი, სტატუსი და ხელფასი, რომელთა ხელფასი მეტია 600 ლარზე.

12.13-ა ნახაზზე 1-ელ მოთხოვნას შეესაბამება SQL-1, ხოლო მეორეს კი - SQL-2 “select” კონსტრუქცია. ისინი ორ ღილაკზეა მიმაგრებული. საილუსტრაციო მაგალითში შედეგები გამოიტანება listBox1 და listBox2 ველებში.

C#-პროგრამის კოდი ამ ორი ღილაკისთვის მოცემულია 12.1_ლისტინგში.



ნახ.12.13-ა

```
// ლისტინგი_12.1 – C# + Ms Access -------
private void button1_Click(object sender, EventArgs e)
{
  OleDbConnection con = new OleDbConnection();
  OleDbCommand cmd = new OleDbCommand();
  OleDbDataReader reader;

con.ConnectionString =
     "Provider=Microsoft.ACE.OLEDB.12.0;" +
     "Data Source=C:\\C#2010\\WinADO\\DB_AccessUni.accdb";

cmd.Connection = con;
cmd.CommandText = "select * from Student";
try
 {
   con.Open();
   reader = cmd.ExecuteReader();
   listBox1.Items.Clear();
    while (reader.Read())
      {
       listBox1.Items.Add(reader["St_ID"] + " : " +
                    reader["Gvari"] + " : " +
                    reader["Saxeli"] + " : " +
                    reader["sqesi"] + " : " +
```

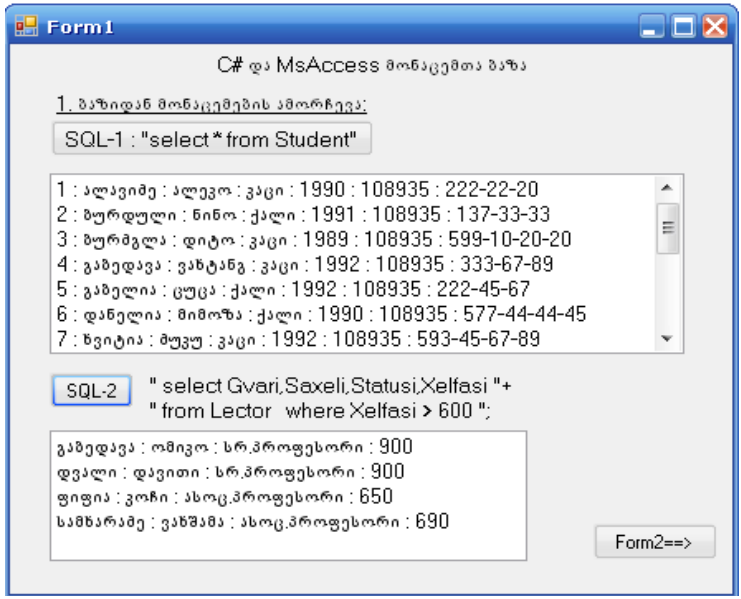
```
                    reader["dab_celi"] + " : " +
                    reader["Jg_ID"] + " : " +
                    reader["telefoni"]);
      }
      reader.Close();
      con.Close();
     }
    catch (Exception ex)
     {
      MessageBox.Show(ex.Message);
     }
  }
// end button1_click ----------------

private void button2_Click(object sender, EventArgs e)
  {
    OleDbConnection con = new OleDbConnection();
    OleDbCommand cmd = new OleDbCommand();
    OleDbDataReader reader;

 con.ConnectionString =
       "Provider=Microsoft.ACE.OLEDB.12.0;" +
       "Data Source=C:\\C#2010\\WinADO\\DB_AccessUni.accdb";
 cmd.Connection = con;
 cmd.CommandText =
     "select  Gvari,Saxeli,Statusi,Xelfasi " +
     "from Lector " +
     "where Xelfasi > 600 ";
 try
  {
    con.Open();
    reader = cmd.ExecuteReader();
    listBox2.Items.Clear();
    while (reader.Read())
     {
      listBox2.Items.Add( reader["Gvari"] + " : " +
                    reader["Saxeli"] + " : " +
                    reader["Statusi"] + " : " +
                    reader["Xelfasi"]);
      }
      reader.Close();
      con.Close();
     }
```

```
    catch (Exception ex)
      {
        MessageBox.Show(ex.Message);
      }
  }
// end button2_click ----------------
```

შენიშვნა: პროგრამის თავში using-სტრიქონებში უნდა ჩაემატოს“

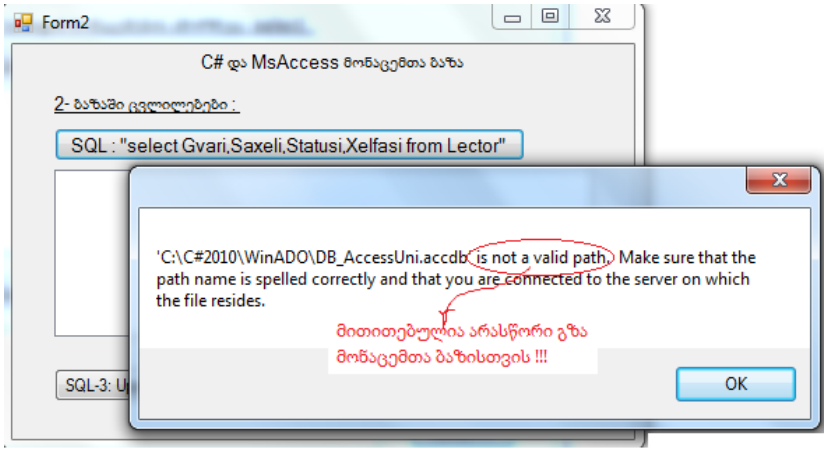
```
using System.Data.OleDb;
```

12.13-ბ ნახაზზე ნაჩვენებია აღნიშნული მოთხოვნების შესრულების შედეგები MsAccess ბაზის Student და Lector ცხრილებიდან. პირველში „ * ”- ნიშნავს „ყველა“ - ველს, ხოლო მეორეში აიღება მხოლოდ „გვარი,სახელი, სტატუსი და ხელფასი“.



ნახ.12.13-ბ

პროგრამაში **try…catch** ბლოკით ხდება მონაცემთა ბაზასთან წვდომის პროცესში შესაძლო შეცდომების აღმოჩენა, რაც აადვილებს პროგრამისტის მუშაობას. მაგალითად, ინფორმაციის მიღება, რომ მონაცემთა ბაზის ფაილი არაა მითითებულ patch-კატალოგში (ნახ.12.14), ან რომ SQL-მოთხოვნის სინტაკსში შეცდომაა და ა.შ.



ნახ.12.14

Open() მეთოდით ხდება პროგრამის კავშირის გახსნა ბაზასთან. შემდეგ, ExecuteReader() მეთოდით მოთხოვნა გადაეგზავნება ბაზას. შედეგები ბრუნდება OleDbReader კლასით, რაც ხორციელდება reader მიმთითებლით (მაჩვენებლით).

ვინაიდან წინასწარ არაა ცნობილი თუ რამდენი სტრიქონი იქნება შედეგში, გამოიყენება ListBox, რომელიც წინასწარ სუფთავდება.

Read() მეთოდი გვაწვდის ბაზიდან ერთ ჩანაწერს (სტრიქონს) და ამავდროულად სპეც-მაჩვენებლით მიუთითებს მომდევნო ჩანაწერზე. თუ ჩანაწერი ბოლოა, მაშინ მაჩვენებლის მნიშვნელობა ხდება false. ეს მართვა ხორციელდება while ციკლით try-ბლოკში.

ჩანაწერის შიგნით ველების მნიშვნელობები შეესაბამება მათ ნომრებს ან დასახელებებს. შესაძლებელია ასევე ყველა ველის გამოტანა (*-ით), რომლებიც სპეც-გამყოფითაა (მაგ., „:“) დაცილებული.

ბოლოს, Reader ობიექტი და კავშირი უნდა დაიხუროს Close() მეთოდით.

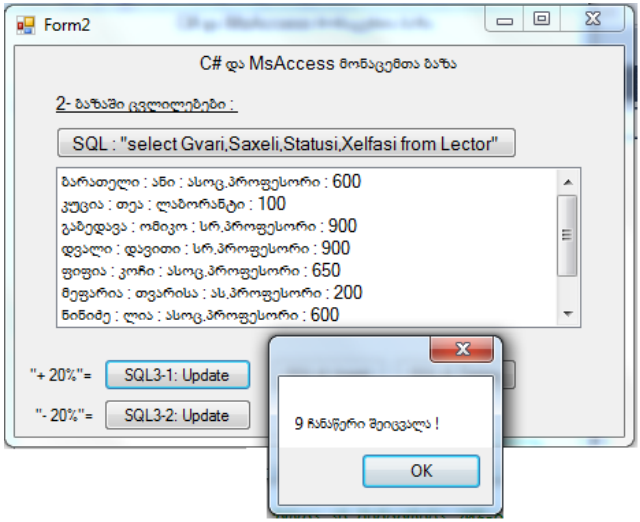
ამოცანა_12.2: მონაცემთა ბაზაში „უნივერსიტეტი“ DB_AccessUni.acce საჭიროა ცვლილებების განხორციელება insert(), delete() და update() მეთოდების გამოყენებით. Form2-ზე

მოვათავსოთ შესაბამისი ელემენტები და განვახორციელოთ ტრანზაქციები:

ა) ყველა ლექტორის ხელფასი გაიზარდოს 20%-ით. (ამასთანავე შესაძლებელი უნდა იყოს საწყისი მონაცემების აღდგენა, ანუ შემცირდეს ხელფასები 20 %-ით);

ბ) 108935 ჯგუფში დაემატოს ახალი სტუდენტი გვარით „ახალაძე“, სახელით „ნოუთბუკა“ და ა.შ.;

გ) ამოიშალოს ბაზიდან 108836 ჯგუფი, რომელმაც დაასრულა 4-წლიანი სწავლების კურსი.

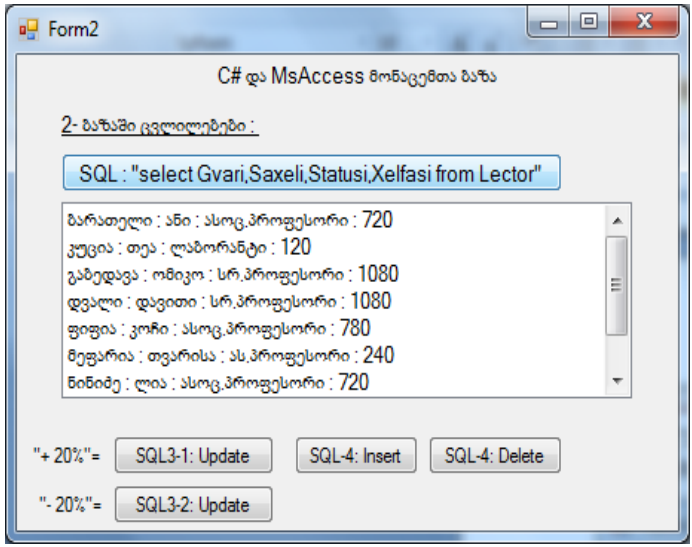


ნახ.12.15

ლექტორთა ხელფასების შეცვლილი შედეგები მოცემულია 12.16 ნახაზზე.

SQL3-2 ღილაკით შემცირდება ხელფასის რაოდენობა 20%-ით, ანუ აღდგება პირველადი მონაცემები ბაზაში.

შესაბამისი update - კოდი მოცემულია 12.2_ლისტინგში.



ნახ.12.16

```
// ლისტინგი_12.2 --- Ms Acees  “update” ------
using System;
using System.Data.OleDb;
using System.Windows.Forms;

namespace WinADO
{
 public partial class Form2 : Form
 {
  public Form2() { InitializeComponent();  }
  private void button1_Click(object sender, EventArgs e)
   {
    OleDbConnection con = new OleDbConnection();
    OleDbCommand cmd = new OleDbCommand();
    int Raod;
    // ხელფასის ზრდა ან შემცირება 20%-ით ----
    con.ConnectionString =
           "Provider=Microsoft.ACE.OLEDB.12.0;" +
           "Data Source=C:\\C#2010\\12\\DB_AccessUni.accdb";
    cmd.Connection = con;
```

```
 if(ReferenceEquals(sender,button1))
     // op = "*"  or " /" -----
  cmd.CommandText = "update Lector set Xelfasi=Xelfasi * 1.2";
 else
  cmd.CommandText = "update Lector set Xelfasi=Xelfasi / 1.2";
 try
  {
     con.Open();
     Raod = cmd.ExecuteNonQuery();
      MessageBox.Show(Raod + " ჩანაწერი შეიცვალა !");
     con.Close();
  }
 catch (Exception ex)
    {
      MessageBox.Show(ex.Message);
    }
}

private void button4_Click(object sender, EventArgs e)
 {
   OleDbConnection con = new OleDbConnection();
  OleDbCommand cmd = new OleDbCommand();
  OleDbDataReader reader;

  con.ConnectionString =
            "Provider=Microsoft.ACE.OLEDB.12.0;" +
           "Data Source=C:\\C#2010\\12\\DB_AccessUni.accdb";

  cmd.Connection = con;
   cmd.CommandText = "select Gvari,Saxeli,Statusi,
                                        Xelfasi from Lector";
  try
   {
     con.Open();
     reader = cmd.ExecuteReader();
     listBox1.Items.Clear();
     while (reader.Read())
       {
        listBox1.Items.Add(reader["Gvari"] + " : " +
                           reader["Saxeli"] + " : " +
                           reader["Statusi"] + " : " +
```

```
                        reader["Xelfasi"]);
      }
    reader.Close();
      con.Close();
     }
catch (Exception ex)
  {
    MessageBox.Show(ex.Message);
  }
```

კოდში update ტიპის ცვლილებისათვის DB_MsAccessUni.accedb ფაილში გამოყენებული კონსტრუქცია:

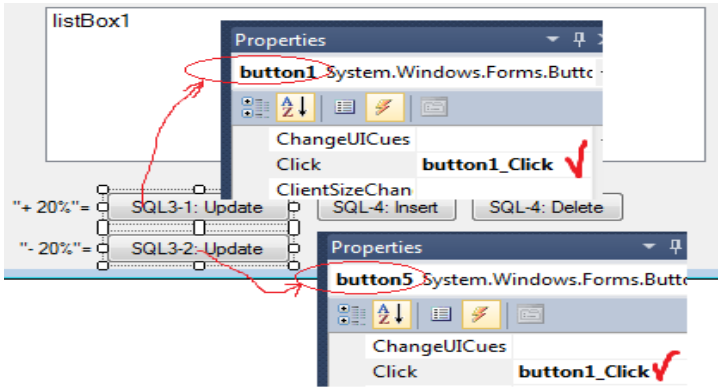
“update Lector set Xelfasi=Xelfasi * 1.2"

რაც ნიშნავს ცხრილის აქტუალიზაციას (update… set), სახელით Lector და ცხრილის ველის (ატრიბუტის) ცვლილების ალგორითმს (Xelfasi=Xelfasi * 1.2), რომელიც უნდა შესრულდეს ჩანაწერებზე;

try…catch ბლოკში იხსნება ბაზა (Open-ით) და გამოიძახება მეთოდი `ExecuteNonQuery()`, რომელიც დააბრუნებს ჩანაწერების რაოდენობას (Raod), რომელთაც შეეხო ცვლილება. ეს ინფორმაცია გამოიტანება MessageBox-ით.

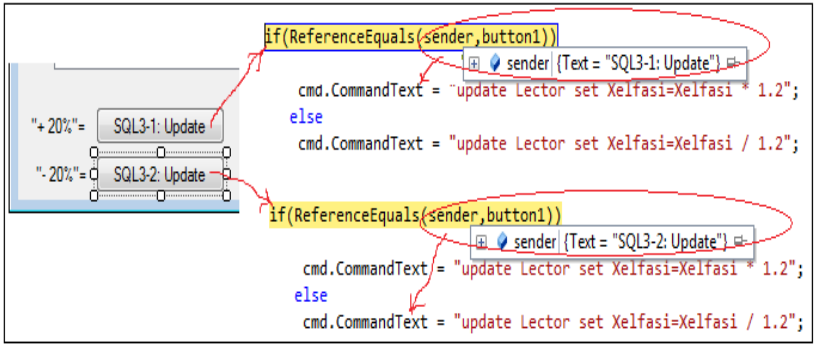
მონაცემთა ბაზის ცხრილში Lector ველისათვის Xelfasi უნდა მოხდეს მნიშვნელობათა გაზრდა 20%-ით (button1: SQL3-1) ან შემცირება (button5: SQL3-2), ნახ.12.17.

ამის განსახორციელებლად კოდში გამოყენებულია მეთოდი ReferenceEquals(sender object). ამ მეთოდით განისაზღვრება - აქვს თუ არა ორ ობიექტს ერთი მისამართი, ანუ ინახება თუ არა ისინი მეხსიერების ერთიდაიმავე უჯრედებში. თუ „კი“, მაშინ „true“ და სრულდება გამრავლება (ხელფასის მომატება), ხოლო თუ სხვადასხვა ობიექტია, მაშინ გვექნება “false” და შესრულდება გაყოფა (ხელფასის შემცირება).



ნახ.12.17

კოდის ReferencialEquals(sender, button1) მეთოდში, იმისდა მიხედვით, თუ რომელი ბუტონი (SQL3-1 თუ SQL3-2) იქნება არჩეული, sender-ს მიენიჭება button1 ან button5 მიმთითებლის მნიშვნელობა (ნახ.12.18).



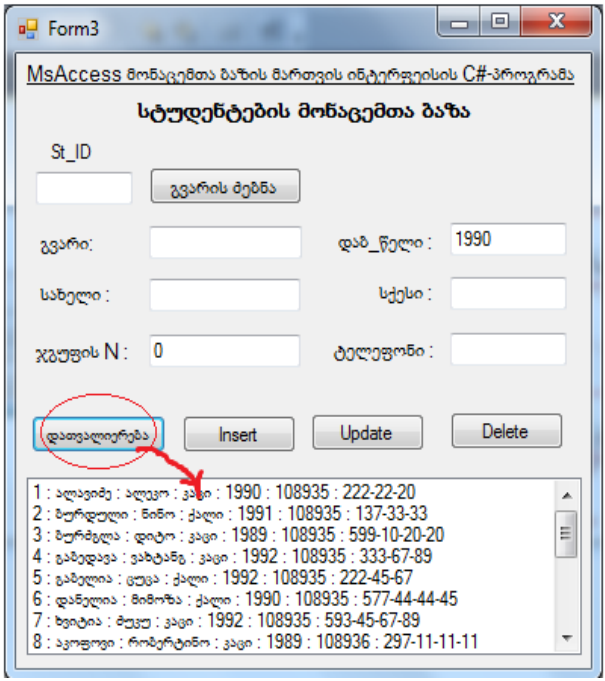
ნახ.12.18

ამგვარად, როდესაც sender და button1 ერთიდაიმავე მისამართზეა, მაშინ ხდება ხელფასის მომატება. დასასრულ, პროგრამის button4_Click ღილაკით listBox1-ში გამოიტანება MsAccess ბაზის Lector ცხრილის განახლებული მონაცემები (მომატებული ან დაკლებული ხელფასებით).

ამოცანა_12.3: საჭიროა აიგოს კოდი მომხმარებლის ინტერფეისი სტუდენტთა რეგისტრაციისათვის, რომელიც Ms Access მონაცემთა ბაზასთან იმუშავებს, განახორციელებს ჩანაწერების ძებნის, ჩამატების, წაშლის და კორექტირების სერვისულ ფუნქციებს.

საწყისი ფორმა 12.19 ნახაზზეა ნაჩვენები. ”დათვალიერება“ ღილაკი კავშირშია ცხრილ სტუდენტთან და გამოაქვს ჩანაწერები textBox1-ში.

პროგრამის საწყისი ფრაგმენტი (ბაზასთან მიერთება, მონაცემთა ინიციალიზაცია) და ღილაკი დათვალიე-რება“ მოცემულია 12.3_ლისტინგში.



ნახ.12.19

```
// ლისტინგი_12.3  --- MsAccess ბაზის მონაცემების მართვა ----
using System;
using System.Collections;
using System.Data.OleDb;
using System.Drawing;
using System.Windows.Forms;
nam espace WinADO
{
 public partial class Form3 : Form
  {
    public Form3() {InitializeComponent(); }
    OleDbConnection con = new OleDbConnection();
    OleDbCommand cmd = new OleDbCommand();
    OleDbDataReader reader;
    ArrayList stNummer = new ArrayList();

 private void Form3_Load(object sender, EventArgs e)
  {
   con.ConnectionString =
      "Provider=Microsoft.ACE.OLEDB.12.0;" +
     "Data Source=C:\\C#2010\\12\\DB_AccessUni.accdb";
    cmd.Connection = con;
  }

 private void button1_Click(object sender, EventArgs e)
  {
    Datvaliereba();
  }
 private void Datvaliereba()
   {
   try
     { con.Open();
       cmd.CommandText = "select * from Student";
       Ekranze_gamotana();
     }
    catch (Exception ex)
      {
        MessageBox.Show(ex.Message);
      }
     con.Close();
     textBox1.Text=""; // St-ID
     textBox2.Text=""; // Gvari
```

```
      textBox3.Text=""; // Saxeli
      textBox4.Text=""; // sqesi
      textBox5.Text="1990"; // dab_celi
      textBox6.Text="0"; // Jg_ID
      textBox7.Text=" "; // telefoni
   }
  private void Ekranze_gamotana()
  {
    DateTime DabTarigi;
    reader = cmd.ExecuteReader();
     listBox1.Items.Clear();
     stNummer.Clear();
     while (reader.Read())
      {
       listBox1.Items.Add(
       reader["St_ID"] + " : " +
       reader["Gvari"] + " : " +
       reader["Saxeli"] + " : " +
       reader["sqesi"] + " : " +
       reader["dab_celi"] + " : " +
       reader["Jg_ID"] + " : " +
       reader["telefoni"]);
      }
     reader.Close();
    }

private void button2_Click(object sender, EventArgs e)
    {  // Insert ღილაკის მეთოდი -----
      int Raod;
      try
       {con.Open();
        cmd.CommandText="insert into Student "+
         "(St_ID,Gvari,Saxeli,sqesi,dab_celi,Jg_ID,telefoni) "+
        " values (" + textBox1.Text +",'"+
              textBox2.Text +"', '" +textBox3.Text + "', '"+
             textBox4.Text +"', " + textBox5.Text + ", "+
             textBox6.Text +", '" + textBox7.Text + "')";
         MessageBox.Show(cmd.CommandText);

         Raod=cmd.ExecuteNonQuery();
         if(Raod >0)
         MessageBox.Show("ერთი სტრიქონი ჩაიწერა !");
```

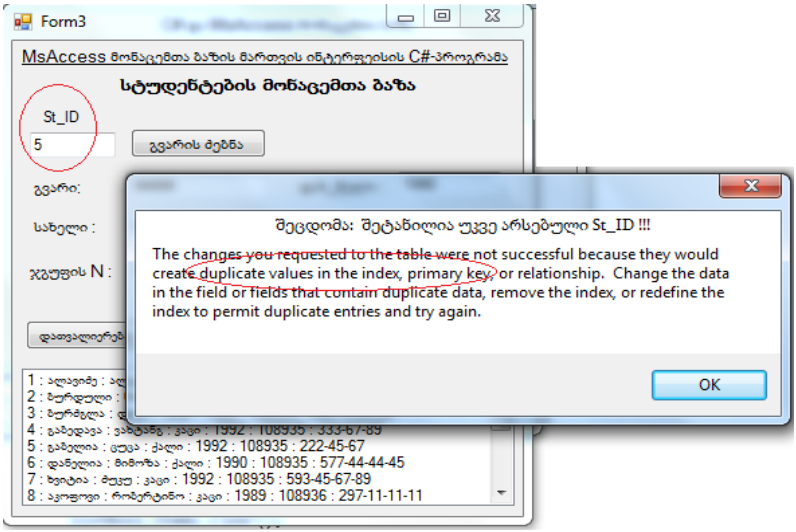
```
        }
       catch(Exception ex)
        {
         MessageBox.Show(ex.Message);
     MessageBox.Show("შეიტანეთ გვარი და სხვა მონაცემები...");
        }
       con.Close();
       Datvaliereba();
   }
  }
}
```

კოდის დასაწყისში Form3() კლასსა და Form3_Load მეთოდში გადმოტანილია ის ძირითადი კონსტრუქციები, რომლებიც საერთოა ფორმაზე მოთავსებული ელემენტებისათვის. ესაა con, cmd, stNummer ობიექტების შექმნა `OleDbConnection()`, `OleDbCommand()` და `ArrayList()` კლასების საფუძველზე. აგრეთვე ბაზასთან მიერთება მისი პროვაიდერისა და კატალოგის გზის მითითებით. SQL მოთხოვნებისა და კავშირის მაჩვენებლებისთვის განისაზღვრა OleDbDataReader-ის reader.

პირველი რეალიზებული მეთოდი, რომლის ლისტინგიც ზემოთ განვიხილეთ, არის აგებული Form3-ის ღილაკისთვის „დათვალიერება“ სტუდენტთა ცხრილისათვის. მისი გამოყენება შესაძლებელია მრავალჯერადად, ბაზის მონაცემთა ცვლილებების მონიტორინგის მიზნით.

აქ try…catch ბლოკში მოთავსებულია ბაზის გახსნა-დახურვის, select-მოთხოვნის სტრიქონი, შეცდომების „დაჭერის“ და შეტყობინების გამოცემის მექანიზმი და ბოლოს, ამორჩეული სტრიქონების გამოტანის მეთოდი `Ekranze_gamotana()`. შედეგები აისახება listBox1-ში.

მეორ რელიზებული მეთოდი ეკუთვნის Insert -ღილაკს, ანუ შესაძლებელია Form3-ზე textBox-ებში ახალი სტუდენტის მონაცემების შეტანა და შემდეგ აღნიშნული ღილაკით (იხ. 12.3_ლისტინგში `button2_Click`) მისი მოთავსება Access ბაზაში.



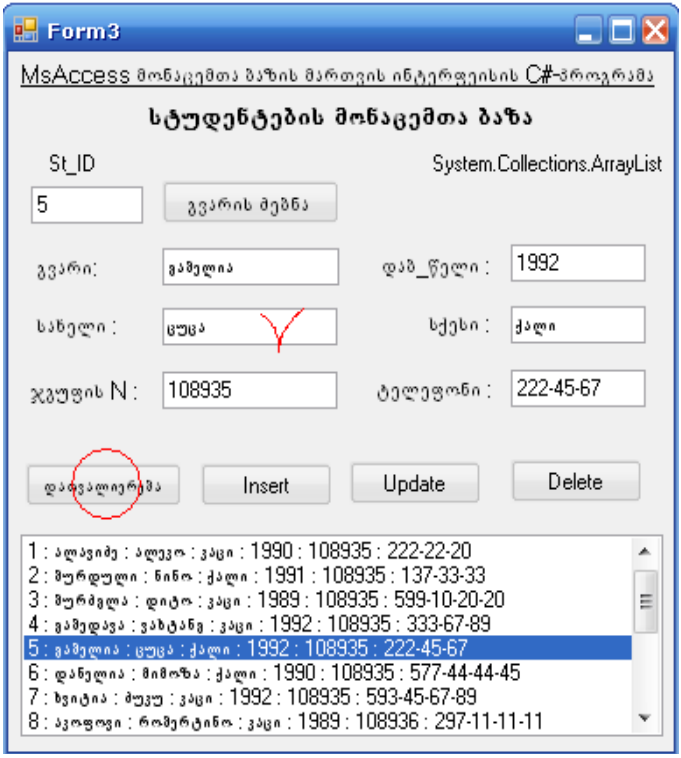
ნახ.12.20

თუ მომხმარებელმა შეცდომით შეიტანა რომელიმე მონაცემი, სისტემა პოულობს მას და გამოაქვს შესაბამისი შეტყობინება. მაგალითად, თუ შეტანილია სტუდენტის ინდექსი St_ID, რომელიც უკვე არსებობს, მაშინ მივიღებთ შეტყობინებას, რომელიც 12.20 ნახაზზეა ნაჩვენები. ბაზაში შეცდომიანი სტრიქონი არ ჩაიწერება.

ახლა განვიხილოთ ბაზაში ცვლილებების შეტანის (Update) და ჩანაწერების წაშლის (Delete) მეთოდების დაპროგრამების საკითხები.

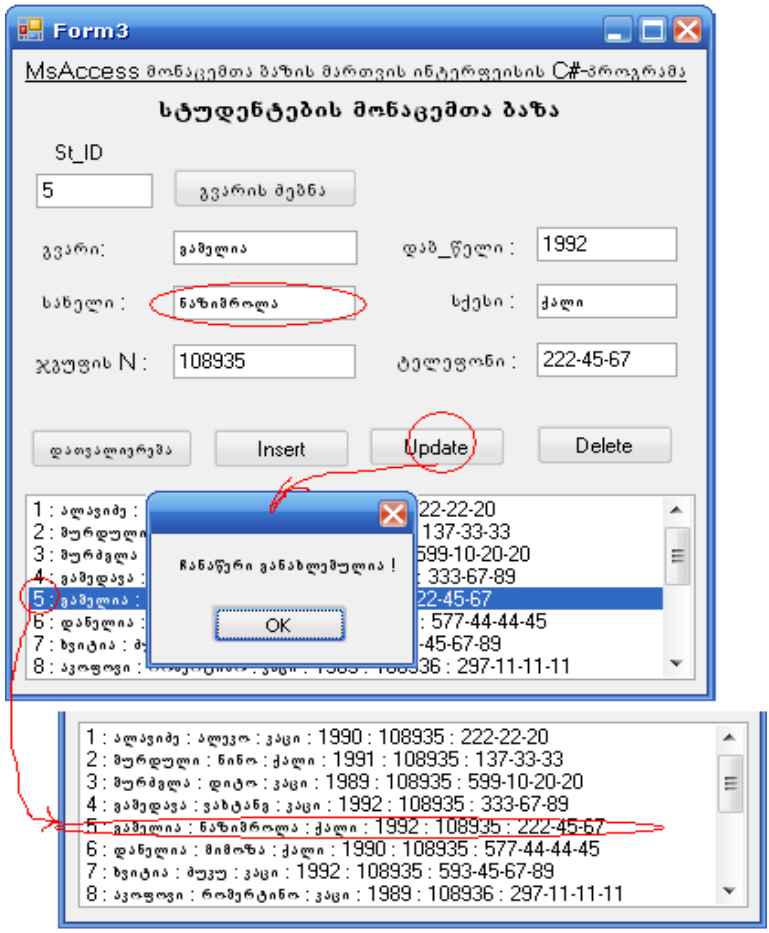
ამოცანა_12.4: Ms Access ბაზაში არსებული “სტუდენტები”-ს ცხრილიდან Windows-ფორმაზე ListBox-ში გამოვიტანოთ ჩანაწერები. ავირჩიოთ შესაცვლელი სტრიქონი, რომლის სვეტის მნიშვნელობები აისახება ფორმის შესაბამის textBox-ველებში. აქ შევცვალოთ ამ ველების მნიშვნელობები. Update ღილაკით ცვლილებები უნდა მოთავსდეს ბაზაში. შედეგების შემოწმების სისწორე განხორციელდება სტრიქონების ხელახალი ასახვით ListBox-ში.

12.21 ნახაზზე მოცემულია ფორმის საწყისი მდგომარეობა. არჩეულია მე-5 სტრიქონი. textBox-ებში ჩაიწერა ამ სტრიქონის მონაცემები. საჭიროა შეიცვალოს **სახელი=”ცუცა”** ახლით.



ნახ.12.21

12.22 ნახაზზე ნაჩვენებია განახლების პროცესის ფრაგმენტები. ველი **სახელი=”ნაზიბროლა”** (შესაძლებელია სხვა ველების ცვლილებაც), შემდეგ ღილაკით Update გამოჩნდება შეტყობინება ”ჩანაწერი განახლებულია” (თუ ტრანზაქცია შესრულდა სწორად). თუ რამე შეცდომაა, გამოვა შესაბამისი შეტყობინება.



ნახ.12.22

12.4_ლისტინგის კოდი არის 12.3_ლისტინგის გაფართოება. აქ ორი მოვლენა პროგრამირდება. ერთი, listBox1-ში სტრიქონის არჩევა ცვლილების მიზნით, რომელსაც textBox-ებში გამოაქვს შესაბამისი მონაცემები. და მეორე, Update ღილაკის დაჭერით შესრულებული პროცედურა (მონაცემთა ბაზაში შენახვა).

```
  //ლისტინგი_12.4 --- Update() ------
private void listBox1_SelectedIndexChanged(object sender,
                                                    EventArgs e)
{  // სტრიქონის არჩევა ----
 try
  {
    con.Open();
    cmd.CommandText = "select * from Student " +
        " where St_ID = " + stNummer[listBox1.SelectedIndex];

       reader = cmd.ExecuteReader();
       reader.Read();

       textBox1.Text="" + reader["St_ID"];
       textBox2.Text="" + reader["Gvari"];
       textBox3.Text="" + reader["Saxeli"];
       textBox4.Text="" + reader["sqesi"];
       textBox5.Text="" + reader["dab_celi"];
       textBox6.Text="" + reader["Jg_ID"];
       textBox7.Text="" + reader["telefoni"];

       reader.Close();
    }
  catch (Exception ex)
    {
      MessageBox.Show(ex.Message);
    }
  con.Close();
}
private void button3_Click(object sender, EventArgs e)
  {   // update -------
    int Raod;
    try
     {
       con.Open();
       cmd.CommandText = "update Student set " +
             "St_ID = " + textBox1.Text + ", "+
              "Gvari = '" + textBox2.Text + "', " +
              "Saxeli ='" + textBox3.Text + "', " +
              "sqesi = '" + textBox4.Text + "', " +
              "dab_celi = " + textBox5.Text + ", " +
              "Jg_ID = " + textBox6.Text + ", " +
```

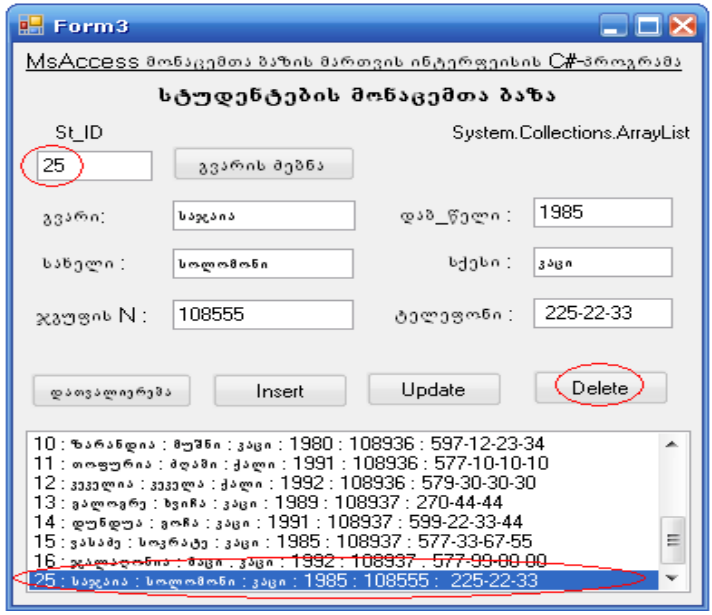
```
            "telefoni = '" + textBox7.Text + "' "+
      " where St_ID =" + stNummer[listBox1.SelectedIndex];
        MessageBox.Show(cmd.CommandText);
          Raod = cmd.ExecuteNonQuery();
          if (Raod > 0)
            MessageBox.Show("ჩანაწერი განახლებულია !");
      }
    catch(Exception ex)
      {
         MessageBox.Show(ex.Message);
         MessageBox.Show("შეასწორეთ ერთი ჩანაწერი " +
                   "აირჩიეთ რომელიმე გვარი," +
                   " ცალსახა სტუდ_ნომერი და " +
                   " სხვა მონაცემები !!!");
      }
    con.Close();
    Datvaliereba();
  }  // end Update
```

ამოცანა_12.5: განვიხილოთ Ms Access ბაზის ცხრილში “სტუდენტები” ჩანაწერის წაშლის პროცედურა, რომელიც Delete ღილაკზე იქნება მიბმული.

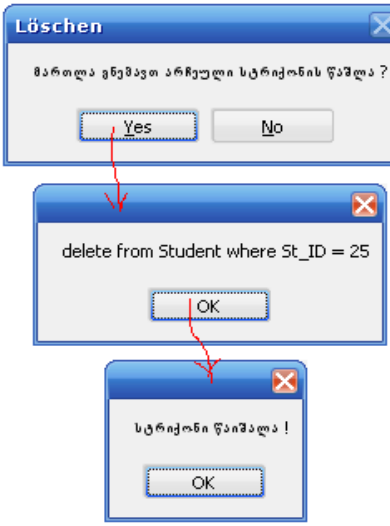
12.23-ა ნახაზზე ნაჩვენებია ფორმაზე მოთავსებული ცხრილის ”სტუდენტები” ჩანაწერები, რომელთაგან, ერთ-ერთი, კერძოდ 25-ე ჩანაწერი არჩეულია წასაშლელად.

Delete-ღილაკის ამოქმედებით მიმდევრობით გამოჩნდება 12.23-ბ ნახაზზე ნაჩვენები შეტყობინებები. თუ ყველაფე-რი ნორმალურადაა, ბაზიდან მოხდება 25-ე ჩანაწერის ამოშლა.

წაშლის პროგრამის ტექსტი მოცემულია 12.5_ლისტინგში. აქაც try…catch ბლოკში cmd.CommandText სტრიქონში ჩაწერილია წაშლის SQL-ბრძანება, რომელიც შემდეგ `ExecuteNonQuery()` მეთოდით გადაეცემა შესასრულებლად. შედეგის ნახვა ხორციელდება პროგრამის ბოლოს Datvaliereba() მეთოდით.



ნახ.12.23-ა



ნახ.12.23-ბ

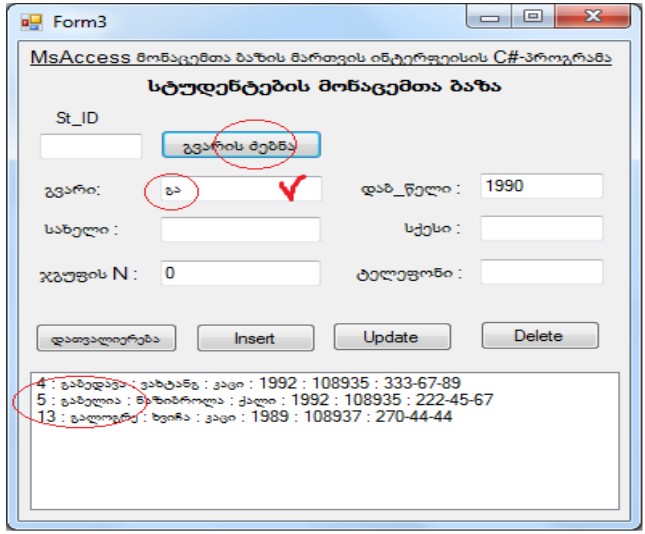
```
// ლისტინგი_12.5 ---  Delete -------
private void button4_Click(object sender, EventArgs e)
 { int Raod;
   if (textBox1.Text == "")
    {
      MessageBox.Show("აირჩიეთ ერთი სტრიქონი !");
      return;
    }
   if (MessageBox.Show("მართლა გნებავთ არჩეული" +
                  " სტრიქონის წაშლა ?", "Löschen",
             MessageBoxButtons.YesNo) == DialogResult.No)
     return;
   try
   {
    con.Open();
    cmd.CommandText = "delete from Student " +
          "where St_ID = " + stNummer[listBox1.SelectedIndex];
      MessageBox.Show(cmd.CommandText);
    Raod = cmd.ExecuteNonQuery();
    if (Raod > 0)
       MessageBox.Show("სტრიქონი წაიშალა !");
   }
  catch (Exception ex)
   { MessageBox.Show(ex.Message); }
  con.Close();
  Datvaliereba();
}  //end Delete -----------
```

ამოცანა_12.6: ავაგოთ პროგრამა, რომელიც გვარის ველში შეტანილი ერთი, ორი ან მეტი სიმბოლოთი იპოვის მრავალჩანაწერიან ბაზაში შესაბამის სტრიქონებს და გამოიტანს მათ listBox1-ში (ნახ.12.24).

ეს კოდი მოცემულია 12.6_ლისტინგში.

```
  // ლისტინგი_12.24 ---- გვარით ძებნა -----
private void button5_Click(object sender, EventArgs e)
 {  try
      {
        con.Open();
        cmd.CommandText = "select * from Student where" +
               " Gvari like '%" + textBox2.Text + "%'";
```

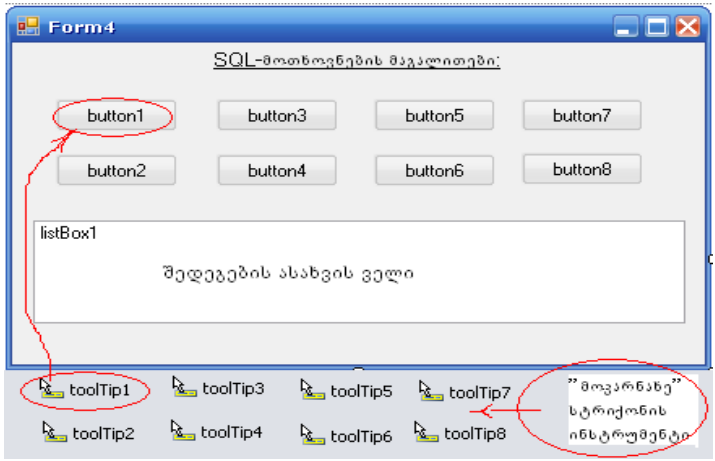
```
            MessageBox.Show(cmd.CommandText);
            Ekranze_gamotana();
           }
          catch (Exception ex)
           {
             MessageBox.Show(ex.Message);
           }
          con.Close();
}
```



ნახ.12.24

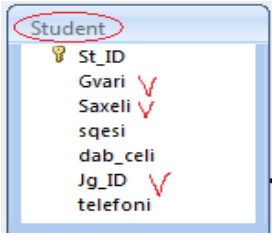
➢ SQL მოთხოვნების დაპროგრამირების საილიუსტრაციო მაგალითები C#-ში MsAccess ბაზისთვის

ამოცანა_12.7: ავაგოთ C# კოდი - მომხმარებლის ინტერფეისი MsAccess მონაცემთა ბაზის რამდენიმე ცხრილთან ერთდროულად სამუშაოდ. გამოვიყენოთ DB_AccessUni ბაზის Table-ები: Student, Group, Lector და Lect_Jgufi (ნახ.12.8-12.10). საჭიროა მოთხოვნების წინასწარ დაპროექტება და მისი SQL-ფორმატით წარმოდგენა. შემდეგ ამ სტრუქტურირებული მოთხოვნის ჩასმა C# კოდში (მიმაგრება button_Click... ღილაკებზე ნახ.12.25)



ნახ.12.25

მოთხოვნა_1: ”ეკრანზე გამოვიტანოთ სტუდენტთა სია: გვარი, სახელი, ჯგუფის_ნომერი, მოწესრიგებული ჯგუფის ნომრით და გვარით” (ნახ.12.26) .



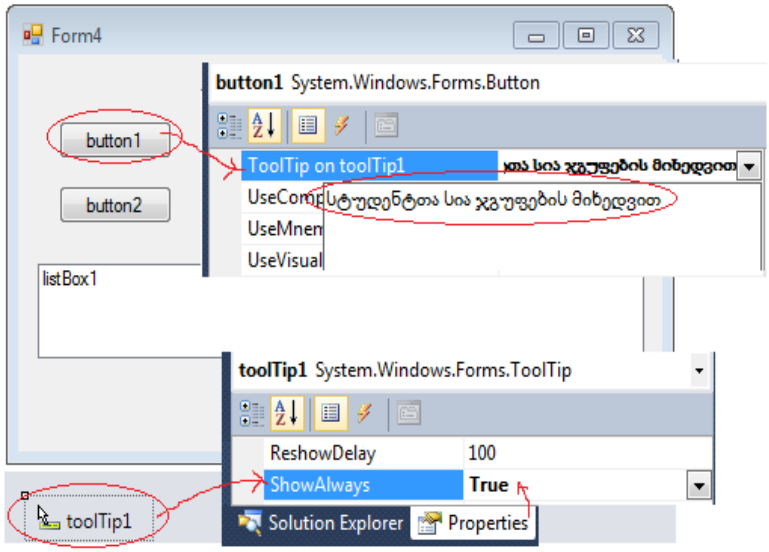
ნახ.12.26

სასურველია მოთხოვნის ღილაკების „გასემანტიკურება“ (მაუსის კურსორის მიტანისას ღილაკზე გამოჩნდეს „მოკარნახე ტექსტი (hint-მინიშნება), თუ რას აკეთებს ეს ღილაკი“).

ამისათვის ToolBox-პანელიდან ფორმაზე გადმოვიტანოთ ToolTip1-არავიზუალური ელემენტი, რომელიც მოთავსდება ფორმის ქვემოთ (ნახ.12.27-ა). მოვნიშნოთ იგი და Properties-ში დავაყენოთ თვისება ShowAllways = true.

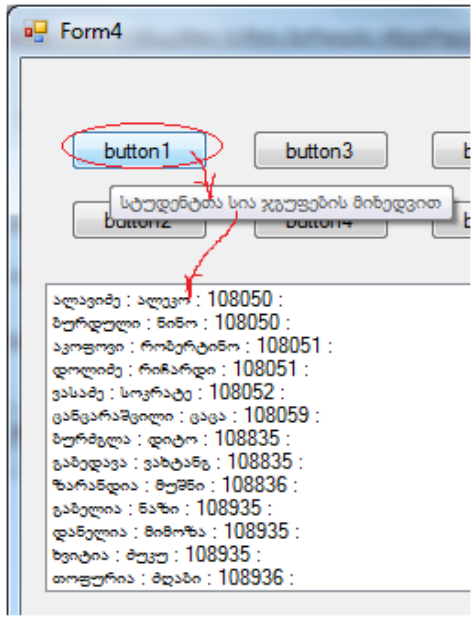
შემდეგ მოვნიშნოთ ღილაკი button1 და Properties-ში თვისებაში ToolTip on toolTip1 ჩავწეროთ „მოკარნახე“ ტექსტი. ავამუშავოთ პროგრამა, მივიტანოთ მაუსის კურსორი button1 ღილაკთან. გამოჩნდება ტექსტი „სტუდენტთა სია ჯგუფების მიხედვით“. თუ არ გვინდა ეს ინფორმაცია, გადავალთ სხვაზე. თუ ავამოქმედებთ button1-ს, მივიღებთ შედეგს (ნახ.12.28).

button1 ღილაკის პროგრამის კოდის ფრაგმენტი, რომელშიც ასახულია SQL-ტიპის მოთხოვნა, მოცემულია 12.25_ლისტინგში.



ნახ.12.27

```
// ლისტინგი_12.25 --- button1 ----
private void button1_Click(object sender, EventArgs e)
 {
   Amorcheva("select * from Student order by Jg_ID, Gvari",
                                  "Gvari", "Saxeli", "Jg_ID");
   MessageBox.Show("select * from Student order by Jg_ID,
                                 'Gvari', 'Saxeli', 'Jg_ID' ");
 }
```

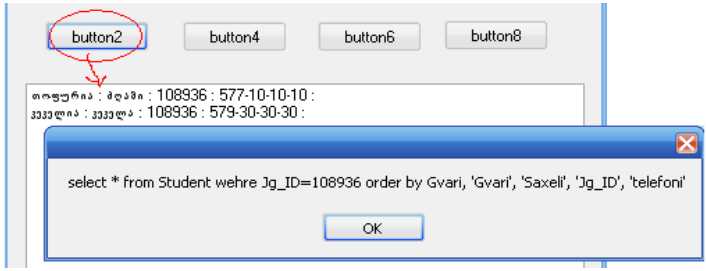


ნახ.12.28

მოთხოვნა_2: ”ვიპოვოთ ის სტუდენტები, რომლებიც სწავლობენ 108936 ჯგუფში. გამოვიტანოთ მათი გვარები, სახელები, ჯგუფის_ნომრები.

```
// ლისტინგი_12.26 --- button2 ----
private void button2_Click(object sender, EventArgs e)
{
Amorcheva("select * from Student where Student.Jg_ID=108936 "+
  "order by Gvari", "Gvari", "Saxeli", "Jg_ID", "telefoni");
MessageBox.Show("select * from Student wehre Jg_ID=108936 "+
  "order by Gvari, 'Gvari', 'Saxeli', 'Jg_ID', 'telefoni' ");
}
```

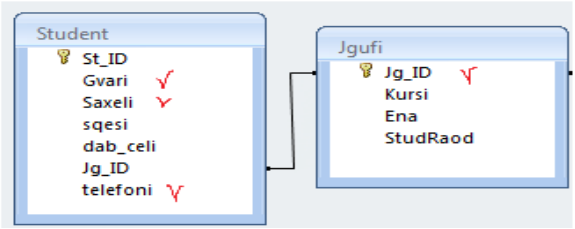
შედეგი მოცემულია 12.29 ნახაზზე.



ნახ.12.29

მოთხოვნა_3: ”ვიპოვოთ იმ სტუდენტთა გვარები, სახელები, ჯგუფის_ნომრები და ტელეფონები, რომლებიც სწავლობენ ინგლისურენოვან ჯგუფში”.

ამ შემთხვევაში საჭიროა ბაზიდან ორი ცხრილის გამოყენება: Student და Jgufi. 12.30 ნახაზიდა ჩანს, რომ ორი ეს ცხრილები კავშირშია ერთმანეთთან ატრიბუტით Jgufi.Jg_ID (პირველადი გასაღები, ანუ უნიკალური ინდექსი) და Student.Jg_ID (მეორადი გასაღები, ანუ არაუნიკალური ინდექსი).

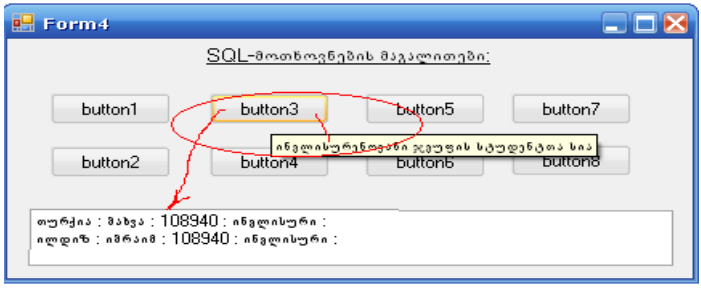


12.30

მათი გათვალისწინებით SQL-მოთხოვნას ექნება 12.27_ლისტინგზე მოცემული სახე.

```
// ლისტინგი_12.27 --- button3 ----
private void button3_Click(object sender, EventArgs e)
{
   Amorcheva("select * from Student,Jgufi " +
     "where Student.Jg_ID=Jgufi.Jg_ID and Ena='ინგლისური' " +
                  "order by Jgufi.Jg_ID, Gvari", "Gvari",
                        "Saxeli", "Student.Jg_ID", "Ena");
  MessageBox.Show("select * from Student,Jgufi "+
    "where Student.Jg_ID=Jgufi.Jg_ID and Ena='ინგლისური' " +
```

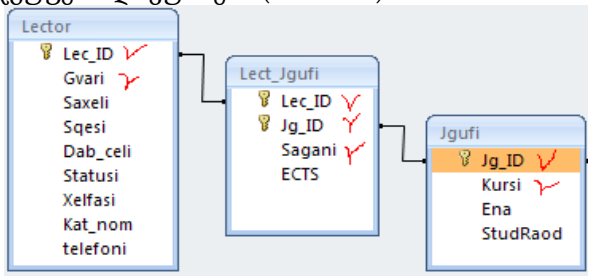
```
                "order by Gvari, 'Gvari', 'Saxeli',
                                'Student.Jg_ID', 'Ena' ");
}
```



12.31

მოთხოვნა_4: რომელი კურსის რომელ ჯგუფებს და რა საგნებს ასწავლის ლექტორი გაბედავა ?

ბაზიდან უნდა გამოვიყენოთ სამი ცხრილი: Lector, Jgufi და Lect_Jgufi, რათა კონკრეტული ლექტორისთვის ვიპოვოთ მისი საგნები, ჯგუფები და კურსები (ნახ.12.32).

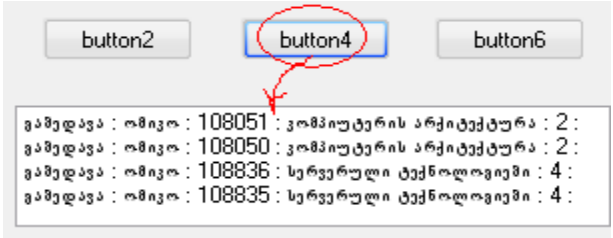


12.33

კოდის ტექსტი მოცემულია 12.28_ლისტინგში, ხოლო შედეგები 12.34 ნახაზზე.

```
// ლისტინგი_12.28 --- button4 ----
private void button4_Click(object sender, EventArgs e)
{
  Amorcheva("SELECT * FROM Lector,Jgufi,Lect_Jgufi " +
            "where Jgufi.Jg_ID = Lect_Jgufi.Jg_ID and "+
               "Lector.Lec_ID = Lect_Jgufi.Lec_ID and "+
                           "Lector.Gvari='გაბედავა' " +
```

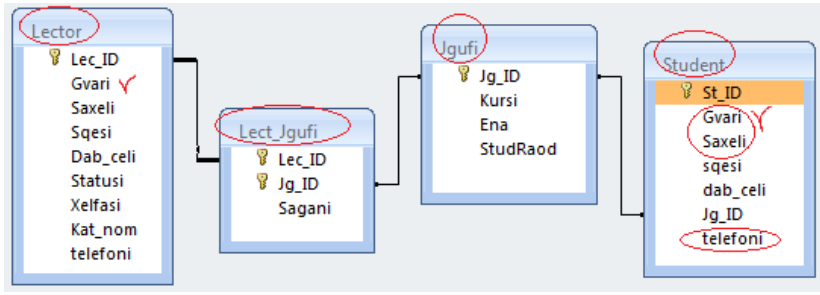
```
            "order by Lect_Jgufi.Sagani ", "Gvari", "Saxeli",
                    "Jgufi.Jg_ID", "Sagani","kursi");

  MessageBox.Show("SELECT * FROM Lector,Jgufi,Lect_Jgufi " +
                  "where Jgufi.Jg_ID = Lect_Jgufi.Jg_ID and "+
                     "Lector.Lec_ID = Lect_Jgufi.Lec_ID and
                        Lector.Gvari='გაბედავა' " +
                   "order by Lect_Jgufi.Sagani , 'Gvari',
                 'Saxeli', 'Jgufi.Jg_ID', 'Sagani', 'kursi'");
}
```



12.34

მოთხოვნა_5: რომელ სტუდენტებს ასწავლის ლექტორი გაბედავა ? საჭიროა გვარი, სახელი და ჯგუფის ნომერი.

მონაცემთა ბაზიდან ამჯერად დაგვჭირდება ოთხივე ცხრილი (ნახ.12.35). კონკრეტული ლექტორიდან (მარცხენა ნაპირა ცხრილი) უნდა მივიდეთ კონკრეტულ სტუდენტამდე (მარჯვენა ნაპირა ცხრილი). შესაბამისი SQL-მოთხოვნის ტექსტი მოცემულია 12.29_ლისტინგში.

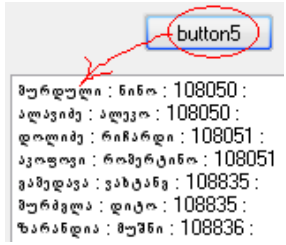


12.35

```
// ლისტინგი_12.29 --- button5 ----
private void button5_Click(object sender, EventArgs e)
{
  Amorcheva("SELECT * FROM Lector,Jgufi,Lect_Jgufi,Student " +
            "where Jgufi.Jg_ID = Lect_Jgufi.Jg_ID and "+
            “Lector.Lec_ID = Lect_Jgufi.Lec_ID and " +
    "Student.Jg_ID=Jgufi.Jg_ID and Lector.Gvari='გაბედავა' " +
             "order by Jgufi.Jg_ID ", "Student.Gvari",
                        "Student.Saxeli", "Student.Jg_ID");
  MessageBox.Show("SELECT * FROM
                        Lector,Jgufi,Lect_Jgufi,Student " +
                  "where Jgufi.Jg_ID = Lect_Jgufi.Jg_ID and
                     Lector.Lec_ID = Lect_Jgufi.Lec_ID and " +
    "Student.Jg_ID=Jgufi.Jg_ID and Lector.Gvari='გაბედავა' " +
                  "order by Jgufi.Jg_ID, 'Student.Gvari',
                         'Student.Saxeli', 'Student.Jg_ID' ");
}
```

შედეგი მოცემულია 12.36 ნახაზზე.

ნახ.12.36

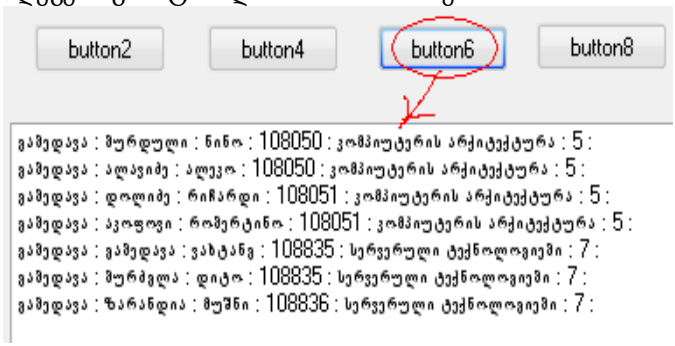


მოთხოვნა_6: ლექტორ გაბედავასთან რომელი ჯგუფის რომელ სტუდენტებთან აქვს ლექციები რომელ საგნებში და რამდენ კრედიტიანებია ეს საგნები ?

```
// ლისტინგი_12.30 --- button6 ----
private void button6_Click(object sender, EventArgs e)
{
 Amorcheva("SELECT * FROM Lector,Lect_Jgufi, Jgufi,Student " +
           "where Lector.Lec_ID=Lect_Jgufi.Lec_ID and "+
                      " Jgufi.Jg_ID = Lect_Jgufi.Jg_ID and "+
    "Student.Jg_ID=Jgufi.Jg_ID and Lector.Gvari='გაბედავა' " +
```

```
            "order by Jgufi.Jg_ID ", "Lector.Gvari",
                     "Student.Gvari", "Student.Saxeli", +
                     "Jgufi.Jg_ID", "Sagani", "ECTS");
    MessageBox.Show("SELECT * FROM Lector,Lect_Jgufi, Jgufi, "+
                                                  "Student " +
                    "where Lector.Lec_ID=Lect_Jgufi.Lec_ID and "+
                          " Jgufi.Jg_ID = Lect_Jgufi.Jg_ID and "+
                           "Student.Jg_ID=Jgufi.Jg_ID and "+
                           "Lector.Gvari='გაბედავა' " +
                    "order by Jgufi.Jg_ID, 'Lector.Gvari', "+
                              'Student.Gvari', 'Student.Saxeli',
                             'Jgufi.Jg_ID', 'Sagani', 'ECTS' ");
}
```

შედეგები გამოტანილია 12.37 ნახაზზე.



ნახ.12.37

მოთხოვნა_7: რა საგნები ისწავლება და რამდენ კრედიტიანებია ?

ეს ინფორმაცია მისაწვდომია ერთი ცხრილიდან Lect_Jgufi. თუ მოთხოვნა ასე დაიწერება:

```
Amorcheva("select Sagani, ECTS from Lect_Jgufi order by
                                  Sagani", "Sagani", "ECTS");
```

მაშინ მივიღებთ ასეთ შედეგს (ნახ.12.38), რაც არაკორექტულია განმეორებადი სტრიქონების გამო:

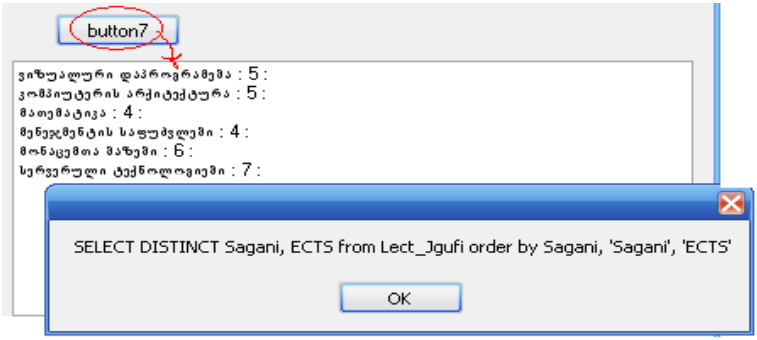
```
ვიზუალური დაპროგრამება : 5 :
ვიზუალური დაპროგრამება : 5 :
კომპიუტერის არქიტექტურა : 5 :
კომპიუტერის არქიტექტურა : 5 :
კომპიუტერის არქიტექტურა : 5 :
მათემატიკა : 4 :
მათემატიკა : 4 :
მენეჯმენტის საფუძვლები : 4 :
მენეჯმენტის საფუძვლები : 4 :
მონაცემთა ბაზები : 6 :
მონაცემთა ბაზები : 6 :
სერვერული ტექნოლოგიები : 7 :
სერვერული ტექნოლოგიები : 7 :
```

ნახ.12.38

საჭიროა SELECT DISTINCT… კონსტრუქციის გამოყენება, რაც გამორიცხავს განმეორებად სტრიქონებს რელაციურ ცხრილებში. ამგვარად შესაძლებელია სტრიქონთა ”სიმრავლის” მიღება. სწორი ტექსტი მოცემულია 12.31_ლისტინგში.

```
// ლისტინგი_12.31 --- button7 ----
private void button7_Click(object sender, EventArgs e)
{
  Amorcheva("select distinct Sagani, ECTS from Lect_Jgufi “+
            “ order by Sagani", "Sagani", "ECTS");
  MessageBox.Show("SELECT DISTINCT Sagani, ECTS “+
         from Lect_Jgufi order by Sagani, 'Sagani', 'ECTS' ");
 }
```

12.39 ნახაზზე ასახულია სწორი შედეგები.



ნახ.12.39

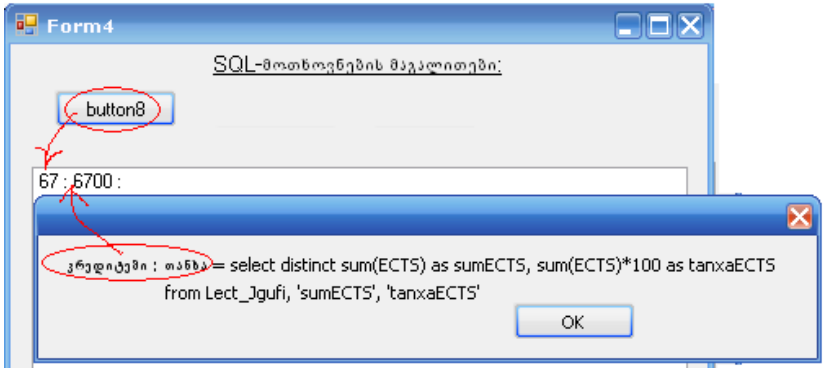
მოთხოვნა_8: რამდენი კრედიტი შეიძინეს სტუდენტებმა დამატებით სემესტრში და რას უდრის ჯამური შემოსავალი თანხაში ?

ცხრილში Lect_Jgufi მოთავსებულია ოპერატიული ინფორმაცია მიმდინარე სემესტრის საგნების შესახებ, ანუ, აქედან უნდა მოხდეს ECTS-ატრიბუტის მნიშვნელობათა შეჯამება და მისი გამრალება 1 კრედიტის ღირებულებაზე. სიმარტივისთვის დავუშვათ, რომ ის 100 ლარია.

12.32 ლისტინგში მოცემულია კოდის ტექსტი შესაბამისი SQL მოთხოვნით.

```
// ლისტინგი_12.32 --- button7 ----
private void button8_Click(object sender, EventArgs e)
{
  Amorcheva("select sum(ECTS) as sumECTS, “+
                  “sum(ECTS)*100 as tanxaECTS “+
            “from Lect_Jgufi ", "sumECTS", "tanxaECTS");
  MessageBox.Show(" კრედიტები : თანხა = select “+
         “sum(ECTS) as sumECTS, sum(ECTS)*100 as tanxaECTS "+
            "from Lect_Jgufi, 'sumECTS', 'tanxaECTS' ");
}
```

შედეგი მოცემულია 12.40 ნახაზზე.



ნახ.12.40

განხილული პროგრამის დასაწყისი და საერთო ნაწილი ყველა მოთხოვნისა მოცემულია 12.33_ლისტინგში.

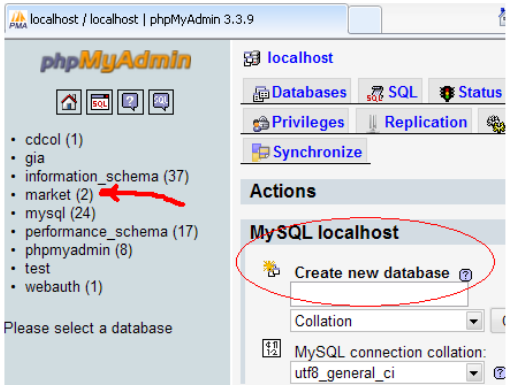
```
// ლისტინგი_12.33 --- Syetem+Contact with DBS+ try…catch ----
using System;
using System.Drawing;
using System.Windows.Forms;
using System.Data.OleDb;
namespace WinADO
{
 public partial class Form4 : Form
  {
   public Form4() { InitializeComponent(); }
   private void Amorcheva(string sqlbefehl,
                                    params string[] velebi)
   {
     OleDbConnection con = new OleDbConnection();
     OleDbCommand cmd = new OleDbCommand();
     OleDbDataReader reader;
     int i;
     string striqoni;
 con.ConnectionString = "Provider=Microsoft.ACE.OLEDB.12.0;" +
        "Data Source=C:\\C#2010\\WinADO\\DB_AccessUni.accdb";
     cmd.Connection = con;
     cmd.CommandText = sqlbefehl;
     try
      {  // აქ ხდება SQLმოთხოვნის წაკითხვა, ანალიზი ატრიბუტებით
       con.Open();
       reader = cmd.ExecuteReader();
       listBox1.Items.Clear();
       while (reader.Read())
        {
          striqoni = "";
          for (i = 0; i < velebi.Length; i++)
             striqoni += reader[velebi[i]] + " : ";

          listBox1.Items.Add(striqoni); // ეკრანზე გამოტანა ---
        }
       reader.Close();
       con.Close();
      }
    catch (Exception ex)
      {    // შეცდომის არსებობისას გამოსცემს შეტყობინებას --------
        MessageBox.Show(ex.Message);
   }} // აქ უერთდება button_Click - მეთოდები -----  // . . .
```

12.6. C# და MySQL

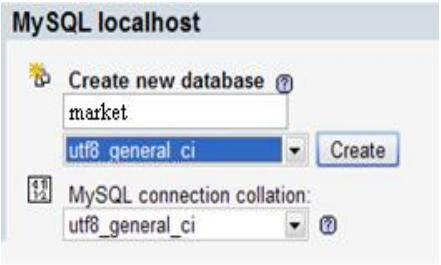
განვიხილოთ C# ენის გამოყენებით მომხმარებელთა ინტერფეისების აგების მაგალითები მონაცემთა ბაზების მართვის სისტემისთვის MySQL. იგი, როგორც განაწილებული რელაციური ბაზა ერთ-ერთი აქტუალური და ფართოდ გამოყენებადი კლიენტ-სერვერული პაკეტია დღეისათვის, განსაკუთრებით php ვებ-ტექნოლოგიებში [ ].

ვგულისხმობთ, რომ MySQL სისტემა დაინსტალირებულია კომპიუტერზე (თუ არა, მაშინ ის ჩამოწერილ უნდა იქნეს http://www.mysql.com საიტიდან და დაინსტალირდეს).

როგორც წესი MySQL სისტემის გამოძახება ხდება რომელიმე ბრაუზერში, მაგალითად Internet Explorer-ში url-მისამართში უნდა მიეთითოს: http://localhost/phpmyadmin/ სტრიქონი. 12.41-ა ნახაზზე გამოტანილია შედეგი. აქ ავირჩევთ ჩვენთვის საჭირო ბაზას, მაგალითად, market, რომელსაც აქვს 2 ცხრილი. თუ საჭიროა ახალი ბაზის შექმნა, მაშინ გამოვიყენებთ ველს: create new database. აქ ჩავწერთ ახალი ბაზის სახელს, Collation ველ-ში ვირეჩევთ სტრიქონს utf8_general_ci, რაც უნიკოდის გამოყენების საშუალებას იძლევა (ნახ 12.41-ბ).



ნახ.12.41-ა



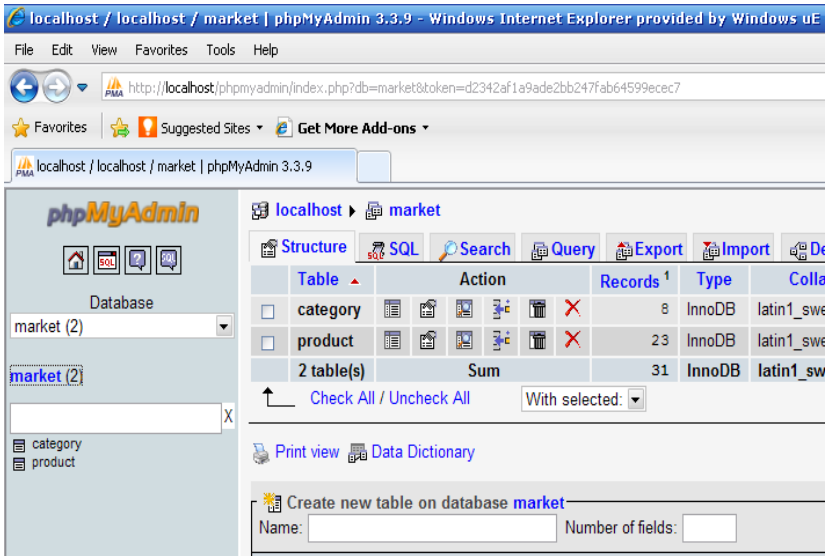
ნახ.12.41-ბ

შემდეგ უნდა შევქმნათ ბაზის ცხრილები, რისთვისაც ველში “Create new table on database”: market - ჩავწერთ ცხრილის სახელს, მაგალითად, product ან category და დავიწყებთ ცხრილის ველების (სვეტების) შექმნას (მაგალითად, ნახ.12.42-ა).

Field	Type	Length/Values[1]
pr_ID	INT	4
Name	VARCHAR	30
prize	DECIMAL	
cat_ID	INT	2

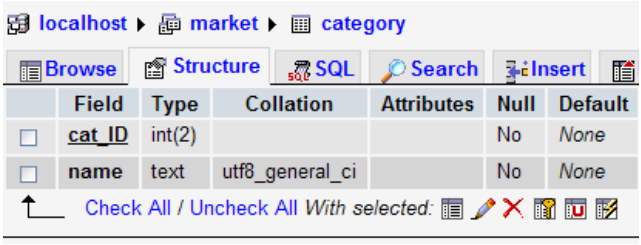
ნახ.12.42-ა

12.42-ბ ნახაზზე ნაჩვენებია ცხრილები: category და product.



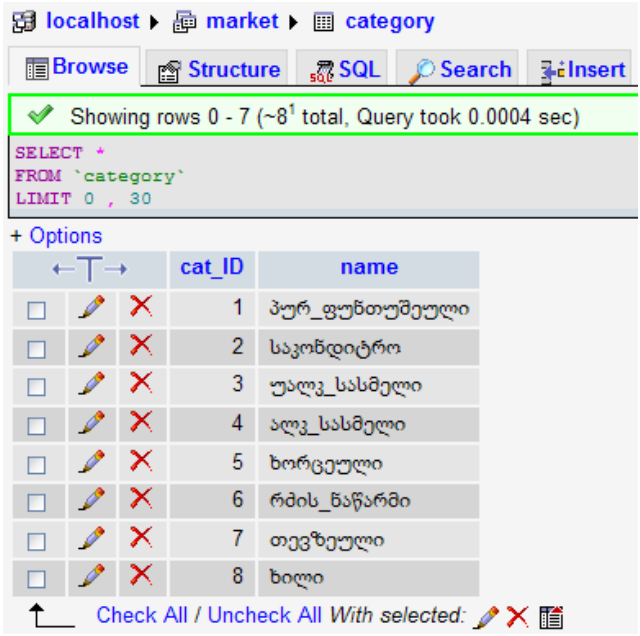
ნახ.12.42-ბ

ავირჩიოთ category ცხრილი და მენიუში Structure. 12.43-ა ნახაზზე ნაჩვენებია შედეგი.



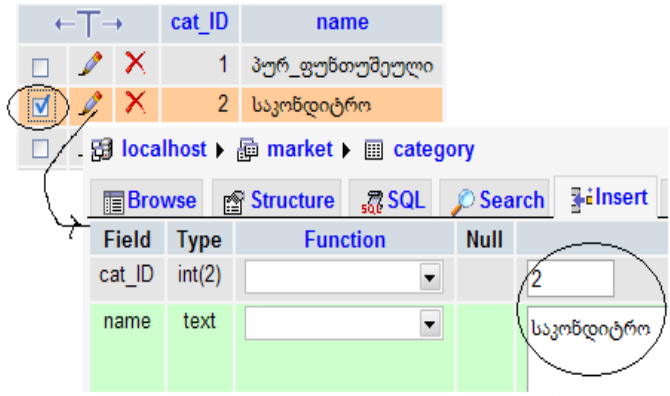
ნახ.12.43-ა

cat_ID გასაღებური ველი (პირველადი ინდექსი) აქ გახაზულია. ახლა უნდა შევავსოთ სტრიქონები კონკრეტული მნიშვნელობებით. ამისათვის ავამოქმედებთ Browse გადამრთველს და გამოჩნდება 12.43-ბ ნახაზზე მოცემული ფანჯარა.



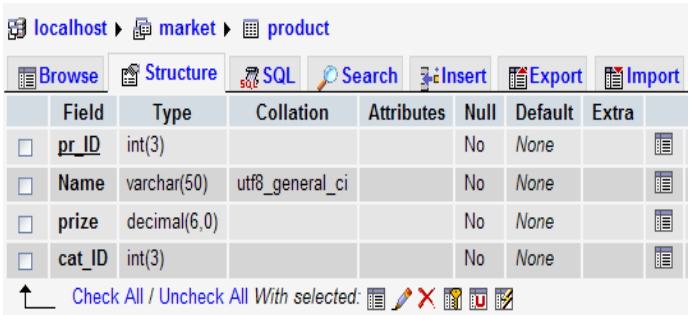
ნახ.12.43-ბ

”ჩეკბოქსის” ჩართვით და ფანქრის არჩევით შევალთ სტრიქონის რედაქტირების რეჟიმში და შევასრულებთ ჩვენთვის საჭირო ფუნქციას (ნახ.12.44).

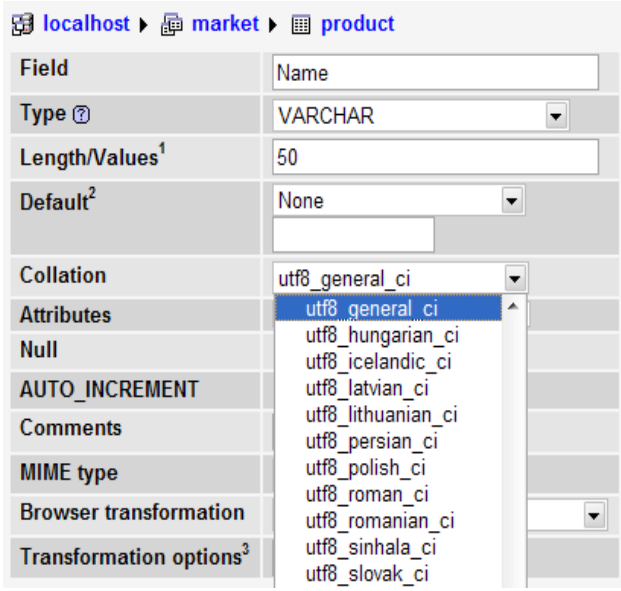


ნახ.12.44

ახლა გადავიდეთ product ცხრილზე. მისი ველების განსაზღვრით და სტრიქონების შეტანით შესაძლოა ქართული უნიკოდების მაგივრად ’????? ????’ სტრიქონის მიღება. ასეთი ველის სტრუქტურაში Collation სვეტში უნდა ჩავსვათ utf8_general_ci (ნახ.12.45-ა,ბ).

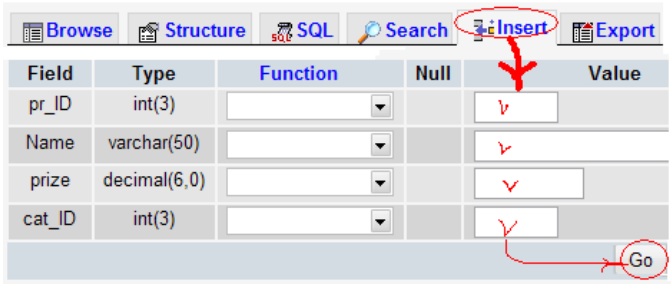


ნახ.12.45-ა



ნახ.12.45-ბ

შევავსოთ პროდუქციის ცხრილი მნიშვნელობებით Browse და Insert გადამრთველების გამოყენებით (ნახ.12.46-ა).



ნახ.12.46-ა

შევსებული ცხრილის ფრაგმენტი მოცემულია 12.46-ბ ნახაზზე.

←T→	pr_ID	Name	prize	cat_ID
	1	არაჟანი	4	6
	2	პური	1	1
	3	ძეხვი	11	5
	4	ხაჭო	3	6
	5	ფუნთუშა ქიშმიშით	1	1
	6	კოკა_კოლა	2	3
	7	ფანტა	2	3
	8	ღვინო ქინძმარაული	18	4
	9	ღვინო ხვანჭკარა	23	4
	10	ტირამუსი	7	2
	11	საქონლის ხორცი	18	5
	12	კონიაკი "ვარციხე"	20	4
	13	არაყი "გომი"	13	4
	14	ასატრიბა	25	7
	15	ვაშლი "გოლდენი"	3	8
	16	კონსერვი "შპროტი"	5	7
	17	ფორთოხალი	3	8
	18	მინერალური "ბორჯომი"	2	3
	19	მინერალური "წაბაღლავი"	1	3
	20	ორცხობილა "ხუშის"	4	2
	21	ნაყინი "ვანილის"	6	2
	22	მინერალური "ლიკანი"	2	3
	23	ღორის სამწვადე	23	5

Check All / Uncheck All *With selected:*

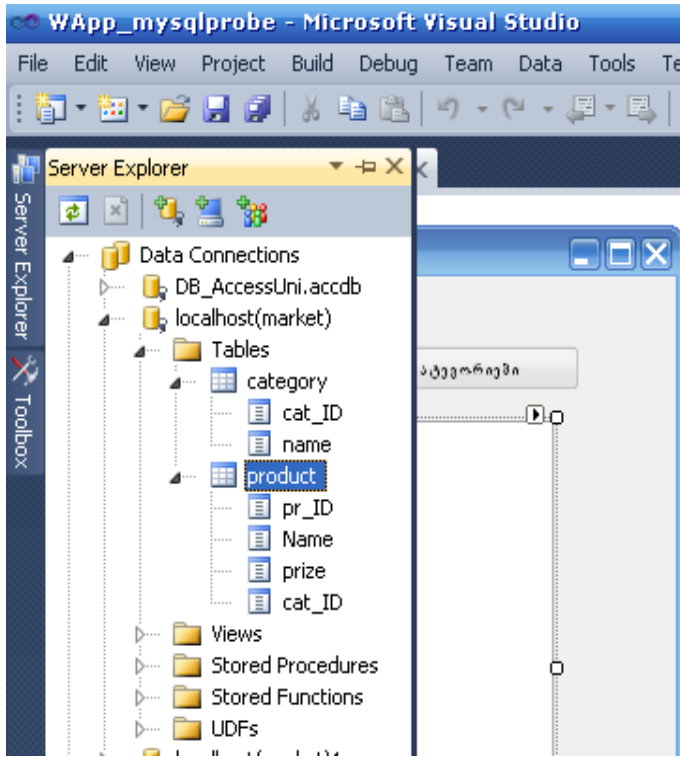
ნახ.12.46-ბ

ამგვარად, MySQL ბაზის ორი ცხრილი category და product მზადაა გამოსაყენებლად. დავხუროთ ეს ბაზა.

ახლა ავამუშავოთ Ms Visual Studio 2010 პაკეტი და C# ენის WindowsForm რეჟიმში Form1-ზე გამოვიტანოთ MySQL ბაზის მონაცემები, სხვადასხვა ჩრილში, მოთხოვნების შესაბამისად.

12.47 ნახაზზე ნაჩვენებია Server Explorer-ის ფანჯარა, სადაც localhost(market) მიერთებულ ბაზაში ჩანს Tables: category და

product თავიანთი ველებით (ატრიბუტებით). მთავარი მომენტია C# კოდიდან MySQL-ის market ბაზის მიერთება სამუშაოდ.



ნახ.12.47

პროგრამის კოდში უნდა მოთავსდეს MySQL ბაზის დასაკავშირებელი რამდენიმე სტრიქონი, რომელიც ქვემოთაა მოცემული.

```
MySql.Data.MySqlClient.MySqlConnection msqlConnection = null;
msqlConnection = new  MySql.Data.MySqlClient. MySqlConnection
     ("server=localhost;"+
      "User Id=user@localhost; database=market;
                                  Persist Security Info=True");
```

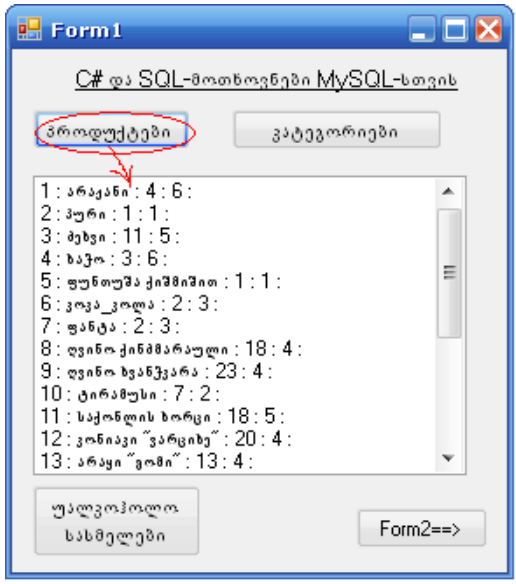
აქ განსაზღვრულია კლიენტის ბაზის მისაერთებლად აუცილებელი მონაცემები, როგორიცაა localhost-სერვერი, მომხმარებლის User-იდენტიფიკატორი და ბაზის სახელი market.

პროგრამის მთლიან კოდს შემდეგ განვიხილავთ. ახლა Form1-ფორმაზე მოთავსებული რამდენიმე ღილაკი განვიხილოთ.

ღილაკით ”**პროდუქტები**” ListBox1-ში გამოიტანება ყველა პროდუქტის სია, იდენტიფიკატორით, დასახელებით, ფასით და კატეგორიის ნომრით. მისი SQL-მოთხოვნის ”ამორჩევის” კოდს ექნება ასეთი სახე:

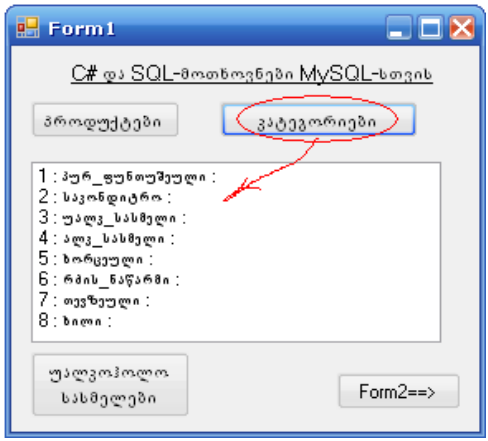
```
private void button1_Click(object sender, EventArgs e)
 {
   Amorcheva("select * from product order by pr_ID, Name",
                        "pr_ID", "Name", "prize", "cat_ID");
 }
```

შედეგები ასახულია 12.48 ნახაზზე მოცემულ ფანჯარაში.



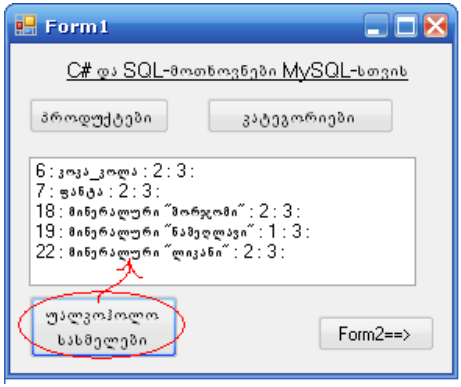
ნახ.12.48

ღილაკით ”**კატეგორიები**” ListBox1-ში გამოიტანება ბაზაში არსებული პროდუქციის ყველა კატეგორიის დასახელება. როგორც 12.49 ნახაზიდან ჩანს, სახეზეა 8 კატეგორია.



ნახ.12.49

ღილაკით ”**უალკოჰოლო სასმელები**” ListBox1-ში გამოიტანება იმ პროდუქტების სია, რომელთა ველში ”კატეგორიის იდენტიფიკატორი” შეესაბამება უალკოფოლო სასმელებს, ანუ ჩვენ შემთხვევაში `cat_ID=3`. შედეგი ასახულია 12.50 ნახაზზე.



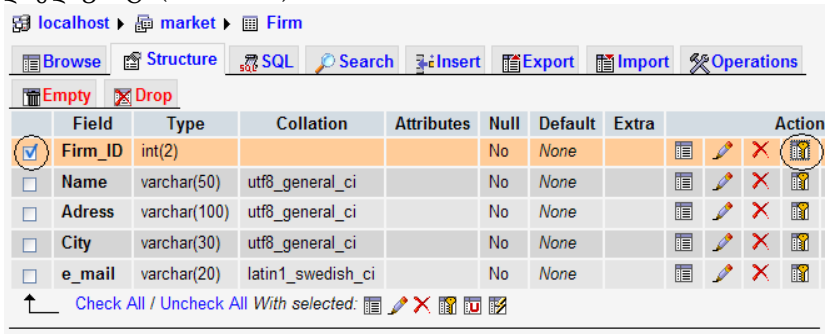
ნახ.12.50

პროგრამის კოდი მოცემულია 12.34 ლისტინგში.

```
// ლისტინგი_12.34 ----------
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace WApp_mysqlprobe
{
 public partial class Form1 : Form
 {
  public Form1() { InitializeComponent(); }
  private void Amorcheva(string sqlbefehl,
                                    params string[] velebi)
   {
    int i;
    string striqoni;
MySql.Data.MySqlClient.MySqlConnection msqlConnection = null;
    msqlConnection = new
MySql.Data.MySqlClient.MySqlConnection("server=localhost;"+
       "User Id=user@localhost;database=market;Persist
                                          Security Info=True");
MySql.Data.MySqlClient.MySqlCommand msqlCommand = new
                        MySql.Data.MySqlClient.MySqlCommand();
   msqlCommand.Connection = msqlConnection;
    msqlCommand.CommandText = sqlbefehl;
 try
   {
     msqlConnection.Open();
    MySql.Data.MySqlClient.MySqlDataReader msqlReader =
                                  msqlCommand.ExecuteReader();
    listBox1.Items.Clear();
    while (msqlReader.Read())
     {
      striqoni = "";
     for (i = 0; i < velebi.Length; i++)
        striqoni += msqlReader[velebi[i]] + " : ";
     listBox1.Items.Add(striqoni);
     }
   }
```

```
  catch (Exception ex)
    {
       MessageBox.Show(ex.Message);
    }
  finally
      {
        msqlConnection.Close();
      }
  }
 private void button1_Click(object sender, EventArgs e)
  {
    Amorcheva("select * from product order by pr_ID, Name",
                          "pr_ID", "Name", "prize", "cat_ID");
  }
 private void button2_Click(object sender, EventArgs e)
  {
    Amorcheva("select * from category order by cat_ID",
                                              "cat_ID", "Name");
  }
private void button3_Click(object sender, EventArgs e)
  {
    Amorcheva("select * from product where cat_ID=3 order by
           pr_ID", "pr_ID", "Name", "prize", "cat_ID");
  }
}}
```

ამოცანა_12.7: განხილულ ბაზას დავამატოთ ახალი ცხრილი პროდუქციის მწარმოებელი ფირმებისათვის, სახელით Firm, რომელსაც ექნება ხუთი ველი: იდენტიფიკატორი, დასახელება, მისამართი, ქალაქი და ელ-ფოსტა (ნახ.12.51-ა).



ნახ.12.51-ა

Browse გადამრთველით გამოვიტანოთ ფირმების ცხრილი და შევავსოთ იგი (ნახ.12.51-ბ).

Firm_ID	Name	Adress	City	e_mail
1	სანტე	წერეთლის 55	თბილისი	sante@gmail.ge
2	საქპური	გორგასლის 111	თბილისი	kalanda@puri.ge
3	ნიკორა	ფუჩიწაძის 200	თბილისი	nikora@gmail.ge
4	კოკა-კოლა	წერეთლის 30	თბილისი	cc@mail.ru
5	მითანა	რიონის 177	ქუთაისი	mitana@mail.ru
6	ნატახტარი	რუსთაველის 3	მცხეთა	natakhtari@gmail.com
7	თელიანი_ველი	ერეკლეს 55	თელავი	teliani@org.com
8	ქუთათური	აღმაშენებლის 222	ქუთაისი	varcikhe@mail.ru
9	So&Co	კოსტავას 77	თბილისი	so&co@gmail.ge
10	გორიციხე	სტალინის 15	გორი	gorivashla@hotmail.c
11	ქობულეთა	რუსთაველის 555	ქობულეთი	Citron@achara.ge
12	ბორჯომული	ლიკანის 20	ბორჯომი	borjomi@ckali.ge
13	ბახმარო	ნასაკირალის 1	ოზურგეთი	nabeglavi@hotmail.ge

ნახ.12.51-ბ

product ცხრილში საჭიროა დავამატოთ ველი Firm_ID, რომლითაც ის დაუკავშირდება ამ პროდუქციის მწარმოებელი ფირმის სტრიქონს. შევიტანოთ product ცხრილის Firm_ID ველის სვეტში შესაბამისი ფირმების იდენტიფიკატორები (ნახ. 12.52).

←T→	pr_ID	Name	prize	cat_ID	Firm_ID
☐ ✎ ✕	1	არაჟანი	4	6	1
☐ ✎ ✕	2	პური	1	1	2
☐ ✎ ✕	3	ძეხვი	11	5	3
☐ ✎ ✕	4	ხაჭო	3	6	1
☐ ✎ ✕	5	ფუნთუშა ქიშმიშით	1	1	3
☐ ✎ ✕	6	კოკა_კოლა	2	3	4
☐ ✎ ✕	7	ფანტა	2	3	6
☐ ✎ ✕	8	ღვინო ქინძმარაული	18	4	7
☐ ✎ ✕	9	ღვინო ხვანჭკარა	23	4	7
☐ ✎ ✕	10	ტირამუსი	7	2	5
☐ ✎ ✕	11	საქონლის ხორცი	18	5	5
☑ ✎ ✕	12	კონიაკი "ვარციხე"	20	4	8
☐ ✎ ✕	13	არაყი "გომი"	13	4	8
☐ ✎ ✕	14	ასატრიწა	25	7	9
☐ ✎ ✕	15	ვაშლი "გოლდენი"	3	8	10
☐ ✎ ✕	16	კონსერვი "შპროტი"	5	7	9
☐ ✎ ✕	17	ფორთოხალი	3	8	11
☐ ✎ ✕	18	მინერალური "ბორჯომი"	2	3	12
☐ ✎ ✕	19	მინერალური "წაბეღლავი"	1	3	13
☐ ✎ ✕	20	ორცხობილა "ნუშის"	4	2	5
☐ ✎ ✕	21	წაყინი "ვანილის"	6	2	1
☐ ✎ ✕	22	მინერალური "ლიკანი"	2	3	12
☐ ✎ ✕	23	ღორის სამწვადე	23	5	5

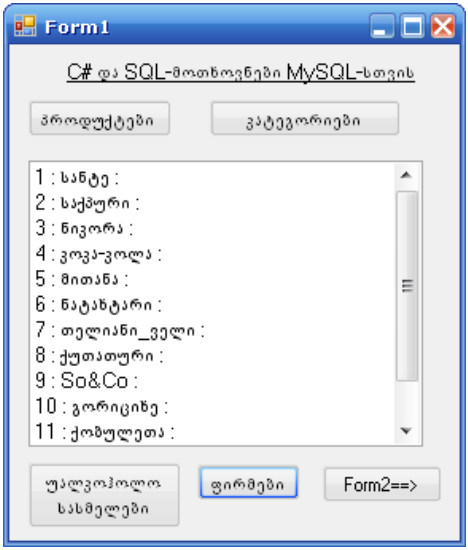
Check All / Uncheck All *With selected:* ✎ ✕

ნახ.12.52

ღილაკით ”**ფირმები**” ListBox1-ში გამოვიტანოთ პროდუქციის მწარმოებელი ფირმების სია. C#კოდის ფრაგმენტს ექნება შემდეგი სახე:

```
private void button5_Click(object sender, EventArgs e)
   {
     Amorcheva("select * from Firm order by Firm_ID",
                                          "Firm_ID", "Name");
   }
```

შედეგები მოცემულია 12.53 ნახაზზე.



ნახ.12.53

12.7. C# და Ms SQL Server

ძირითადი ამოცანა, რომელსაც ჩვენ აქ განვიხილავთ, არის ADO.NET პროგრამული პაკეტის გამოყენებით მომხმარებელთა სამუშაო ინტერფეისის დამუშავების სადემონსტრაციო მაგალითის აგება. ამასთანავე, მონაცემთა ბაზების სახით უნდა გამოვიყენოთ SQL Server პაკეტით დამუშავებული ცხრილები.

Microsoft Visual Studio NET სისტემაში შესვლის შემდეგ პირველ რიგში შევამოწმოთ მონაცემთა ბაზასთან კავშირი. ამისათვის მენიუდან View | Server Explorer-ით გამოვიტანოთ ფანჯარა (ნახ.12.54) და ავირჩიოთ HOMESERVER.BusinProc.dbo. გამოჩნდება SQL Server ბაზა, ცხრილები და ველები. DIAGRAM1-ის არჩევით მივიღებთ ცხრილებს კავშირებით, რომელიც იდენტურია SQL Server-ში ჩვენს მიერ შექმნილი ბაზისა. ამგვარად თავსებადობა კარგადაა რეალიზებული. 12.55 ნახაზზე ნაჩვენებია ცხრილთა კავშირების ფრაგმენტი, მხოლოდ მათი სათაურების ჩვენებით.

ახლა დავიწყოთ მომხმარებლის ინტერფეისის (ერთი სამუშაო ფორმის) დამუშავება. მენიუდან ავირჩიოთ

File | New | Project

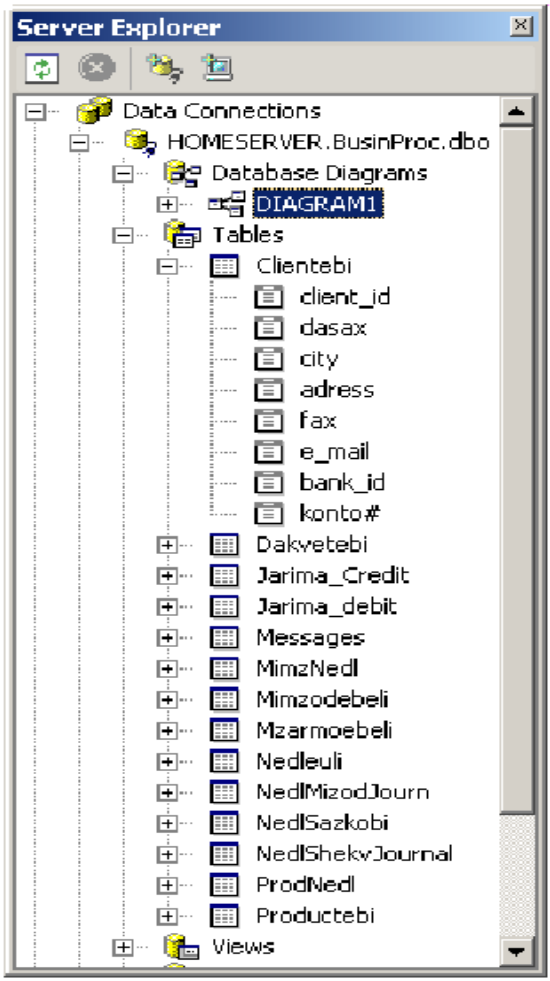
კომბინაცია და მივიღებთ 12.56 ნახაზზე ნაჩვენებ ფანჯარას. აქ შევარჩევთ სახელს (Name), ვინჩესტერზე შესანახ გზას და პაპკის სახელს (Location). Template-ში ავირჩევთ Windows Application.

:

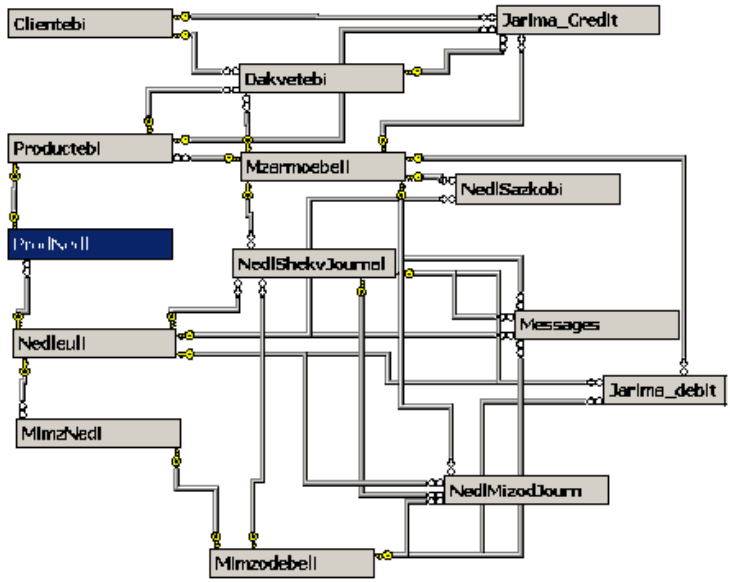
File | New Item

და მივიღებთ 12.57 ნახაზზე მოცემულ ფანჯარას.

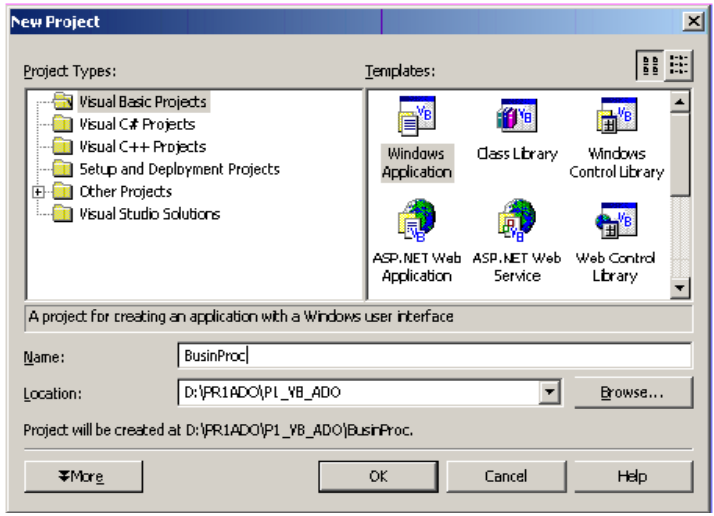
ავირჩიოთ Data Form Wizard და ეკრანზე გამოჩნდება ახალი ფორმა (ნახ.12.58). აქ ღილაკით Next გადავალთ შემდეგ ბიჯზე (ნახ.12.59).



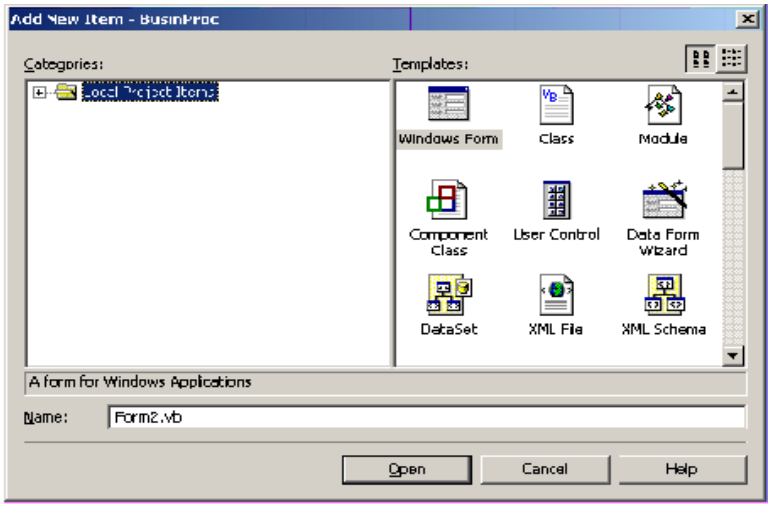
ნახ.12.54. მონაცემთა ბაზასთან დაკავშირება



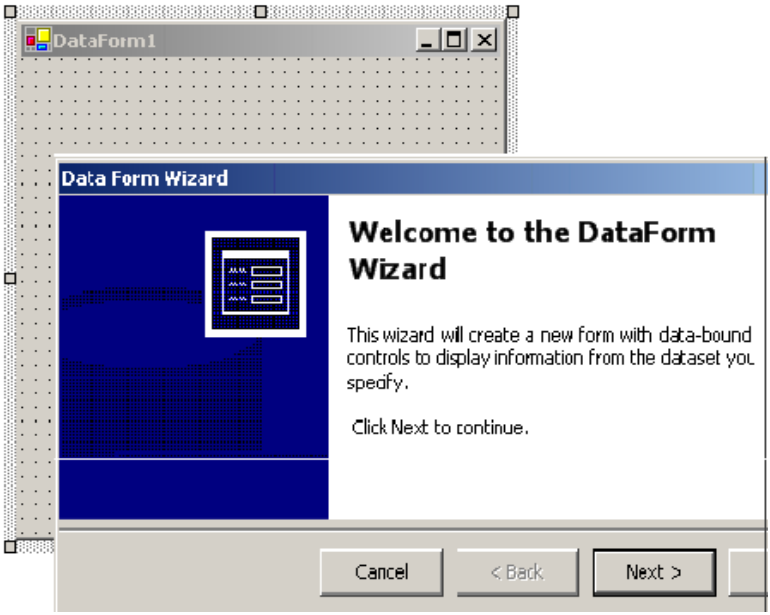
ნახ.12.55. SQL Server მონაცემთა ბაზის ასახვა ADO.NET-ში



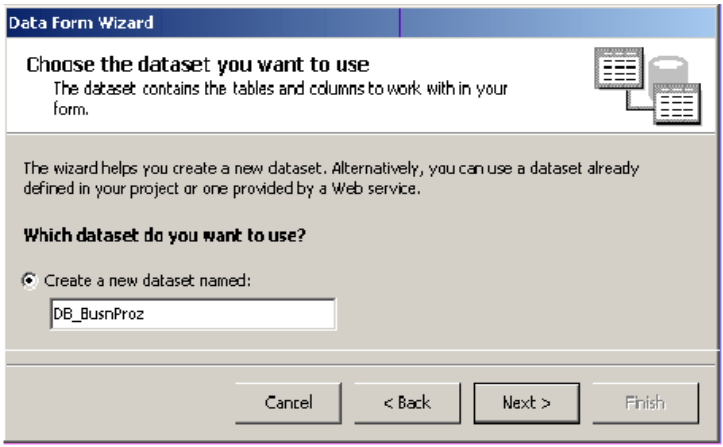
ნახ.12.56



ნახ.12.57



ნახ.12.58

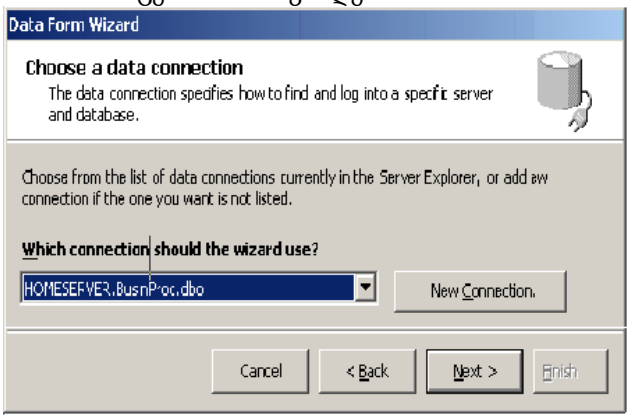


ნახ.12.59

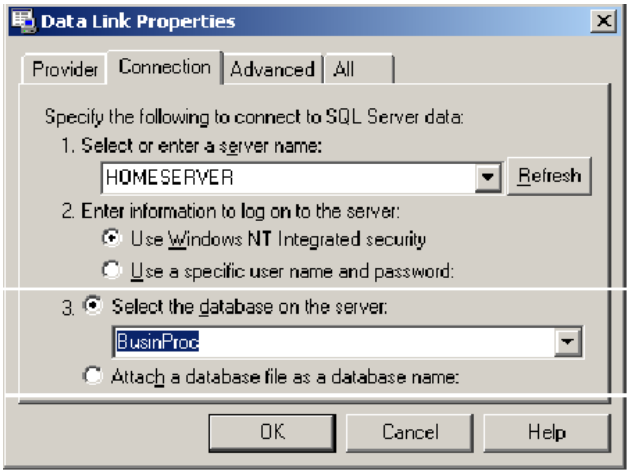
შევარჩიოთ მონაცემთა ბაზის სახელი და Next.

მონაცემთა ბაზასთან კავშირის დასამყარებლად 12.60 ნახაზის შესაბამისი ფანჯრიდან New Connection-ში შევარჩევთ სახელს.

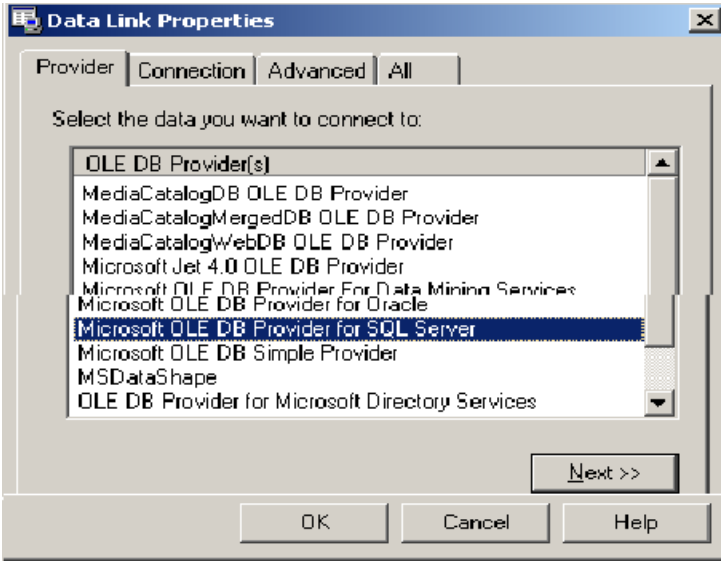
ამისათვის უნდა განვიხილოთ მომდევნო სამი ფანჯარა, შესაბამისად ნახ.12.61-ა,ბ,გ. როგორც ვხედავთ, ჩვენ შევარჩიეთ SQL Server-ის მონაცემთა პროვაიდერი.



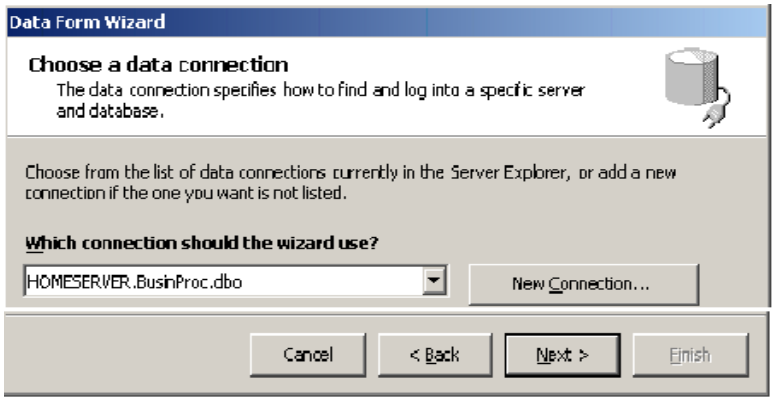
ნახ.12.60



ნახ.12.61-ა



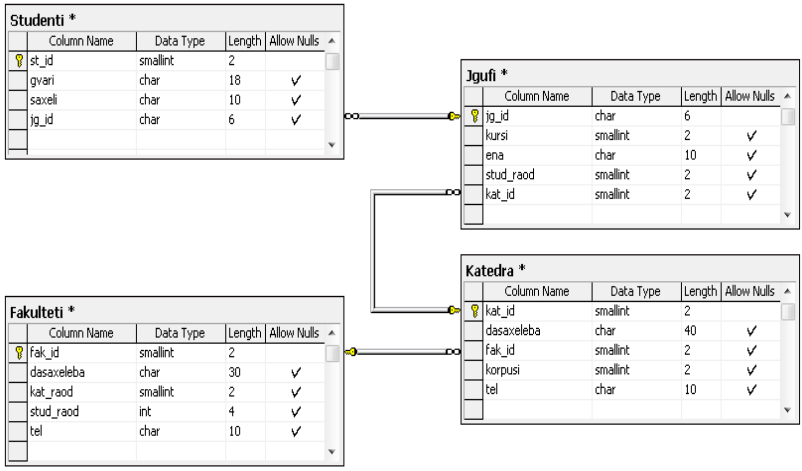
ნახ.12.61-ბ



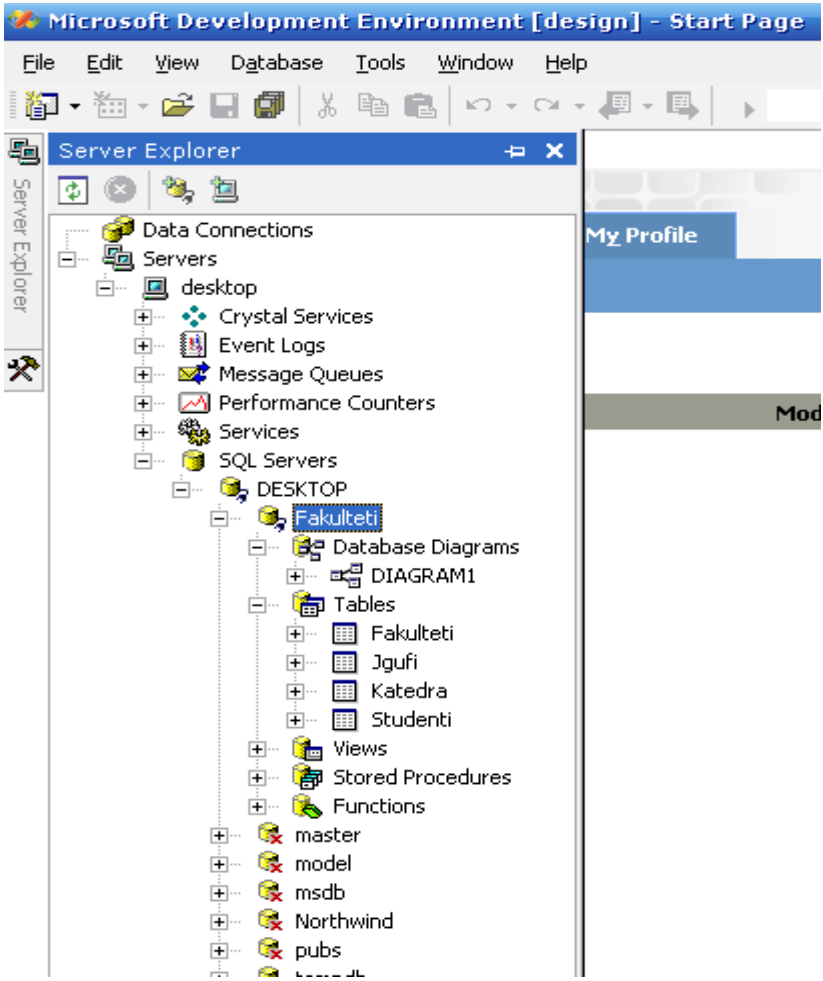
ნახ.12.61-გ

➢ სადემონსტრაციო მაგალითი MsSQL Server ბაზის, ADO.NET და C#ის გამოყენებით

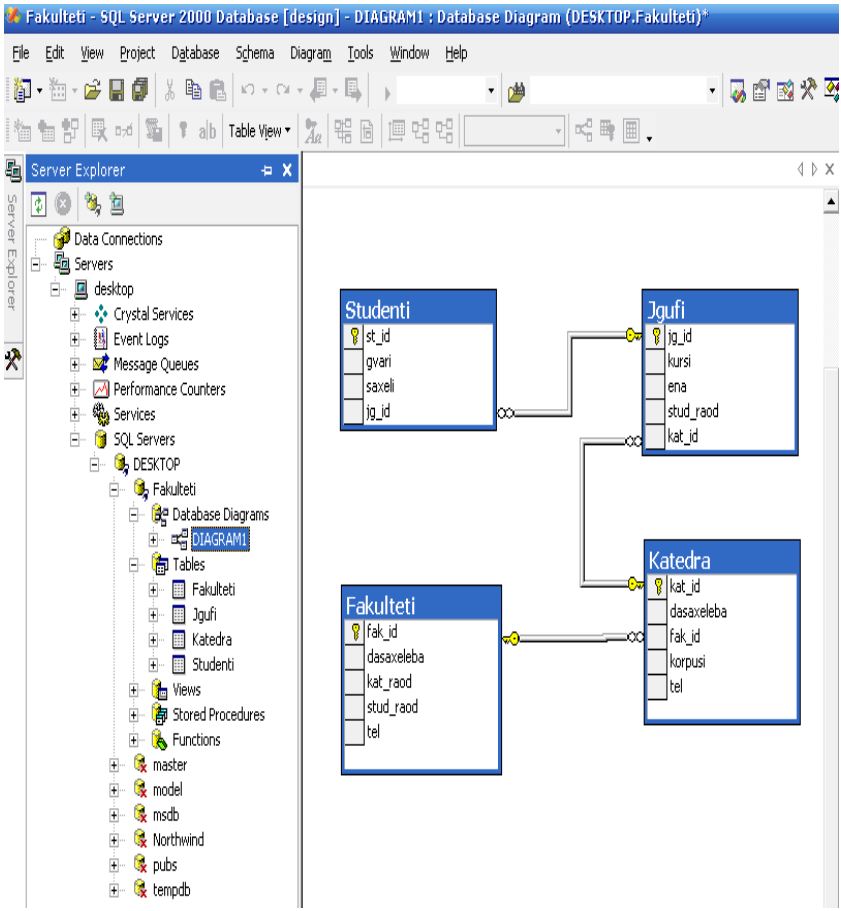
განვიხილოთ მაგალითი MsSQL_Server-ის მონაცემთა ბაზისათვის “ფაკულტეტი” (ნახ.12.62-12.67) ცხრილებით “სტუდენტები”, “ჯგუფები”, “ლექტორები” და “კათედრები”.



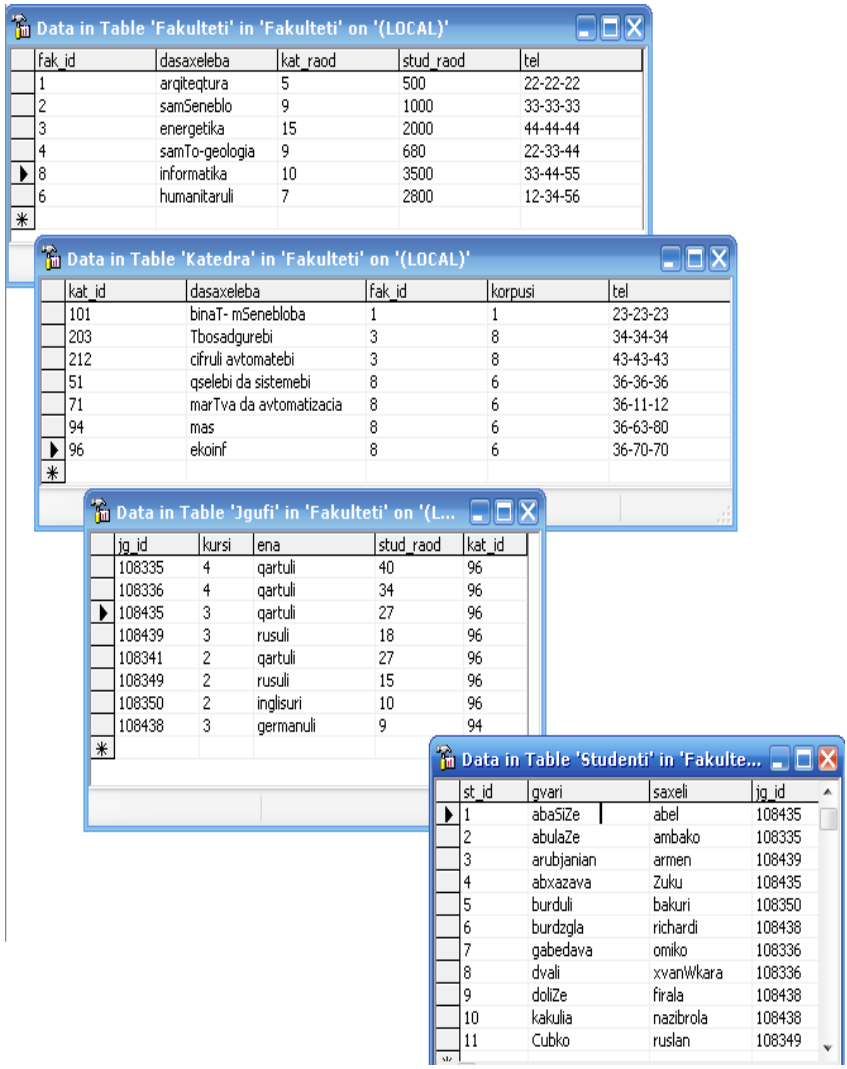
ნახ.12.62. MsSQL Server ბაზის სტრუქტურა



ნახ.12.63. MsVisual Studio ბაზასთან მიერთება



ნახ.12.64. ADO.NET ბაზის დიაგრამა



Data in Table 'Fakulteti' in 'Fakulteti' on '(LOCAL)'

fak_id	dasaxeleba	kat_raod	stud_raod	tel
1	arqiteqtura	5	500	22-22-22
2	samSeneblo	9	1000	33-33-33
3	energetika	15	2000	44-44-44
4	samTo-geologia	9	680	22-33-44
8	informatika	10	3500	33-44-55
6	humanitaruli	7	2800	12-34-56

Data in Table 'Katedra' in 'Fakulteti' on '(LOCAL)'

kat_id	dasaxeleba	fak_id	korpusi	tel
101	binaT- mSenebloba	1	1	23-23-23
203	Tbosadgurebi	3	8	34-34-34
212	cifruli avtomatebi	3	8	43-43-43
51	qselebi da sistemebi	8	6	36-36-36
71	marTva da avtomatizacia	8	6	36-11-12
94	mas	8	6	36-63-80
96	ekoinf	8	6	36-70-70

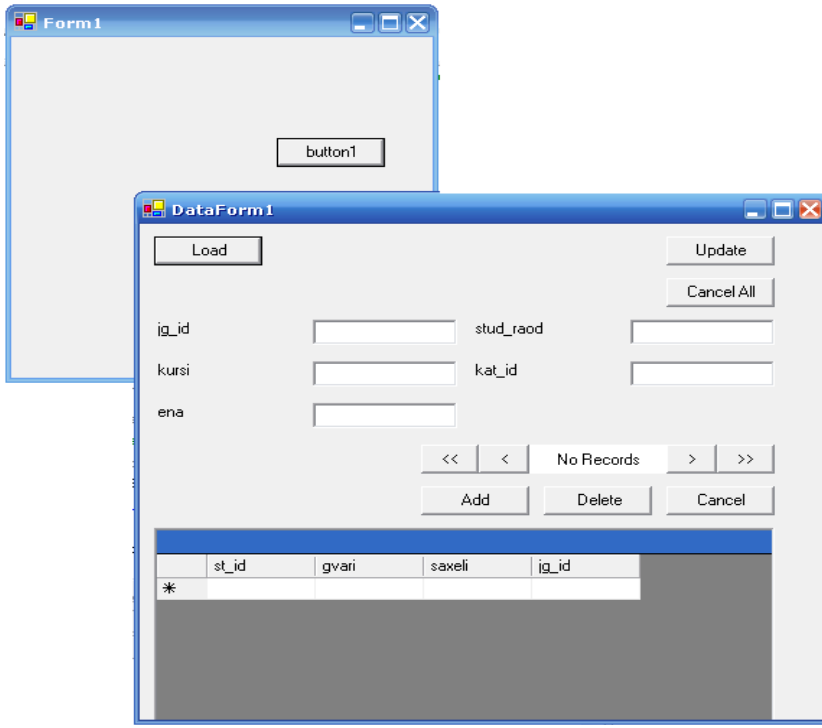
Data in Table 'Jgufi' in 'Fakulteti' on '(L...

jg_id	kursi	ena	stud_raod	kat_id
108335	4	qartuli	40	96
108336	4	qartuli	34	96
108435	3	qartuli	27	96
108439	3	rusuli	18	96
108341	2	qartuli	27	96
108349	2	rusuli	15	96
108350	2	inglisuri	10	96
108438	3	germanuli	9	94

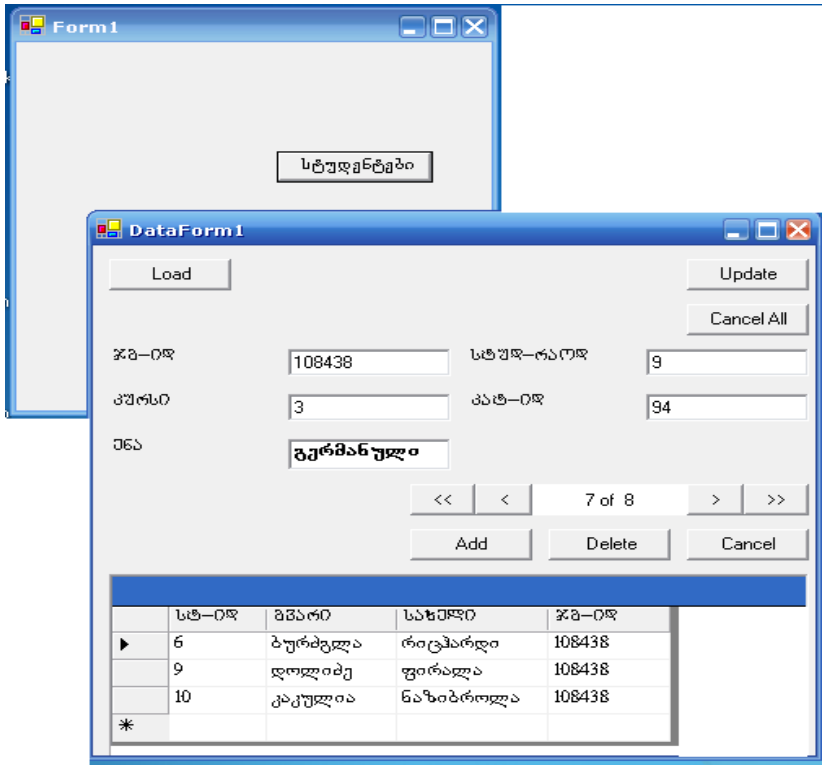
Data in Table 'Studenti' in 'Fakulte...

st_id	gvari	saxeli	jg_id
1	abaSiZe	abel	108435
2	abulaZe	ambako	108335
3	arubjanian	armen	108439
4	abxazava	Zuku	108435
5	burduli	bakuri	108350
6	burdzgla	richardi	108438
7	gabedava	omiko	108336
8	dvali	xvanWkara	108336
9	doliZe	firala	108438
10	kakulia	nazibrola	108438
11	Cubko	ruslan	108349

ნახ.12.65. ADO.NET ბაზის ცხრილები



ნახ.12.66. C# -ის ინტერფეისი



ნახ.12.67. Visual Studio-C# -ის შედეგები

შედეგად მიიღება C#.NEთ–ის ორი ფანჯარა, რომლის ცხრილებიც დაკავშირებულია MsSQL_Server-ის მონაცემთა ბაზასთან ADO.NET პროვაიდერის საშუალებით.

12.8. კითხვები და სავარჯიშოები:

12.1. რას წარმოადგენს ADO.NET და როგორია მისი სტრუქტურა ?

12.2. როგორ გამოიყენება ობიექტი DataSet ელემენტი მონაცემებთან მიმართვისას ?

12.3. რისთვის და როგორ გამოიყენება DataTable ობიქტი მისი DataColumns და DataRows ერთობლიობებით ?

12.4. რა ფუნქციას ასრულებს DataSet ობიექტის DataRelations ერთობლიობა ?

12.5. რისთვის და როგორ გამოიყენება პროვაიდერები: SQL Server .NET Data Provider და OleDb .NET Data Provider ?

12.6. რისთვის და როგორ გამოიყენება ობიექტი Connection ?

12.7. ააგეთ მონაცემთა ბაზა Ms Acess ორი დაკავშირებული ცხრილით (მაგალითად, „პროდუქტები“ და „კატეგორიები“). განახორციელეთ პროგრამიდან ცხრილებთან მუშაობა (ჩაამატეთ, შეცვალეთ, წაშალეთ და მოიძიეთ საჭირო მონაცემები ბაზიდან).

12.8. ააგეთ მონაცემთა ბაზა Ms Acess სამი დაკავშირებული ცხრილით (მაგალითად, „პროდუქტები“, „კატეგორიები“ და „ფირმები“). გამოიყენეთ DataSet ობიექტი და განახორციელეთ ცხრილებთან მუშაობა (ჩაამატეთ, შეცვალეთ, წაშალეთ და მოიძიეთ საჭირო მონაცემები ბაზიდან).

12.9. ააგეთ მონაცემთა ბაზა MySQL ორი დაკავშირებილი ცხრილით (მაგალითად, „სტუდენტები“ და „ჯგუფები“). განახორციელეთ ცხრილებთან მუშაობა (ჩაამატეთ, შეცვალეთ, წაშალეთ და მოიძიეთ საჭირო მონაცემები ბაზიდან).

12.10. ააგეთ მონაცემთა ბაზა MySQL ოთხი დაკავშირებილი ცხრილით (მაგალითად, „სტუდენტები“, „ლექტორები”, „ჯგუფები“ და „ლექტორ-ჯგუფები“). გამოიყენეთ DataSet ობიექტი და განახორციელეთ ცხრილებთან მუშაობა (ჩაამატეთ, შეცვალეთ, წაშალეთ და მოიძიეთ საჭირო მონაცემები ბაზიდან).

12.11. ააგეთ მონაცემთა ბაზა Ms_SQL_Server ორი დაკავშირებილი ცხრილით (მაგალითად, ევროლიგის „ფეხბურთელები“ და „გუნდები“). განახორციელეთ ცხრილებთან მუშაობა (ჩაამატეთ, შეცვალეთ, წაშალეთ და მოიძიეთ საჭირო მონაცემები ბაზიდან).

12.12. ააგეთ მონაცემთა ბაზა MySQL სამი დაკავშირებილი ცხრილით (მაგალითად, ევროპის „ქვეყნები“, „ფეხბურთელები“ და „გუნდები“). გამოიყენეთ DataSet ობიექტი და განახორციელეთ ცხრილებთან მუშაობა (ჩაამატეთ, შეცვალეთ, წაშალეთ და მოიძიეთ საჭირო მონაცემები ბაზიდან).

III ნაწილი:

Visual C# და ASP.NET: Web-აპლიკაციების აგება

13.1. შესავალი ASP.NET სისტემაში

ASP.NET (Active Server Pages – აქტიური სერვერული გვერდები) – არის NET-პლატფორმის ნაწილი და ტექნოლოგია, რომელიც დინამიკურად ქმნის დოკუმენტებს Web-სერვერზე, როცა ისინი მოითხოვება HTTP-ს საშუალებით.

ASP.NET ტექნოლოგია ანალოგიურია PHP, ColdFusion და სხვ. ტექნოლოგიების, მაგრამ მათ შორის მნიშვნელოვანი განსხვავებაცაა. ASP.NET, როგორც მისი დასახელება გვიჩვენებს, შეიქმნა სპეციალურად NET პლატფორმასთან სრული ინტეგრაციის მიზნით, რომლის ნაწილიც ითვალისწინებს C# ენის მხარდაჭერას.

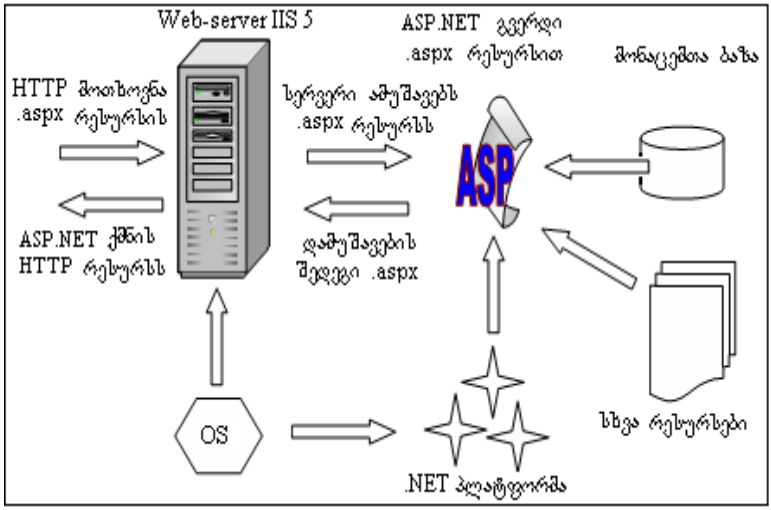
როგორც ცნობილია, Web-გვერდების დასაპროგრამებლად გამოიყენება ისეთი სცენარების ენები, როგორიცაა VBScript ან JScript. ეს სკრიპტული ენები მუშაობდა, მაგრამ ხშირად გარკვეულ პრობლემებს უქმნიდა დაპროგრამების “ნამდვილი” ენების პროგრამისტებს სხვადასხვა ადმინისტრატორული დავალებათა შესრულებისას, რაც საბოლოო ჯამში აისახებოდა სისტემის მწარმოებლურობის დაქვეითებაში.

მაღალგანვითარებული ენების გამოყენების შემთხვევაში, მაგალითად, შესაძლებელია მუშაობის პროცესის უზრუნველყოფა სრული სერვერული ობიექტური მოდელით. ASP.NET ახორციელებს მიმართვას გვერდის ყველა მმართველ ელემენტთან, როგორც ზოგადად გარემოს ობიექტებთან. სერვერის მხარესაც კი ხორციელდება წვდომა .NET–ის ყველა საჭირო კლასთან.

გვერდების მართვის ელემენტები ფუნქციონალურია და ფაქტობრივად, შესაძლებელია ყველაფრის გაკეთება, რასაც Windows-ის ფორმის კლასებთან ვაკეთებდით, რაც უფრო მოქნილს ხდის სისტემას. ამის გამო ASP.NET-ის გვერდებს, რომლებიც ქმნის HTML შედგენილობას, ხშირად უწოდებენ Web-ფორმებს.

13.2. ASP.NET–ის საბაზო არქიტექტურა

ASP.NET გამოიყენებს ინტერნეტის ინფორმაციულ სერვერს (IIS – Internet Information Server) HTTP მოთხოვნებზე საპასუხო შინაარსის მისაწოდებლად. ASP.NET გვერდები მოთავსებულია ფაილებში .aspx გაფართოებით. მისი საბაზო არქიტექტურა იხ. 13.1 ნახაზზე.



ნახ.13.1. საბაზო არქიტექტურა

ASP.NET-ის დამუშავების დროს მისაწვდომია .NET-ის ყველა კლასი, სპეციალური კომპონენტები, შექმნილი C# ან სხვა ენებზე, მონაცემთა ბაზები და ა.შ. ფაქტობრივად, სახეზეა ყველა ის შესაძლებლობა, რომელსაც იყენებს C# დანართის აგებისას. ე.ი., C#-ის გამოყენება ASP.NET-ში ეფექტურს ხდის დანართის შესრულებას.

ASP.NET ფაილი შეიძლება შეიცავდეს ნებისმიერ ელემენტს შემდეგი სიიდან:

- ინსტრუქციის დამუშავება სერვერისთვის;

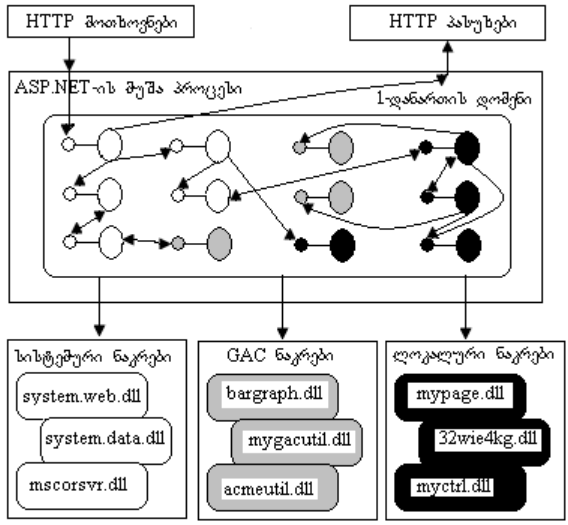
- კოდები ენებზე: C#, VB.NET, Jscript.NET ან სხვ., რომელთა მხარდაჭერა ხდება .NET პლატფორმით;

- შინაარსი ნებისმიერი ფორმით, რომელიც გენერირდება რესურსის სახით HTML–ით;

- ASP.NET-ის ჩადგმული სერვერული მართვის ელემენტები.

ამგვარად, შესაძლებელია ASP.NET ფაილის არსებობა, რომელიც მხოლოდ ერთი სტრიქონისგან შედგება, მაგალითად, Hello:, ყოველგვარი სხვა კოდის ან ინსტრუქციის გარეშე. ამის საფუძველზე იქმნება დასაბრუნებელი HTML გვერდი, რომელშიც მოთავსებულია აღნიშნული ტექსტი.

ამგვარად, ASP.NEთ-ის ბირთვია .NET-ის კლასების ერთობლიობა, რომელიც ემსახურება HTTP მოთხოვნების დამუშავებას. ზოგიერთი მათგანი ამ კლასებიდან განსაზღვრულია სისტემურ ნაკრებში (Assembly), როგორც .NET-ის საბაზო კლასების ბიბლიოთეკის ნაწილი. ზოგი შეიძლება მოთავსებული იყოს გლობალური კეშის ნაკრებში (GAC – Global Assembly Cache), ზოგიც შეიძლება ჩაიტვირთოს ლოკალური ნაკრებიდან, რომელიც განთავსებულია დანართის ვირტუალურ კატალოგში. ყველა კლასი, რომლებიც ემსახურება HTTP მოთხოვნებს, ჩაიტვირთება დანართის დომენში (დანართის არე – Application area) ASP.NET-ის მუშა პროცესში, და ურთიერთქმედებს ერთმანეთთან, აფორმირებს მოთხოვნებზე პასუხებს. 13.2 ნახაზი ასახავს ASP.NET-ის საბაზო არქიტექტურას დონეების მიხედვით. იგი 13.1 ნახაზის დეტალურად ასახსნელადაა შემოტანილი.



ნახ.13.2. საბაზო არქიტექტურა დონეების მიხედვით

ASP.NET-ის ძირითადი სიახლე მის წინამორბედებთან შედარებით ისაა, რომ აქ ყვლა არსი აისახება კლასის საშუალებით, რომელიც ნაკრებიდან ჩაიტვირთება.

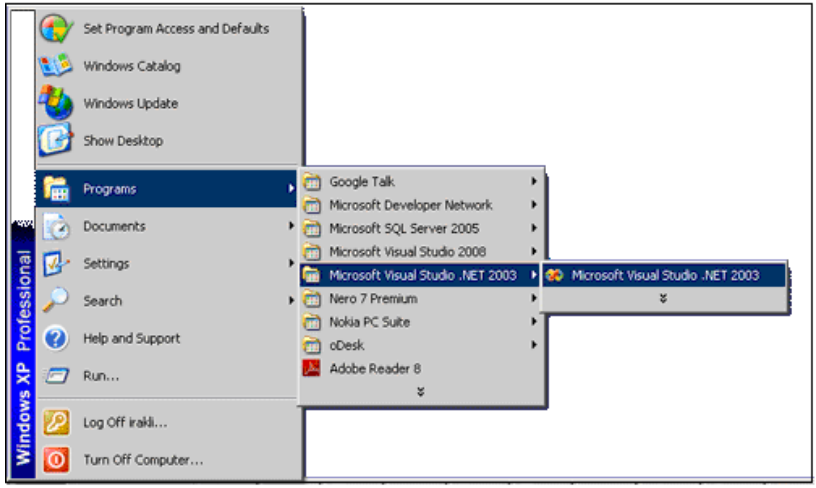
დანართების აგება ASP.NET-ში ხდება კლასების კონსტრუირებით, რომლებიც ურთიერთმოქმედებს სხვა კლასებთან. ზოგი კლასი იწარმოება პლატფორმის საბაზო კლასებიდან, ზოგს შეუძლია ინტერფეისების რეალიზება ამ პლატფორმაში, ზოგიც ურთიერთქმედებს პლატფორმის საბაზო კლასებთან მათი მეთოდების გამოძახების გზით.

ASP.NET-ის კლასიკური სინტაკსი ისევ შენარჩუნებულია, ოღონდაც მისი კოდები, რომლებიც ინახება ნაკრების ფაილებში სერვერის მხარეს, გარდაქმნილია კლასების განსაზღვრების სახით.

13.3. ASP.NET აპლიკაციის შექმნის ეტაპები

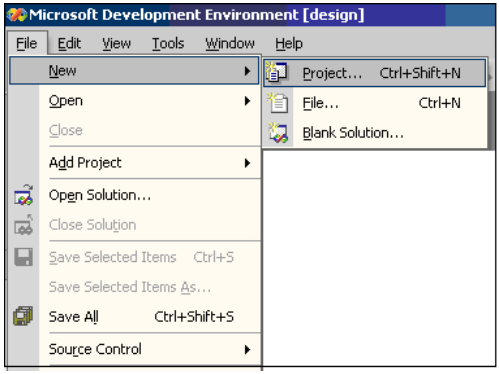
ASP.NET აპლიკაციის შესაქმნელად საჭიროა შემდეგი საფეხურების შესრულება:

1. პროგრამების პანელიდან ავირჩიოთ:
   Start->Programs->Microsoft Visual Studio .NET 2003



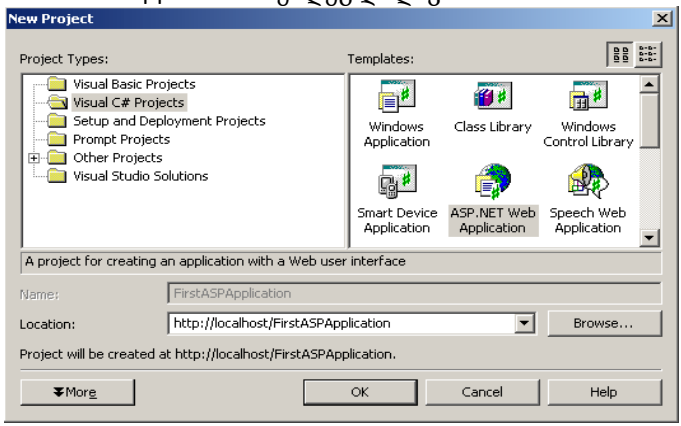
ნახ.13.3

2. პროგრამის გაშვების შემდეგ აირჩიეთ მენიუს პუნქტი:
   File -> New -> Project.



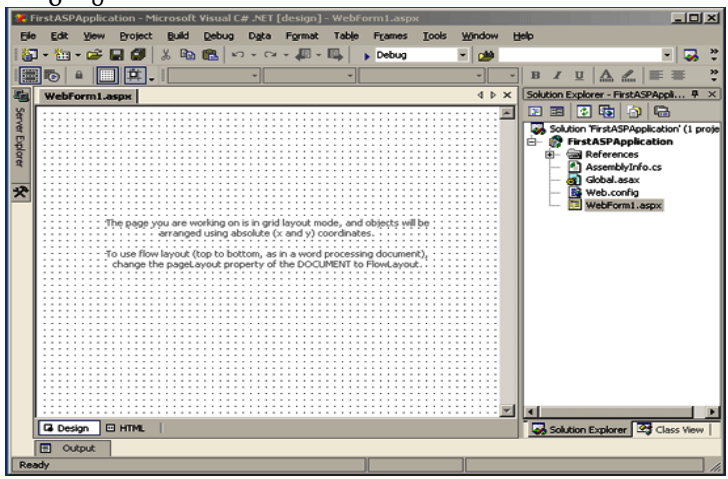
ნახ.13.4

3. გაიხსნება ახალი პროექტების ტიპის არჩევის ფანჯარა. Project Types ფანჯარაში აირჩიეთ Visual C# Project, ხოლო Templates ფანჯარაში ASP.NET Web Application. Location ველში http://localhost/ შემდეგ აკრიფეთ აპლიკაციის სახელი, ამ შემთხვევაში აპლიკაციის სახელია FirstASPApplication. შემდეგ ღილაკი OK.



ნახ.13.5

4. Visual Studio შექმნის აპლიკაციას და გახსნის Microsoft Visual C#.NET გარემოში.



ნახ.13.6

შექმნილ პროექტს შექმისთანავე შეიცავს რამდენიმე ფაილს:

- AssemblyInfo.cs
- Global.asax
- Web.config
- WebForm1.aspx და WebForm1.aspx.cs

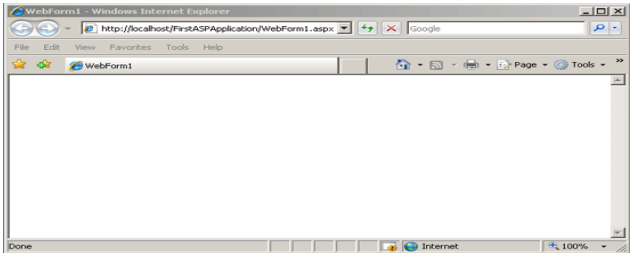
AssemblyInfo.cs არის შეიცავს ინფორმაციას აპლიკაციის შესახებ, როგორიცაა აპლიკაციის სახელი, ავტორი, ვერსია და სხვა.

Global.asax ფაილი ემსახურება აპლიკაციის დონის მოვლენების დამუშავებას, როგორიცაა შეცდომების დაჭერა, ახალი სესიის შექმნა, სესიის დასრულება და სხვა.

Web.config არის XML ფაილი რომელიც შეიცავს აპლიკაციის კონფიგურაციის მონაცემებს: სესიის პარამეტრები, მონაცემთა ბაზასთან კავშირის პარამეტრები, მომხმარებლების ავტორიზაციასა და აუთენთიფიკაციის კონფიგურაციის პარამეტრები.

WebForm1.aspx და WebForm1.aspx.cs ქმნიან ერთ ვებ-გვერდს. WebForm1.aspx ფაილი შეიცავს ვიზუალურ ელემენტებს, ხოლო WebForm1.aspx.cs ვებ-ფორმის კლასის მოვლენების დამუშავების მეთოდებს და ბიზნეს ლოგიკას.

5. შექმნილი ვებ-გვერდის ნახვა შესაძლებელია CTRL+F5 დაჭერით ან მენიუს პუნქტი Debug -> Start Without Debugging არჩევით:



ნახ.13.7

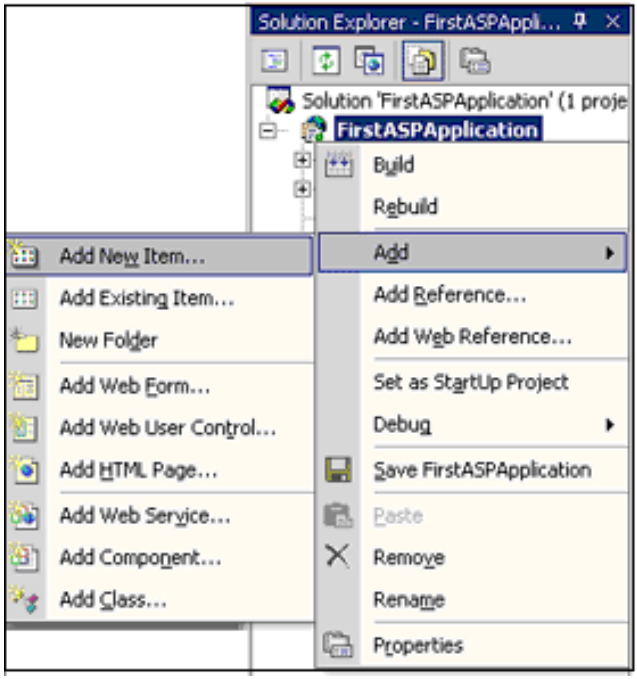
გახსნილი ვებ-გვერდი ცარიელია, რადგან არ შეიცავს ვიზუალურ ელემენტებს.

13.4. ახალი ვებ-გვერდის შექმნა

შექმნილ აპლიკაციას დავამატოთ ახალი ASPX ვებ-გვერდი.

ახალი ფაილის დამატება შესაძლებელია კლავიატურის ღილაკების კომბინაცის CTRL+SHIFT+A საშუალებით, ან მონიშნეთ პროექტი Solution Explorer ფანჯარაში, თაგუს მარჯვენა ღილაკზე დაჭერით გამოიძახეთ კონტექსტური მენიუ და აირჩიეთ:

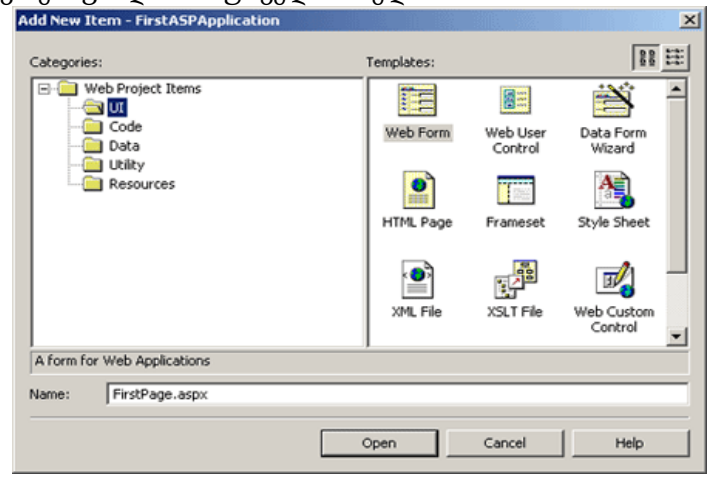
Add->Add New Item



ნახ.13.8

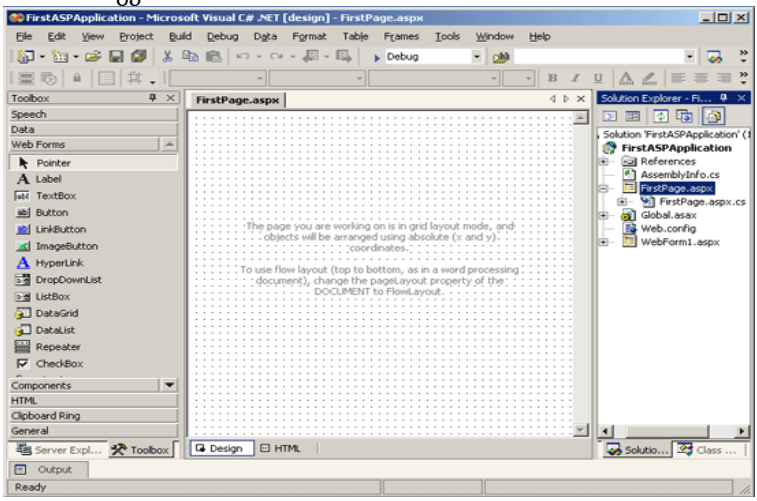
გაიხსნება ახალი ფაილის არჩევის ფანჯარა, მარცხენა ფანჯარაში შესაძებელია ფაილების კატეგორიის არჩევა, ხოლო მარცხენაში ფანჯარაში წინასწარ განსაზღვრეული შაბლონებიდან შესაბამისი ტიპის ფაილების არჩევა. კატეგორიების ფანჯარაში

აირჩიეთ UI ხოლო შაბლონების ფანჯარაში Web Form. Name ველში შეიყვანეთ ფაილის სასურველი სახელი:



ნახ.13.9

Open ღილაკზე დაჭერით პროექტს დაემატება ახალი ფაილი FirstPage.aspx და FirstPage.aspx.cs. გვერდი ავტომატურად გაიხსნება დიზაინის რეჟიმში.



ნახ.13.10

13.5. Web-გვერდის ვიზუალური და პროგრამული ნაწილების განცალკევება

ASP.NET გვერდები შეიძლება შედგებოდეს 2 ფაილისგან: 1. ვიზუალური გვერდისგან, სადაც განთავსებულია HTML კოდი და კომპონენტები; 2. კოდის ფაილისგან, სადაც მუშავდება ვებ-ბრაუზერის მომხმარებლის მიერ ინიცირებული სხვადასხვა შეტყობინებები და ბიზნეს ლოგიკის პროცედურები და ფუნქციები. ეს მეთოდი ცნობილია Codebehind-ის სახელწოდებით. განვიხილოთ მაგალითი, ამ მეთოდის გამოყენებით:

1. შევქმნათ ახალი ვებ-აპლიკაცია ან გავხსნათ უკვე არსებული;
2. დავამატოთ ახალი ვებ-გვერდი VisualPage.aspx (ცარიელი ASPX ფაილი);
3. დავამატოთ ახალი C# ფაილი ProgramPage.cs (ცარიელი C# ფაილი);
4. VisualPage.aspx შევავსოთ HTML კოდით და სერვერული კონტროლებით:

```
<%@ Page Language="C#" %>
<html>
 <head>
 </head>
 <body>

 </body>
</html>
```

ნახ.13.11

5. ProgramPage.cs ფაილში საჭიროა შემდეგი ტექსტის შეტანა:

```
using System;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;
```

```
public class ProgramPage : System.Web.UI.Page
{  private void Page_Load(object sender,
     System.EventArgs e) {   }

override protected void OnInit(EventArgs e)
  {
   InitializeComponent();
   base.OnInit(e);
  }

  private void InitializeComponent()
  {
   this.Load += new
System.EventHandler(this.Page_Load);
  }
 }
```

6. VisualPage.aspx ფაილში <body></body> ტეგებს შორის უნდა დაემატოს სერვერული კონტროლი:

```
<body>
      <asp:Label Runat="server"
                 ID="CurrentDate"></asp:Label>
</body>
```

ნახ.13.12

7. VisualPage.aspx გვერდის დასაწყისში დავამატოთ დირექტივა:

```
 <%@ Page language="c#"
Codebehind="ProgramPage.cs"
AutoEventWireup="false"
Inherits="ProgramPage" %>
```

ეს დირექტივა მიუთითებს, რომ ამ გვერდის მოვლენები დამუშავდება ფაილში ProgramPage.cs და გვერდის შესაბამისი კლასის სახელია ProgramPage.

8. VisualPage.aspx მოთავსებულია სერვერულ კონტროლზე მიმართვისთვის პროგრამის კოდში. საჭიროა ეს კონტროლი აღიწეროს ProgramPage.cs კლასში, ამიტომ მას დავამატოთ ველი:

```
public class ProgramPage : System.Web.UI.Page
 {
      protected System.Web.UI.WebControls.Label
      CurrentDate;
...
```

ნახ.13.13

System.Web.UI.WebControls.Label მიუთითებს რომ ამ ველის ტიპია სერვერული კონტროლი Label-ის კლასი.

9. მიმდინარე თარიღის და დროის გამოსატანად სერვერული კონტროლის საშუალებით გვერდის ჩატვირთვის მოვლენის დამუშავების პროცედურაში-Page_Load მეთოდში კონტროლის მიმართვა ხდება მისი იდენტიფიკატორის მიხედვით CurrentDate და მის ატრიბუტს Text მიენიჭება მიმდინარე თარიღი და დროის მნიშვნლეობა:

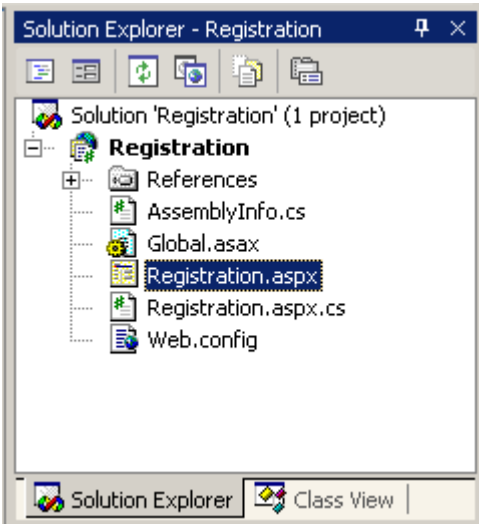
```
  private void Page_Load(object sender,
System.EventArgs e)
  {
   CurrentDate.Text =
System.DateTime.Now.ToString();
  }
```

ნახ.13.14

13.6. ინტერაქტიული Web-გვერდის შექმნა

შევქმნათ ვებ-გვერდი რომელზეც მომხმარებელი შეიყვანს საკუთარ მონაცემებს და გადააგზავნის სერვერზე.

შექმენით ახალი ASP.NET აპლიკაცია სახელით Registration. დაამატეთ ფაილები Registration.aspx და Registration.aspx.cs.



ნახ.13.15

ვებ-გვერდის მოთავსებულია სერვერული კონტროლები: form, asp:TextBox, asp:TextBox, asp:DropDownList, asp:CheckBoxList, asp:Button, asp:Label. გვერდის ჩატვირთვისადსს ღილაკზე დაჭერისას გამოიძახება onclick მოვლენაზე მიბმული მეთოდი Register_Click.

გახსენით Registration.aspx და შეიყვანეთ შემდეგი კოდი:

```
<%@ Page language="c#"
Codebehind="Registration.aspx.cs"
  AutoEventWireup="false"
Inherits="FirstASPApplication.Registration" %>
<HTML>
 <HEAD>
  <title>რეგისტრაციის ფორმა</title>
 </HEAD>
 <body>
  <form method="post" runat="server"
id="registration">
   შეიყვანეთ საკუთარი მონაცემები:
   <table border="1">
```

```
    <tr>
     <td>სახელი:</td>
     <td>
      <asp:TextBox id="FirstName"
runat="server"></asp:TextBox></td>
    </tr>
    <tr>
     <td>გვარი:</td>
     <td>
      <asp:TextBox id="LastName"
runat="server"></asp:TextBox></td>
    </tr>
    <tr>
     <td>სქესი:</td>
     <td><asp:RadioButtonList id="Sex"
runat="server"
                  RepeatDirection="Horizontal">
       <asp:ListItem
Value="მდედრობითი"></asp:ListItem>
       <asp:ListItem
Value="მამრობითი"></asp:ListItem>
      </asp:RadioButtonList></td>
    </tr>
    <tr>
     <td>ქალაქი</td>
     <td><asp:DropDownList id="City"
runat="server">
       <asp:ListItem
Value="თბილისი"></asp:ListItem>
       <asp:ListItem
Value="ქუთაისი"></asp:ListItem>
       <asp:ListItem
Value="რუსთავი"></asp:ListItem>
       <asp:ListItem Value="გორი"></asp:ListItem>
       <asp:ListItem Value="ბათუმი"></asp:ListItem>
       <asp:ListItem Value="თელავი"></asp:ListItem>
      </asp:DropDownList></td>
    </tr>
```

```
    <tr>
     <td>ინტერესების სფერო:</td>
     <td>
      <asp:CheckBoxList id="Interests"
runat="server">
       <asp:ListItem Value="საინფორმაციო
ტექნოლოგიები"></asp:ListItem>
       <asp:ListItem
Value="სამართალმცოდნეობა"></asp:ListItem>
       <asp:ListItem Value="ეკონომიკა და
მენეჯმენტი"></asp:ListItem>
       <asp:ListItem Value="სამშენებლო
სფერო"></asp:ListItem>
      </asp:CheckBoxList>æ/td>
    </tr>
   </table>
   <asp:Button id="Register" runat="server"
Text="რეგისტრაცია"></asp:Button>
   <br />
   <asp:Label id="Message"
runat="server"></asp:Label>
  </form>
 </body>
</HTML>
```

ნახ.13.16

ვხსნით Registration.aspx.cs და შეგვაქვს შემდეგი კოდი:

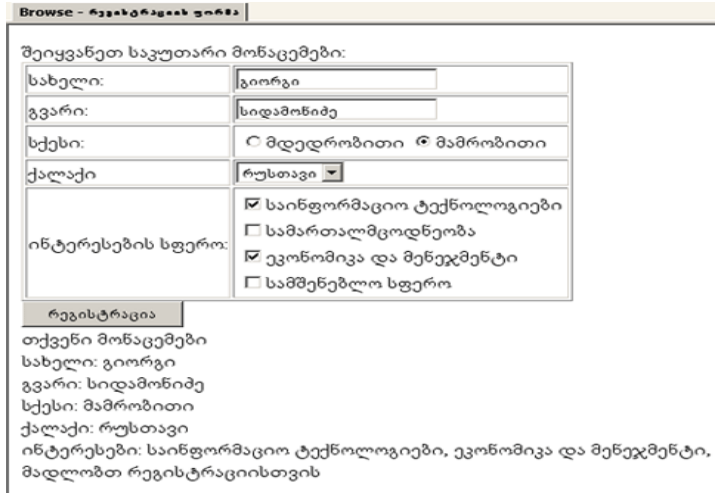
```
using System;
using System.Text;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;
// გაგრძელება შემდეგ გვერდზე
```

```
namespace FirstASPApplication
{
 public class Registration : System.Web.UI.Page
  {
   protected System.Web.UI.WebControls.TextBox
FirstName;
   protected System.Web.UI.WebControls.TextBox
LastName;
   protected
System.Web.UI.WebControls.RadioButtonList Sex;
   protected System.Web.UI.WebControls.DropDownList
City;
   protected System.Web.UI.WebControls.CheckBoxList

Interests;
   protected System.Web.UI.WebControls.Label Message;
   protected System.Web.UI.WebControls.Button
Register;

    private void Register_Click(object sender,
                                System.EventArgs e)
   {
      StringBuilder sb = new StringBuilder();
      sb.Append("თქვენი მონაცემები<br>");
      sb.AppendFormat("სახელი: {0}<br>",
FirstName.Text);
      sb.AppendFormat("გვარი: {0}<br>",
LastName.Text);
      sb.AppendFormat("სქესი: {0}<br>",
Sex.SelectedValue);
    sb.AppendFormat("ქალაქი: {0}<br>",
City.SelectedValue);
      sb.Append("ინტერესები: ");
    foreach(ListItem item in Interests.Items)
     {
       if(item.Selected)
        sb.AppendFormat("{0}, ", item.Value);
     }
```

```
    sb.Append("<br>მადლობთ რეგისტრაციისთვის");
    Message.Text = sb.ToString();
  }

   override protected void OnInit(EventArgs e)
   {
    InitializeComponent();
    base.OnInit(e);
   }
  private void InitializeComponent()
   {
    this.Register.Click += new
       System.EventHandler(this.Register_Click);
   }
 }
}
```

ნახ.13.17

ვებ-გვერდი და მისი შესრულების შედეგი ჩანს 3.18 ნახაზზე:

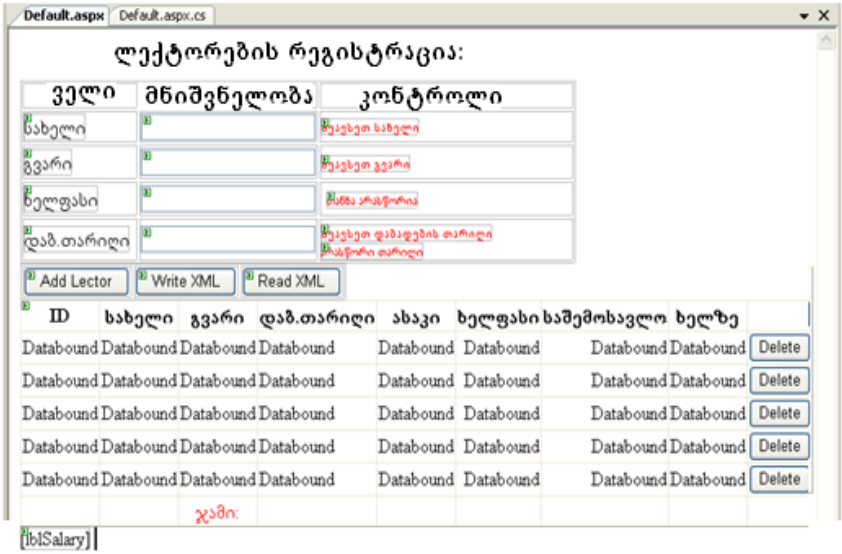


ნახ.13.18

13.7. DataSet / GridView -თან მუშაობა და XML ფაილი

ერთ-ერთი მნიშვნელოვანი საკითხია ინტერაქტიულ რეჟიმში მონაცემების შეტანისას მათი კონტროლის პროცედურების დამუშავება. შეტანილ მონაცემთა ეკრანზე ასახვის საშუალებების - GridView / DataSet გამოყენება. აგრეთვე მეტად მოსახერხებელია შედეგების XML - ფაილში შენახვა და XML- ფაილიდან მათი ამოღების პროცედურების შექმნა.

ამოცანა_13.1: Visual Studio .NET გარემოში ავაგოთ Web-პროექტი ლექტორთა სარეგისტრაციო მონაცემების შესატანად. განვახორციელოთ მონაცემთა ვიზუალური და ავტომატური კონტროლის საშუალებების გამოყენება. Add Lector-ღილაკით შეტანილი მონაცემები აისახოს GridView ცხრილში უნიკალური ID-ს მქონე სტრიქონის სახით, მომზადდეს სარეგისტრაციო ცხრილი ახალი ინფორმაციის შესატანად. ავტომატური კონტროლისთვის გამოყენებულ იქნეს: ToolBox->Validation;



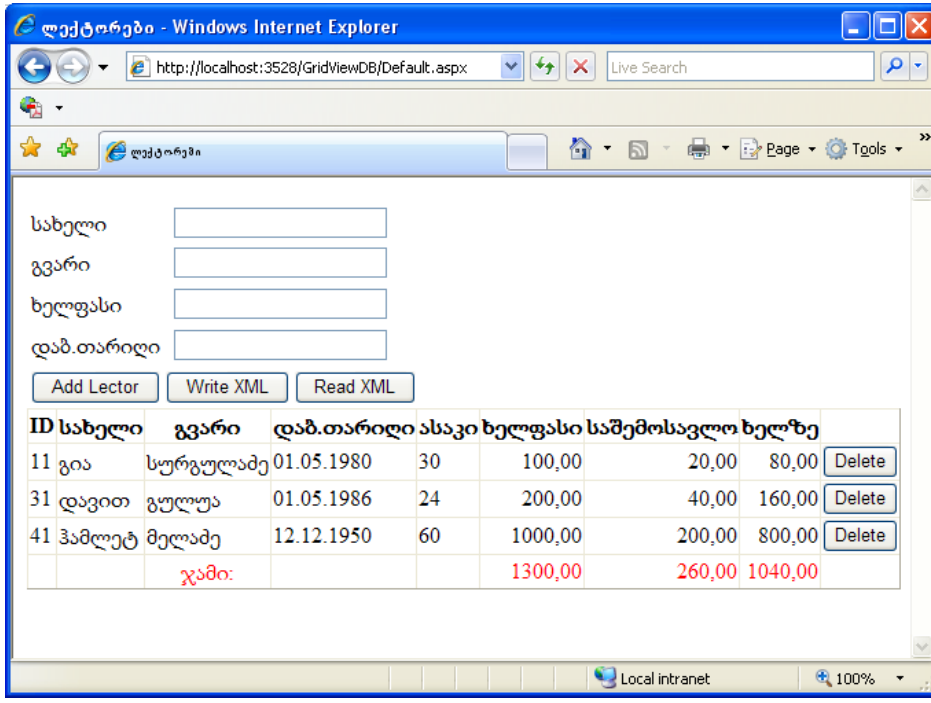
ნახ.13.19

3. GridView ცხრილში ავაგოთ სვეტები შესაბამისი დასახელებებით. გარდა სარეგისტრაციო მონაცემებისა, აქ დავამატოთ გაანგარიშებადი ველებიც: ასაკი, საშემოსავლო_გადასახადი, ხელზე_ასაღები_თანხა. ეს ველები განისაზღვრება .CS-კოდში;

4. GridView ცხრილში დავამატოთ ახალი სვეტი Delete-ფუნქციით, არასასურველი სტრიქონის ოპერატიულად წასაშლელად;

5. GridView ცხრილის Fother-სტრიქონში გამოვიტანოთ „ჯამი:“ ხელფასის, საშემოსავლოს და ხელზე_თანხის სვეტებისათვის.

13.20 ნახაზზე ნაჩვენებია ვებ-გვერდის მაკეტი ბრაუზერში.



ნახ.13.20

ქვემოთ აღიწერება ღილაკები, რომლებიც ფორმაზეა განლაგებული. მათი დანიშნულებაა მონაცემთა დამატება, კონტროლი, შენახვა XML ფაილში, შემდეგ კი ამ ფაილიდან ამოღება. განვიხილოთ მათი პროგრამული კოდები:

Add Lector:

```
protected void Button1_Click(object sender, EventArgs e)
  {
    DataSet dsLectors = Session["MyDataSet"] as DataSet;
    DataTable dtLectors = dsLectors.Tables["Lectors"];

    DataRow newlector = dtLectors.NewRow();
    //newlector["ID"] = "1";
    newlector["FirstName"] = txtFirstName.Text;
    newlector["LastName"] = txtLastName.Text;
    if (!String.IsNullOrEmpty(txtSalary.Text))
       newlector["Salary"] = Decimal.Parse(txtSalary.Text);

    newlector["BirthDate"] = DateTime.Parse(txtBirthDate.Text);

    dtLectors.Rows.Add(newlector);

    object sumSalary = dtLectors.Compute("SUM(Salary)", "");
    object sumTax = dtLectors.Compute("SUM(Tax)", "");
    object sumNettoSalary = dtLectors.Compute("SUM(NettoSalary)", "");
    //lblSalary.Text = sumSalary.ToString();
    GridView1.Columns[5].FooterText = String.Format("{0:F2}",
sumSalary);
    GridView1.Columns[6].FooterText =
String.Format("{0:F2}",sumTax);
    GridView1.Columns[7].FooterText = String.Format("{0:F2}",
sumNettoSalary);

    Session["MyDataSet"] = dsLectors;
```

```
        GridView1.DataSource = dsLectors;
        GridView1.DataBind();
    }
```

6. დავამატოთ ორი ღილაკი: Write_XML (სტრიქონების შესანახად XML-ფაილში) და Read_XML (სტრიქონების ამოსაღებად XML ფაილიდან).

```
// Write_XML ------------------
protected void Button2_Click(object sender, EventArgs e)
    {
        DataSet ds = Session["MyDataSet"] as DataSet;
        ds.WriteXml(Request.PhysicalApplicationPath + "\\lectors.xml");
        //Response.Redirect("~/lectors.xml");
    }

//  Read_XML ----------------
protected void Button3_Click(object sender, EventArgs e)
    {
        DataSet ds = Session["MyDataSet"] as DataSet;
        ds.ReadXml(Request.PhysicalApplicationPath + "\\lectors.xml");
        DataTable dtLectors = ds.Tables["Lectors"];
        object sumSalary = dtLectors.Compute("SUM(Salary)", "");
        object sumTax = dtLectors.Compute("SUM(Tax)", "");
        object sumNettoSalary = dtLectors.Compute("SUM(NettoSalary)", "");
        //lblSalary.Text = sumSalary.ToString();
        GridView1.Columns[5].FooterText = String.Format("{0:F2}",
                                                        sumSalary);
        GridView1.Columns[6].FooterText = String.Format("{0:F2}",
                                                         sumTax);
        GridView1.Columns[7].FooterText = String.Format("{0:F2}",
                                                       sumNettoSalary);
        GridView1.DataSource = ds;
        GridView1.DataBind();
    }
```

7. GridView ცხრილთან სამუშაოდ გამოიყენება DataSet - ობიექტი, რომელსაც C#-კოდში აქვს შემდეგი სახე:

```
private DataSet GetDataSet()
   {
      DataTable lectors = new DataTable("Lectors");
      //Add the DataColumn using all properties

      DataColumn id = new DataColumn("ID");
      id.DataType = typeof(int);
      id.Unique = true;
      id.AutoIncrement = true;
      id.AutoIncrementSeed = 1;
      id.AutoIncrementStep = 10;
      id.AllowDBNull = false;
      id.Caption = "ID";
      lectors.Columns.Add(id);

      //Add the DataColumn using defaults
      DataColumn firstName = new DataColumn("FirstName");
      firstName.DataType = typeof(string);
      firstName.MaxLength = 35;
      firstName.AllowDBNull = false;
      lectors.Columns.Add(firstName);

      DataColumn lastName = new DataColumn("LastName");
      lastName.DataType = typeof(string);
      lastName.MaxLength = 50;
      lastName.AllowDBNull = false;
      lectors.Columns.Add(lastName);

      DataColumn salary = new DataColumn("Salary", typeof(decimal));
      salary.DefaultValue = 0.00m;
      lectors.Columns.Add(salary);

      DataColumn birthDate = new DataColumn("BirthDate",
                                                typeof(DateTime));
      //birthDate.DefaultValue = DateTime.Now;
      birthDate.AllowDBNull = true;
```

```
        lectors.Columns.Add(birthDate);

        DataColumn age = new DataColumn("Age", typeof(DateTime));
        age.ColumnMapping = MappingType.Hidden;
        age.Expression = "BirthDate";
        lectors.Columns.Add(age);

        DataColumn tax = new DataColumn("Tax", typeof(decimal));
        tax.ColumnMapping = MappingType.Hidden;
        tax.DataType = typeof(decimal);
        tax.Expression = "salary*0.2";
        lectors.Columns.Add(tax);

        DataColumn netto = new DataColumn("NettoSalary",
typeof(decimal));
        netto.ColumnMapping = MappingType.Hidden;
        netto.DataType = typeof(decimal);
        netto.Expression = "salary - salary*0.2";
        lectors.Columns.Add(netto);

        ////Derived column using expression
        //DataColumn lastNameFirstName = new DataColumn("LastName and
FirstName");
        //lastNameFirstName.DataType = typeof(string);
        //lastNameFirstName.MaxLength = 70;
        //lastNameFirstName.Expression = "lastName + ', ' + firstName";
        //employee.Columns.Add(lastNameFirstName);
        DataSet ds = new DataSet();
        ds.Tables.Add(lectors);
        return ds;
    }
```

8. XML-ფაილს მასში სტრიქონების (ობიექტების) ჩაწერის შემდეგ ექნება ასეთი სახე:

```
<?xml version="1.0" standalone="yes"?>
<NewDataSet>
  <Lectors>
    <ID>11</ID>
```

```
  <FirstName>გია</FirstName>
  <LastName>სურგულაძე</LastName>
  <Salary>100</Salary>
  <BirthDate>1980-05-01T00:00:00+04:00</BirthDate>
 </Lectors>
 <Lectors>
  <ID>31</ID>
  <FirstName>დავით</FirstName>
  <LastName>გულუა</LastName>
  <Salary>200</Salary>
  <BirthDate>1986-05-01T00:00:00+04:00</BirthDate>
 </Lectors>
 <Lectors>
  <ID>41</ID>
  <FirstName>გიორგი</FirstName>
  <LastName>სურგულაძე</LastName>
  <Salary>1900</Salary>
  <BirthDate>1980-12-30T00:00:00+04:00</BirthDate>
 </Lectors>
</NewDataSet>
```

13.8. კითხვები და სავარჯიშოები:

13.1. რას წარმოადგენს ASP.NET და რისთის გამოპიყენება იგი ?

13.2. როგორია ASP.NET სტრუქტურა და შედგენილობა ?

13.3. როგორ ხდება Web-აპლიკაციისთვის ახალი პროექტის გახსნა და გვერდის შექმნა ?

13.4. როგორ ნაწილდება ფუნქციები ASP.NET-ში HTML და C# ენებს შორის ?

13.5. დაახასიათეთ ASP.NET აპლიკაციის შექმნის ეტაპები.

13.6. ASP.NET-ის გამოყენებით ააგეთ ვებ-გვერდი: „კათედრა“.

13.7. ააგეთ ინტერაქტიური ვებ-გვერდი „სტუდენტთა რეგისტრაცია“, რომელშიც შესაძლებლი იქნება კლიენტის მხრიდან სერვერზე მონაცემების გადაცემა.

IV ნაწილი. C# ენის გრაფიკული საშუალებანი

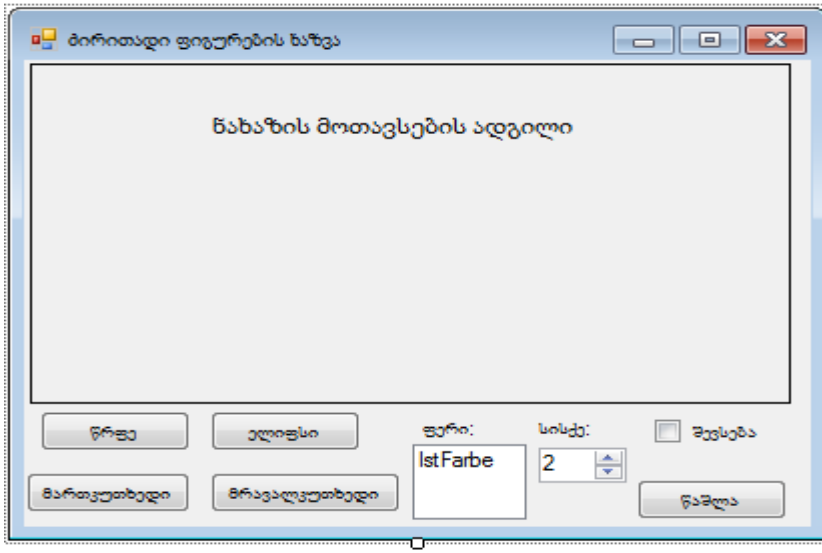
14. გრაფიკული ინტერფეისის ინსტრუმენტი

14.1. ძირითადი გრაფიკული ფიგურების აგება - GDI+ (Graphics Device Interface)

ძირითად გრაფიკულ ფიგურებს მიეკუთვნება წრფე, მართკუთხედი, ელიფსი და მრავალკუთხედი, რომელთა საფუძველზე აიგება სხვა სახის ფიგურები.

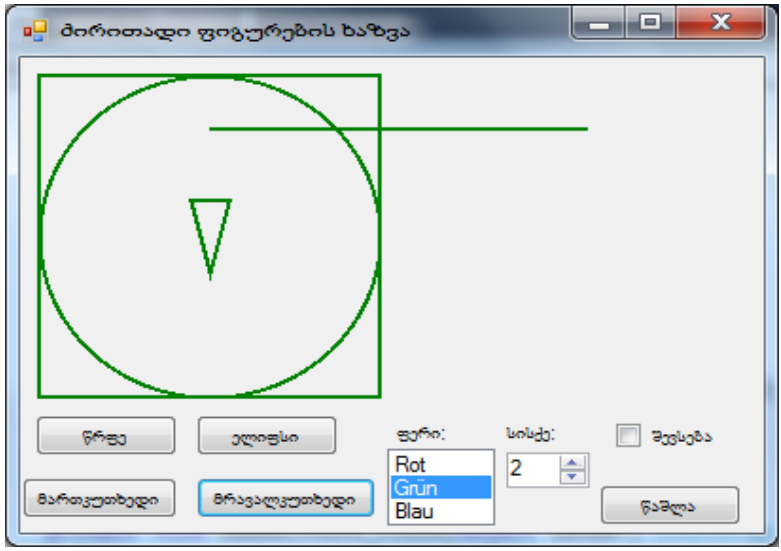
ფიგურები თავსდება ფორმაზე, რისთვისაც მათ სჭირდება მდებარეობის პარამეტრები და ზომები. აგრეთვე მათ მიერ შემოსაზღვრული ფართობის შევსება რომელიმე ფერით და ა.შ.

14.1 ნახაზზე მოცემულია გრაფიკების ასაგები ფორმა და ძირითად გრაფიკთა აგებისათვის შესაბამისი ღილაკები.

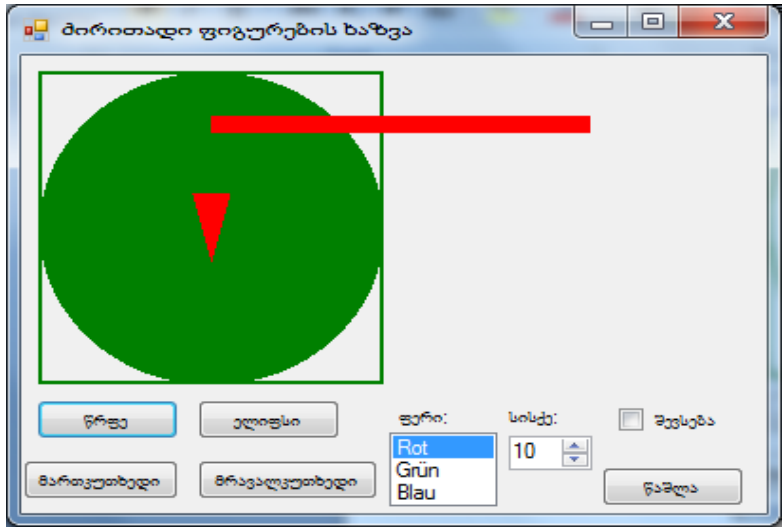


ნახ.14.1

მომდევნო ნახაზებზე ნაჩვენებია თითოეული გრაფიკის ბუტონის ამუშავებით მიღებული სურათი (ნახ. 14.2-14.4).



ნახ.14.2



ნახ.14.3

ცალკეული ბუტონის შესაბამისი კოდები აღწერილია 15_1 ლისტინგში.

```
// ლისტინგი_14.1 ---CreateGraphics() ---
using System;
using System.Drawing;
using System.Windows.Forms;

namespace ZeichnenGrundformen
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        Graphics z;
        Pen stift = new Pen(Color.Red, 2);
        SolidBrush pinsel = new SolidBrush(Color.Red);

   private void Form1_Load(object sender, EventArgs e)
        {
            z = CreateGraphics();

            lstFarbe.Items.Add("წითელი");
            lstFarbe.Items.Add("მწვანე");
            lstFarbe.Items.Add("ლურჯი");
            lstFarbe.SelectedIndex = 0;
        }

   private void cmdLinie_Click(object sender, EventArgs e)
        {
            z.DrawLine(stift, 100, 40, 300, 40);
        }

   private void cmdRechteck_Click(object sender, EventArgs e)
        {
            if (chkFüllen.Checked)
            {
                z.FillRectangle(pinsel, 10, 10, 380, 180);
                chkFüllen.Checked = false;
            }
```

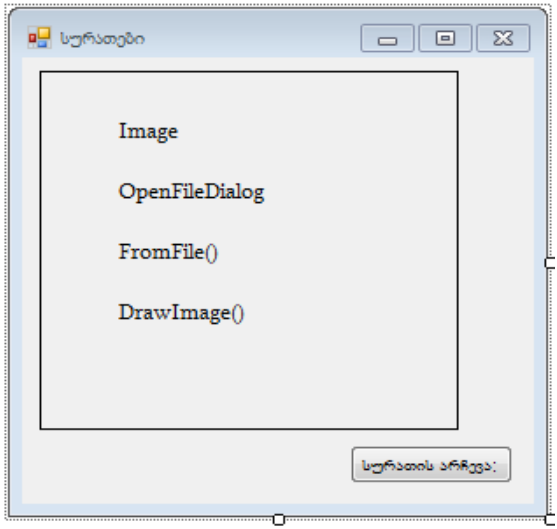
```
            else
                z.DrawRectangle(stift, 10, 10, 380, 180);
        }

    private void cmdPolygon_Click(object sender, EventArgs e)
        {
            Point[] point_feld =
                    {new Point(90, 80),
                     new Point(110, 80),
                     new Point(100, 120)};

            if (chkFüllen.Checked)
            {
                z.FillPolygon(pinsel, point_feld);
                chkFüllen.Checked = false;
            }
            else
                z.DrawPolygon(stift, point_feld);
        }

    private void cmdEllipse_Click(object sender, EventArgs e)
        {
            if (chkFüllen.Checked)
            {
                z.FillEllipse(pinsel, 10, 10, 180, 180);
                chkFüllen.Checked = false;
            }
            else
                z.DrawEllipse(stift, 10, 10, 180, 180);
        }

    private void numPenWidth_ValueChanged(object sender,
                                                EventArgs e)
        {
            stift.Width = (float) numPenWidth.Value;
        }

    private void lstFarbe_SelectedIndexChanged(object sender,
                                                EventArgs e)
        {
          Color[] color_feld = { Color.Red, Color.Green,
                                                Color.Blue };
```

```
            stift.Color = color_feld[lstFarbe.SelectedIndex];
            pinsel.Color = color_feld[lstFarbe.SelectedIndex];
        }

    private void cmdClear_Click(object sender, EventArgs e)
        {
            z.Clear(BackColor);
        }
    }
}
```

14.2. სურათების წარმოდგენა

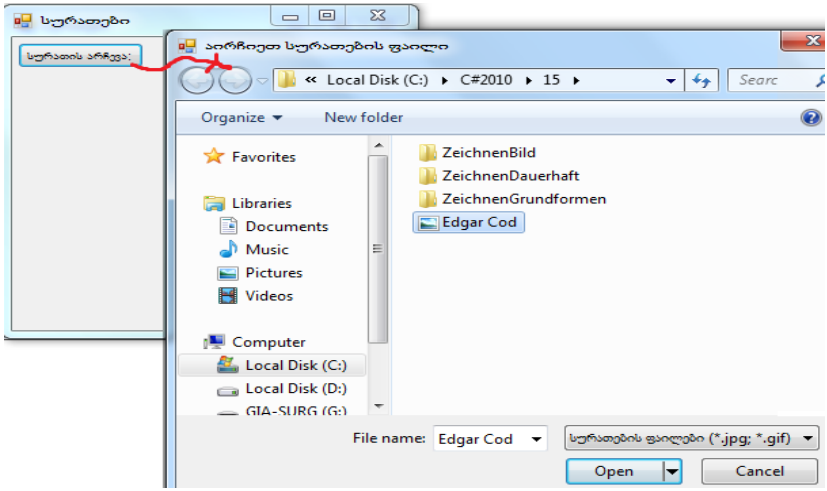
სურათის ასაგებად ფორმაზე გამოიყენება Image კლასი. მისი სტატიკური მეთოდით FromFile() შესაძლებელი ხდება სურათის ჩამოტვირთვა ფაილიდან და მოთავსება ფორმაზე.

14.4 ნახაზზე ნაჩვენებია ფორმა, რომელზეც ღილაკით „სურათის არჩევა“ ხდება Image კლასის გამოყენება, რომელსაც აქვს მეთოდი FromFile() სურათის ჩამოსატვირთად ფაილიდან ფორმაზე.



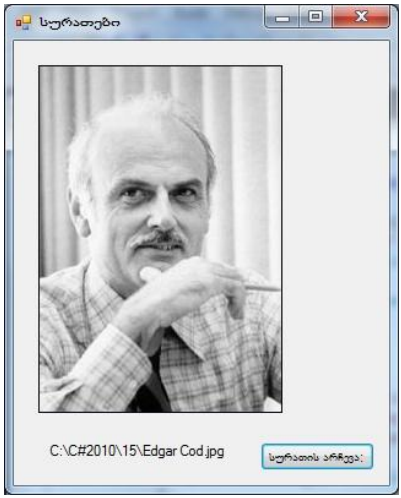
ნახ.14.4

ღილაკის ამოქმედებისთანავე იხსნება დიალოგის ფანჯარა OpenFileDialog (ნახ.14.5).



ნახ.14.5

შედეგი სურათით ნაჩვენებია 14.6 ნახაზზე. პროგრამის კოდი მოცემულია 14.2 ლისტინგში.



ნახ.14.6. ედგარ კოდი

```
// ლისტინგი_14.2 --- სურათები ------------
using System;
using System.Drawing;
using System.Windows.Forms;

namespace ZeichnenBild
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

    private void cmdAuswahl_Click(object sender, EventArgs e)
        {
            Graphics z = CreateGraphics();
            Font df = new Font("Verdana", 11);
            SolidBrush pinsel = new SolidBrush(Color.Black);

            OpenFileDialog ofd = new OpenFileDialog();
            Image bild;

            z.Clear(BackColor);

            ofd.InitialDirectory = "C:\\C#2010\\15";
            ofd.Title = "აირჩიეთ სურათების ფაილი";
            ofd.Filter = "სურათების ფაილები (*.jpg;
                                    *.gif)|*.jpg; *.gif";

            if (ofd.ShowDialog() == DialogResult.OK)
            {
                bild = Image.FromFile(ofd.FileName);
                z.DrawImage(bild, 20, 40);
                z.DrawString("სიგანე: " + bild.Width +",
               სიმაღლე: " + bild.Height, df, pinsel, 20, 20);
            }
            else
               MessageBox.Show("არაა სურათის ფაილი !");
        }
    }
}
```

14.9. კითხვები და სავარჯიშოები:

14.1. რას წარმოადგენს GDI+ ბიბლიოთეკა ?

14.2. როგორ ხდება გრაფიკული რეჟიმის ინიციალიზაცია ?

14.3. როგორ იხაზება გრაფიკულ რეჟიმში ძირითადი ფიგურები: ხაზი, მართკუთხედი, მრავალკუთხედი, ელიფსი ?

14.4. როგორ ხდება ფიგურების დასახაზი ფერის შერჩევა და ფიგურების გაფერადება ?

14.5. გამოხაზეთ ფიგურა - „ცილინდრიანი კაცი“ და გააფერადეთ.

14.6. როგორ ხდება სურათის ჩასმა ფორმაზე მითითებულ ადგილას გარე ფაილიდან ?

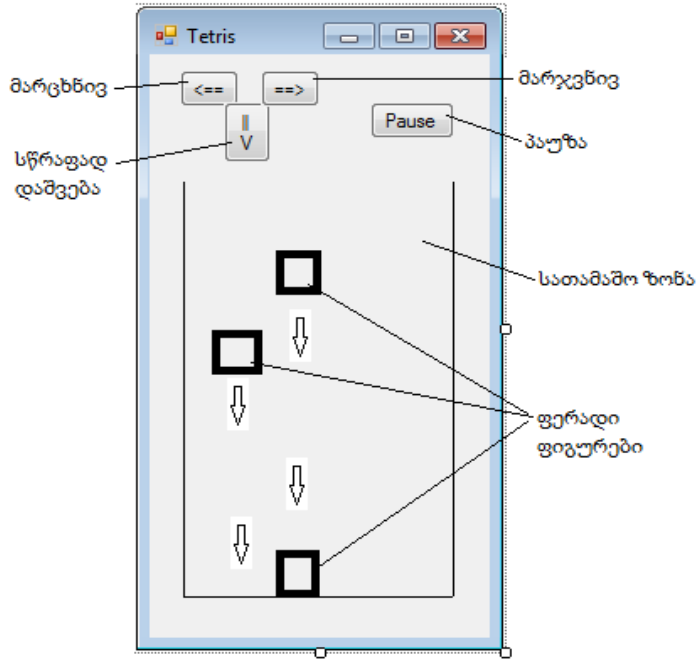
14.7. ააგეთ ფორმა „გალერეა“, რომელზეც მოთავსდება C, C++, Java, C# და UML ენების შემქნელთა ფოტოები (მოიძიეთ ინტერნეტში) და ექნება მცირე ბიოგრაფიული კომენტარები.

15. პროექტი: სათამაშო ანიმაციური პროგრამა “Tetris” (ვიზუალური დაპროგრამების პრინციპები)

თამაშების დაპროგრამება ერთ-ერთი მნიშვნელოვანი მიმართულებაა პროგრამული ბიზნესის ბაზარზე. იგი აქტუალური და სწრაფად განვითარებადი დარგია.

ჩვენ განვიხილავთ „Tetris“ - თამაშს, რომელიც საკმაოდ „ასაკოვანია“, მაგრამ დღესაც დიდი პოპულარობით სარგებლობს. შექმნილია არა ერთი ნაირსახეობა ამ თამაშისა და კვლავ ვითარდება. ჩვენი მიზანია C#_2010 ენის ინსტრუმენტით და მისი მდიდარი კლასთა ბიბლიოთეკით, ვიზუალური ელემენტების გამოყენებით ავაგოთ ამ თამაშის მარტივი მოდელი და პროგრამოს კოდი [18].

თამაშის ინტერფეისი 15.1 ნახაზეა მოცემული.



ნახ.15.1

თამაშის მიზანი: არ გაივსოს ჭურჭელი ფერადი ფიგურებით !

ტაქტიკა: ერთფეროვანი ფიგურები დალაგდეს „ერთ ვერტიკალში“ ან „ერთ ჰორიზონტალში“. როცა რამდენიმე ერთფეროვანი ფიგურა მოგროვდება (მაგალითად, 3), მაშინ სამივე ქრება, ანუ ჭურჭელი თავისუფლდება ახალი ფიგურებისთვის.

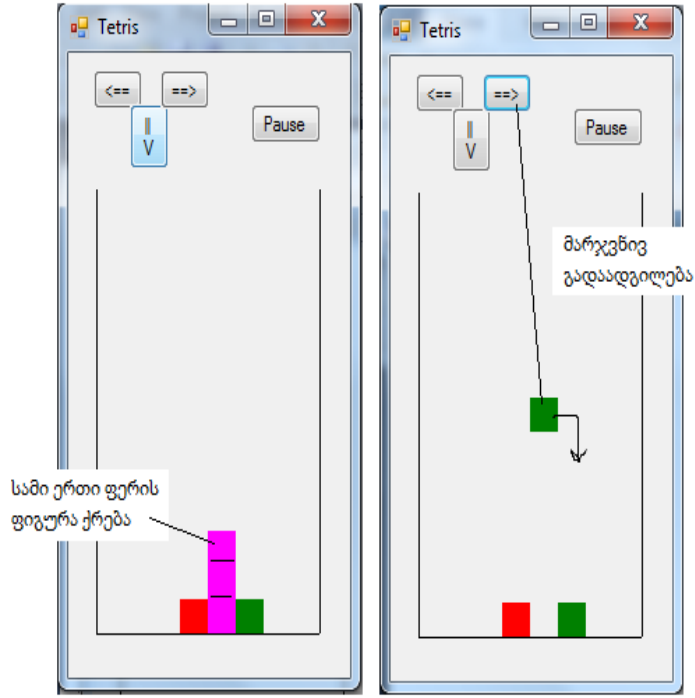
ფერადი ფიგურები ჩნდება შემთხვევით რიცხვთა გენერატორით და ეშვება ქვემოთ. „მარცხენა“ და „მარჯვენა“ ღილაკებით ჩვენ უნდა გადავაადგილოთ სასურველ ვერტიკალში.

თუ ორი ერთნაირი ფერის ფიგურაზე დაჯდა სხვა ფერის ფიგურა, მაშინ ის იბლოკება მანამ, სანამ ზედა ახალი ფერის ფიგურები არ გაქრება.

თამაშის ტემპი (ფიგურების დაშვების სისწრაფე ჭურჭლის ფსკერზე) პირველ ეტაპზე ნორმალურია. როცა მოთამაშე კარგად ართმევს თავს ამ ეტაპს, ანუ ფერადი ფიგურებით არ ივსება ჭურჭელი, მაშინ პროგრამა უმატებს ახალი ფერადი ფიგურების გაჩენის და ფსკერზე დაშვების სისწრაფეს (რთულდება პროცესის მართვა). თუ ამასაც გაართვა თავი მოთამაშემ, მაშინ კიდევ ახალი ეტაპი იწყება და ა.შ.

პროგრამის ასაგებად უნდა გამოვიყენოთ:

- ორგანზომილებიანი ველი;
- დროითი Timer;
- შემთხვევით რიცხვთა გენერატორი;
- ფერადი ფიგურების (კვადრატები) გაჩენა და გაქრობა მუშაობის პროცესში;
- ღილაკების (მოვლენების) მართვის ორგანიზება „ახალგაჩენილ“ ფერადი ფიგურებისთვის.



ნახ.15.2

პროგრამაში „ფერადი ფიგურა“ რეალიზებულია Panel-ტიპის ელემენტით, რომელსაც შეუძლია რვა ფერიდან ერთ-ერთის მიღება.

```
// ლისტინგი_15.1 ---  “Tetris” C# პროგრამის კოდი ------
using System;
using System.Collections;
using System.Drawing;
using System.Windows.Forms;
namespace Tetris
{
    public partial class Form1 : Form
    {
       public Form1() { InitializeComponent();  }
```

```
// აქტუალური პანელის ინდექსი Index
int PX;

// სათამაშო ბილიკების რაოდენობა
int[,] F = new int[15, 10];

// სტრიქონები და სვეტები აქტუალური პანელის
int PZ, PS;

// სირთულის დონე
int Stufe;

// სათამაშო პანელების საწყისი ცარიელი სია
ArrayList PL = new ArrayList();

//  პანელების ფერთა სიმრავლე
Color[] FarbenFeld = {Color.Red,
    Color.Yellow, Color.Green, Color.Blue,
    Color.Cyan, Color.Magenta, Color.Black,
    Color.White};

// ბილიკის სტატუსის კონსტანტები
const int Leer = -1;
const int Rand = -2;

// შემთხვევით რიცხვთა გენერატორი: გაჩენა და გააქტიურება
Random r = new Random();

private void Form1_Load(object sender, EventArgs e)
{
    int Z, S;

    // ბილიკი დაკავებულია
    for (Z=1; Z<14; Z++)
    {
        F[Z, 0] = Rand;
        for (S=1; S<9; S++)
            F[Z, S] = Leer;
        F[Z, 9] = Rand;
    }
```

```
            for (S=0; S<10; S++)
                F[14, S] = Rand;

            // ინიციალიზირება
            Stufe = 1;
            NächstesPanel();
        }

        private void NächstesPanel() // შემდეგი პანელი
        {
            int Farbe;
            Panel p = new Panel();

            // ახალი პანელის შეტანა მასივის სიაში
            PL.Add(p);

            // ”დაკლიკვის”* მოვლენის დამუშავება
            p.Click += new EventHandler(PanelClickReaktion);

            // ახალი პანელის ადგილზე ჩასმა
            p.Location = new Point(100, 80);
            p.Size = new Size(20, 20);

            // ახალი პანელისთვის ფერის შერჩევა
            Farbe = r.Next(0,8);
            p.BackColor = FarbenFeld[Farbe];

            // ახალი პანელის ფორმაზე შეტანა
            Controls.Add(p);

            // მომავალი წვდომის ინდექსის განსაზღვრა
            PX = PL.Count - 1;

            // პანელის საინფორმაციო ინდექსის დაფიქსირება
            p.Tag = PX;

            // აქტუალური სტრიქონი და სვეტი
            PZ = 1;
            PS = 5;
        }
    private void PanelClickReaktion(object sender, EventArgs e)
```

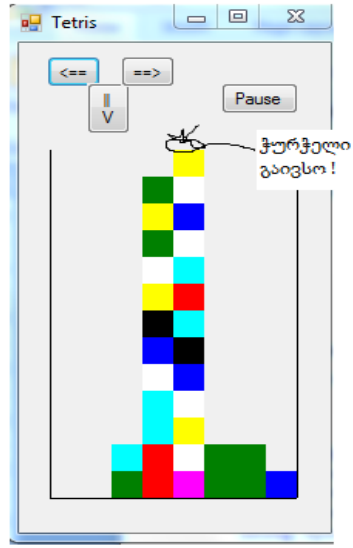
```
{
    // რომელი პანელი დაიკლიკა
    Panel p = (Panel) sender;

    // დაკლიკული პანელის თვისებების შეცვლა
    lblPNr.Text = "P " + p.Tag;
    p.BorderStyle = BorderStyle.FixedSingle;
}

private void timT_Tick(object sender, EventArgs e)
{
    // თუ დასრულდა ან აღარ მუშაობს
    if (F[PZ + 1, PS] != Leer)
    {
        // მიღწეულია ზედა სტრიქონი
        if (PZ == 1)
        {
            timT.Enabled = false;
            MessageBox.Show("აბა რა !");
            return;
        }

        F[PZ, PS] = PX;        // დაკავება
        AllePrüfen();
        NächstesPanel();
    }
    else
    {
        // თუ გრძელდება თამაში კიდევ
        Panel p = (Panel) PL[PX];
        p.Top = p.Top + 20;
        PZ = PZ + 1;
    }
}

private void AllePrüfen()
{
    int Z, S;
    bool Neben, Über;
    Neben = false;
    Über = false;
```

```
        // სამი ერთფეროვანი პანელი გვერდი-გვერდაა ?
        for(Z=13; Z>0; Z--)
        {
            for(S=1; S<7; S++)
            {
                Neben = NebenPrüfen(Z, S);
                if (Neben) break;
            }
            if (Neben) break;
        }

        // სამი ერთფეროვანი პანელი ერთმანეთზეა ?
        for(Z=13; Z>2; Z--)
        {
            for(S=1; S<9; S++)
            {
                Über = ÜberPrüfen(Z, S);
                if (Über) break;
            }
            if (Über) break;
        }

        if (Neben || Über)
        {
            // უფრო სწრაფად
            Stufe = Stufe + 1;
            timT.Interval = 5000 / (Stufe + 9);

            // ალბათ შესაძლებელია კიდევ ერთი რიგის ამოგდება
            AllePrüfen();
        }
    }

    // თუ სამი ბილიკი ერთმანეთის გვერდით დაკავებულია
    private bool NebenPrüfen(int Z, int S)
    {
        int ZX, SX;
        bool ergebnis = false;

        if (F[Z, S] != Leer &&
                F[Z, S + 1] != Leer &&
                F[Z, S + 2] != Leer)
```

```
    {
        Panel p = (Panel) PL[F[Z, S]];
        Panel p1 = (Panel) PL[F[Z, S + 1]];
        Panel p2 = (Panel) PL[F[Z, S + 2]];

        /* თუ სამი ერთნაირი ფერია
        if (p.BackColor == p1.BackColor &&
                p.BackColor == p2.BackColor)
        {

            for(SX=S; SX<S+3; SX++)
            {
                /* PL წაიშალოს ფორმიდან
                Control c = (Control) PL[F[Z, SX]];
                Controls.Remove(c);
                /* ბილიკი თავისუფლდება */
                F[Z, SX] = Leer;

          // ზედა პანელი ათავისუფლებს ქვედა დაბლოკილს
                ZX = Z - 1;
                while (F[ZX, SX] != Leer)
                {
                    Panel px =
                        (Panel) PL[F[ZX, SX]];
                    px.Top = px.Top + 20;

                    // ბილიკი კვლავ კავდება
                    F[ZX + 1, SX] = F[ZX, SX];
                    F[ZX, SX] = Leer;
                    ZX = ZX - 1;
                }

            }
            ergebnis = true;
        }
    }
    return ergebnis;
}

// თუ სამი ერთფეროვანია ერთმანეთზე
private bool ÜberPrüfen(int Z, int S)
{
```

```
            int ZX;
            bool ergebnis = false;

            if (F[Z, S] != Leer && F[Z - 1, S] != Leer &&
                    F[Z - 2, S] != Leer)
            {
                Panel p = (Panel) PL[F[Z, S]];
                Panel p1 = (Panel) PL[F[Z - 1, S]];
                Panel p2 = (Panel) PL[F[Z - 2, S]];

                // თუ სამი ფერი ერთნაირია
                if (p.BackColor == p1.BackColor &&
                        p.BackColor == p2.BackColor)
                {

                    // სამი პანელი ქრება
                    for (ZX=Z; ZX>Z-3; ZX--)
                    {
                        // PL იშლება ფორმიდან
                        Control c = (Control) PL[F[ZX, S]];
                        Controls.Remove(c);
                        // ბილიკი თავისუფლდება
                        F[ZX, S] = Leer;
                    }
                    ergebnis = true;
                }
            }
            return ergebnis;
        }

    private void cmdLinks_Click(object sender, EventArgs e)
        {
            if (F[PZ, PS - 1] == Leer)
            {
                Panel p = (Panel) PL[PX];
                p.Left = p.Left - 20;
                PS = PS - 1;
            }
        }

    private void cmdRechts_Click(object sender, EventArgs e)
        {
```

```
            if (F[PZ, PS + 1] == Leer)
            {
                Panel p = (Panel) PL[PX];
                p.Left = p.Left + 20;
                PS = PS + 1;
            }
        }

    private void cmdUnten_Click(object sender, EventArgs e)
        {
            while (F[PZ + 1, PS] == Leer)
            {
                Panel p = (Panel) PL[PX];
                p.Top = p.Top + 20;
                PZ = PZ + 1;
            }
            F[PZ, PS] = PX;        // დაკავება
            AllePrüfen();
            NächstesPanel();
        }

   private void cmdPause_Click(object sender, EventArgs e)
        {
            timT.Enabled =
!timT.Enabled;
        }
    }
}
```



ნახ.15.3

ჭურჭლის გავსებით სრულდება თამაში.

➢ კითხვები და სავარჯიშოები

15.1. რას წარმოადგენს ანიმაციური პროგრამა და რა ძირითად ელემენტებს უნდა ფლობდეს იგი ?

15.2. როგორ ხდება ფიგურების გადაადგილების პროგრამული რეალიზაცია ?

15.3. როგორ ხდება დროითი ელემენტის გამოყენება დინამიკურ პროცესში ?

ლიტერატურა:

1. სურგულაძე გ., დოლიძე თ., ყვავაძე ლ. კომპონენტურ-ვიზუალური დაპროგრამება: ინტერფეისების აგება C# da C++ ენებზე მონაცემთა განაწილებული ბაზებისათვის. სტუ, თბილისი, 2006.

2. ბოტჰე კ., სურგულაძე გ. და სხვ. თანამედროვე პროგრამული პლატფორმები და ენები: (C,C++, Java, XML). სტუ, თბ., 2003.

3. სურგულაძე გ., ბულია ი., თურქია ე. Web-აპლიკაციების დამუსავება მონაცემთა ბაზების საფუძველზე (ADO.NET, ASP.NET, C#). სახელმძღვ., სტუ, თბ., 2009.

4. გოგიჩაიშვილი გ., სურგულაძე გ., შონია ო. დაპროგრამების მეთოდები (C,C++). სახელმძღვ., სტუ, თბ., 1997.

5. ფრანგიშვილი ა., სურგულაძე გ., ვაჭარაძე ი. ბიზნეს-პროგრამების ექსპერტულ შეფასებებში მხარდამჭერი გადაწყვეტილებათა მიღების მეთოდები და მოდელები. მონოგრ., სტუ, თბ., 2009.

6. სამხარაძე რ. Visual C#.NET. სახელმძღ., სტუ, .თბ., 2009

7. სურგულაძე გ. დაპროგრამების ვიზუალური მეთოდები და ინსტრუმენტები. სახელმძღვ. სტუ, თბ., 2005.

8. სურგულაძე გ. ობიექტ-ორიენტირებული დაპროგრამების მეთოდი. სტუ. თბ., 2005.

9. სურგულაძე გ., ბულია ი., თურქია ე. Web-აპლიკაციების აგება ASP.NET & C# პაკეტებით .NET პლატფორმაზე. დამხმ.სახ., ლაბ.პრაქტიკუმი. სტუ, თბ., 2009.

10. ბახტაძე თ. ობიექტზე ორიენტირებული პროგრამირების ენა C#. სახელმძღვ., სტუ, თბ., 2006

11. სურგულაძე გ. ობიექტ-ორიენტირებული დაპროგრამების მეთოდი. სტუ. თბ., 2005.

12. სურგულაძე გ. დაპროგრამების მეთოდები. მეთოდური სახელმძღვანელო საკურსო პროექტის შესასრულებლად. სტუ, თბ., 2007

13. სურგულაძე გ., შონია ო., ყვავაძე ლ. მონაცემთა განაწილებული ბაზების მართვის სისტემები (MsSQL Server, Access, InterBase, JDBC, Oracle). სტუ, თბილისი, 2004

14. თურქია ე. თურქია ე. ბიზნეს–პროექტების მართვის ტექნოლოგიური პროცესების ავტომატიზაცია. მონოგრ., სტუ. თბ., 2010,

15. დეიტელი ვ., დეიტელი კ. დაპროგრამება C და C ++ ენებზე. რუს.ენაზე, მოსკოვი, 2005.

16. Developing Windows-Based Applications with Ms VisualBasic.NET and Ms Visual C#.NET. MCAD/MCSD Training Kit. Exam 70-306 and 70-316. Перю с англ., М., “Русская Редакция”. 2003

17. Volz B. Einstieg in Visual C# 2008. Galileo Computing. Bonn. 2008

18. Theis T. Einstieg in Visual C# 2010. Galileo Computing. Bonn. 2010

19. Sceppa D. Microsoft ADO.NET. Пер. с англ., М., «Русская Редакция», 2003.

20. Microsoft Corporation. Разработка Windows-приложений на Ms VB и Ms VC#.NET. Пер. с англ., М., «Русская Редакция», 2003.

21.Майо Дж. C#: Искусство программирования. Энциклопедия программиста. Пер.с англ., "DiaSoft", СПб., 2002.

22. Robinson S., Cornes O., Glynn J., Harvey B., McQueen C., Moemeka J., Nagel C., Skinner M., Watson K. Professional C#. Bimingham, WroxPress, 2001.

იბეჭდება ავტორთა მიერ
წარმოდგენილი სახით

საგამომცემლო სახლი ,,ტექნიკური უნივერსიტეტი‘‘
თბილისი, მ. კოსტავას 77