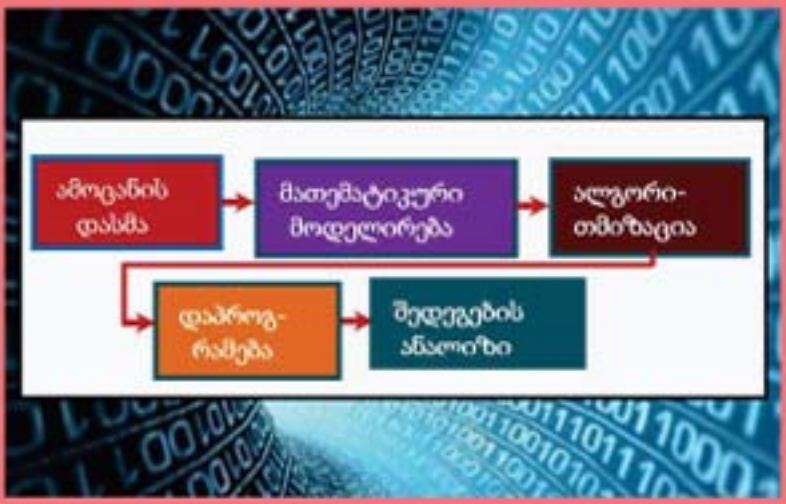
ალექსანდრე დუნდუა

კომპიუტერი, ინტელექტი და ლინგვისტიკა

(აგებულია ინტერდისციპლინარულობის პრინციპების გამოყენებით)



საგამომცემლო სახლი
„ტექნიკური უნივერსიტეტი“

საქართველოს ტექნიკური უნივერსიტეტი

ალექსანდრე დუნდუა

კომპიუტერი, ინტელექტი და ლინგვისტიკა

რეკომენდებულია საქართველოს
ტექნიკური უნივერსიტეტის
სარედაქციო-საგამომცემლო საბჭოს
მიერ. 25.12.2024, ოქმი №2

თბილისი

2025

უაკ 004.8

დამხმარე სახელმძღვანელოში განხილულია კომპიუტერული სისტემებისა და ხელოვნური ინტელექტის გენეზისის საკითხები. იგი იწყება ისეთი ტრივიალური პრობლემის გადაწყვეტის გადმოცემით, როგორიცაა სპეციალიზებული არითმეტიკული მანქანის სტრუქტურის სინთეზი და ბოლოვდება ხელოვნური ინტელექტის ფორმირების ჩვენებით. მოკლედაა მოყვანილი არითმეტიკული და ლოგიკური მანქანების განვითარების ისტორიები. გადმოცემულია როგორც სამყაროს, ისე ბუნებრივი ინტელექტის წარმოშობისა და თანამედროვე ადამიანის ჩამოყალიბების საკვანძო საკითხები; განხილულია დაპროგრამების პარადიგმები, მოხდენილია კომპიუტერული ენების კლასიფიცირება და დაინტერესებული მკითხველისათვის გასაგებ ენაზეა აღწერილი დაპროგრამების სასწავლო (Pascal, Qbasik) ენათა სტრუქტურები.

განკუთვნილია სატრანსპორტო სისტემებისა და მექანიკის ინჟინერიის ფაკულტეტის („ტრანსპორტის“ საგანმანათლებლო პროგრამა) ბაკალავრებისათვის. შეიძლება გამოყენებული იქნეს აღნიშნული სფეროს მაგისტრანტების, დოქტორანტებისა და ახალგაზრდა სპეციალისტების მიერ. იგი დააინტერესებს კომპიუტერული სისტემების, ლოგიკური მოწყობილობების, ხელოვნური ინტელექტისა და დაპროგრამების საკითხებით დაინტერესებულ მკითხველთა ფართო ფენას.

რეცენზენტები: საქართველოს ტექნიკური უნივერსიტეტის
ენერგეტიკის ფაკულტეტის პროფესორი
სერგო დადუნაშვილი,

საქართველოს ტექნიკური უნივერსიტეტის
სატრანსპორტო სისტემებისა და მექანიკის
ინჟინერიის ფაკულტეტის ასოცირებული
პროფესორი მურთაზ პაპასკირი

ISBN 978-9941-512-94-0

http://www.gtu.ge

ვუძღვნი მეუღლეს, დოდო ნადარეიშვილს

ს ა რ ჩ ე ვ ი

წ ი ნ ა თ ქ მ ა

რთული საკითხის გადაწყვეტისათვის, რომლის რაობის სრულად გაცნობიერება ერთი ვიწრო სპეციალიზაციის ჩარჩოებში ვერ ხერხდება, გამოიყენება სხვადახვა დისციპლინის ცოდნის, მეთოდებისა და ცნებების ინტეგრაციაზე დაფუძნებული მიდგომა, რომელსაც **ინტერდისციპლინარულობა** (*ლათ. Inter – „გაერთიანება“, disciplina – „დისციპლინა“*) ეწოდება. ერთ-ერთი ასეთი რთულ საკითხთაგანია ბუნებრივი ინტელექტის მსგავსი უნარის მქონე ხელოვნურ მოწყობილობად (კომპიუტერად) ჩვეულებრივი **არითმეტიკული მანქანის** გარდაქმნის საკითხი. ამიტომ ჩვენი ნაშრომი **ინტერდისციპლინარეული** მიდგომის გამოყენებითაა აგებული. იგი სამი ნაწილისაგან შედგება, რომელთაგანაც ახალი მიდგომებით **პირველი ნაწილი** გამოირჩევა. **არითმეტიკული მანქანები** ნულოვანი თაობის კომპიუტერებად ითვლება. ამიტომ კომპიუტერებზე დაწერილ სახელმძღვანელოთა ავტორები, როგორც წესი, მხოლოდ ამ მანქანების განხილვით კმაყოფილდებიან. ისინი საუბარს იწყებენ უძველესი სათვლელი საშუალებების (აბაკების) ჩამოთვლით, შემდეგ გვაცნობენ **1623** წელს **ვიჰელმ შიკარდისა** და **1643** წელს **ბლეზ პასკალის** მიერ აგებულ არითმეტიკულ მანქანებს, საუბარს განაგრძობენ სხვადასხვა სახის არითმომეტრების კონსტრუქციათა აღწერით და, ისე გადადიან კომპიუტერთან დაკავშირებულ ძირითადი საკითხების განხილვაზე, რომ განუხილველად ტოვებენ **6**-საუკუნოვანი ისტორიის მქონე **ლოგიკურ მანქანებს**. ამის გამო ბუნდოვანი რჩება მექანიზმი, რომლის მეშვეობითაც არითმეტიკული მანქანა გარდაიქმნა კომპიუტერად, ამ უკანასკნელში კი ჩაისახა ხელოვნური ინტელექტი. აღნიშნული ნაკლის გამოსასწორებლად ჩვენს წიგნში ცალკე თავი დავუთმეთ ლოგიკური მანქანების განხილვას და მასში შევიტანეთ ლოგიკური ალგებრის ელემენტები. ეს მეტად მნიშვნელოვანია და საკითხისადმი **ახლებურ მიდგომად** უნდა ჩაითვალოს.

მეორე სიახლეც „კარგად დავიწყებული ძველის“ გახსენებასთანაა დაკავშირებული. ბევრი არ იხსენებს იმ ფაქტს, რომ ტერმინი „**კომპიუტერი**“ პირველად გერმანელმა მეცნიერმა **გოტფრიდ ლაიბნიცმა** გააჟღერა თავის ცნობილ გამოთქმაში, რომელიც ინგლისურად ასე გამოითქმება: „**Gentlmen, let us COMPUTE!**“. ზოგადად, კომპიუტერების

აგების საქმეში **ლაიბნიცის** შეტანილი წვლილის აღნიშვნას ბევრი თავს არიდებს. საილუსტრაციოდ ერთ ფაქტის გახსენებაც სამარისია. **ლაიბნიცმა 1703** გამოცემულ ნაშრომში „**Explication de l'Arithmétique Binaire**" პირველად აღწერა **თვლის ორობითი სისტემა**. მან პრაქტიკულადავე დაამტკიცა არითმეტიკულ მანქანებში **თვლის ორობითი სისტემის** გამოყენების ღირსებები. ამის „კარგად დავიწყების" გამო ზოგიერთი გამომგონებელი, **XX** საუკუნის ორმოციან წლებში, მაინც შეეცადა კომპიუტერში **თვლის ათობითი სისტემა** გამოეყენებინა. შეცდომა მალე შენიშნეს, გამოასწორეს და კომპიუტერის სამუშაო ენად **ორობითი ენა** ისე შემოიტანეს, რომ ... **ლაიბნიცის** დამსახურების ხსენებაც კი საჭიროდ არ ჩათვალეს. მსგავსი მიდგომების შესაცვლელად **პირველ ნაწილში** საკმაოდ დიდი ადგილი დავუთმეთ კომპიუტერის შექმნაში **ლაიბნიცის** დამსახურებათა ჩვენებას, რაც **მეორე ახლებურ** მიდგომად შეიძლება მივიჩნიოთ.

მესამე სიახლე ხელოვნურ ინტელექტს შეეხება. მის შექმნაზე მუშაობა ოფიციალურად **1956** წლის **15** ივნისს დაწყო. **ბუნებრივ ინტელექტის** შექმნაზე მუშაობის დაწყების დროის დადგენაზე კი თეორეტიკოსი მეცნიერები დღემდე იტეხენ თავს და ამის შესახებ მრავალი სხვადასხვა ჰიპოთეზა შექმნეს. ყველაზე სარწმუნოდ დღეს „**დიდი აფეთქების**" ჰიპოთეზაა მიჩნეული. მის თანახმად, **ბუნებრივ ინტელექტზე მუშაობის პროცესი** დაახლოებით **13,82** მილიარდი წლის წინ დაიწყო. **კიბერნეტიკა** გვასწავლის, რომ ცოცხალ ორგანიზმებში არსებული მართვის პროცესების ცოდნა ხელოვნური მმართველი მანქანების აგებაში გვეხმარება. ამას, სამწუხაროდ, ხშირად ივიწყებენ და **ხელოვნური ინტელექტის** აგების პრობლემების განხილვისას **ბუნენებრივი ინტელექტის** წარმოშობის პროცესებზე საუბარს თავს არიდებენ. ასეთი მიდგომის დასაძლევად **პირველ ნაწილში** მთელი თავი დავუთმეთ „**დიდი აფეთქების**" შედეგად **ბუნებრივი ინტელექტის** წარმოშობის საკვანძო პროცესების პოპულარულად გადმოცემას. ჩვენი აზრით, ეს ახალგაზრდების თვალთახედვის გაფართოებას შეუწყობს ხელს, რაც მათ პოტენციურ შესაძლებლობებს გაააქტიურებს.

მეორე და მესამე ნაწილები კომპიუტერული ლინგვისტიკის საკითხებს ეთმობა. **მეორე ნაწილში** მოცემულია კომპიუტერული ენების კლასიფიკაცია და ძირითადი ცნობები დაპროგრამების პარადიგმების შესახებ. **მესამე ნაწილში** განხილულია სასწავლო ენები - **პასკალი** და

ბეისიკი, რომელთა ინსტრუმენტარიუმი როგორც **სტრუქტურული**, ისე **ობიექტზე ორიენტირებული დაპროგრამირებისას** შეიძლება გამოვიყენოთ. ისინი დაპროგრამების ჩვევების პროფესიულად გამოყენების თვალსაზრისით ყველაზე პერსპექტული ენებია: პირველი ახალგაზრდებს სტრუქტურული, ხოლო მეორე - ვიზუალური დაპროგრამების ელემენტების ათვისებას გაუადვილებს. ამ ნაწილებში კომპიუტერული ლინგვისტიკის ტრადიციული საკითხებია გადმოცემული; **სიახლეებად** მხოლოდ მათი **წარმოდგენის ფორმები** შეიძლება მივიჩნიოთ, რადგან, ცნობილი ფილოსოფიური კატეგორიის თანახმად, **ფორმა** დიალექტურადაა დაკავშირებული **შინაარსთან** და მის ახლებურად ჩამოყალიბებას უწყობს ხელს.

ერთ-ერთ ადრე გამოცემულ წიგნში ვწერდი [**წ. 6**]: „ჩვენი მიზანია ქართველი სტუდენტებისათვის ეტაპობრივად მოვამზადოთ ... სახელმძღვანელოები“. გამოქვეყნებული ნაშრომების ქვემოთ მოყვანილი სია ადასტურებს, რომ დაპირება შეძლებისამებრ შემისრულებია და **ტექნიკურ უნივერსიტეტში 53**-წლიანი უწყვეტი პედაგოგიური საქმიანობის განმავლოვაში დაგროვილი ცოდნის დიდი ნაწილის ახალგაზრდობისათვის დატოვება მომიხერხებია. ვიმედოვნებ, რომ მათ იგი თავიანთ საქმიანობაში გარკვეულწილად დაეხმარება.

2025 – 1990 წლებში ჩვენს მიერ გამოცემული სასწავლო ლიტერატურა

წ. 1. ა. დუნდუა. კომპიუტერი ინტელექტი და ლინგვისტიკა - თბ, საგამომცემლო სახლი „ტექნიკური უნივერსიტეტი“, 2024. -235 გვ.

წ. 2. ა. დუნდუა. ავტომატიკისა და ტელემექანიკის სასადგურო სისტემების დაპროექტების საფუძვლები. - თბ, საგ. სახლი „ტექნიკური უნივერსიტეტი“, **2022**. – 132 გვ.

წ. 3. ა. დუნდუა. კომპუტერული სისტემების პროგრამული საშუალებები (**Software**). თბ, საგამომცემლო სახლი „ტექნიკური უნივერსიტეტი“, **2019**. - 164 გვ.

წ. 4. ა. დუნდუა. კომპუტერული სისტემების ტექნიკური საშუალებები (**Hardware**). თბ, საგამომცემლო სახლი „ტექნიკური უნივერსიტეტი“, **2019**. - 144 გვ.

წ. 5. ა. დუნდუა. კომპიუტერული სისტემებისა და საინფორმაციო ტექნოლოგიების თეორიული საფუძვლები. თბ, „ტექნიკური უნივერსიტეტი“, **2014**. - 258 გვ.

წ. 6. ა. დუნდუა. ტრანსპორტზე მიკროპროცესორული ტექნიკის გამოყენების საფუძვლები. თბ, „ტექნიკური უნივერსიტეტი“, **2017**. - 344 გვ.

წ. 7. ა. დუნდუა. ავტომატიკისა და ტელემექანიკის სასადგურო და საგადასარბენო სისტემები, ნაწ. II - თბ, „ტექნიკური უნივერსიტეტი“. - **2014**. - 478 გვ.

წ.8. ა. დუნდუა. ავტომატიკისა და ტელემექანიკის სასადგურო და საგადასარბენო სისტემები. ნაწ. I თბ, „ტექნიკური უნივერსიტეტი“, **2009**. - 291 გვ.

წ.9 **ა. დუნდუა, ვ. საპოჟნიკოვი, ვ. საპოჟნიკოვი.** - დისკრეტულ მოწყობილობათა თეორიის საკითხები. - თბ, „ საქართველოს ტექნიკური უნივერსიტეტი“, **1990** – 119 გვ.

პირველი ნაწილი

კომპიუტერიდან ხელოვნურ ინტელექტამდე

1.1. კომპიუტერი დაიწყო ... ცხრილების გამოთვლით

დიდი ბრტანეთისათვის, როგორც დიდი საზღვაო იმპერიისათვის, მეთვრამეტე საუკუნეში მეტად მნიშვნელოვანი იყო გემების უსაფრთხოდ მოძრაობის პრობლემა. ამიტომ **1776** წლიდან იქ დაიწყეს „**საზღვაო ალმანახის**“ (*იწგ.«Nautical Almanac»*) ყოველწლიური გამოცემა. მასში მოცემული იყო ასტრონომიული და სანავიგაციო ცხრილები. მათ შესადგენად ჩასატარებელი იყო დიდი რაოდენობის რთული ლოგარითმული და ტრიგონომეტრიული გამოთვლები. ამ სამუშაოს შესრულებაზე დაკავებული იყო მრავალი გამომთვლელი. მათ სწორ მუშაობაზე დიდად იყო დამოკიდებული გემების უსაფრთხო მოძრაობა. გამომთვლელების მონდომებული მუშაობის მიუხედავად, ცხრილებში მაინც იპარებოდა ათობითი და ასობითი შეცდომა; საქმეს ვერც მათ აღმოსაჩენად და გასასწორებლად გამომცემლების მიერ დაქირავებული კორექტორების მთელი შტატი შველოდა.

სწორედ ამ პერიოდში გამოვიდა სამეცნიერო ასპარეზზე ახალგაზრდა მათემატიკოსი **ჩარლზ ბებიჯი** (*იწგ. Charles Babbage, 1791 - 1871*). მას, როგორც თავისი ქვეყნის პატრიოტს, დიდად აწუხებდა ეს პრობლემა და მის გადაწყვეტაზე დაიწყო მუშაობა. გადაწყვეტის მოსაძებნად იგი ბიბლიოთეკებში, როგორც მხატვრულად იტყვიან, აღამებდა და ათენებდა. **1864** წელს დაწერილ ავტოგრაფიულ წიგნში „**ფილოსოფოსის ცხოვრების ფურცლებში**“ იგი წერს „ ... ერთხელ საღამოს, როდესაც კემბრიჯის ანალიზური საზოგადოების ერთ-ერთ

ოთახში ჩემ წინ არსებულ ლოგარითმების გადაშლილ ცხრილთან ვთვლემდი, ოთახში საზოგადოების რომელიღაც წევრი შემოვიდა და, დაინახა რა, რომ მე თითქმის მეძინა, ჩამძახა: „ო! რაზე ოცნებობ, ბებიჯი?“, რაზედაც ვუპასუხე: „იმაზე, რომ ყველა ეს ცხრილი შეიძლება გამოითვალოს მანქანის დახმარებით“ ... ეს შემთხვევა შეიძლება მომხდარიყო **1812** ან **1813** წელს“.

სწორედ ზემოთ აღნიშნული პერიოდიდან დაიწყო **21** წლის **ბებიჯმა** ლოგარითმებიანი ტრიგონომეტრიული ცხრილების ავტომატურად გამომთვლელი მანქანის გამოგონებაზე ფიქრი. ასეთი მანქანის აგების იდეა გერმანელმა ინჟინერმა **იოჰან ჰელფრიხ ფონ მიულერმა** *(გერ. Johann Helfrich von Müller, 1746-1830)* გამოთქვა თავის ჯერ კიდევ **1788** წელს გამოცემულ წიგნში. **მილერმა** აგრეთვე გამოთქვა ასეთი მანქანის შესაქმნელად **სხვაობის მათემატიკური მეთოდის** გამოყენების **მეორე იდეაც**, და სავარაუდო მანქანას „**სხვაობითი მანქანა**“ უწოდა. ბებიჯს გაცნობილი ჰქონდა **მიულერის** მიერ ამ საკითხზე დაწერილი სტატიის ინგლისური თარგმანი, მაგრამ არ შეიძლება იმის მტკიცება, იყო თუ არა ბებიჯი მიულერის გავლენის ქვეშ. სამაგიეროდ დადასტურებულია, რომ მან თავისი პროექტისათვის ძირითადი იდეა ამოიღო ფრანგი მათემატიკოსის, პარიზის მეცნიერებათა აკადემიის წევრისა და ლონდონის სამეფო საზოგადოების უცხოელი წევრის **გასპარ დე პრონის** *(ფრ. Gaspard Clair François Marie Riche, baron de Prony,1755 – 1839)* ნაშრომებიდან. **პრონის** დაავალეს მეტრულ სისტემაში შესაყვანად მომზადების მიზნით შეემოწმებინა და გაეუმჯობესებინა ლოგარითმებიანი ტრიგონომეტრული ცხრილები. **პრონიმ** სამუშაოს სამ დონეზე განაწილების იდეა წამოაყენა. უმაღლეს **ზედა დონეზე** რიცხვითი გამოთვლებისათვის საჭირო გამოსახულებების დასამუშავებლად დასაქმდებოდა **5-6** გამოჩენილი მათემატიკოსი; **მეორე დონეზე** მაღალკვალიფიციური **7 – 8** მათემატიკოსი ერთმანეთისაგან ხუთი ან ათი ინტერვალით დაშორებული არგუმენტებისათვის გამოითვლიდნენ ლოგარითმული ფუნქციების მნიშვნელობებს, რომლებიც ცხრილში საყრდენ რიცხვებად იქნებოდა გამოყენებული. ამის შემდეგ ლოგარითმული ფუნქციები **6**-ზე არაუმეტესი ხარისხიანი **მრავალწევრების** სახით წარმოდგინდებოდა და ისინი გამოსათვლელად გადაეცემოდა **გამომთვლელებად** წოდებულ **მესამე დონეზე** დასაქმებულ პირებს, რომელთა რაოდენობა **90**-ს აღწევდა. დავალების შესასრულებლად მათ მხოლოდ მარტივი არითმე-

ტიკული ოპერაციების აკურატულად გამოთვლა დასჭირდებოდათ. ეს მაქსიმალურად შეამცირებდა მათ მიერ შეცდომების დაშვების ალბათობას. **პრონის** საფრანგეთში ყოფნისას გაეცნო **ბებიჯი**. მისმა ზემოთ მოყვანილმა წინადადებამ მას ზემოთ აღნიშნული „**სხვაობითი მანქანის**“ პრაქტიკული რეალიზებისაკენ უბიძგა. ეს შესაძლებელს გახდიდა **მესამე დონეზე** დასაქმებული **გამომთვლელები**, რომლებიც, სამწუხაროდ, თავიანთი საქმიანობისას მაინც უშვებდნენ გარკვეული რაოდენობის შეცდომებს, მთლიანად ჩაენაცვლებინა ამ საქმის უშეცდომოდ შემსრულებელი „**სხვაობითი მანქანით**“.

1822 წელს გამოქვეყნებულ სტატიაში ბებიჯმა აღწერა ასეთი მანქანა და შეუდგა მის პრაქტიკულ რეალიზაციას. მისთვის, როგორც მათემატიკოსისთვის, ცნობილი იყო მრავალწევრებით ფუნქციების **აპროქსიმაციისა** და სასრული სხვაობების გამოთვლის პროცესები. მათი ავტომატიზებისათვის **ბებიჯმა** დაიწყო მანქანის დაპროექტება, რომელსაც **სხვაობითი მანქანა** (ინგ. Difference engine) უწოდა. მას ავტომატურად უნდა გამოეთვალა ექვსზე არაუმეტესი ხარისხის მქონე მრავალწევრების **18**-ნიშნიანი მნიშვნელობები.

*შევნიშნავთ, რომ **აპროქსიმაცია** (ლათ. „proxima” - „უახლოესი“) მეცნიერული მეთოდია, რომლთაც გარკვეულ ობიექტები შეიძლება შევცვალოთ მათთან მიახლოებული სხვა მარტივი ობიექტებით; **სასრული სხვაობა** - ინტერპოლირებასა და რიცხვით დიფერენცირებაში ფართოდ გამოყენებული მათემატიკური ტერმინია; **ინტერპოლირება** (ლათ. Interpolis –„გაგლუვება“) მეთოდია, რომელიც საშუალებას გვაძლევს ფუნქციის ნაცნობი მნიშვნელობების დისკრეტული ნაკრების საშუალებით მისი უცნობი საშუალედო მნიშვნელობები ვიპოვოთ.*

ბებიჯმა 1822 წელსვე სპეციალური ლილვაკებითა და კბილანებით ააგო სხვაობითი მანქანის მოდელი და მისი სრულმასშტაბიანი დამუშავებისათვის დიდი ბრიტანეთის მთავრობისაგან მიიღო **1500** ფუნტი სტერლინგი. პროექტის რეალიზებისათვის **ბებიჯი**ს მიერ სუბსიდიების საერთო ჯამმა **17000** ფუნტი სტერლინგი შეადგინა. მას მანქანის რეალიზებისათვის სამი წლის ვადა მისცეს. სამწუხაროდ, ბებიჯმა ვერ გათვალა მანქანის აგებასთან დაკავშირებული ყველა სიძნელე და, არა თუ ვერ ჩაეტია სამ წელში, არამედ ცხრა წლის მუშაობის შემდეგ იძულებული გახდა **1933** შეეჩერებინა მუშაობა და სხვა სამუშაოს შესრულება დაეწყო. მანქანის უკვე დასრულებული ნაწილი შესანიშნავად მუშაობდა და გამოთვლებს მოსალოდნელზე უფრო

მაღალი სიზუსტით ასრულებდა. აღსანიშნავია კიდევ ორი ფაქტი. **პირველი** (და, როგორც ქვემოთ დავინახავთ, მთავარი) ისაა, რომ „სხვაობითი მანქანაში“ არითმეტიკული ოპერაციების შესასრულებლად **თვლის ათობითი სისტემა იყო გამოყენებული**, ხოლო მე**ორე** კი (რომელიც გამოთვლითი ტექნიკის ისტორიით დაინტესებული პირებისათვის იქნება მნიშვნელოვანი) ისაა, რომ **ბებიჯმა** ამ მანქანიდან ინფორმაციის გამოსატანი საინტერესო მოწყობილობაც შექმქმნა; იგი ინფორმაციას ფოლადის შტამპით ამოტვიფრავდა სპილენძის პატარა დაფაზე და, ფაქტობრივად, თანამედროვე პრინტერების წინასახე - **მექანიკურ პრინტერი** იყო.

მოგვიანებით ბებიჯმა განაგრძო „სხვაობით მანქანაზე“ მუშაობა და **1847-49** წლებში დამუშავებულ მის გაუმჯობესებულ ვარიანტს „**№2 სხვაობითი მანქანა**“ უწოდა.

ჩარლზ ბებიჯის დაბადების ორასი წლის იუბილის პერიოდში, **1989**-დან **1991** წლებში, **მეცნიერების ლონდონის მუზეუმში** „**№2 სხვაობითი მანქანის**“ ასლი ააგეს, ხოლო **2000** წელს იმავე მუზეუმში აამუშავეს ბებიჯის მიერ საკუთარი მანქანისათვის აგებული **პრინტერი**. ძველ ნახაზებში უმნიშვნელო კონსტრუქციული უზუსტობების აღმოფხვრის შემდეგ, ორივე კონსტრუქცია წუნდაუდებლად ამუშავდა. ამ ექსპერემენტებმა დაასრულა **ჩარლზ ბებიჯის** კონსტრუქციათა პრინციპული მუშაობის უნარიანობის შესახებ მიმდინარე ხანგრძლივი დებატები.

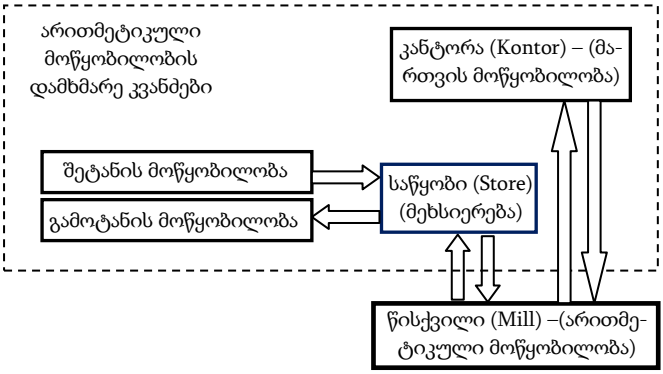
1.1.1. ბებიჯის მთავარი იდეა

ბებიჯის მიერ „**სხვაობით მაბქანაზე**“ მუშაობის შეწყვეტის მიზეზი, უსახსრობის გარდა, ის გარემოებაც იყო, რომ იგი **1834** წელს **ახალმა იდეამ** გაიტაცა. მან გადაწყვიტა შეექმნა მანქანა, რომელიც არა მარტო ცხრილს, არამედ ისეთი მათემატიკურ ამოცანებსაც გამოითვლიდა, რომელთა გადაწყვეტაზე „თავს იტეხდნენ“ ინჟინრები და მათემატიკოსები. სწორედ ეს იყო ბებიჯისეული მთავარი, შეგვიძლია ვთქვათ, **ზოგადსაკაცობრიო მნიშვნელობის** იდეა. იმ პერიოდში, არსებული დაბალი ტექნოლოგიური დონის გამო, ასეთი იდეისთვის პრაქტიკული ხორცშესხმა შეუძლებელი იყო და ვერც **ბებიჯმა** შეძლო, მაგრამ იდეის რეალიზებისათვის აუცილებელი თეორიული მასალა მან წარმატებით დაამუშავა. ასეთ მანქანას **ბებიჯმა** „**ანალიზური მანქანა**“ უწოდა. იგი თანამედროვე **ციფრული გამომთვლელი მანქანის წინასახეა.** თანამედროვე გამო-

მთვლელი მანქანის (კომპიუტერის) აგების პრინციპები **XIX** საუკუნეში იქნა დამუშავებული. ამგვარად ბებიჯმა მთელი საუკუნით გაასწრო დროს. მან შორმჭვრეტელურად განსაზღვრა, რომ „**ანალიზური მანქანის**“ ასაგებად აუცილებელი იყო კარდინალურად გამარტივებულიყო მანქანის არითმეტიკული კვანძი - ამჯამავი სქემა. დაძაბული შრომის შემდეგ მან შექმნა **გაჭოლი გადატანიანი ამჯამავი სქემა**, რომელიც ერთ-ერთ უთვალსაჩინოესი გამოგონებაა.

„ანალიზური მანქანა“ შეიცავდა შემდეგი ნაწილებს **(ნახ. 1.1**):

1) საწყობს (Store), რომელშიც უნდა შენახულიყო გამოთვლებში მონაწილე ცვლადების მნიშვნელობები და ოპერაციათა ჩატარებისას მიღებული შედეგები. თანამედროვე კომპიუტერში მას **მეხსიერება** ეწოდება;



ნახაზი 1.1. „ანალიზური მანქანის“ („დაპროგრემადი კალკულატორის“) ბლოკ-სქემა

2) წისქვილს (**Mill**), რომელსაც ცვლადებზე არითმეტიკული ოპერაციები უნდა შეესრულებინა. თანამედროვე კომპიუტერში მას **არითმეტიკული მოწყობილობა** ეწოდება.

3) კანტორას (**Konto**), რომელსაც უნდა დაეცვა ოპერაციათა შესრულების თანამიმდევრობა. თანამედროვე კომპიუტერში მას **მართვის მოწყობილობა** ეწოდება;

4) შეტანა-გამოტანის მექანიზმებს (მოწყობილობებს), რომლებიც მუყაოს ბარათების (ისინი პერფობარათების წინასახეა) მეშვეობით ასრულებდა საკუთარ ოპერაციებს. განასხვავებდნენ „ოპერაციულ ბა-

რათებს“, „ცვლადების ბარათებს“, „მმართველ ბარათებს“, „მიმწოდებელ ბარათებს“ და „ნულოვან ბარათებს“.

„ოპერაციული ბარათებიან“ **მექანიზმს** სათანადო ბარათზე არსებული ნახვრეტების შესაბამისი გარკვეული არითმეტიკული ოპერაციის შესასრულებლად უნდა მოემზადებია „წისქვილი“ („არითმეტიტიკული მოწყობილობა“).

„ცვლადების ბარათებიან“ **მექანიზმს** უნდა ემუშავა „მმართველ ბარათებთან“ და ცვლადები უნდა გადაეტანა „საწყობიდან“ (მეხსიერებიდან) „წისქვილში“ ან პირიქით.

„მიმწოდებელ ბარათები“ და „ნულოვან ბარათები“ გამოიყენებოდა „საწყობსა“ და „წისქვილს“ შორის რიცხვების გადასატანად. ამასთანავე პირველ მათგანს რიცხვები გადაჰქონდა „საწყობიდან“ (მეხსიერებიდან) წისქვილში (არითმეტიკულ მოწყობილობაში), ხოლო „ნულოვან ბარათებს“ - პირიქით.

„ანალიზური მანქანის“ შექმნაზე ბებიჯმა **1834** წელს დაიწყო. მან სხვადასხვა მექანიზმების ასაგებად დაამუშავა **200**-ზე მეტი დაწვრილებითი ნახაზი და მანქანის დაახლოებით **30** ვერსია. იგი სიცოცხლის ბოლო წუთამდე მის სრულყოფაზე მუშაობდა.

„**ანალიზური მანქანის**“ („**Analytical Engine**“) სქემა იმდენად ახლოა თანამედროვე კომპიუტერის მოწყობილობასთან, რომ **ბებიჯი** თავისუფლად შეგვიძლია მივიჩნიოთ **კომპიუტერული ერის** წინამორბედად. შესაქმნელად დარჩა მხოლოდ შენახვადი პროგრამის სქემა, რომელიც **100** წლის შემდეგ განხორციელდა.

სამწუხაროდ, ყველა მისი იდეა მხოლოდ ქაღალდზე დარჩა - მოქმედი მოდელი არ იქნა აგებული. მიუხედავად ამისა, ბებიჯი იყო ადამიანი, რომელმაც პირველად შექმნა კომპიუტერის ზოგადი კონცეფცია და დროს თითქმის მთელი საუკუნით გაუსწრო. საელემენტო ბაზის არარსებობამ, ტექნოლოგიების დაბალმა დონემ და დაუფინანსებლობამ ხელი შეუშალა მას ხორცი შეესხა საკუთარი ინოვაციური იდეისათვის.

„**ანალიზურ მანქანაზე**“ მუშაობა მან ფაქტობრივად სამ წელიწადში - **1837** წელს დაასრულა, როდესაც გამოვიდა მისი სტატია „გამომთვლელი ძრავის მათემატიკური ძალა“. მასში მან სრულად აღწერა აღნიშნული მანქანა და სიცოცხლის დარჩენილ დროში მხოლოდ მის სრულყოფაზე მუშაობდა.

მოკლედ შევჩერდეთ **„ანალიზური მანქანის“** შესაძლო ზომების შეფასებაზე. ბებიჯის აზრით, მის ასამოქმედებლად უნდა გამოყენებულიყო **ორთქლის მანქანა**, რადგან **1834** წელს, როდესაც ბებიჯმა **„ანალიზური მანქანის“** სქემა დაამუშავა, კაცობრიობას ელექტრული ენერგიის გამოყენება ჯერ კიდევ არ ჰქონდა მოფიქრებული. მხედველობაში თუ მივიღებთ იმ ფაქტსაც, რომ წინასწარი გათვლებით, მის ასაგებად **50 000**-ზე მეტი კბილანათვალი იქნებოდა საჭირო, უნდა ვივარაუდოთ, რომ მისი ზომები ორთქმავლის ზომებზე ნაკლები არ უნდა ყოფილიყო. პრობლემის გადაწყვეტა ძალიან ძნელი იყო და ამიტომ მიაჩნდა **ჩარლზ ბებიჯს**, რომ მიზნის მიღწევას ნახევარი საუკუნე მაინც დასჭირდებოდა.

აღსანიშნავია, რომ **ჩარლზ ბებიჯმა** მარტოდმარტომ დაამუშავა ანალიზური მანქანის კონსტრუქცია. მისი უახლოესი მეგობარი, დამხმარე და ერთადერთი თანამოაზრე იყო მხოლოდ **ადა ავგუსტა ლავლეისი** (ინგ. *Augusta Ada King Byron, Countess of Lovelace; 1815 -1852*). **1840** წელს იტალიელმა მათემატიკოსებმა **ჩარლზ ბებიჯი** ტურინში მიიწვიეს, სადაც მან წაიკითხა ლექცია თავისი **„ანალიზური მანქანის“** შესახებ. იტალიელმა გენერალმა, მეცნიერმა, სახელმწიფო მღვაწემ, რომელიც ორ წელზე მეტი ხნის განმავლობაში ხელმძღვანელობდა იტალიის მინიტრთა საბჭოს, ტურინის საარტელირიო აკადემიის მასწავლებელმა **ლუიჯი ფედერიკო მენაბრეამ** (*Luigi Federico Menabrea; 1809 - 1896*) შეადგინა ბებიჯის ლექციის კონსპექტი და ფრანგულ ენაზე გამოაქვეყნა. ამით მან ძალიან დიდი საქმე გააკეთა, რადგან ამით **ადა ლავლეისს** საშუალება მისცა, გაცნობოდა ლექციის შინაარსს. მან კონსპექტი ინგლისურად თარგმნა და გამოაქვეყნა საკუთარი კომენტარებით, რომელის მოცულობა თავად ტექსტის მოცულობას აღემატებოდა. კომენტარებში მან აღწერა მანქანა და მისი დაპროგრამების ინსტრუქცია. ეს იყო მსოფლიოში შექმნილი პროგრამა. სწორედ ამიტომ **ადა ლავლეისი** სამართლიანად ითვლება პირველ დამპროგრამებლად. დასასრულს, უნდა აღვნიშნო, იგი არის შესანიშნავი ბრიტანელი პოეტის **ჯორჯ ბაირონის** ქალიშვილი.

1.2. ცნობები არითმეტიკული მანქანებისა და არითმეტიკების შესახებ

ჩარლზ ბებიჯმა „**საზღვაო ალმანახთან**“ დაკავშირებული მათემატიკური ამოცანის გადასაწყვეტად (ლოგარითმული ცხრილის ასაგებად) გადაწყვიტა ხორცი შეესხა გერმანელი მეცნიერის **მიულერის მიერ** დამუშავებული **„სხვაობითი მანქანის**“ შექმნის იდეისათვის. აღნიშნული მანქანა ვინაიდან კონკრეტული ამოცანის გადასაწყვეტად იქმნებოდა, ამიტომ იგი **სპეციალიზებულ მათემატიკურ მანქანად** უნდა ჩაითვალოს. მასზე მუშაობისას **ბებიჯს** დაებადა **უნივერსალური (ნებისმიერი სახის) მათემატიკური მანქანის** შექმნის საკუთარი იდეა. ამ უკანასკნელს მან „**ანალიზური მანქანა**“ უწოდა. მაშასადამე, ორივე მათგანი, (როგორც სხვაობითი, ისე ანალიზური), მანქანა **მათემატიკური მანქანაა.** დაწვრილებით განვიხილოთ თითოეული მათგანი.

„**სხვაობითი მანქანა**“ აპარატურულად (და არა პროგრამულად!) **რეალიზებული სპეციალიზებული არითმეტიკული მანქანა** იყო, რომელიც **„საზღვაო კალენდრისათვის**“ საჭირო ლოგარითმული ცხრილების შედგენაზე დასაქმებული გამომთვლელელი ადამიანების ჩასანაცვლებად შეიქმნა. იგი გამოიყენებოდა არა ცალკეული არითმეტიკული ოპერაციების შესასრულებლად, არამედ ლოგარითმულ ფუნქციების შესაბამის **მრავალწევრთა მნიშვნელობების გამოსათვლელად**. მიღებული მნიშვნელობებით შეივსებოდა სათანადო ლოგარითმული ცხრილების ცარიელი (იხ. წინა თავი) უჯრები.

„**ანალიზური მანქანა**“ არა მარტო მრავალწევრების, არამედ ნებისმიერი არითმეტიკული გამოსახულებების მნიშვნელობათა გამომთვლელი პროგრამულად რეალიზებული **უნივერსალური არითმეტიკული მანქანაა**. მისი ფუნქციონირებისათვის წინასწარ უნდა დამუშავდეს სპეციალური პროგრამა და შეტანილი იქნეს მანქანაში. პირველი ასეთი პროგრამა, როგორც ზემოთ აღვნიშნეთ, **ადა ლავლეისს** ეკუთვნის, რის გამოც მან **პირველი დამპროგრამების** სახელი დაიმკვიდრა.

არითმეტიკული მოწყობილობა ანალიზური მანქანის სტრუქტურის შემადგენელი ელემენტია და არა ავტონომიურად ფუნქციონირებადი მანქანა. დამოუკიდებლად ფუნქციონირებად არითმეტიკულ მოწყობილობას **არითმეტიკული მანქანა** ვუწოდოთ. პირველი არითმეტიკული მანქანა ჯერ კიდევ **XVII** პირველ ნახევარში იქნა დამუშავებული. თითქმის **300** წლის განმავლობაში მის პირველ გამომგონებლად ითვლებოდა ფრანგი მეცნიერი **ბლეზ პასკალი** (*ფრან; Blaise Pascal; 1623 - 1662*), რომელმაც **1642** წელს გადასახადების ამკრებად მომუშავე მამას სამუშაოს შესამსუბუქებლად დღეს „**პასკალინად**“ წოდებული საანგარიშო მანქანა - **ამჯამავი მანქანა** შეუქმნა. მანქანას შეკრების გარდა სხვა არითმეტიკული ოპერაციების შესრულებაც შეეძლო, მაგრამ ამისათვის განმეორებადი შეკრებების საკმაოდ მოუხერხებელი პროცედურების შესრულება იყო საჭირო. რთული კონსტრუქციის გამო „**პასკალინას**“ მხოლოდ რამდენიმე ეგზემპლარი იქნა დამზადებული.

შტუდგარტის საქალაქო ბიბლიოთეკაში **1957** წელს მუშაობისას კეპლერის სამეცნიერო ცენტრის დირექტორმა **ფრანც ჰამერმა** აღმოაჩინა **სანკტ-პეტერბურგთან** ახლოს მდებარე პულკოვოს ობსერვატორიაში დაცული გამოჩენილი ასტრონომისა და მათემატიკოსის **იოჰან კეპლერის** არქივში არსებული უცნობი სათვლელი მოწყობილობის ესკიზის ფოტოასლი; მან დაამტკიცა, რომ იგი იყო **ებერჰარდისა და კარლის სახელობის ტიუბენგენის უძველესი** (*დაარსებულია 1477 წელს*) **უნივერსიტეტის** (*გერ. Eberhard-Karls-Universität Tübingen*) პროფესორის **ვილჰელმ შიკარდის** მიერ **კეპლერისათვის 1624** წლის **25** თებერვალს მიწერილი წერილის დანართი, სადაც იგი აღწერს მის მიერ **1623** წელს გამოგონებულ სათვლელ მანქანას.

ზემოთ აღნიშნულის თანახმად, **პირველი არითმეტიკული მანქანა** შეიქმნა არა **1642**, არამედ **1623** წელს. ვინაიდან იგი **ბლეს პასკალმა** და **ვილჰელმ შიკარდმა** ერთმანეთისაგან სრულიად დამოუკიდებლად გამოიგონეს, ამიტომ ისინი სამართლიანად ითვლებიან არითმეტიკული მანქანის შექმნის **პიონერებად**.

მომდევნო პერიოდში სხვადასხვა კონსტრუქციის ბევრი არითმეტიკული მანქანა აიგო, მაგრამ აუცილებლად მიგვაჩნია თქვენი ყურადღება ამ სფეროში **XVII** საუკუნის გენიოსის, **გოტფრიდ ვილჰელმ ლაიბნიცის** (*გერ. Gottfried Wilhelm von Leibniz; 1646-1716*), მიერ შეტანილ წვლილზე შევაჩეროთ.

არითმეტიკული მანქანის შექმნის იდეა **ლაიბნიცს** ცნობილ ჰოლანდიელ ასტრონომ, ფიზიკოს და მათემატიკოს **ქრისტიან ჰიუგენსთან** (*ნიდერ. Christiaan Huygens; 1629 - 1695*) ურთიერთობისას დაებადა. იგი განაცვიფრა იმ ფაქტმა, რომ მუშაობისას უდიდეს მეცნიერს კოლოსალური სიდიდის გამოთვლების ხელით შესრულება უხდებოდა. მან ჩათვალა, რომ მის მსგავს „შესანიშნავ ადამიანებს“ თავიანთი ძვირფასი დრო იმ გამოთვლების შესასრულებლად არ უნდა დაეკარგათ, რომლებსაც სპეციალური მანქანის დახმარებით ნებისმიერი ადამიანი გაართმევდა თავს.

გადაწყვიტელების მიღების შემდეგ **გოტფრიდ ლაიბნიცი** დიდი გატაცებით ჩაუჯდა საქმეს. მისი შთაგონების ერთი წყარო **იყო ბლეზ პასკალის** არითმეტიკული მანქანა. საკუთარი არითმეტიკული მანქანის პირველი პროტოტიპი მან **1673** შექმნა და მისი დემინსტრირება მოახდინა **ლონდონის სამეფო საზოგადოების სხდომაზე.** თავისი გამოსვლის დასასრულს მან აღიარა საკუთარი მოწყობილობის არასრულყოფილება და მსმენელებს მასზე მუშაობის განგრძობას დაპირდა. შემდგომ, **1654 – 1676** წლებში, მან დიდი სამუშაო ჩაატარა მანქანის გასაუმჯობესებლად და ლონდონში კალკულატორის ახალი ვარიანტი მიიტანა. ეს იყო პრაქტიკული გამოთვლისათვის გამოუსადეგარი მცირეთანრიგიანი მთვლელი მანქანის მოდელი. მხოლოდ **1694** წელს **ლაიბნიცმა** ააგო **12**-თანრიგიანი მოდელი. შემდეგ მან კიდევ რამდენდენჯერ გადაამუშავა იგი და უკანასკნელი ვარიანტი მხოლოდ **1710** წელს დაამზადა. საბოლოო ანგარიშით, მის აგებაზე ლაიბნიცმა **37** წელი დახარჯა. მსგავს მანქანებს შორის იგი იყო პირველი, რომელსაც დიდ რიცხვებზე ყველა ძირითადი არითმეტიკული (შეკრების, გამოკლების, გამრავლების, ახარისხების, კვადრატული და კუბური ფესვების ამოღების) ოპერაციების შეესრულება შეეძლო. იგი უნდა ჩაითვალოს **არითმომეტრის** პროტიპად, რომელიც **1820** წლიდან **XX** საუკუნის **60**-იან წლებამდე ფართოდ გამოიყენებოდა. მასში არსებული „**ლაიბნიცის თვლად**“ წოდებული ორიგინალური მოწყობილობა, რომელსაც მომდევნო ვარიანტებში **ცილინდრს** უწოდებდნენ, მანქანას საშუალებას აძლევდა გამრავლებისა და გაყოფის ოპერაციები საოცრად სწრაფად შეესრულებინა.

ამერიკელმა გამომგონებელმა, მაუსის, აგრეთვე სამომხმარებლო ინტერფეისის, ჰიპერტექსტისა და ტექსტური რედაქტორისა და ჯგუფური ონლაინ-კონფერენციის გამომგონებელმა, ადამიან-მანქანური

ინტერფეისის მკვლევარი **დუგლას კარლ ენგელბარტი** (*ინგ. Duglas Carl Engelbart; 1925 - 2013*) **1968** ფეხზე წამოუდგა და აპლოდისმენტები უძღვნა **ლაიბნიცის** გამომთვლელ მანქანა.

ამსტერდამის თავისუფალი უნივერსიტეტის *(ნიდერ. Vrije Universiteit Amsterdam*) პროფესორმა, კომპიუტერული სისტემების დამმუშავებელთა ჯგუფის ხელმძღვანელმა, **ენდრიუ სტიუარტ ტანენბაუმმა** (*ინგ. Andrew Stuart Tanenbaum; 1944*), არითმეტიკულ გამომთვლელ მანქანებს **ნულოვანი თაობის კომპიუტერები** უწოდა. ამ მანქანებმა, სხვა ობიექტებთან ერთად, შექმნეს ნოყიერი ნიადაგი, რომლიდანაც აღმოცენდა **თანამედროვე ციფრული კომპიუტერები. „**ბებიჯისეული სხვაობითი მანქანა“ ამ ნიადაგიდან აღმოცენებულ ყლორტს შეგვიძლია შევადაროთ, რომელიც შემდგომი ევოლუციური განვითარების შედეგად პირველ ციფრულ კომპიუტერად ჩამოყალიბდა. აქვე აღვნიშნავთ, რომ **ელექტრონულ გამომთვლელ მანქანებად** წოდებულ მის პირველ წარმომადგენლებს თავდაპირველად **დაპროგრამებად კალკულატორებს უწოდებდნენ.** როგორც **1.1.** ნახაზიდან ჩანს, მათში ძირითადი მუშა მოწყობილობა **მათემატიკური მოწყობილობა** იყო. დანარჩენი ელემენტები (შეტანის მოწყობილობა, მეხსიერება, მართვის მოწყობილობა და გამოტანის მოწყობილობა) არითმეტიკული მოწყობილობის ფუნქციონირების უზრუნველყოფისათვის საჭირო დამხმარე საშუალებებებს შეიძლება შევადაროთ. ამ მოწყობილობას **მხოლოდ არითმეტიკული ოპერაციების შესრულება** შეეძლო. ასე რომ, მივადექით მის უდიდებულესობა **არითმეტიკას**, რომლის შესახებაც ყველა დროის ერთ-ერთი უდიდესი გერმანელი მათემატიკოსი **იოჰან კარლ ფრიდრიხ გაუსი** (*გერმ. Johann Carl Friedrich Gauß; 1777 - 1855*) ამბობდა: „არითმეტიკა მათემატიკის დედოფალია, მათემატიკა კი - ყველა მეცნიერების“. მისი ნათლად გააზრების გარეშე სრულფასოვნად ვერ შევაფასებთ კომპიუტერის უშუალო წინამორბედის - „**ანალიზური მანქანის**“ და, აქედან გამომდინარე, თავად თანამედროვე კომპიუტერის კონსტრუქციულ თავისებურებებს. არითმეტიკის შესწავლით დაიწყო ჩვენი დაწყებითი სასკოლო განათლება, ახლა კი მას მოწიფული ადამიანის თვალით შევხედოთ.

1.2.1. არითმეტიკა იწყებოდა ასე

ტერმინი „**არითმეტიკა**“ მიღებულია ბერძნული სიტყვისაგან „**arithmys**“, რაც „**რიცხვს**“ ნიშნავს. **2023** წლის **10** სექტემბერს ფორმულირებული განსაზღვრების ძალით, **არითმეტიკა** მათემატიკის ნაწი-

ლია, რომელიც **რიცხვებისა** და **მათზე ჩასატარებელი მოქმედებების** უმარტივეს თვისებებს შეისწავლის. იგი ერთ-ერთი უძველესი მეცნიერებაა: მისი ისტორია ჩვენს წელთაღრიცხვამდე III – II ათას-წლეულიდან იღებს სათავეს. თანამედროვე სახის მიღებამდე ბევრჯერ მოხდა არითმეტიკის სახეცვლილება. უცნაურადაც შეიძლება მოგეჩვენოთ, მაგრამ მან რიცხვებისა და მათზე ჩასატარებელ მოქმედებათა თვისებების შესწავლა იმ პერიოდიდან დაიწყო, როდესაც ჯერ კიდევ ვერ ხდებოდა რიცხვების ფორმალური გამოსახვა: ამისათვის დღეს საყოველთაოდ მიღებული ნიშნები - ციფრები - გაცილებით გვიან პერიოდში შეიქმნა.

ცხრ. 1.1. ქართული ანბანის ასოებით რაოდენობების გამოსახვა

ა - 1	ბ - 2	გ - 3	დ - 4	ე - 5	ვ - 6	ზ - 7	ჱ - 8
თ - 9	ი- 10	კ- 20	ლ- 30	მ- 40	ნ- 50	ჲ- 60	ო - 70
პ - 80	ჟ - 90	რ- 100	ს - 200	ტ - 300	უ - 400	ფ - 500	ქ- 600
ღ - 700	ყ- 800	შ- 900	ჩ- 1000	ც- 2000	ძ- 3000	წ- 4000	ჭ - 5000
ხ - 6000	ჴ - 7000	ჯ - 8000	ჰ - 9000	ჵ - 10000			

სპეციალური პირობითი ნიშნების არარსებობის პერიოდში ადამიანები რაოდენობების აღნიშვნისათვის სხვადასხვა საშუალებას, კერძოდ, იეროგლიფებს, დიაგრამებს, ანბანის ასოებს და ა. შ., იყენებდნენ. მაგალითისათვის **1. 1**. ცხრილში მოყვანილია ქართული ანბანის ასოების მეშვეობით რაოდენობების აღნიშვნისათვის საქართველოში გამოყენებული მეთოდის მაგალითი. რაოდენობების აღნიშვნისათვის მასში გამოყენებულია ციფრები, რომლებსაც დღეს **არაბულ ციფრებს** ვუწოდებთ. მოცემული ცხრილის გამოყენებით რიცხვი **3748** ასე ჩაიწერება: **ჯღმჱ = 3000 + 700 + 40 + 8 = 3748**. აღნიშნული მეთოდის გამოყენებისას საკმაოდ რთულია არითმეტიკული ოპერაციათა პროცესების ფორმალიზება.

1.2.2. ათობითი არითმეტიკის შექმნა და მისი განზოგადება

თანამედროვე არითმეტიკის შექმნისაკენ პირველი ნაბიჯი გადაიდგა ჩვენი წელთაღრიცხვის **595** წელს, როდესაც ინდოეთში რიცხვების გამსასახვისათვის შეიქმნა ციფრები. თავდაპირველად მათში არ არსებობდა ციფრი **0.** რიცხვებში ნულების შემცველი თანრიგებს შეუვსებელს ტოვებ-

დნან, ე. ი. „პრობელებს“ გამოიყენებდნენ.

თანამედროვე არითმეტიკისა და ალგებრის შექმნა დაკავშირებულია **ჰორეზმში** *(უზბეკეთის ისტორიულ ოლქი)* ***IX*** საუკუნეში მცხოვრები დიდი არაბი მათემატიკოსის, ასტრონომის, გეოგრაფისა და ისტორიკოსის **აბუ აბდულაჰ** (ან აბუ ჯაფარ) **მუჰამედ იბნ მუსა ალ-ჰორეზმის** (*არაბ.* الخوارزمی مو سی بـ ن محمد ابو عـ بد الله; *დაახლ. 783 - დაახლ. 850*) სახელთან.

ისტორიული წყაროების თანახმად **ალ-ჰორეზმმა** ოცზე მეტი სამეცნიერო ნაშრომი შექმნა. ჩვენამდე მხოლოდ ათმა ნაშრომმა მოაღწია, რომელთაგანაც მხოლოდ ორია მათემატიკური ხასიათის. ესენია: **„წიგნი ინდური თვლის შესახებ“** (*არაბ. „ფი ჰიტაბ ალ-ჰინდ“*) და **„მოკლე წიგნი აღდგენისა და დაპირისპირების შესახებ“** *(არაბ. „ალ-ჰიტაბ ალ-მუჰტასაბ ფი ჰისაბ ალ-ჯაბარ ვა ალ-მუქაბლა“).* პირველმა მათგანმა **ათობით არითმეტიკას** დაუდო სათავე, მეორემ კი იგი განაზოგადა. მოგვიანებით განზოგადებულმა არითმეტიკამ **სასკოლო ალგებრის** სახელწოდება მიიღო.

● მათემატიკის მეცნიერების მნიშვნელოვანი ნაწილის - **არითმეტიკის** თვისობრივი ცვლილების სფეროში მნიშვნელოვანი როლი შეასრულა **ალ-ჰორეზმის** მიერ ***IX*** საუკუნეში დამუშავებულმა არითმეტიკულმა ტრაქტატმა **„წიგნი ინდური თვლის შესახებ“**. იგი დიდი ხნის განმავლობაში დაკარგულად ითვლებოდა, მაგრამ **1857** წელს **კემბრიჯის უნივერსიტეტის** ბიბლიოთეკაში იპოვნეს **XII** საუკუნეში შესრულებული მისი ლათინური თარგმანი სათაურით „***Algorizmi de numero Indorum***“. მასში არაა შენარჩუნებული **ალ-ჰორეზმისეული** ორიგინალური ტექსტის ყველა ნიუანსი (მაგალითად, გამოტოვბულია კვადრატული ფესვის ამოღების წესი), მაგრამ აღწერილია ოთხი არითმეტიკული ოპერაციის (შეკრების, გამოკლების, გამრავლებისა და გაყოფის) შესრულების წესები. ისინი პრაქტიკულად არ განსხვავდება დღეს გამოყენებული წესებისაგან. ამ წიგნის ლათინური თარგმანის პირველი სტრიქონი იწყება სიტყვებით «**Dixit Algorizmi**» - **„თქვა ალგორიზმა“** (*აქ „**ალგორიზმი**“ სახელ **ალ-ჰორეზმის** ლათინიზებული შესატყვისია*). ძალიან სწრაფად ავტორის სახელი **ალ-ჰორეზმი** საზოგადო სახელ „**ალგორიზმად**“ გარდაიქმნა. იგი თავდაპირველად აღნიშნავდა **არითმეტიკას**, შემდეგ კი გარკვეულ წესზე დაქვემდებარებული გამოთვლათა ნებისმიერ სისტემას. **1957** წელს გამოცემულ ინგლისური ენის ავტორიტეტულ ლექსიკონში „**Webster's New World Diction-**

ar“ (***2022*** *წლიდან მას უშვებს ინგლისურ-ამერიკული საგამომცემლო კომპანია HarperCollins Pub-lishers LLC*) სიტყვა „**ალგორითმი**“ მოძველებულ ტერმინად იწოდება და განმარტებულია როგორც „არაბული ციფრების დახმარებით არითმეტიკული ოპერაციების შესრულება“. ასე შემოვიდა ჩვენს ცხოვრებაში ტერმინი „**ალგორითმი**“, რომელიც შემდეგ შეუმჩნევლად მათემატიკიდან კიბერნეტიკაში გადაბარგდა. გარდა ამისა, რადგან **ინდოეთში** შექმნილ ციფრებს ევროპა არაბულ ენაზე შექმნილი **ალ-ჰორეზმი**ს ნაშრომით გაეცნო, ამიტომ მათ **არაბულ ციფრები** უწოდეს და დღესაც ამ სახელით მოიხენიებენ მთელ მსოფლიოში.

არაბული ციფრებით რიცხვების გამოსახვის წესმა ევროპაში თავდაპირველად აღიარება ვერ პოვა. **1299** წელს იტალიურ **ფლორენციაში** გამოვიდა კანონი, რომელიც კრძალავდა არაბული ციფრების გამოყენებას, მაგრამ რადგან ვაჭრებმა დაიწყეს არაბული ციფრების ფართო გამოყენება, ამიტომ **XVI** საუკუნეში ჯერ ევროპში, და შემდეგ, მთელი მსოფლიოში, ამ ციფრების გამოყენება დაიწყეს. თავად დისციპლინას კი „**ალგორითმის**“ ნაცვლად „**არითმეტიკა**“ (*ძვ. ბერძ. ἀριθμητική, arithmētikḗ - მიღებულია ἀριθμός, arithmós-დან: „რიცხვი“*) უწოდეს. ვინაიდან მასში რიცხვები გამოსახვისათვის თვლის ათობითი სისტემაა გამოყენებული, ამიტომ იგი ფაქტობრივად „**ათობითი არითმეტიკაა**“. დაწყებით სკოლაში ჩვენ მათემატიკას **ალ-ჰორეზმის** მიერ შემნილი **ათობითი არითმეტიკის** მეშვეობით „ვეზიარეთ“.

დღევანდელი შინაარსით ტერმინ **„ალგორითმის**“ გამოყენება **ციფრული ელექტრული მანქანის** (**კომპიუტერის**) გამოჩენის შემდეგ დაიწყეს. ამ ტერმინით აღინიშნება გარკვეული პროცესის ორგანიზებისათვის საჭირო მოქმედებების ერთობლიობა. აქ იგულისხმება არა მარტო გარკვეული არითმეტიკული ამოცანის, არამედ ნებისმიერი სახის ამოცანის გადაწყვეტის პროცესი. ჩვენთვის დღეს ტრივიალურად ითვლება ისეთი გამონათქვამები, როგორიცაა „ქუჩაზე გადასვლის ალგორითმი“, „გარკვეული კერძის მომზადების ალგორითმი“, „კონკრეტული სამუშაოს შესრულების ალგორითმი“, „სარეცხი მანქანის გამოყენების ალგორითმი“ და ა. შ.

● **ალ-ჰორეზმი** არ დაკმაყოფილებულა მარტო ინდოეთში შექმნილი ციფრების პოპულიზებითა და მათი საშუალებით ჩაწერილ რიცხვებზე შესასრულებელ ოპერაციათა წესების (ალგორითმების) ფორმირებით. მეორე მათემატიკურ ტრაქტატში **„მოკლე წიგნი აღდგენისა**

და დაპირისპირების შესახებ“ (*არაბ. „ალ-ჰიტაბ ალ-მუჰტასაბ ფი ჰისაბ* ***ალ-ჯაბარ*** *ვა ალ-მუქაბლა“*) მან ჩამოაყალიბა ნებისმიერი ბუნების სიმრავლის ელემენტებზე ამ ოპერაციების ჩატარების წესები, რის მეოხებითაც განაზოგადა შეკრებისა და გამრავლების ჩვეულებრივი ოპერაციები. ამ ნაშრომმა უმნიშვნელოვანესი როლი ითამაშა მათემატიკის ისტორიაში. მან საფუძველი ჩაუყარა მათემატიკის კიდევ ერთ ნაწილს **ალგებრას**, რომლის სახელწოდება აღნიშნული ტრაქტატის სათაურში არსებული სიტყვა „**ალ-ჯაბარიდან**“ იქნა მიღებული.

არსებული განმარტების ძალით **ალგებრა** მათემატიკის ნაწილია, რომელშიც არითმეტიკული და ფორმალური მანიპულაციები სრულდება არა კონკრეტულ რიცხვებზე, არამედ აბსტრაქტულ სიმბოლოებზე, ე. ი. იგი ფაქტობრივად არითმეტიკის განზოგადების შედეგადაა მიღებული. მას ევროპა და მთელი მსოფლიო **ალ-ჰორეზმის** ზემოთ აღნიშნული ტრაქტატის მეშვეობით ეზიარა. არსებობს ლათინურ ენაზე მისი ორი თარგმანი. პირველი მათგანი **1140** წელს შეასრულა ინგლისელმა მათემატიკოსმა **რობერტ ჩესტერელმა** *(ინგ. Robert of Chester; წლები უცნობია*). მისი სახელწოდებაა „წიგნი ალგებრისა და ალ-მუმუკაბალეს შესახებ“ და შენახულია კემბრიჯის უნივერსიტეტის ბიბლიოთეკაში. მეორე თარგმანი ეკუთვნის **იოან სევილიელს** (*ესპ. Juan el Sevillano; ლთJohannes Hispaniensis, Johannes Toletanus; დაახლ1100 - 1180*).

ნაშრომი შედგება სამი ნაწილისაგან. პირველ (თეორიულ) ნაწილში განხილულია პირველი და მეორე ხარისხის განტოლებები, დანარჩენ ორ ნაწილში კი ნაჩვენებია ალგებრის პრაქტიკული გამოყენებები. სიტყვა „**ალ-ჯაბარ**“ („შევსება“) ნიშნავდა უარყოფითი წევრის გადატანას განტოლების ერთი ნაწილიდან მეორეში, ხოლო „**ალ-მუკაბალა**“ („დაპირისპირება“) - განტოლების ორივე ნაწილში არსებული ურთიერთტოლი წევრების შეკვეცას.

1.2.3. ორობითი არითმეტიკის შექმნა

ალ-ჰორეზმის მერ **IX** საუკუნეში დამუშავებულ **ათობით არითმეტიკას** ევროპა **XII** საუკუნეში გაეცნო. ამის შემდეგ დაახლოებით სამი საუკუნე გახდა საჭირო იმისათვის, რომ იგი ევროპასა და დანარჩენ მსოფლიოში დამკვიდრებულიყო. დაახლოებით ასეთი ბედი ერგო მეორე გენიოსის, **გოტფრიდ ლაიბნიცის** მიერ შექმნილ მეორე არითმეტიკას - **ორობით არითმეტიკას.** მისი ჭეშმარიტი მნიშვნელობა კაცობრიობამ მეოცე

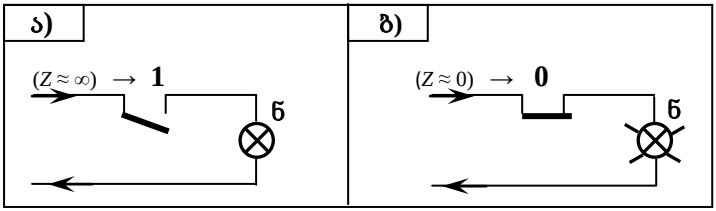
საუკუნის შუა პერიოდში გაიაზრა, როდესაც **მანქანური ენის** შესაქმნელად ორობითი რიცხვების გამოყენება დასჭირდათ.

ორობითად წოდებულ თვლის სისტემას ერთ-ერთი უძველთაგანი სისტემაა. ძალიან არასრულყოფილი ფორმით იგი გვხვდება **ავსტრალიისა** და **პოლინეზიის** ტომებში. დანომვრის ორობითი ხერხის გამოგონებას მიაწერენ ჩვენს წელთაღრიცხვამდე მეოთხე ათასწლეულში მცხოვრებ ჩინელ იმპერატორს **ფო გი**-ს. დაისმის კითხვა: ვინ გამოიგონა თვლის ორობით სისტემაში არითმეტიკული ოპერაციების (შეკრების, გამოკლების, გამრავლებისა და გაყოფის) წესები? დღეისთვის უტყუარადაა დამტკიცებული, რომ ასეთი პატივი ერგო **გოტფრიდ ლაიბნიცს,** რომელმაც **1697** წელს შექმნა **ორობითი არითმეტიკა**. საკუთარმა აღმოჩენამ იგი იმდენად აღაფრთოვანა, რომ **ფო გის** მიმართ მადლიერების ნიშნად დაამზადა ოქროს სპეციალური მედალი, რომელზეც ნატურალური რიცხვების საწყისი მწკრივის ორობითი გამოსახულებაა დატანილი. მეცნიერების ისტორიაში შეიძლება ეს ერთადერთი შემთხვევაა, როდესაც უმაღლესი პატივი ერგო არა კონკრეტულ ადამიანს, არამედ მათემატიკურ აღმოჩენას!

როგორც წინა თავში აღვნიშნავდით, **გოტფრიდ ლაიბნიც** თავის არითმეტიკულ მანქანის შექმნაზე **1673-1710** წლების განმავლობაში მუშაობდა. იგი ამ პერიოგში არ ივიწყებდა მის მიერ დამუშავებული ორობით სისტემასაც. **1679** წლის **15** მარტით დათარიღებულ ლათინურ ენაზე შესრულებულ ხელნაწერში იგი განმარტავდა თუ როგორ შეიძლებოდა თვლის ორობით სისტემაში გამოთვლების, კერძოდ, გამრავლების ოპერაციის, შესრულება; მოგვიანებით მან ზოგადი სახით დაამუშავა **თვლის ორობით სისტემაში მომუშავე გამომთვლელი მანქანის პროექტს**. იგი წერს: „ამგვარი სახის გამოთვლა შეიძლება მანქანაზე შევასრულოთ. უეჭველია, რომ ეს ძალიან მარტივად და განსაკუთრებული დანახარჯების გარეშე შეგვიძლია შემდეგნაირად მოვახდინოთ: საჭიროა ქილაში იმგვარი **ნახვრეტები** გავაკეთოთ, რომ შეგვეძლოს მისი გაღება და დახურვა. უნდა გავაღოთ **1**-ის შესაბამისი, ხოლო დავხუროთ - **0**-ის შესაბამისი ნახვრეტები. ღია ნახვრეტებიდან ღარში ჩავადება პატარა ბურთულები, ხოლო დახურული ნახვრეტებიდან არაფერი არ ჩავარდება. ქილა იმგვარად უნდა გადანაცვლდეს სვეტიდან სვეტისაკენ, როგორც ამას მოითხოვს გამრავლება. ღარებით წარმოდგენილი იქნება სვეტები, ამასთანავე მანქანის ამუშავებამდე ვერცერთი ბურთულა რომელიმე ღარიდან

სხვა რომელიმე ღარში ვერ უნდა მოხვდეს ...“. შემდეგ იგი არაერთხელ დაუბრუნდა ოროობით არითმეტიკას თავის მრავალრიცხოვან წერილებსა და **1703** წელს შექმნილ ტრაქტატში „**Explicatition de l'Arithmetique Binari**“. გამომთვლელ მანქანაში თვლის ოროობითი სისტემის გამოყენების **ლაიბნიცისეული** იდეა თითქმის სამი საუკუნის განმავლობაში ყველასგან იყო დავიწყებული.

ზემოთ აღწერილი მოწყობილობის შემოთავაზებით **ლაიბნიცმა** გაასწრო თავის დროს: მან ფაქტობრივად შემოგვთავაზა **ელექტრომაგნიტური რელეს** მექანიკური ანალოგი. აღნიშნული რელე **1835** წელს გამოიგონა ერთ-ერთ უდიდეს ამერიკელ ფიზიკოსად მიჩნეულმა **ჯოზეფ ფრანკლინმა** (*იხგ. Joseph Henry; 1797 - 1878*). **2.1.** ნახაზზე მოყვანილია ამ რელეს კონტაქტით **ნ** ნათურის ჩართვის სქემა. კონტაქტი ორი შესაძლო, ჩამოშვებული (ნახ. **1.2,ა**) ან მიზიდული (ნახ. **1.2,ბ**), მდგომარეობიდან შეიძლება ერთ-ერთში იმყოფებოდეს. **პირველი მდგომარეობა**, როდესაც კონტაქტის ***Z*** წინაღობა ძალიან დიდია (ე. ი. $Z \approx \infty$), პირობითად აღვნიშნოთ ციფრით **1**, ხოლო **მეორე მდგომარეობა**, როდესაც კონტაქტის წინაღობა ძალიან მცირეა ($Z \approx 0$) - ციფრით **0** .



***ნახ. 1.2**. რელეს კონტაქტის გამოყენების მაგალითი*

განვიხილოთ ორი შემთხვევა:

პირველი შემთხვევა. დავუშვათ, რომ **ელექტრომაგნიტური რელეს** კონტაქტის მდებარეობა **1** ემთხვევა ლაიბნიცისეული **„მექანიკური რელეს“** ნახვრეტის მდგომარეობა **1**-ს. ამ დროს:

ა) ელექტრომაგნიტური რელეს **კონტაქტი გათიშულია (ნახ. 1.2,ა)**, ამიტომ მისი წინაღობა უსასრულოდ დიდია, კონტაქტი დენს (მიმართულად მოძრავ ელექტრონებს) არ ატარებს და **ნ** ნათურა ჩამქრალია,

ბ) ლაიბნიცისეული „მექანიკური რელეს“ **ნახვრეტი დახურულია**, ბურთულას არ ატარებს და ბურთულა ღარში ვერ ვარდება.

მეორე შემთხვევა. დავუშვათ, რომ **ელექტრომაგნიტური რელეს** კონტაქტის მდებარეობა **0** ემთხვევა ლაიბნიცისეული **„მექანიკური რელეს“** ნახვრეტის მდგომარეობა **0**-ს. ამ დროს:

ა) ელექტრომაგნიტური რელეს **კონტაქტი შერთულია** (**ნახ. 1.2,ბ**), ამიტომ მისი წინაღობა თითქმის ნულის ტოლია, კონტაქტი დენს (მიმართულად მოძრავ ელექტრონებს) თითქმის უდანაკარგოდ ატარებს და **ნ** ნათურა ანთებულია,

ბ) ლაიბნიცისეული „მექანიკური რელეს“ **ნახვრეტი ღიაა**, იგი ბურთულას ატარებს და ბურთულა ღარში ვარდება.

ანალოგური თანაფარდობა შეიძლება დავამყაროთ **ლაიბნიცისეულ მექანიკურ რელესა** და გასაღების რეჟიმში მომუშავე **ტრანზისტორს** შორის. მისი დაკეტილი და ღია მდგომარეობები შეიძლება შესაბამისად შესაბამისად **1**-ითა და **0**-ით შეიძლება აღვნიშნოთ. უფრო მეტიც - ადვილად შეიძლება ერთმანეთს შევუსაბამოთ აღნიშნულ მექანიკური რელე და **ნეირონი**, რომელიც მხოლოდ ორ, აგზნებული და აღუგზნებელ, მდგომარეობიდან მხოლოდ ერთ-ერთში შეიძლება იმოფებოდეს. ერთ-ერთ მათგანს თუ აღვნიშნავთ ციფრ **1**-ით, მაშინ მე-ორე ციფრ **0**-ით უნდა აღინიშნოს.

გამომთვლელ მანქანაზე მუშაობის დაწყებისას **ლაიბნიცს** საყრდენი გამოეცალა - **1673** წელს გარდაიცვალა მისი მფარველი **მაინცის არქიეპისკოპოსი** და იგი იმ პრობლემების პირისპირ აღმოჩნდა, რომელთა წინაშე უძლურნი აღმოჩნდნენ კაცობრიობის მრავალი გონიერი პიროვნებები. ეკონომიკური პრობლემების დასაძლევად **ლაიბნიცმა** გასაღების მიზნით საკუთარი გამომთვლელი მანქანების დამზადება დაიწყო. მის მთავარ საზრუნავად გადაიქცა უმოკლეს დროში კომერციულად მომგებიანი მანქანის აგებაზე ზრუნვა და **ლაიბნიცი** თანდათან დაშორდა საკუთარი გრანდიოზული ჩანაფიქრების რეალიზების პროცესს. არსებობს არც თუ ისე უსაფუძვლო მოსაზრება იმის შესახებ, რომ აღნიშნული სამწუხარო გარემოება რომ არ შექმნილიყო, შეიძლებოდა **ლაიბნიცს** შეეძლო **ბებიჯზე** თითქმის **150** წლით ადრე შეექმნა მსოფლიოში პირველი კომპიუტერი. მიუხედავად ამისა, **ლაიბნიცმა** მაინც შეძლო შესანიშნავი მემკვიდრეობის დატოვება. კომპიუტერული ტექნოლოგიების დამუშავების სფეროში, სამწუხაროდ, მისი ღვაწლი ჯერ კიდევ სათანადოდ არაა შეფასებული.

1.2.4. არითმეტიკების შექმნის ფორმალური მეთოდი

ათობითი არითმეტიკის შექმნაში **ალ-ჰორეზმის**, ხოლო ორობითი არითმეტიკის შექმნაში - **გოტფრიდ ლაიბნიცის** შრომა შეგვიძლია უმაღლეს ხელოვნებას შევადაროთ. მათი მოღვაწეობის პერიოდში ჯერ კიდევ არ იყო დამუშავებული სხვადასხვა სახის არითმეტიბის შექმნის **ფორმალური მეთოდი**, რომლის არსებობამ არითმეტიკის შექმნის პროცესი რუტინულ საქმიანობად გადააქცია. ამ მეთოდის თანახმად თავდაპირველად უნდა განისაზღვროს **თვლის სასურველი სისტემა** და შემდეგ - მიღებული თვლის სისტემით წარმოდგენილ რიცხვებზე ჩასატარებელი შეკრების, გამოკლების, გამრავლებისა და გაყოფის **ბინარული ოპერაციები**.

თვლის სიტემა (*ინგ. numeral system* ან *system of numeration*) არის საწერი ნიშნებით რიცხვების ჩაწერის სიმბოლური მეთოდი.

თვლის იდეალურ სისტემას უნდა შეეძლოს:

- წარმოადგინოს რიცხვთა სიმრავლეები (მაგალითად, ყველა მთელი ან ნამდვილი რიცხვების სიმრავლეები);
- თითოეულ რიცხვს მიანიჭოს უნიკალური სახე;
- ასახოს რიცხვთა ალგებრული და არითმეტიკული სტრუქტურები.

თვლის სისტემები იყოფა:

- პოზიციურ (*ინგლ. positional system, place-value notation*) სისტემებად;
- არაპოზიციურ სისტემებად;
- შერეულ სისტემებად.

თვლის პოზიციურ სისტემებში ერთსა და იმავე რიცხვის ჩანაწერში ერთსა და იმავე რიცხვით ნიშანს (ციფრს) თავისი განთავსების ადგილის (თანრიგის) მიერ დაკავებულ ადგილზე დამოკიდებულებით აქვს სხვდასხვა მნიშვნელობა;

ასეთი სისტემების რიცხვს მიეკუთვნება თვლის ათობითი სისტემა, რომელიც შექმნა **ალ-ჰორეზმიმ** და ოროობითი სისტემა, რომლის ავტორია **გოტფრიდ ლაიბნიცი.**

თვლის პოზიციური სისტემა მთელი დადებითი $b > 1$ რიცხვით განისაზღვრება, რომელსაც თვლის სისტემის **ფუძე** ეწოდება. თვლის სისტემას, რომელშიც ფუძედ $b > 1$ რიცხვი გამოიყენება, ეწოდება თვლის ***b***-ობითი სისტემა. *მაგალითად, **b** = **2; 3; 10**-ის შემთხვევებში შესაბამისად გვაქვს თვლის ოროობითი, სამობითი და ათობითი სისტემები.*

უნიშნო მთელი x რიცხვი თვლის b-ობით სისტემაში გამოისახება b რიცხვის სასრული წრფივი კომბინაციის სახით: $x = \sum_{k-0}^{n-1} a_k \ b^k$, სა -
-დაც a_k არის ციფრებად წოდებული მთელი რიცხვები, რომლებიც $0 \le a_r \le (b-1)$ უტოლობას აკმაყოფილებს.

ასეთ გამოსახვისას თითოეულ ბაზისურ b^k ელემენტს **თანრიგი** (პოზიცია) ეწოდება. თანრიგებისა და მათი შესაბამისი ციფრების უფროსობა თანრიგის k ნომრით (ხარისხის მაჩვენებლის მნიშვნელობით) განისაზღვრება. თვლის b-ობით სისტემაში n რაოდენობის თანრიგების საშუალებით შეგვიძლია 0-დან $(b^n - 1)$-მდე დიაპაზონში არსებული b^n რაოდენობის სხვადასხვა მთელი რიცხვი ჩავწეროთ.

სხვადასხვაგვარი გაგება თუ არ წარმოიშობა (*თითოეული ციფრი უნიკალური თუ წერითი ნიშნებითაა გამოსახული*), მაშინ x რიცხვი მარცხნიდან მარჯვნივ თანრიგების კლებადი უფროსობით განთავსებული მისი b-ური ციფრების მიმდევრობის სახით ჩაიწერება:

$$x = a_{n-1}\, a_{n-2} \dots a_0;$$

მაგალითად, *ხუთასოცდაჩვიდმეტი* თვლის ათობით სისტემაში ასე ჩაიწერება: $537 = 5 \cdot 10^2 + 3 \cdot 10^1 + 7 \cdot 5^0$.

დღეისთვის გამოყენებადი თვლის ყველაზე ხშირად პოზიციური სისტემებია:

- ორობითი სისტემა, სადაც $b = 2$ (*გამოიყენება დისკრეტულ არითმეტიკაში, კომპიუტერულ მეცნიერებაში, დაპროგრამებაში*);
- სამობითი სისტემა, სადაც $b = 3$ *(პერსპექტულია კომპიუტერულ მეცნიერებაში მისი გამოყენება*);
- რვაობითი სისტემა, სადაც $b = 8$ *(გამოიყენება მიკროპროცესორულ ტექნიკაში*);
- ათობითი სისტემა, სადაც $b = 10$ *(საყოველთაოდ გამოყენებადი სისტემა*);
- თორმეტობითი სისტემა, სადაც $b = 12$ (*დუჟინობით თვლა*);
- თექვსმეტობითი სისტემა, სადაც $b = 16$ *(გამოიყენება კომპიუტერულ მეცნიერებაში, დაპროგრამებაში*);
- ოცობითი სისტემა, სადაც $b = 20$;
- სამოცობითი სისტემა, სადაც $b = 60$ *(გამოიყენება დროის, კუთხეების და კოორდინატების, გრძედებისა და განედების გაზომვებისათვის*).

პოზიციურ სისტემებში თვლის სისტემის ფუძის სიდიდის გაზრდით რიცხვების ჩანაწერში მცირდება ციფრების რაოდენობა.

1.3. დაპროგრამებადი კალკულატორიდან - კომპიუტერამდე

კომპიუტერის შექმნისკენ მიმართული სამუშაოები მეორე მსოფლიო ომის პერიოდში განახლდა. ამერიკის შეერთებული შტატების სამხედრო საზღვაო ფლოტმა გადაწყვიტა **ჩარლზ ბებიჯის** თეორიული შრომების პრაქტიკული რეალიზაცია მოეხდინა და საკუთარი პრობლემები გადაწყვეტისათვის აეგო გამომთვლელი მანქანა. მან გენერალურ კონტრაქტორად აირჩია ფირმა ***IBM***. უკანასკნელმა ქვეკონტრაქტორად შეარჩია ხუთი გამომთვლელისაგან შემდგარი ჯგუფი, რომელსაც ხელმძღვანელობდა სამხედრო საზღვაო ფლოტის მე-2 რანგის კაპიტანი **ჰოვარდ ეიკენი** (*ინგ. Howard Hathaway Aiken; 1900 – 1973*). ამის წყალობით იგი ამერიკული კომპიუტერთმშენებლობის პიონერი გახდა. სამუშაოს დასრულების შემდეგ გამომთვლელი მანქანა ამერიკის სამხედრო ფლოტს გადაეცა და იქ გამოიყენებოდა მეორე მსოფლიო ომის დასკვნით ეტაპზე.

ჰოვარდ ეიკენის მანქანა ამერიკის შეერთებულ შტატებში შექმნილი ერთ-ერთი პირველი გამომთვლელი მამქანაა. კორპუსზე დატანილი წარწერის თანახმად მისი სრული სახელწოდებაა „Aiken-IBM Automatic Sequence Controlled **Calculator Mark** I“, ანუ „მიმდევრობითი მართვის ავტომატური Aiken-IBM **კალკულატორი Mark I**“. შემოკლებით მას „**Mark I**“-ს უწოდებენ. ტერმინი კალკულატორი მიღებულია ლათინური სიტყვისაგან „**Calculatio**“, რაც „დათვლას“, „ანგარიშს“ ნიშნავს. ტესტების წარმატებით დასრულების შემდეგ „**Mark I**“ გადაიტანეს ჰარვარდის უნივერსიტეტში და ფორმალურად **1944** წლის **7** აგვისტს ამუშავდა. მის აგებაზე დაიხარჯა **500** ათასი დოლარი, იგი შეიცავს დაახლოებით **765000** დეტალს (ელექტრომაგნიტურ რელეებს, გადამრთველებს და ა. შ.); მისი სიგრძე თითქმის **17** მეტრია, სიმაღლე **2,5** მეტრს აღემატება და იწონის **4,5** ტონას (**ჰარვარდის უნივერსიტეტში** რამდენიმე ათეული კვადრატული მეტრი ფართობი უკავია); ერთდროულად შეუძლია დაამუშაოს **23** ათობითი თანრიგიანი **72** რიცხვი, წამში ასრულებს შეკრება-გამოკლების სამ ოპერაციას, გამრავლების ოპერაციის შესასრულებლად სჭირდება **6**, ხოლო გაყოფის ოპერაციის შესასრულებლად - **15,**3 წამი, ლოგარითმებისა და

ტრიგონომეტრიული ფუნქციების გამოთვლაზე კი ერთ წუთზე მეტ დროს ხარჯავს.

„**Mark I**“, ფაქტობრივად, ჩვეულებრივი **დაპროგრამებადი კალკულატორი** იყო, რომელსაც არითმომეტრებით აღჭურვილი დაახლოებით **20** ოპერატორის შეცვლა შეეძლო. დაპროგრამების უნარის გამო ზოგიერთები მას რეალურად ფუნქციონირებად **კომპიუტერსაც** უწოდებენ. „**Mark I**“ ვერ ასრულებდა პირობით გადასვლებს, რის გამოც თითოეული პროგრამის ჩასაწერად საკმაოდ გრძელ ლენტურ რულონს მოითხოვდა, ციკლების ორგანიზებისათვის კი წასაკითხი ლენტის დასაწყისისა და დაბოლოების ერთმანეთზე გადაწებება იყო საჭირო. ასეთი შეზღუდულობების მიუხედავად, „**Mark I**“ იყო პირველი სრულად ავტომატიზებული მანქანა, რომელიც გამოთვლის პროცესში ადამიანის ჩარევას არ საჭიროებდა.

1.3. 1. ჩარლზ ბებიჯის იდეის პრაქტიკულად რეალიზება

ჰოვარდ ეიკენის ჯგუფის მიერ დამუშავებული „**Mark I“** არ წარმოადგენდა გამონაკლისს. საწყის პერიოდში შექმნილ გამომთვლელი მანქანებიც ფაქტობრივად დაპროგრამებადი კალკულატორები იყო. ასე მაგალითად, პენსილვანიის უნივერსიტეტში **1946** წელს ამერიკელი მეცნიერების **ჯონ ადამ პრესპერ ესკერტ-უმცროსისა** (*ინგ. John Adam Presper Eckert; 1919 - 1995*) და **ჯონ უილიამ მოკლის** (*ინგ. John William Mauchly; 1907 - 1980*) მიერ შექმნილი გამომთვლელი მანქანის აბრევიატურა **ENIAC** გაიშიფრება როგორც „**E**lectronic **N**umerical **I**ntegrator and **C**alculator“, ანუ **„ელექტრონული ციფრული ინტეგრატორი და კალკულატორი**“; ასევე **1949** წელს კემბრიჯის უნივერსიტეტში (ინგლისი) ბრიტანელი მეცნიერის **მორის ვინსენტ უილკსის** (*ინგ. Maurice Vincent Wilkes; 1913 - 2010*) ხელმძღვანელობით მომუშავე ჯგუფის მიერ აგებული გამომთვლელი მანქანის აბრევიატურა **EDSAC** გაიშიფრება „***E**lectronic **D**elay **S**torage **A**utomatic **C**alculator“*, ანუ **„დაყოვნებით შემნახავი ელექტრონული ავტომატური კალკულატორი“**.

სამივე განხილულ შემთხვევაში კომპიუტერის ნაცვლად მიღებული მანანქანები **დაპროგრამებად კალკულატორებად** იწოდებოდა. დაპროგრამებად კალკულატორსა და კომპიუტერს შორის არსებული განსხვავების ილუსტრირებისათვის განვსაზღვროთ დაპროგრამება-

დი კალკულატორის ფუნქციური დანიშნულება. ამ მიზნით გავიხსენოთ ტერმინ „პარამეტრის“ მნიშვნელობა.

პარამეტრი (*ძვ. ბერძ.* παραμετρέω – „*გამზომავი*“; *სადაც* <u>*παρά*</u>*:* „გვერდით“, „დამხმარე“, „დაქვემდებარებული“; და <u>*μέτρον*</u>: „გაზომვა“) არის სიდიდე:

1. რომლის მნიშვნელობების მიხედვით ერთმანეთისაგან განასხვავებენ რაიმე სიმრავლის ელემენტებს;

2. რომელიც მუდმივია მოცემული მოვლენის ან ამოცანის ფარგლებში, მაგრამ იცვლის საკუთარ მნიშვნელობას სხვა მოვლენაში ან ამოცანაში გადასვლისას.

ზოგჯერ **პარამეტრებს** სხვა სიდიდეებთან (ცვლადებთან) შედარებით ნელად ცვლად სიდიდეებსაც უწოდებენ.

პარამეტრი ობიექტის ან სისტემის გაზომვადი თვისება ან მაჩვენებელია; სისტემის პარამეტრის გაზომვით მიიღება რიცხვი ან პარამეტრის მნიშვნელობა, თავად სისტემა შეიძლება იმ პარამეტრების სიმრავლედ წარმოვიგინოთ, რომელთა გაზომვასაც სისტემის ქცევის მოდელირებისათვის საჭიროდ მივიჩნევთ.

სხვადასხვა სფეროში საქმიანობის პერიოდში ადამიანებს ხშირად გარკვეული **პარამეტრების** მნიშვნელობების ცოდნა სჭირდებათ. ამის გამო ისინი სხვადასხვა, უმეტესწილად მათემატიკური, მეთოდების დახმარებით იწყებენ მათ გამოთვლას. გამოთვლების პროცესში სრულიად მოულოდნელად მათ წინაშე წინასწარ გაუთვალისწინებელი ისეთი პრობლემები წამოიჭრება, რომ მათი დაძლევისათვის ადამიანებს მეტად არაორდინალური სამუშაოების ჩატარება უხდებათ. ასეთი არაორდინალური პრობლემების სათავე გახდა დიდ ბრიტანეთში **1776** წლიდან გამომავალი „**საზღვაო ალმანახი**“ (იხ პირველი თავი). მასში შესატანი ლოგარითმული და ტრიგონომეტრიული ფუნქციების მნიშვნელობების გამოთვლისას იმდენ შეცდომები იქნა დაფიქსირებული, რომ მათი აღმოფხვრის აუცილებლობამ **ჩარლზ ბებიჯი** აიძულა შედგომოდა **„სხვაობითი მანქანის“** აგებას, რომლის პროცესში მას „**ანალიზური მანქანის**“ შექმნის იდეა დაებადა. არსებული დაბალი ტექნოლოგიური დონის გამო **ჩარლზ ბებიჯმა** ვერ შეძლო იდეისათვის ხორცი შეესხა, მაგრამ თავისი ტალანტისა და არნახული შრომისმოყვარეობის წყალობით, შეძლო დაეტოვებინა მეტად მნიშვნელოვანი თეორიული მასალა, რომლის მიხედვით შედგენილი „**ანალიზური მანქანის**“ გამარტივებული ბლოკ-სქემა **1.1** ნახაზზეა

მოყვანილი. მისი ერთადერთი მოქმედი მუშა ელემენტია **არითმეტიკული მოწყობილობა** და ამიტომ იგი **მხოლოდ არითმეტიკული ამოცანების გადაწყვეტაზეა** ორიენტირებული. არითმეტიკის მიზანია შეისწავლოს მხოლოდ ციფრებით გამოსახული რიცხვების თვისებები და მათზე ჩასატარებელი ოპერაციები. ე.ი. შეასრულოს თვლასთან და შეჯამებასთან დაკავშირებული მოქმედებები. ლათინური სიტყვა **Calculatio** წმინდა არითმეტიკული ოპერაციების აღსანიშნავად გამოიყენება (იხ. გვ. **29**). მისგან ნაწარმოები ტერმინი „**კალკულატორი**“ დაერქვა მხოლოდ არითმეტიული ოპერაციების შემსრულებელ მანქანას. განასხვავებენ **არადაპროგრამებად** და **დაპროგრამებად** კალკულატორებს.

არადაპროგრმებადი კალკულატორი გამოიყენება ცალკეული არითმეტიკული ოპერაციების შესასრულებლად. პირველი კალკულატორებია **XVII** საუკუნეში **ვილჰელმ შიკარდისა** და **ბლეზ პასკალის** მიერ შექმნილი უმარტივესი არითმეტიკული მანქანები. მათგან განსხვავებით „**Mark 1**“-ს, **ENIAC**-სა და **EDSAC**--ს შეუძლიათ სპეციალური პროგრამების მეშვეობით, ადამიანის ჩაურევლად, შეასრულოს **პარამეტრების მნიშვნელობების გამოსათვლელად** საჭირო ყველა არითმეტიკული ოპერაცია. ამიტომ მათ **დაპროგრამებადი კალკულატორები** ეწოდება.

დაპროგრამებადი კალკულატორების შექმნით ხორცი შეესხა **ჩარლზ ბებიჯის** იდეას: შესაძლებელი გახდა ნებისმიერი სახის არითმეტიკული ოპერაციების დასახული სიზუსტითა და უშეცდომოდ შესრულება და გამოირიცხა მცდარი **პარამეტრებიანი** ნებისმიერი სახის კალენდრის გამოცემის ალბათობა. ამგვარად, **ჩარლზ ბებიჯის** მიერ **XIX** საუკუნის შუა პერიოდში დასახული მიზანი დაპროგრამებადი **კალკულატორების შექმნა იყო**. ეს მიზანი მხოლოდ **XX** საუკუნის **40**-იან წლების დასაწყისში იქნა მიღწეული.

ჩარლზ ბებიჯისაგან განსხვავებით, **გოტფრიდ ლაიბნიცს XVII** საუკუნეში არა ცალკეული **პარამეტრების**, არამედ **ლოგიკური დასკვნების** ავტომატურად გამოთვლელი მანქანის შექმნის იდეა დაებადა. მისი რეალიზებისათვის **ალ-ჰორეზმის** მიერ დამუშავებული **ათობითი ალგებრა** არაა საკმარისი: მას ორობითი არითმეტიკის განზოგადებით მიღებული **ლოგიკური ალგებრაც** უნდა დაემატოს. **ორობითი არითმეტიკის** „მამა“ კი **ლაიბნიცია**. კომპიუტერის შექმნაში მისი დიდი ღვაწლის ნათლად წარმოჩნენა ამ წიგნის ერთ-ერთი მიზანია.

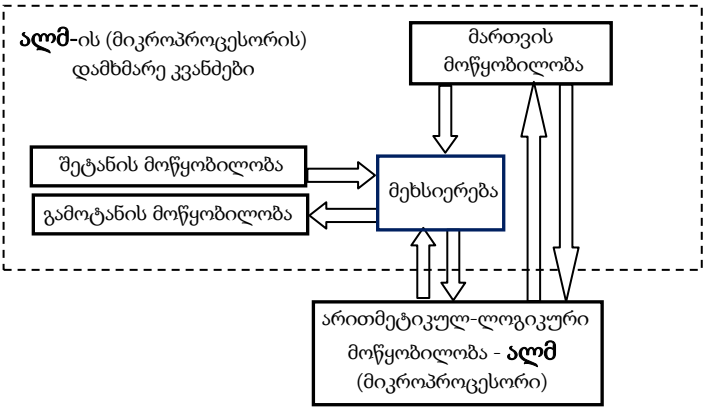
1.3. 2. „ბატონო, მოდით გამოვთვალოთ!“ ანუ კალკულატორიდან - კომპიუტერამდე!

ინგლისელ გამომგონებლის **რობერტ ჰუკისათვის** (*ინგ. Robert Hooke; 1635-1703*) **1676** წელს მიწერილ წერილში **ისააკ ნიუტონი** (*ინგ. Isaac Newton; 1642 - 1727*) წერდა: „სხვებზე შორს თუ ვიხედებოდი, იმიტომ რომ გიგანტების მხრებზე ვიდექი“. **გოტფრიდ ლაიბნიცისათვის** ერთ-ერთი ასე გიგანტი იყო ძველი საბერძნეთის უდიდესი ფილოსოფოსი **არისტოტელე** (*384 – 322 ჩვენს წელთაღრიცხვამდე*), რომელმაც საკუთარ ტრაქტატებში დაწვრილებით გამოიკვლია ლოგიკის ტერმინოლოგია, დეტალურად განიხილა დასკვნის მიღების თეორია, აღწერა მთელი რიგი ლოგიკური ოპერაციები და **ჩამოყალიბა აზროვნების ძირითადი კანონები**, კერძოდ, წინააღმდეგობრიობისა და გამრიცხული მესამის კანონები. **ლოგიკის მეცნიერების** ჩამოყალიბებაში **არისტოტელეს ღვაწლი** იმდენად დიდია, რომ ზოგიერთი მეცნიერი ლოგიკას **არისტოტისეულ ლოგიკის** სახელწოდებითაც მოიხსენიებენ.

განსაკუთრებით ხაზგასასმელია ის, რომ **არისტოტელემ** პირველმა შენიშნა ლოგიკასა და მათემატიკას (რომელსაც იგი არითმეტიკას უწოდებდა) შორის არსებული მჭიდრო კავშირი. იგი შეეცადა ერთმანეთთან შეერთებინა მის მიერ დაფუძნებული ლოგიკა და მათემატიკა. უფრო სწორად, იგი შეეცადა აზროვნება, ანუ **დასკვნის გამოტანის პროცესი** საწყისი დებულებების საფუძველზე ჩატარებულ გამოთვლებამდე დაეყვანა. ერთ-ერთ საკუთარ ტრაქტატში, **არისტოტელე** მჭიდროდ მიუახლოვდ მათემატიკური ლოგიკის ერთ-ერთ განყოფილებას **მტკიცებულებათა თეორიას.**

არისტოტელეს შემდეგ ბევრი ფლოსოფოსი და მათემატიკოსი ავითარებდნენ ლოგიკის ცალკეული დებულებებს და თითქმის სახავდნენ თანამედროვე დისციპლინის - **გამოთქმების აღრიცხვის** კონტურებს, მაგრამ **მათემატიკური ლოგიკას** ყველაზე მჭიდროდ **XVII** მიუახლოვდა **გოტფრიდ ლაიბნიცი**. მან გვიჩვენა გვიჩვენა სრული გზა, რომლითაც შეგვეძლო **სრული გაურკვევლობის სამეფოდან** გადავსულიყავით **მათემატიკის სამეფოში,** სადაც აბსოლუტურად ზუსტად შევძლებდით განგვესაზღვრა ობიექტებსა და გამოთქმებს შორის არსებული მიმართებები. **ლაიბნიცი იმედოვნებდა**, რომ მომავალში ადამიანები უნაყოფო კამათში ძვირფასი დროის კარგვის ნაცვლად ხელ-

ში აიღებდნენ ფანქარსა და ქაღალდს, ან ცარცით ხელში მივიდოდნენ დაფასთან და ერთმანეთს თავაზიანად შესთავაზებდნენ: „**ბატონო, მოდით გამოვთვალოთ!**“ ამ დამოთქმის ინგლისური თარგმანი ასეთია: „**Gentlmen, let us COMPUTE !**“.



ნახაზი 1.3. თანამედროვე კომპიუტერის ბლოკ-სქემა

ლათინური სიტყვა **Computare**, თვლისა და შეჯერების გარდა გამოთვლის ოპერაციასაც აღნიშნავს, ხოლო ტერმინი „**გამოთვლა**,“ არითმეტიკული ოპერაციების შესრულების გარდა, მათემატიკური საშუალებებით რაიმეს დამტკიცებასაც, ე. ი. **ლოგიკური ოპერაციების** შესრულებასაც გულისხმობს.

ლოგიკა (*ძვ. ბერძ. Λογική – „საუბრისა და ფიქრის მეცნიერება და ხელოვნება“; მიღებულია ბერძნული სიტყვისაგან λόγος - „სიტყვა“, „აზრი“, „მსჯელობა“ „ცნება“*), ფართო გაგებით, მატერიალური და იდეალური მოვლენებისა და საგნებისათვის ცხადად ან არაცხადად დამახასიათებელი გონივრულობა, რაციონალურობა, შინაგანი კანონზომიერება და თანამიმდევრობაა; ვიწრო გაგებით კი, ესაა მეცნიერება სწორად აზროვნების ფორმების, პრინციპებისა და კანონების, ცოდნის ზრდისა და მისი ჭეშმარიტობის პირობების შესახებ.

ლოგიკური ოპერაციები ეწოდება მოქმედებებს, რომელთა შედეგადაც უკვე არსებული ცნებებით იბადება ახალი ცნებები. უფრო ვი-

წრო გაგებით, ლოგიკური ოპერაციის ცნება გამოიყენება მათემატიკურ ლოგიკაში. **მათემატიკური ლოგიკა** მათემატიკის ნაწილია, რომელიც შეისწავლის მათემატიკურ აღნიშვნებს, ფორმალურ სისტემებს, მათემატიკური მსჯელობების მტკიცებადობას, მათემატიკური მტკიცებადობის ზოგად ბუნებას, გამოთვლადობას და მათემატიკის საფუძვლის სხვა ასპექტებს.

ჩარლზ ბებიჯის მიზანი თუ „**საზღვაო ალმანახის**“ პრობლემების გადასაწყვეტად საჭირო დაპროგრამებადი კალკულატორის შექმნა იყო, **გოტფრიდ ლაიბნიცის** მთელი ძალისხმევა ლოგიკური ფუნქციების შემსრულებული მათემატიკური მანქანის შექმნისაკენ იყო მიმართული. ასეთი მანქანის ფუნქციური შესაძლებლობები **დაპროგრამებადი კალკულატორების** ფუნქციურ შესაძლებლობებზე წარმოუდგენლად ფართოა და მას ადამიანისათვის დამახასიათებელი ბევრი ქცევის იმიტირების უნარიც აქვს. ასეთი პოტენციური შესაძლებლობიანი მანქანისათვის **დაპროგრამებადი კალკულატორის** სახელის დარქმევა წარმოუდგენელი უსამართლობა იქნებოდა და მას მოეძებნა **გოტფრიდ ლაიბნიცის** ზემოთ მოყვანილ ცნობილ გამოთქმაში გამოყენებული სიტყვა „**COMPUTE**“-დან ნაწარმოები სიტყვა **კომპიუტერი** !

კომპიუტერი შეგვიძლია მივიჩნიოთ დაპროგრამებადი კომპიუტერის ფუნქციური შესაძლებლობების გაფართოების შედეგად მიღებულ პროდუქტს, რადგან ფაქტობრივად:

კომპიუტერი = დაპროგრამებად კალკულატორს + ლოგიკა.

სამწუხაროდ თუ საბედნიეროდ, ადამიანის, როგორც იარაღის მკეთებელი ცხოველის, საქმიანობის თანამიმდევრობის განსაზღვრისას დომინანტურია **პრაქტიკული ამოცანების** გადაწყვეტა. ხოლო გონებრივი აღმაფრენის მეშვეობით ფორმირებული იდეები ხშირად მეორე პლანზე გადაიწევს ხოლმე. მართლაც, ისტორიულად კომპიუტერის შექმნის იდეა **გოტფრიდ ლაიბნიცს** XVII საუკუნეში, ხოლო **ჩარლზ ბებიჯს** დაპროგრამებადი კალკულატორის იდეა ორი საუკუნის დაგვიანებით - XIX საუკუნეში დაებადა. ასე რომ ისტორიული თანამიმდევრობის დაცვის შემთხვევაში ჯერ კომპიუტერი უნდა შექმნილიყო და შემდეგ - დაპროგრამებადი კალკულატორი. მაგრამ უშეცდომო „**საზღვაო ალმანახი**“ ადამიანებისათვის მეტად მნიშვნელოვან საქმიანობას - საზღვაო გადაადგილებას უქმნიდა საფრთხეს, ამიტომ ადამიანებმა პირველად ამ საფრთხის აღმოსაფხვრელად აუცილებელი „დაპროგრამებადი კალკულატორის“ სქემა დაამუშავა და შემ-

დეგ დაიწყო კომპიუტერის სქემის დამუშავებაზე მუშაობა. უნდა აღვნიშნოთ, რომ ორივე მიმართულებით წარმატებას დიდხანს არ დაუგვიანებია. დაპროგრამებადი კალკულატორების იდეის პრაქტიკული რეალიზაციიდან უმოკლეს ვადაში, კერძოდ, **1951** წელს, დაპროგრამებად კალკულატორ **ENIAC**-ის შემქმნელებმა **ჯონ მოჩლიმა** და **პრესპერ ეკერტმა 1927** წლიდან **1955** წლამდე ამერიკაში ფუნქციონირებად გამომთვლელი მანქანების მწარმოებელ კორპორაცია **Remington Rand**-ში შექმნეს „**UNIV**ersal **A**utomatic **C**omputer“ (**UNIVAC**), ანუ „უნივერსალური ავტომატური კომპიუტერი“. მომდევნო ექვსი წლის განმავლობაში აიგო მილიონი დოლარის ღირებულების ორმოცდაექვსი ასეთი კომპიუტერი, რომელთაგანაც ბოლო მათგანი **1970** წლამდე წარმაატებით მუშაობდა.

1950-იანი წლების დასაწყისში **UNIVAC** იმდენად პოპულარული იყო, რომ ფართო მომხმარებლი ნებისმიერ ელექტრონულ კომპიუტერს **UNIVAC**-ს უწოდებდა. ძალიან მალე **Remington Rand**-ს პირველობა **IBM**-მა ჩამოართვა. **1960**-იან წლებში **UNIVAC**-ი ექვს საუკეთესო გამომთვლელ მანქანათა ჩამონათვალში შედიოდა, ხოლო ბოლო ასეთი კომპიუტერი **1980** წელს აიგო!

„**დაპროგრამებად კომპიუტერის**“ ბლოკ-სქემაში თუ მუშა ელემენტად გამოყენებული იყო არითმეტიკული მოწყობილობა (იხ. **ნახ. 1.1**), კომპიუტერში მუშა ელემენტად გამოყენებული უნდა იყოს არითმეტიკული და ლოგიკური მოწყობილობების ინტეგრირებით მიღებული **ა**რითმეტიკულ-**ლ**ოგიკური **მ**ოწყობილობა, ანუ **ალმ**, (ნახ. **1.3**). ერთმანეთს თუ შევადარებთ **1.1** და **1.3** ნახაზებს, დავინახავთ, რომ დაპროგრამებად კალკულატორი და კომპიუტერი ერთმანეთისაგან მხოლოდ ერთი ნაბიჯითაა დაშორებული: საკმარისია დაპროგრამებად კალკულატორში არსებული არითმეტიკული მოწყობილობა **ალმ**-ით შევცვალოთ და კომპიუტერს მივიღებთ! ამ ერთი ნაბიჯის გადასადგმელად საჭირო გახდა ინგლისში ღარიბ ოჯახში დაბადებული იყო უნიჭიერესი ახალგაზრდა, რომელიც თვითგანათლებით გახდებოდა მათემატიკის პროფესორი და შექმნიდა უპრეცედენტო ალგებრას - **ლოგიკის ალგებრას**, რომელსაც დღეს მისი შემქმნელის - **ჯორჯ ბულის** (*ინგ. George Boole; 1815 - 1864*) საპატივცემლოდ **ბულის ალგებრას ვუწოდებთ.** მასზე დაწვრილ;ებით მომდევნო პარაგრაფში ვისაუბრებთ. მანამდე კი შევნიშნავთ, რომ არითმეტიკულ-ლოგი-

კური მოწყობილობა **XX** საუკუნის **70**-იან წლებამდე კომპიუტერის განუყოფელ ნაწილად ითვლებოდა. შემდგომში, დიდი ინტეგრალური სქემების კონსტრუირების წყალობით, იგი მობილურობის უნარიან მიკროსქემად - **მიკროპროცესორად** - გადაიქცა. **მიკროპროცესორული ტექნიკა** თანამედროვე ინფორმაციული სისტემების ასაგებ საელემენტო ბაზად გამოიყენება.

1.3. 3. ჯორჯ ბულის უჩვეულო ალგებრა

მათემატიკური ლოგიკის შექმნის ლაიბნიცისეული ოცნების ხორცშეხმის პრობლემებით **XIX** საუკუნის პირველ ნახევარში დაინტერესდა უსაზღვრო მიზანსწრაფულობით გამორჩეული ინგლისელი მათემატიკოსი **ჯორჯ ბული**. მისი მშობლების მატერიალური მდგომარეობა იმდენად მძიმე იყო (მამამისი მეწაღის საქმიანობით არჩენდა ოჯახს), რომ პატარა **ჯორჯმა** მხოლოდ ღარიბებისათვის განკუთვნილი დაწყებითი სკოლის დამთავრება შეძლო. მამის დასახმარებლად ოჯახის შენახვის მძიმე ტვირთი ტარება მან **16** წლის ასაკიდან აიღო თავის თავზე. გამოიცვალა რა რამდენიმე პროფესია, გარკვეული დროის გავლის შემდეგ მან გახსნა პატარა სკოლა, სადაც **9** წლის განმავლობაში ასრულებდა როგორც ხელმძღვანელის, ისე მასწავლებლის ფუნქციებს. იგი დიდ დროს უთმობდა თვითგანათლებას და მალე **მათემატიკითა და სიმბოლური ლოგიკის** იდეებით დაინტერტესდა. მან დაიწყო დაიწყო ალგებრულ განტოლებებთან დაკავშირებული მათემატიკური სტატიების ბეჭდვა. **1836** წელს იგი გაეცნო და დაუმეგობრდა ახალგაზრდა მათემატიკოსს **გრეგორი დუნკანს** (*ინგ. Gregory Duncan; 1813 - 1947*), რომელმაც **1836** წელს კემბრიჯის უნივერსიტეტთან დააარსა მათემატიკური ჟურნალი „**Cambridge Mathematikal Journal**". **გრეგორს** ძალიან ხიბლავდა **ბულის** სტატიები და მათ აქვეყნებდა საკუთარ ჟურნალში. **1843** წელს **ბულმ**ა მას მიაწოდა სტატია „**ანალიზის ზოგადი მეთოდი**" (*„A General Method"*), რომლის მოცულობა აღემატებოდა ჟურნალში გამოსაქვეყნებელი სტატიების მოცულობას. **გრეგორმა** იგი სათანადო გადაწყვეტილების მისაღებად გადასცა **ლონდონის სამეფო საზოგადოებას** (*Royal Society London*). საზოგადოებამ ამ სტატიის არა მარტო დაბეჭდვის ნებართვა გასცა, არამედ იგი, როგორც ბოლო სამი წლის განმავლობაში შემოსული საუკეთესო სამეცნიერო სტატია, სამეფო მედლით დაჯილდოვდა. ეს მაშინ როდესაც **ჯორჯ ბულს** არა მარტო საუნივერსიტეტო, არამედ საერთოდ სათანადო განათლებაც

არ ჰქონდა! ამით მოულოდნელობები არ დამთავრებულა. მა-ლე, კერძოდ, **1849** წელს ქალაქ **კორკში** (*Kork*) გაიხსნა სამეფო კოლეჯი (დღეს კორკის საუნივერსიტეტო კოლეჯი), სადაც თვითნასწავლმა **ჯორჯ ბულმა** დაიკავა **მათემატიკოსის პირველი პროფესორის** თანამდებობა!

კორკის კოლეჯში მუშაობისას **1849** წელს **ჯორჯ ბულმა** გამოაქვეყნა თავისი კაპიტალური **424** გვერდიანი ნაშრომი **„აზროვნების კანონების გამოკვლევა, რომელზეც დაფუძნებულია ლოგიკისა და ალბათობათა თეორიები“** („An Investigation of the Laws of Thought on Which are Founded the Mathematical Theories of Logic and Probabilities“**).** ამ ნაშრომით მან დააფუძნა მანამდე უცნობი ალგებრს მეცნიერება - „**ლოგიკის ალგებრა**“. დამფუძნებლის საპატივცემლოდ დღეს მას „**ბულის ალგებრის**“ სახელითაც მოვიხსენიებთ. ტრადიციულ მიდგომის ძალით **„ათობითი ალგებრის**“ (ანუ სასკოლო ალგებრის) მსგავსად ზოგიერთი ავტორი მას „**ორობითი ალგებრასაც**“ უწოდებს, მაგრამ, ზემოთ ნახსენები „უჩვეულობის“, გამო მეჩვენება, რომ ეს ტერმინი მთლად ზუსტად ვერ უნდა გამოხატავდეს სკოლის მეშვეობით ფართოდ რეკლამირებული ალგებრის არსს.

ცხრ. 1.2. „ლოგიკის ალგებრის“ ლოგიკური ოპერაციები

გამეორების ოპერაცია: $y = x$	პირსის ოპერაცია: $y = \overline{x_1 + x_2}$
ინვერსია: $y = \bar{x}$	Mod 2-ის ოპერაცია: $y = x_1 \oplus x_2$
კონიუნქცია: $y = x_1 x_2$	ეკვივალენტობა: $y = x_1 \sim x_2$
დიზიუნქცია: $y = x_1 + x_2$	იმპლიკაცია: $y = x_i \rightarrow x_j$
შეფერის ოპერაცია: $y = \overline{x_1 x_2}$	იმპლიკაციის უარყოფა: $y = \overline{x_i \rightarrow x_j}$

ათობითი არითმეტიკისა და მისი განზოგადებით მიღებული **ათობითი ალგებრის** ავტორი იყო **IX** საუკუნის უდიდესი მოაზროვნე და მათემატიკოსი **ალ-ჰორეზმი (*იხ. §1.2.2.*)**. ორივე დისციპლინაში ავტორმა გამოიყენა დღეს **არითმეტიკულ ოპერაციებად წოდებული** ერთი და იგივე ოპერაციები *(შეკრება, გამოკლება, გამრავლება, გაყოფა, ამოფესვა და ახარისხება)*.

„ლოგიკის ალგებრის“ შემთხვევაში. მასში გამოყენებული ცვლადები მხოლოდ **0**-ია ან **1**-ის ტოლი მნიშვნელობებს იღებს და მათზე შეიძლება შესრულდეს როგორც არითმეტიკული, ისე ლოგიკური

ოპერაციები, რომელთა რაოდენობა **10**-ის ტოლია (ცხრ. **1.2**). ლოგიკური ოპერაციები დეტალურად [**1; 5; 4**]-ში გვაქვს განხილული.

1.4. ლოგიკური მანქანების ისტორიიდან

არითმეტიკული და ლოგიკური მანქანები ქმნის ქვაკუთხედს, რომელზედაც დაფუძნებულია უახლოეს ტექნოლოგიად მიჩნეული კომპიუტერი. კომპიუტერული მეცნიერების სფეროში გამოცემულ ლიტერატურაში ღირსეულადაა დაფასებული კომპიუტერის სტრუქტურის აგებაში **არითმეტიკული მანქანების** მიერ შეტანილი წვლილი. ამის დასადასტურებლად იმის აღნიშვნაცაა საკმარისი, რომ ამსტერდამის თავისუფალი უნივერსიტეტის (*ნიდერ. Vrije Universiteit Amsterdam*) პროფესორმა, კომპიუტერულ მეცნიერებებში რეიტინგული წიგნების ავტორმა ავტორმა **ენდრიუ სტიუარტ ტანენბაუმმა** (*ინგ. Andrew Stuart Tanenbaum*) არითმეტიკულ მანქანებს **ნულოვანი თაობის კომპიუტერები** უწოდა. კომპიუტერული მეცნიერების წიგნების აბსოლუტური უმრავლესობა იწყება **ვილჰელმ შიკარდისა** და **ბლეზ პასკალის** არითმეტიკული მანქანების მოხსენიებით, არაფერი რომ არ ვთქვათ სხვა არითმეტიკულ მანქანებზე.

ლოგიკური მანქანებს, სამწუხაროდ, ასეთი ხვედრი არ ეღირსა და, ალბათ, ამიტომ, ქართულ ტექნიკურ ლიტერატურაში ისინი საკმაოდ მწირადაა განხილული. აღნიშნული ნაკლის ნაწილობივად გამოსწორებისთვის მათ განხილვას შედარებიდ დიდ ადგილს დავუთმობ.

1.4. 1. ლოგიკური მანქანის რაობა

ლოგიკური მანქანები და ზემოთ განხილული **არითმეტიკული მანქანები** ერთმანეთისაგან განსხვავდება არა მარტო ფუნქციური დანიშნულებით, არამედ მათი პრაქტიკული გამოყენებადობის ნიშნითაც. არითმეტიკულ მანქანებს ადამიანი აგებდა საკუთარი პრაქტიკული მიზნების მისაღწევად. ამის დამადასტურებელი კლასიკური მაგალითია **ბლეზ პასკალის** მიერ აგებული გამომთვლელი მანქანა „**პასკალინას**“ შექმნის ისტორია. იგი **19** წლის **ბლეზ პასკალმა** შექმნა წმინდა უტილიტარული მიზნის მისაღწევად: გადასახადების ამკრეფად მომუშავე მამისათვის შრომის

შესამსუბუქებლად. მაღალი პრაქტიკული გამოყენებადობის გამო კალკულატორებად წოდებული არითმეტიკულ მანქანებით დღესაც მასობრივად მარაგდება ტექნიკის ბაზარი.

ზემოთ აღნიშნულისაგან განსხვავებით, დიდი ხნის განმავლობაში **არცერთი ლოგიკური** მანქანა არ გამოიყენებოდა პრაქტიკული საქმიანობისათვის. ისინი ლოგიკური ოპერაციების მექანიზების შესაძლებლობათა მადემონსტრირებელ მანქანებად ითვლებოდა, რომებიც შეგვეძლო დამხმარე სახელმძღვანელოებად გამოგვეყენებინა. ამერიკელი ინჟინერი, კრიპტოანალიტიკოსი და მათემატიკოსი **კლოდ ელვუდ შენონი** (*ინგ. Claude Elwood Shannon; 1916 - 2016*) არის მეცნიერი, რომელმაც დიამეტრულად შეცვალა დამკვიდრებული შეხედულება **ლოგიკურ მანქანებზე**. მან თავდაპირველად თავის სადიპლომო ნაშრომში, ხოლო მოგვიანებით, **1938** წელს გამოქვეყნებულ სამეცნიერო სტატიაში **„რელეური და გადამრთველი სქემების სიმბოლური ანალიზი**“ („*A Symbolic Analysis of Relay and Switching Circuits*“), გვაჩვენა რელეურ (კომუტირებად) სქემებსა და ლოგიკურ ფუნქციებს შორის არსებული იზომორფიზმი; აღნიშნულის გამო, ნებისმიერი რელეური და გადამრთველი მოწყობილობა ლოგიკური ფუნქციების მარეალიზებელი ლოგიკური ელემენტებით შეგვიძლია ავაგოთ. ამის წყალობით ლოგიკურმა მანქანებმა უაღრესად დიდი პრაქტიკული მნიშვნელობა შეიძინა და კიდევ ერთხელ დაადასტურა ცნობილი ავსტრიელი ფიზიკოსის **ლუდვიგ ედუარდ ბოლცმანის** (*გერმ. Ludwig Eduard Boltzmann*) ცნობილი სენტენციის ჭეშმარიტება, რომლის თანახმადაც „**კარგ თეორიაზე უფრო პრაქტიკული არაფერი არ არის**“. ლოგიკური ელემენტების შესახებ დამატებით ინფორმაციას შეგიძლიათ [**1**]-ში გაეცნოთ.

შეიძლება ზოგიერთები გააოცოს იმ ფაქტმა, რომ ლოგიკური მანქანების ისტორია მთელი **სამი საუკუნით აღემატება** არითმეტიკული მანქანების ისტორიას. კიდევ უფრო გასაოცარია ის გარემოება, რომ თავად ტერმინი „**ლოგიკური მანქანა**“ ლიტერატურაში გაცილებით გვიან, კერძოდ, **XIX** საკუნეში შემოვიდა. იგი შემოიტანა ლოგიკის, ფილოსოფიისა და პოლიტეკონომიის ინგლისელმა პროფესორმა, პოლიტეკონომიაში მათემატიკური თეორიისა ფუძემდებელმა, ზღვრული სარგებლიანობის თეორიის ერთ-ერთმა შემქმნელმა **უილიამ სტენლი ჯევონსმა** (*ინგ.William Stanley Jevons*; *1835 - 1882*). ასეთი გვიანი „ნათლობის“ მიზეზი, ჩვენი აზრით, მისი პრაქტიკული მნიშვნელობის დაუნახავობა უნდა ყოფილიყო. ლოგიკური მანქანის ცნების

პოპულიზებში დიდია ამერიკელი ფილოსოფოსის, ლოგიკოსის, მათემატიკოსის, სემიოტიკოსისა და პრაგმატიზმის ფუძემდებლის, **ჩარლზ სანდერს პირსის** (*იხგ.Charles Sanders Peirce; 1839 - 1914*), მიერ შეტანილი წვლილი.

ამერიკელმა ფსიქოლოგმა, ფილოსოფოსმა და სოციოლოგმა **ჯეიმს მარკ ბოლდუინმა** (*იხგ. James Mark Baldwin; 1861 - 1934*) **1901** წელს გამოცემულ თავის „ლექსიკონში“ (*იხგ. „**Dictionary of philosophy and psychology**“*) ტერმინი „ლოგიკური მანქანა“ ასე განსაზღვრა: **„ლოგიკური მანქანა** არის ლოგიკურ სიმბოლოებსა და დიაგრამებზე მექანიკური ხერხით ჩასატარებელი მოქმედებების შესასრულებლად შექმნილი ინსტრუმენტი“.

ლოგიკური მანქანებისათვის მიძღვნილ პირველ, და დღემდე ერთადერთ, მონოგრაფიაში „**Logic Machines and Diagrams**“, რომელიც **1958** წელს გამოსცა ამერიკელმა მათემატიკოსმა და მეცნიერების ცნობილმა პოპულიზატორმა **მარტინ გარდნერმა** (*იხგ.Martin Gardner*; *1914 - 2010*), **ლოგიკური მანქანა** განმარტებულია როგორც ფორმალური ლოგიკის ამოცანების გადასაწყვეტად სპეციალურად შექმნილი მექანიკური ან ელექტრული ხელსაწყო. ტერმინის შემდგომი დაზუსტება-ცვლილება არ მომხდარა. ეს იმ გარემოებან განაპირობა, რომ სპეციალიზებული **ლოგიკური მანქანების** შექმნის უკანასკნელი მცდელობები სწორედ **გარდნერის** წიგნის გამოსვლისას მოხდა. ამ პერიოდში გამოქვეყნდა უნივერსალური გამომთვლელი მანქანების (კომპიუტერების) გამოყენებით იმავე ამოცანების პროგრამული რეალიზებისათვის მიძღვნილი შრომები, რომელთა გადასაწყვეტად იქმნებოდა ლოგიკური მანქანები. ასე რომ, **ავტონომიური სახით** ლოგიკური მანქანების შექმნის ეპოქა კომპიუტერების გამოსვლით დასრულდა.

1.4. 2. ასე იწყებოდა ლოგიკური მანქანა

ლოგიკური მანქანების ისტორია შეიძლება სამ, კერძოდ **მექანიკურ,** ელექტრომექანიკურ და **ელექტრონულ** ეტაპებად დავყოთ. **მექანიკურ ეტაპს** საფუძველი ჩაუყარა **XIII** საუკუნეში მოღვაწე კატალონიელმა მისიონერმა **რაიმონდ ლულიუსმა** (*ლათ. Raymundus Lullius; კატალ. Ramon Llull; 1232/33 – 1315/16*); **ელექტრომექანიკურ ეტაპზე** მოღვაწე მეცნიერებიდან უნდა გამოვყოთ ცნობილი ბრიტანელი ექიმი, სამეფო საზოგადოების წევრი **ალფრედ სმი** (*იხგ. Alfred Smi;*

1818-1877), ხოლო **ელექტრონულ-ლოგიკურ** ეტაპის ფუძემდებელია **კლოდ შენონი** (იხ. ზემოთ).

მოკლედ განვიხილოთ სამივე ეტაპი.

I ეტაპი

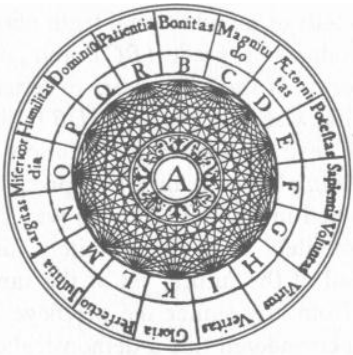
მოცემული ეტაპი მიჩნეულია მექანიკურ ეტაპად. ამ დროს პირველად დაისვა ლოგიკური მსჯელობების პროცესის მექანიზების ამოცანა და პირველი ნაბიჯები გადაიდგა ნაბიჯები „მოაზროვნე“ მექანიკური მოწყობილობების გამოგონების მიმართულებით. დღემდე მოღწეული ფაქტოლოგიური დოკუმენტების თანახმად მსჯელობების მექანიზების არაცხადად ფორმულირებული ამოცანის დასმის პრიორიტეტი ეკუთვნის **რაიმონდ ლულიუსს**. ითვლება, რომ სწორედ იგი შეეცადა პირველად აეგო მექანიკური ლოგიკური მოწყობილობა. **ლულიუსმა** ამ მოწყობილობას საფუძვლად დაუდო მის მიერვე საფუძვლიანად დამუშავებული **კომბინატორული მეთოდი**. ლიტერატურიდან ცნობილია, რომ ცნობილმა ინგლისელმა სატირიკოსმა მწერალმა და პუბლიცისტმა **ჯონათან სვიფტმა** (*ინგ. Jonathan Swift; 1667 - 1745*) ლულიუსის მანქანას „კომპიუტერი“ უწოდა და თავის „გულივერის მოგზაურობაში“ იგი სატირულად აღწერა.

თავისი მოწყობილობა **რაიმონდ ლულიუსმა** თავის ტრაქტატ „**Ars Magna**“-ში („დიად ხელოვნებაში“) და მთელ რიგ სხვა ნაშრომში აღწერა. ამ ნაშრომების თანახმად, **ლულიუსს** მიაჩნდა, რომ ცოდნის თითოეულ სფეროში შესაძლებელია გამოიყოს პირველადი კატეგორიები, ანუ ცნებები, რომელთა კომბინირებითაც შევძლებთ სამყაროს შესახებ მოვიპოვოთ ყველა მოაზრებადი ცოდნა და საგნებს შორის არსებული კავშირები აღმოვაჩინოთ.

პირველადი კატეგორიები **ლულიუსიმა** აღნიშნა ლათინური მთავრული ასოებით, კომბინაციების მისაღებად კი გამოიყენა სექტორებად (ანუ, როგორც იგი უწოდებდა, **კამერებად**) დაყოფილი ორი კონცენტრირებული რგოლი. თითოეულ კამერაში ჩაწერილი იყო კატეგორიები, ან მათი ასოითი აღნიშვნები. რგოლების ბრუნვით ადვილად მიიღებოდა სხვადასხვა კომბინაციათა ცხრილები. ასე წარმოიქმნემნებოდა გარკვეული ფიგურა.

ლულიუსმა შემოგვთავაზა შვიდი ასეთი ფიგურა, რომელთაგანნაც პირველ ორ **ნახაზ 1.4,ა,ბ**-ზეა ნაჩვენები.

ა)



ბ)

გ)

ნახაზი 1.4. ა) ღმერთის ატრიბუტების ფიგურა; **ბ)** სულის ატრიბუტების ფიგურა; **გ)** გამარტივებული ცხრაკამერიანი ფიგურა.

პირველი და მთავარი ფიგურა ღმერთს ეძღვნებოდა და **16** კამერად დაყოფილი ორი კონცენტრირებული რგოლის ცენტრში არსებული **A** ასოთი იყო აღნიშნული (იხ. ნახ **1.4,ა**). გარე რგოლში ჩაწერილი იყო **ღმერთის ატრიობუტები**, ხოლო შიგა რგოლში - მათი აღმნიშვნელი მთავრული ასოები. მაგალითად, **B** – bonitas (სიკეთე), **C** – magnitude (სიდიადე), **D** – Aeternitas (მარადისობა), **E** – sapienta (სიბრძნე) და ა. შ. გარე რგოლის მიმართებით შიგა რგოლის ბრუნვით ვიღებთ **240** კომბინაციას, რომელთაგანაც თითოეული გვაძლევს გარკვეულ ცოდნას. ასე, მაგალითად, შეიძლებოდა გაგვეგო, რომ უზენაესის (ღმერთის) სიკეთე უდიადესია (BC) და მარადიულია (CD), სიბრძნე - უსასრულოა (ED) უკვდავება ჭეშმარიტია (DI) და ა. შ.

S ასოთი აღნიშნული მეორე ფიგურა (ნახ. **1.4,ბ**) **ლულიუსმა** მიუძღვნა ადამიანის **სულს**. წინა ფიგურიდან ცნობილი კონცენტრირებულ რგოლებს გარდა ფიგურაში დამატებითაა გამოყენებული სხვადასხვა ფერიანი ოთხი კვადრატი, რომელთა წვეროები ლათინური მთავრული ასოითაა აღნიშნული. ასეთებია ცისფერი **BCDE**, შავი **FGHI**. წითელი **KLMN** და მწვანე **ROPQ** კვადრატები. ამ კომბინაციებზე ფიქრით მრავალი თეოლოგიური პრობლემის განმარტების მიღებაა შესაძლებელი.

ცხრ. 1.3. ადამიანთა უნარები, რომლებიც შეეთანადება ლულიუსის მეორე ფიგურის კვადრატთა წვეროების აღმნიშვნელ ასოებს

კვადრატთა ფ ე რ ე ბ ი	დამახსოვრება	გონი	ნება	Σ
ცისფერი კვადრატი	**B**	**C**	**D**	**E** = B ∪ C ∪ D
შავი კვადრატი	**F**	**G**	**H**	**I** = F ∪ G ∪ H
წითელი კვადრატი	**K**	**L**	**M**	**N** = K + L + M
მწვანე კვადრატი	**R**	**O**	**P**	**Q** = R + O + P

ცისფერი BCDE კვადრატი არის სულის ნორმალური მდგომარეობა, სადაც: **B** აღნიშნავს **მეხსიერებას**, რომელსაც ახსოვს; **C** - **გონს**, რომელიც მცოდნეა; **D** - **ნებას**, რომელიც სიყვარულითაა სავსე და **E** - ამ უნარების გაერთიანებას.

შავი HFGI კვადრატის **F** და **G** ასოთა მნიშვნელობები ემთხვევა წინა კვადრატის **B** და **C** ასოებით აღნიშნული უნართა მნიშვნელობებს, **H** აღნიშნავს ადამიანის **ნებას**, რომელსაც სამართლიანად სძულს (მაგალითად, ცოდვა), **I** - ამ უნარების გაერთიანებას.

წითელი KLMN კვადრატი ყოველივეს მოძულე ადამიანის სულია, სადაც, **K** აღნიშნავს **მეხსიერებას**, რომელსაც არ ახსოვს; **L** - **გონს**, რომელიც უმეცარია, **M** - **ნებას**, რომელიც სიძულვილითაა სავსე და **N** - ამ უნარების გაერთიანებას.

მწვანე ROPQ კვადრატი ეჭვიანი ადამიანის სულია, სადაც **R** აღნიშნავს მეხსიერებას, რომელიც იმახსოვრებს და ივიწყებს კიდეც; **O** - გონს, რომელიც მცოდნეცაა და უმეცარიც, **P** - ნებას, რომელსაც სიყ-

ვარულიც შეუძლია და სიძულვილიც, ხოლო **Q** - ყველა ამ უნარის გაერთიანებაა.

მოცემული ფიგურის შიგა რგოლის ბრუნვით ვიღებთ **136** ცოდნას.

დანარჩენი ფიგურების განხილვით არ გადავტვირთავთ თქვენს გონებას, რადგან განხილული მაგალითები ლულიუსის მოწყობილობის რგოლების აგების ზოგადი პრინციპების გასაგებად საკმარისია.

დიადი ხელოვნების მომდევნო შრომებში **ლულიუსი** ორი მიმართულებით მიდიოდა: **ერთი მხრივ** ართულებდა საკუთარ ფიგურებს და, **მეორე მხრივ**, საწყის ცნებათა რაოდენობების შემცირებით ამარტივებდა მათ. ლულიუსის გამომგონებლობის მწვერვალად გადაიქცა ლითონის თოთხმეტი დისკისაგან შემდგარი **figura universalis**, რომლის მეშვეობითაც დაახლოებით $\mathbf{18 \cdot 10^{15}}$ რაოდენობის შეუღლებების მიღებაა შესაძლებელი; ეს, ლულიუსის აზრით, საკმარისი უნდა ყოფილიყო იმისათვას, რომ მოწყობილობას თავის თავში ხორცი შეესხა **ყოვლისმომცველი გონებისათვის**. მიუხედავად ამისა, უნდა აღვნიშნოთ, რომ დიდი პოპულარობით **ლულიუსის** გამარტივებული ცხრაკამერიანი ფიგურები (იხ. ნახ. **1.4,გ**) სარგებლობდა. **ლულიუსმა** მათ დაუმატა თავისებური ალფაბეტა, რომლის **B**-დან **K**-მდე ჩათვლით ყოველი ასო ცნებათა გარკვეულ ჯგუფს აღნიშნავდა. მათ ჩვენ არ განვიხილავთ.

ლულიუსს უსათუოდ გაცნობიერებული ჰქონდა, რომ „პირველადი პრინციპების“ მხოლოდ შეპირაპირება საკმარისი არ იქნებოდა. იგი იმედოვნებდა, რომ „**პირველადი პრინციპების**“ მექანიკური კომბინირებით შეძლებდა განესაზღვრა დასკვნების მისაღებად აუცილებელი „სამშენებლო ბლოკები“. ამგვარად, **ლულიუსს** მიაჩნდა, რომ „დიადი ხელოვნების“ ცოდნით შეიარაღებული მკვლევარის **პირველი ამოცანა** იყო თითოეული **მეცნიერებისათვის** შეედგინა ფუძემდებლური ცნებების ამომწურავი ჩამონათვალი. სამწუხაროდ, ჯერ კიდევ დიდი დრო იყო მანამდე, როდესაც ადამიანი დარწმუნდებოდა, რომ ცნებები **შემეცნების შედეგია** და თავისთავად ნათელი **„პირველადი პრინციპები**“ არცერთ მეცნიერებაში არ არსებობს. მაშასადამე, ლულიუსის **პირველი ამოცანა** გადაუწყეტადია. იგივე არ შეიძლება ითქვას მეორე და მესამე ამოცანაზე. **მეორე ამოცანის** მთავარი მიზანი იყო მანქანა ყველა შესაძლო ცნების კომბინირებისათვის გამოგვეყენებინა. ეს ისეთი ამოცანაა, რომელიც არა მარტო შეუძლია შეასრულოს

მანქანამ, არამედ აუცილებელიცაა ამისათვის მისი გამოყენება; და, ბოლოს, ასევე სწორადაა დასმული **მესამე ამოცანაც**. ამ ამოცანის თანახმად თითოეული კომბინაციის ჭეშმარიტობა ადამიანის უნდა დაედგინა. ამგვარად, მანქანა **ლულიუსისათვის** მხოლოდ ინსტრუმენტი იყო, რომელსაც უნდა მოემზადებინა ყველა შესაძლო ვარიანტი და სათანადო გადაწყვეტილების მისაღებად მიეწოდებინა ადამიანისათვის!

ლულიუსის იდეები განავითარეს ისეთმა გამოჩენილმა ადამიანებმა, როგორიც იყვნენ რომის კათოლიკური ეკლესიის კარდინალი, **XV** საუკუნის უდიდესი გერმანელი მოაზროვნე, ფილოსოფოსი და მათემატიკოსი **ნიკოლაი კუზანელი** (*ლათ. Nicolaus Cusanus; 1401 - 1464*), იტალიელი კათოლიკური მღვდელი, ფილოსოფოსი და პოეტი **ჯორდანო ბრუნო** (*იტალ. Giordano Bruno;1548 - 1600*), გერმანელი მეცნიერი და გამომგონებელი, იეზუტების ორდენის ბერი, მათემატიკის პროფესორი **ათანასე კირხერი** (*გერმ. Athanasius Kircher*; *1602 - 1680*) და სხვები.

განსაკუთრებით უნდა აღვნიშნო ის ფაქტი, რომ **რაიმონდ ლულიუსის** ნაშრომმა „**Ars Magna**“ განსაკუთრებულად დოდი ზეგავლენა მოახდინა **გოტფრიდ ლაიბნიცის** ფილოსოფიური და მეცნიერული თვალსაზრისის ფორმირებაზე, რომელიც მან ამ სიტყვებით გამოხატა: ჩემი მიზანია არა უბრალოდ ახალი შედეგების მიღება (მაგალითად, მათემატიკაში), არამედ ამ ახლის პოვნის უზრუნველმყოფი ზოგადი ფორმალური მეთოდის დამუშავება! **ლაიბნიცის** მიხედვით, ეს მეთოდი ადამიანური ცოდნის **„საყოველთაო ალფაბეტაზე**“ უნდა დაფუძნებულიყო. **ლულიუსისაგან** განსხვავებით **ლაიბნიცი** თვლიდა, რომ მისთვის დამატებით შეექმნა აღნიშვნების სისტემა - უნივერსალური მახასიათებელი - და დაემუშავებინა უმარტივესი ცნებებიდან რთული ცნებების მიღების წესები. გვიანდელ ფილოსოფიურ თხზულებებში **ლაიბნიცი** წერდა, რომ ღმერთი არსებული რეალობების აღრიცხვისა და მათგან საუკეთესოს ამორჩევის გზით ქმნის ახალს. ლულიუსის მეთოდი, ფაქტობრივად, ასეთივე მიდგომას იყენებდა.

ლულიუსისთვის დამახასიათებელი სქოლასტიკურმა ხასიათმა ბევრჯერ დაიმსახურა სხვადასხვა ცნობილი პიროვნების არაერთი საკმაოდ მკვახე შეფასებები. თქვენს ყურადღებას შევაჩერებ მხოლოდ ინგლისელ მწერალ **ჯონათან სვიფტზე.** მან თავისი **„გულივერის მოგზაურობის**“ მესამე თავში ქალაქ **ლაგოდოში** დიდი აკადემიის აღწე-

რით, რომლის წევრები აგებდნენ განყენებული ჭეშმარიტებების აღმოჩენისათვის განკუთვნილ განსაკუთრებულ მექანიკურ ხელსაწყოს, მასხრად აიგდო **ლულიუსის** იდეები. ეს მაგალითი თუნდაც იმიტომ იმსახურებს აღნიშვნას, რომ **უილიამ სტენლი ჯევონსმა** მასში აღწერილ მანქანას პირველად უწოდა **ლოგიკური მანქანა** და მხოლოდ ამის შემდეგ დამკვიდრდა ეს სახელი მეცნიერებაში.

რაიმონდ ლულიუსის მიერ **XIII** საუკუნეში დაწყებული მექანიკური ლოგიკური მანქანების ეტაპმა თავის კულმინაციას **XIX** საუკუნეში მიაღწია. ამ პერიოდში კონკრეტული ლოგიკური ამოცანების გადაწყვეტისათის შეიქმნა ისეთი უმნიშვნელოვანეს მექანიკურ ლოგიკური მანქანები, როგორიცაა ბრიტანელი მათემატიკოსის, მექანიკოსისა და გამომგონებლის **ჩარლზ სტენჰოპის** (*ინგ.Charles Stanhope; 1753-1816*) დემონსტრატორი, რუსი გამომგონებლის **სემიონ კორსაკოვის** (*რუს.Семён Никола́евич Корса́ков, 1787-1853*) ინტელექტუალური მანქანები, **უილიამ სტენლი ჯევონსის 21** კლავიშიანი „**ლოგიკური პიანინო**“ და პრისტონის უნივერსიტეტის პროფესორის **ალან მარკვანდის** (*ინგ.Allan Marquand; 1853-1924*) ლოგიკური მანქანა.

ალან მარკვანდი თვლიდა, რომ მან პრაქტიკულად **ჯევონსის** მანქანის კოპირება მოახდინა, მაგრამ უნდა აღვნიშნოთ, რომ **მარკვანდმა** მასში ნოვაციებიც შეიტანა. **ჯევონსის** მანქანასთან შედარებით, **მაკვადის** მანქანის უპირატესობის აღიარებისათვის თუნდაც იმის აღნიშვნაცაა საკმარისი, რომ ამ უკანასკნელში კლავიშების რაოდენობა **10**-მდე იქნა შემცირებული. ცნობილი ამერიკელი ლოგიკოსი **ჩარლზ სანდერს პირსი** (*ინგ.Charles Sanders Peirce*; *1839 - 1914*) თავის ერთ-ერთ ნაშრომში აღნიშნავდა, რომ **მარკვანდის** მანქანა **ჯევონსის** მანქანაზე გაცილებით ეფექტური იყო, თუმცა იქვე აღნიშნავდა ორივე მანქანისათვის დამახასიათებელ შეზღუდულობასაც.

მარკვანდის შესახებ ცოტა ქვემოთ კიდევ ერთხელ ვისაუბრებთ, ახლა კი მინდა აღვნიშნო ლოგიკური მანქანების განვითარების საქმეში **უილიამ სტენლი ჯევონის** დიდი დამსახურება.

უილიამ სტენლი ჯევონის სახელს, უსამართლოდ, საკმაოდ იშვიათად ახსენენ კომპიუტერების განვითარების ისტორიაში. ივიწყებენ ბულის (ლოგიკის) ალგებრისათვის თანამედროვე სახის მიღების საქმეში მის მიერ შეტანილ ღვაწლს. პრაქტიკულად, სწორედ **ჯევონსის** შრომების წყალობით ჩამოყალიბდა **ბულის ალგებრა** თანამედროვე სახით. მხოლოდ ეს იქნებოდა სრულიად საკმარისი იმისათვის, რომ

ჯევონსს საპატიო ადგილი დაეკავებინა კომპიუტერული დიდების პანთეონში, მაგრამ მას კიდევ ემატება ის, რომ, იგი გახდა რეალურად მოქმედი პირველი ლოგიკური მანქანის („**ლოგიკური პიანინოს**“) შემქმნელი, რადგან **ლულიუსისა** და **სტენჰოუპის** მოწყობილობებს ძალიან ძნელია **მანქანები** ეწოდოს.

II ეტაპი

მოცემულ ეტაპს ეწოდება **ელექტრომექანიკური ეტაპი**. მის ფორმირებაში დიდი ღვაწლი მიუძღვის ინგლისელ ექიმს, სამეფო საზოგადოების წევრს **ალფრედ სმის** (*ინგ. Alfred Smee,;1918 - 1877*), რომელმაც დაამუშავა **ელექტრობიოლოგიად** წოდებული თეორია. მისი მიზანი იყო, შეესწავლა ადამიანის ორგანიზმზე ელექტრული გავლენა; ამავე დროს მას განსაკუთრებით თავის ტვინის მუშაობასთან ნერვული სისტემის ელექტრული სტიმულირების კავშირი აინტერესებდა. **1851** წელს ლონდონში მან გამოსცა წიგნი **„სიტყვებთან და ენასთან ადაპტირებული აზროვნების ანალიზი**“ (*„Process if Thought Adapted to Words and Language“*). მასში ავტრმა გადმოსცა დაკვნების გამოტანის იმ ხელოვნური სისტემის აგების გეგმა, რომელიც ახდენს ადამიანის ტვინის მუშაობის მექანიზმების მოდელირებას“. ამ დროს იგი ეყრდნობოდა არა ამ მექანიზმების ცოდნას (ეს ცოდნა იმ პერიოდში არ არსებობდა), არამედ ამა თუ იმ თვისებების არსებობა-არარსებობას. მან ივარაუდა, რომ ეს ინფორმაცია ნერვული ბოჭკოებით გადაცემული ელექტრული სიგნალებით აისახებოდა ადამიანის ტვინში. ამგვარად, თითოეული ცნება ელექტრონული იმპულსების გამტარებელი ნერვული ბოჭკოების ნაკრების სახით წარმოიდგინება. **ალფრედ სმის** ელექტრობიოლოგიური მანქანა თანამედროვე **ნეირონული კომპიუტერის** პირველსახეა.

აქვე უნდა აღვნიშნოთ, რომ დაახლოებით **1885** წელს **მარკვანდმა** შეადგინა ისტორიაში პირველი **ელექტრომექანიკური ლოგიკური მანქანის** პროექტი. მის არქივში შენახული ამ მანქანის ერთადერთი ნახაზი მხოლოდ **1853** წელს გამოქვეყნდა. არსებობს შეხედულება, რომ **მარკვანდს** არ უცდია მისი განხორციელება. მონაცემების ნაკლებობა ამ სქემის სხვადასახაგარად ინტერპრეტირების საშუალებას იძლევა, მაგრამ ნებისმიერ შემთხვევაში ამ მანქანას ერთ-ერთი გამორჩეული ადგილი უკავია არა მარტო ლოგიკური მანქანების, არამედ მთლიანად გამოთვლითი ტექნიკის ისტორიაში. მხოლოდ შეგვიძლია ვივა-

რაუდოთ, თუ როგორი გზებით წავიდოდა გამომთვლელი მანქანების განვითარება, მარკვანდს რომ განეხორციელებინა საკუთარი პროექტი და მას მოეპოვებინებინა ის აღიარება, რასაც იგი უთუოდ იმსახურებდა.

III ეტაპი

ლოგიკის მანქანების განვითარების მესამე ეტაპის დაწყებას საფუძველი ჩაუყარა **1938** წელს **კლოდ შენონის** მიერ გამოქვეყნებულმა საეტაპო მნიშვნელობის სამეცნიერო ნაშრომმა. რომელზეც **§1.4.1**-ში ვსაუბრობდით. მან საშუალება მოგვცა ჯერ ელექტრომაგნიტურ რელეთა კონტაქტების, შემდეგ - ტრანზისტორებისა და, ბოლოს, ინტეგრალური სქემებით მოგვეხდინა ლოგიკური მანქანების სინთეზი. დაიწყო მესამე - **ელექტრონული ეტაპი** (იხ. §1.4.2). იგი ბრწყინვალედ დააგვირგვინა **XX** საუკუნეში შექმნილმა პროგრამულმა მოწყობილობამ - ციფრულმა ელექტრონულმა **კომპიუტერმა**, რომელმაც ლოგიკური მანქანების აპარატურული სინთეზის მეთოდი ჩაანაცვლა პროგრამული სინთეზის მეთოდით. ეს იყო ბრწყინვალე დადასტურება ფუნდამენტურმა დებულებისა, რომლის თანახმადაც აპარატურულად რეალიზებული ნებისმიერი მოწყობილობის რეალიზება პროგრამულადაცაა შესასაძლებელი და პირიქით. ამან **დაასრულა სპეციალიზებული ლოგიკური მანქანების აპარატურულად რეალიზების ერა** და მათი პროგრამულად რეალიზების ერას დაუდო სათავე. მანამდე შექმნილი ლოგიკური მანქანები ან ალგორითმულად (პროგრამულად) იქნა რეალიზებული ან პოტენციურად შესაძლებელია მათი ამგვარი რეალიზება.

1.5. ბაზისური ცნობები ინტელექტისა და ხელოვნური ინტელექტის შესახებ

დღეს ყველასათვის ცნობილი ტერმინ „**ინტელექტით**“აღნიშნავენ **ადამიანის ზოგად უნარს** შეიმეცნოს და გადაწყვიტოს მის წინაშე წარმოშობილი ყველა სიძნელე. მასში გაერთიანებულია ადამიანისათვის დამახასიათებელი ყველა შემეცნებითი უნარი: შეგრძნობა, აღქმა, დამახსოვრება, აზროვნება, წარმოსახვა და შემეცნება. ინტელექტის ასეთი გააზრებისათვის დამახასიათებელია გარკვეული შეზღუდულობა, რადგან იგი ინტელექტზე აბსოლუტურ მონოპოლიურ უფლებას მხო-

ლოდ ადამიანს ანიჭებს და მხედველობის გარეშე ტოვებს იმ ფაქტს, რომ ინტელექტის გარკვეული ნიშნები ცხოველებსა და მცენარებშიც შეინიშნება; ამიტომ ქცევითი გენეტიკის მოძრაობის დამფუძნებელ-მა, ინგლისელმა **ფრენსის გალტონმა** (*ინგ.. Francis Galton, 1822 - 1911*) შემოგვთავაზა ინტელექტის ასეთი განსაზღვრება:

ინტელექტი (*ლათ. Intellectus „აღქმა“, „მიხვედრა“, „გაგება“, “შეგნება“, „წარმოდგენა“*) არის ფსიქიკის თვისება, რომელიც შედგება ახალი სიტუაციების შეცნობის, სწავლების, გამოცდილების შედეგად დამახსოვრების, აბსტრაქტული კონცეფციების გაგება-გამოყენებისა და საკუთარი ცოდნის მეშვეობით გარემომცვლი წრის მართვის უნარებისაგან.

ბრიტანელ-ამერიკელი ფსიქოლოგის **რეიმონდ ბერნარ კეტელის** (*ინგ. Raymond Bernard Cattell*; 1905-1998) თეორიის თანახმად ინტელექტი პირობითად შეიძლება დაიყოს **მოძრავ** და **კრისტალიზებად ინტელექტად. პირველი** გულისხმობს ლოგიკურად აზროვნების, გაანალიზებისა და წინა ცდის ფარგლებიდან გასული ამოცანის გადაწყვეტის, ხოლო **მეორე** - დაგროვილ გამოცდილებით ათვისებული ცოდნისა და ჩვევების გამოყენების უნარს.

ინტელექტის მატარებელია ბუნებრივად გაჩენლი სუბიექტები ან ობიექტები. შეგვიძლია ერთმანეთისაგან განვასხვავოთ ბუნებრივი და ხელოვნური ინტელექტი. კერძოდ, ბუნებრივ ინტელექტად - მივიჩნიოთ ბუნების „პროდუქტების“ - მცენარისთვის, ცხოველისა და ადამიანისთვის დამახასიათებელი ინტელექტი, ხოლო ხელოვნურ ინტელექტად - ადამიანის მიერ შექმნილი მანქანის ან პროგრამის ინტელექტი. მოკლედ განვიხილოთ თითოეული მათგანი.

მცენარის ინტელექტი.

ბევრი მეცნიერი, სადავოობის მიუხედავად, საკმაოდ დიდი ხანია თვლის, რომ მცენარეებს აქვთ გარკვეული ინტელექტი. **2022** წელს ჩატარებულ საკუთარ კვლევებზე დაყრდნობით, ამასვე ამტკიცებენ იტალიელი ბოტანიკოსი **სტეფანო მანკუზო** (*Stefano Mancuso,* 1965), ბრიტანული კოლუმბიის უნივერსიტეტის პროფესორი **სუზანა სმარდი** (*Suzanne Simard*), იტალიელი მეცნიერი **ბარბარა მაცოლაი** (*Barbara Mazzolai*) და ტელ-ავივის უნივერსიტეტის ბიოლოგიის ფაკულტეტის დეკანი, წიგნის - „რა იცის მცენარემ“ - ავტორი, **დენიელ შამოვიცი** (*Daniel Chamovitz*).

მიღებული განმარტების ძალით, **მცენარეების მეხსიერება და ინტელექტი** განპირობებულია მოლეკულარული ქსელებით, რომლებიც მათ დამახსოვრების, ამორჩევისა და სტრესული გაღიზიანებადობის საფუძველზე საკუთარი რესურსების ოპტიმიზების საშუალებას აძლევს. ამგვარად, **ნერვული სისტემის არარსებობის მიუხედავად**, მცენარეებს საკუთარი რესურსების ოპტიმიზების უნარი აქვს, რაც **ჩანასახოვანი სახის ინტელექტად უნდა ჩაითვალოს**. იგი თვისობრივად განსხვავდება როგორც ცხოველის, ისე ადამიანისათვის დამახასიათებელი ინტელექტისაგან, მაგრამ მის მსგავს შედეგებს იწვევს.

ცხოველის ინტელექტი

ცხოველების აზროვნების უნარზე ჯერ კიდევ ანტიკური დროიდან კამათობდნენ. ჩვენს წელთაღრიცხვამდე მეოთხე საუკუნეში ბერძენმა ფილოსოფოსმა **არისტოტელემ** (*ძვ. ბერძ. Ἀριστοτέλης,; 384-322 ძვ. წელთაღ.*) შენიშნა, რომ ცხოველებს აქვს სწავლის უნარი და მან დაუშვა ცხოველებში გონების არსებობის შესაძლებლობა. **ცხოველის ინტელექტად** თვლიან ფსიქიკური ფუნქციების ერთობლიობას, რომლებსაც მიეკუთვნება აზოვნება, სწავლისა და კომუნიკაციის უნარი; ისინი ვერ აიხსნება ინსტიქტებით ან პირობითი რეფლექსებით და მოითხოვს კოგნიტური **ეტოლოგიის** (*ზოოფსიქოლოგიის განყოფილება, რომელიც შეისწავლის ცხოველების ინტელექტს*), **შედარებითი ფსიქოლოგიისა** (*ფსიქოლოგიის ნაწილი, რომელიც შეისწავლის ფსიქიკის ევოლუციას*) და **ზოოფსიქოლოგიის** გამოყენებას.

მაღალგანვითარებული ცხოველები განიცდის ემოციებისა და მოტივების გავლენას. სავარაუდოა, რომ ზოგიერთ მათგანს შეუძლია ლოგიკური ვარაუდი და ახალ გარემოში გადაწყვეტილებების როგორც დაგეგმვა, ისე რეალიზება, აქვს სხვა ცხოველებთან ურთიერთობის უნარი.

ადამიანის ინტელექტი

ამერიკელი სოციოლოგის **ლინდა სუზანა გოტფრედსონის** (*ინგ. Linda Susanne Gottfredson*; *1947*) თანახმად **ადამიანის ინტელექტი** უაღრესად ზოგადი გონებრივი უნარია, რომელიც საშუალებას აძლევს ადამიანს დაასკვნას, დაგეგმოს, პრობლემები გადაწყვიტოს, აბსტრაქტულად იაზროვნოს, რთული იდეები გაიგოს, შესასწავლი მასალა სწრაფად აითვისოს, ცდების საფუძველზე გაამდიდროს თავისი ცოდნა. ეს არ არის უბრალოდ წიგნების, ვიწრო აკადემიური ცოდნის ან ტესტების გასავლელად საჭირო ჩვევების შესწავლა.

მეცნიერის აზრით, ადამიანის ინტელექტი ასახავს გარემომცველი სამ-ყაროს უფრო ფართოდ და ღრმად შემეცნების უნარს, იმის უნარს, რომ ჩაწვდეს საგნების არსს და მიხვდეს, თუ როგორ მოიქცეს ამა თუ იმ სიტუაციაში.

ინგლისელმა ფსიქოლოგმა, მათემატიკური სტატისტიკის მრავალი მეთოდიკის დამმუშავებლმა, **ჩარლზ ედვარდ სპირმენმა** (*ინგ. Charles Edward Spearman; 1863-1945*) **XX** საუკუნის დასაწყისში გვიჩვენა, რომ თუ ადამიანს შეუძლია გარკვეული სახის ამოცანების გადაწყვეტა, მას სხვა სახის ამოცანების წარმატებით გადაწყვეტის უნარიც ექნება. მანვე **1904** წელს შემოიტანა **ზოგადი ინტელექტის ფაქტორის,** ანუ ზოგადი ინტელექტის „**g ფაქტორის**“ ცნება; მას ფსიქოლოგები ზოგადი ინტელექტის გამოსავლინებელ სპეციალურ ტესტებში იყენებენ. იგი აფასებს, თუ რამდენად ეფექტურად შეუძლია ადამიანს შემეცნებითი ამოცანების გადაწყვეტა. პრაქტიკამ გვიჩვენა, რომ „g ფაქტორის“ პირდაპირი გაზომვა ძალიან ძნელია, მაგრამ მის საფუძველზე ფორმულირებული იქნა სიდიდეები, რომელთა გაზომვა შესაძლებელია და რომლებიც გვიჩვენებს **g**-ს მიახლოებით მნიშვნელობას.

სამხედრო საქმეში გამოიყენება ტერმინი „**დისპოზიცია**“, რაც ნიშნავს ჯარის ან ფლოტის საბრძოლველად განლაგებას. დისპოზიციაზე მდგარი ჯარი მზადაა საკუთარი ფუნქციების შესასრულებლად. ანალოგიურად, ადამიანის ინტელექტი არის მისი **აზროვნების დისპოზიცია**, ე. ი. საკუთარი (აზროვნებითი) ფუნქციების შესასრულებლად გონების მზადყოფნა.

ხელოვნური ინტელექტი.

„**ხელოვნული ინტელექტი**“, ბუნებრივი ინტელექტისაგან განსხვავებით, ადამიანის საქმიანობის შედეგად წარმოშობილი პროდუქტია. მის შესახებ შედარებით დაწვრილებით ადამიანის ინტელექტუალური საქმიანობის განხილვისას შევეხებით. ახლა მხოლოდ ტერმინ **„ხელოვნური ინტელექტის“** განმარტებით დავკმაყოფილდებით და მომდევნო თავში ბუნებრივი ინტელექტი ფორმირების საკითხებზე ვისაუბრებთ.

ტერმინ „ხელოვნური ინტელექტის“ ავტორის, ენა **ლისპის** შემქმნელის, ფუნქციური დაპროგრამების ფუძემდებლის, ტიურინგის პრემიის ლაურეატის **ჯონ მაკარტის** (*ინგ. John McCarthy; 1927-2011*) თანახმად **ხელოვნური ინტელექტი** (*ინგ.artificial intelligence*) არის ინტელექ-

ტუალური სისტემების თვისება შეასრულოს ტრადიციულად ადამიანის პრეროგატივად მიჩნეული შემოქმედებითი ფუნქციები.

1.6. დიდი აფეთქებიდან - ჰომო საპიენსის გამოჩენამდე

ჩვენი **სამყარო** მუდმივად არ არსებობდა. იგი ≈13,82 მილიარდი წლის წინ გაჩნდა და დაიწყო თვითგანვითარების პროცესი. ≈4,54 მილიარდი წლის წინ წარმოიშვა დედამიწა. დედამიწაზე ჩამოყალიბდა „**ბუნებრივი ინტელექტი**“. პირველად იგი **მცენარეების ინტელექტის** სახით მოგვევლინა, შემდეგ **ცხოველების ინტელექტად** ტრანსფორმირდა და, ბოლოს, **ადამიანის ინტელექტის** სახით დაამთავრა თავისი გენეზისი. ამის შემდეგ საქმეში ადამიანი ჩაერია და მოახდინა ინტელექტის კონსტრუირება, რომელსაც „**ხელოვნურს**“ ვუწოდებთ. მაშასადამე, იგი ადამიანის ინტელექტის „პირმშოა“.

ტერმინი „**ინტელექტის**“ განმარტება მრავალ საცნობარო ლიტერატურაშია მოცემული და მისი კიდევ ერთხელ გამეორება, ერთი შეხედვით, ძნელი არ უნდა ყოფილიყო: იგი შეგვეძლო ტრივიალური საკითხების ჩამონათვალისათვის მიგვეკუთვნებინა. ასეთი მოლოდინი მოწმენდილ ცაზე ატეხილი მოულოდნელი ჭექა-ქუხილივით დაარღვია ტერმინ „**ხელოვნური ინტელექტის**“ ავტორმა (**1956** წელს) დაპროგრამების ენა **ლისპის** გამომგონებელმა (**1958** წელს), ფუნქციური დაპროგრამების ფუძემდებელმა, ტიურინგის პრემიის ლაურეატმა (**1971** წელს) ამერიკელმა მეცნიერმა **ჯონ მაკარტიმ** (*იხგ. John McCarthy;1927-2011*). ერთ-ერთ თავის ბოლო სტატიაში მან განაცხადა, რომ ტერმინ **ინტელექტის** მყარი, ტერმინ „**ადამიანის ინტელექტისაგან**“ აბსოლუტურად დამოუკიდებელი, განსაზღვრება, არ არსებობს. იგი **ინტელექტს** სამყაროში მიზნის მიღწევისა უნარის **გამოთვლით ნაწილად** მიიჩნევს და თვლის, რომ ჯერჯერობით ჩვენ ინტელექტის მხოლოდ ზოგიერთი მექანიზმის შეცნობა მოვახერხეთ. დანარჩენები კი დღემდე შეუცნობელი გვრჩება, რაც ინტელექტის ტერმინის მყარად განასაზღვრის საშუალებას არ გვაძლევს.

ჯონ მაკარტი მიიჩნევს, რომ სხვადასხვა სახისა და ხარისხის ინტელექტი გვხვდება **ყველა** ადამიანში, **მრავალ** ცხოველსა და **ზოგიერთ** მანქანაში. განხილვის არეს გასაფართოვებლად ამ ჩამონათვალ-

ში მცენარების მოხსენიებაც სასურველია.

ბუნებრივი ინტელექტის მფლობელებიდან მხოლოდ ადამიანს გაუჩნდა **ხელოვნური ინტელექტის** შექმნის პრეტენზია და მან ამ მიმართულებით მნიშვნელოვან წარმატებებსაც მიაღწია. **კიბერნეტიკის** თანახმად, **ცოცხალ ორგანიზმებში** არსებული მართვის პროცესების ცოდნა ხელოვნური მანქანებში მართვის პროცესების ორგანიზებაში გვეხმარება.

საყოველთაოდ მიღებული ჰიპოთეზის თანახმად, „**დიდი აფეთქებით**“ ინიცირებული პროცესები არის **ცოცხალი ორგანიზმების** წარმოშობის მიზეზი. მიზეზ-შედეგობრივი კანონის ძალით, შედეგებზე (*ცოცხალ ორგანიზმებზე*) საუბრის დაწყებამდე საჭიროა იმ მიზეზების (*დიდი აფეთქებით ინიცირებული პროცესების*) ელემენტალური ცოდნა მაინც გვქონდეს, რომლთა მეშვეობითაც წარმოიქმნა ეს შედეგები. მოცემულ თავში სწორედ მათზე ვისაუბრებთ.

სამყაროს შექმნის პრობლემაზე დიდი ხნის განმავლობაში მრავალი შესანიშნავი მეცნიერის მუშაობის შედეგად კაცობრიობა დღეისათვის მივიდა ერთადერთ დასკვნამდე: სივრცეში არსებობს წერტილი, რომლის იქითაც ჩვენთვის ცნობილი ფიზიკის კანონები ვერ სრულდება და იქაური რეალობის შეუცნობა შეუძლებელია. ამ წერტილს კოსმოლოგები „**სინგულარულ**“ (*ინგ. Singularis – „ერთადერთი“, განუმეორებელი“*) **წერტილს** უწოდებენ. იგი მიიჩნევა **სამყაროს საწყის მდგომარეობად**, რომლისათვის დამახასიათებელია უსასრულოდ დიდი სიმჭიდროვე და ტემპერატურა. სამყარომ ამ წერტილისეული მდგომარეობიდან დაიწყო უსწრაფესი (**ფეთქებადი**) გაფართოება, რასაც **დიდი აფეთქება** ეწოდება, **დიდი აფეთქების თეორია** (The Big Band Theiry) იმის ახსნის მცდელობაა, თუ სინგულარული (უსასრულო სიმჭიდროვისა და ტემპერატურის მქონე წერტილოვან) მდგომარეობიდან სამყარომ თუ როგორ მიიღო ისეთი სახე, როგორც მას დღეს აქვს. თავიდანვე უნდა აღვნიშნოთ, რომ დღეისთვის დიდი აფეთქების თეორია სამყაროს წარმოშობა-განვითარების თეორიებს შორის, **ყველაზე ავტორიტეტული თეორიაა.** იგი კოსმოსისა და მატერიის წარმოქმნის საუკეთესო ახსნადაა მიჩნეული. მის გადმოცემამდე მოკლედ მოგახსენებთ, მეცნიერებმა როგორ დაასკვნეს, რომ კოსმოსში არსებობდა „სინგულარული“ წერტილი.

1.6. 1. დიდი აფეთქების ლოგიკური და ექსპერიმენტული საფუძვლები

დიდი აფეთქების თეორია დამყარებულია სინგულარულ მდგომარეობაში სამყაროს არსებობაზე, რომლის დროსაც მას ჰქონდა მაქსიმალური სიმჭიდროვე და ტემპერატურა. ამ დასკვნამდე კაცობრიობა ჯერ ლოგიკური ლოგიკურად მივიდა. მისი სისწორე მოგვიანებით, სრულიად მოულოდნელად, ექსპერიმენტულადაც დამკიცდა. ყოველივე ეს განვიხილოთ თანამიმდევრულად.

ყელაფერი დაიწყო შორეული **1915** წლიდან. საყოველთაოდ ცნობილმა თეორეტიკოს-ფიზიკოსმა **ალბერტ აინშტაინმა** (*ინგ. Albert Einstain; 1879-1955*) გამოაქვეყნა **ფარდობითობის ზოგადი თეორია**. მასში იგი თავდაპირველად ამტკიცებდა, რომ **სამყარო დინამიკური ობიექტი იყო** და იგი ბუნებრივად **ან ფართოვდებოდა ან იკუმშებოდა**. იმ დროინდელი ფიზიკოსების უმრავლესობის მსგავსად, აინშტაინი სამყაროს **სტატიკურ ობიექტად** მიიჩნევდა. ამიტომ მან საკუთარ განტოლებებს სამყაროს სტიტიკური ბუნების დასადასტურებლად საჭირო წევრებიც დაამატა.

საკმაოდ დიდი დროის გავლის შემდეგ, კერძოდ, **1924** წელს, სამყაროს წარმოშობის კვლევის საქმეში თავისდაუნებურად ჩაერთო ამერიკელი ასტროფიზიკოსი და კოსმოლოგი **ედვინ პაუელ ჰაბლი** (*ინგ. Edvin Pawell Hubble; 1889-1953*). თავდაპირველად იგი თითქოსდა **რუტინულ საქმეს ასრულებდა**: ზომავდა დედამიწიდან ორ სპირალურ ნისლეულებამდე არსებულ მანძილს და მოულოდნელად აღმოაჩინა, რომ: **1)** აღნიშნული ნისლეულები ფაქტობრივად ნისლეულები კი არა გალაქტიკები ყოფილა; **2)** ორივე გალაქტიკა სხვადასხვა მიმართულებით მოძრაობდა და შორდებოდა როგორც ერთმანეთს, ისე დედამიწას. ასეთმა მოძრაობამ ისეთი შთაბეჭდილება დატოვა, რომ ისინი თითქოს ერთსა და იმავე (**სინგულარუად** წოდებულ) წერტილიდან რაღაცა ძალის მიერ იყო გატყორცილი (იხ. დანართი, გვ. 230),

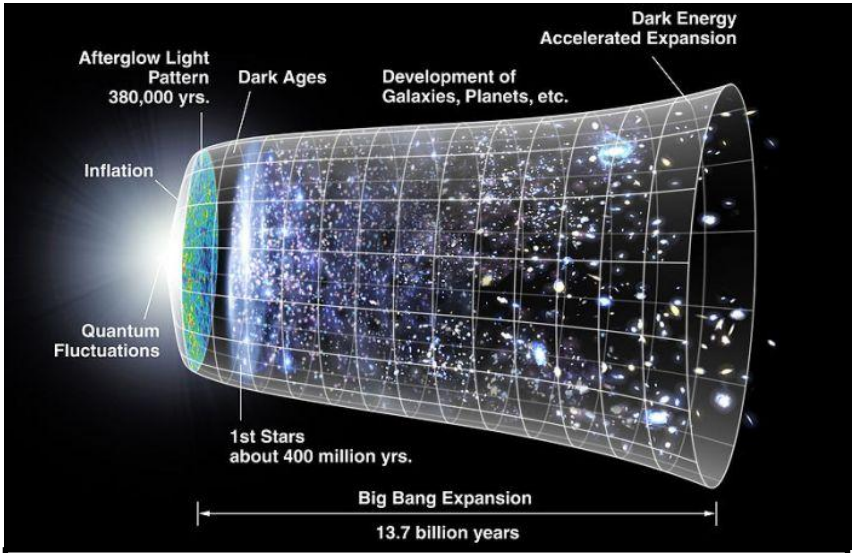
სანამ ასტრონომები **ჰაბლის** აღმოჩენილ შედგებზე კამათობდნენ, კერძოდ, **1931** წელს, საქმეში ჩაერთო ბელგიელი ფიზიკოსი და რომის კათოლიკე მღვდელი **ჟორჟ ანრი ჟოზეფ ედუარდ ლემეტრი** (*ფრან. Georg Henri Joseph Efuard Lemetr; 1894-1966*). მან როგორც **აინშტაინის** მტკიცებულობა, ისე **ჰაბლის** აღმოჩენა, აბსოლუტურ ჭეშმარიტებად მიიჩნია და წამოაყენა **გაფართოებადი სამყაროს იდეა**! იგი ლოგიკურად

ასე მსჯელობდა: რადგან სამყაროს გაფართოებისას იზრდება მისი ზომები, ხოლო მცირდება სიმჭიდროვე და ტემპერატურა, ამიტომ რეტროსპექტიული პროეცირების მეშვეობით, გამოდის, რომ გაფართოების დასაწყისში სამყაროს უნდა ჰქონოდა უსასრულოდ მცირე, კერძოდ წერტილოვანი, ზომები და უსასრულოდ დიდი, როგორც სიმჭიდროვე, ისე ტემპერატურა. ასე დაიბადა **ინოვაციური ფილოსოფიური თეორია**, რომლის თანახმადაც დროში უკან გადაადგილების კვალობაზე სამყარო თანდათან უნდა დაპატარავებულიყო და, ბოლოს, ერთ წერტილში, კერძოდ „**პირველყოფილ ატომში**“, შეყურსულიყო. ამ „პირველყოფილი ატომის“ გაფართოების შედეგად უნდა წარმოქმნილიყო როგორც დრო, ისე სივრცის თანამედროვე ქსოვილი. გაფართოება უნდა მომხდარიყო წარმოუდგენლად დიდი სიჩქარით, რომელსაც **დიდი აფეთქების** სახე უნდა ჰქონოდა ამ დიდი აფეთქებით უნდა დაწყებულიყო ჩვენი სამყაროს განვითარების პროცესი, რომელიც დღესაც გრძელდება.

ლემეტრის იდეას ფიზიკოსების უმრავლესობა თავდაპირველად სკეპტიკურად შეხვდა: **ჰაბლის** შედეგების პრაქტიკულად დამტკიცების გარეშე საფუძველი ეცლებოდა **ლემეტრის** იდეასაც, ხოლო ამ მიმართულებით ჩატარებული სამუშაოები უშედეგოდ მთავრდებოდა.

ყველაფერი სრულიად მოულოდნელად გადაწყდა დადებთად. **1964** წელს ამერიკული სამრწველო კვლევითი და სამეცნიერო კომპანიის **Bell Labs**-ის ორი რადიოინჟინერი, შემდგომში ამერიკელი ასტროფიზიკოსები, პროფესორები ნობელის პრემიის ლაურეატები, **არნო ალან პენზიასი** (*ინგ. Arno Allan Penzias; 1933-2024*) და **რობერტ ვუდრო უილსონი** (*ინგ. Robert Woord Wilson; 1936*), ახდენდნენ ახალი მიკროტალღოვანი რადიომიმღების გამოცდას. დიდი მცდელობების მიუხედავად ისინი ვერ იცილებდნენ **ფონურ ხმაურს**, რომელსაც განუწყვეტლივ აფიქსირებდა რადიომიმღები. პრობლემის ახსნის ძიების პროცესში ისინი შეხვდნენ ფიზიკოსების ჯგუფს, რომლებიც ეძებნენ სათანდო ფინანსებს სწორედ ასეთი ხელსაწყოს ასაგებად. აღმოჩნდა, რომ **ფონურ ხმაურს** იწვევდა გამოსხივება, რომელიც სამყაროს გავარვარებული მჭიდრო **პლაზმიდან** ნაკლებად ცხელ ნეიტრალურ **აირად გადაქცევის** შემდეგ დარჩა. მას ეწოდება **კოსმოსური მიკროტალღოვანი ფონი** და იგი **დიდი აფეთქების** გაგებისათვის იყო საჭირო. იგი არა მარტო დიდი აფეთქების თეორიის ჭეშმარიტობას ამტკიცებს,

არამედ უშორეს წარსულში გადასახედი სარკმლის ფუნქციასაც ასრულებს, როდესაც ჩვენი სამყარო თავის დღევანდელ ზომებზე და-



• Quantam Eluktuation - **კვანტური ფლუქტუაცია;** • Infation - **ინფლაცია**; • Afterglow Light Pattern **380 000** yzs - **რელიქტიური ფონის გამოსხივება 380 000 წელი;** • Dark Ages - **ბნელი საუკუნეები;** •1st stars abut **400** million yrs - **პირველი ვარსკვლავები დაახლ. 400 მლნ წელი;** • Development of Gallxiens, Planets, ets - **გალაქტიკების, პლანეტების და ა.შ. ფორმირება;** • Dark Energi Accelerated Expansion - **ბნელი ენერგიის დომინირება**; • Big Band Expranse **13.77** billion years - **გაფართოების დაწყებიდან 13.77 მლრდ წელი;**

ნახ. 1.5. *სამყაროს გაფართოების ზოგადი სურათი (წლები ათვლილია „დიდი აფთქების“ მომენტიდან)*

ახლოებით მილიონჯერ ნაკლები იყო. მისი ტემპერატურა **10 000 C**^{0}-ს უდრიდა და იგი პლაზმის მდგომარეობაში იმყოფებოდა. გაფართოებისა და გაგრილების კვალობაზე პლაზმა გარდაიქმნა ნეიტრალურ აირად, რომელშიც წარმოიქმნა პირველი ატომები. ამის გამო გამოიყო დიდი რაოდენობის გამოსხივება, რომელიც **კოსმოსური მიკროტალღოვანი** (**რელიქტური**) **ფონის** გამოსხივების სახით აქამდეა შენარჩუნებული. მასზე მოდის სამყაროში არსებული გამოსხივების

99,999%. აღნიშნული ფონი დიდი აფეთქებიდან **380 000** წლის შემდეგ წარმოიშვა (ნახ. **1. 5**). **მეტაფორულად** ეს ფონი, რომლიდან წამოსულმა სხივებმა ჩვენამდე დღეს მოაღწია, დიდი აფეთქებასთვის მაშინ გადაღებულ სურათად შეიძლება მივიჩნიოთ.

სამყაროს ასაკი უდრის სამყაროს გაფართოების დაწყებიდან (დიდი აფეთქებიდან) დღევანდელ დღემდე გავლილ წლებს. დიდი ხნის განმავლობაში მის ასაკად ≈**13,787** წელი ითვლებოდა (**1. 5** ნახაზზე ამ ასაკად **13,77** წელია მითითებული). კოსმოსური მიკროტალღოვანი ფონის დაწვრილებითი ანალიზით დადგინდა, რომ **დედამიწის ასაკი 13,82 მილიარდ წელზე ნაკლები არ უნდა იყოს.** ამ ასაკს თუ მიკროტალღოვანი ფონის ასაკს შევადარებთ, მაშინ ამ ფონის (ანუ, დიდი აფეთქების) აღნიშნული სურათი საბავშვო სურათის ეკვივალენტური გამოვა, რომელიც აფეთქებას მოხდენიდან („დაბადებიდან“) სულ რაღაც ... **10** საათის შემდეგ გადაუღეს!

1.6. 2. გაზომვების ნატურალური ერთეულები „დიდი აფეთქების“ სამსახურში

პენზიასისა და **უილსონის** ექსპერიმენტულმა მონაცემებმა გადამწყვეტი როლი ითამაშა დიდი აფეთქების თეორიის ჩამოყალიბებაში, მაგრამ, როგორც რუსი მეცნიერი **დიმიტრი მენდელეევი** (*რუს. Дмитрий Иванович Менделеевж 1834 – 1907*) წერს, „მეცნიერება იწყება გაზომვებიდან. ზუსტი მეცნიერება წარმოუდგენელია გაზომვის გარეშე“. **გაზომვის პროცესის** ორგანიზებისათვის **გაზომვის ერთეულთა სისტემის**, ანუ შემოკლებით, **ერთეულთა სისტემის** ფორმირებაა აუცილებელი. ჩვენთვის ყველასათვის ცნობილი ერთეულების **საერთაშორისო Si სისტემა**, რომელიც ძირითადად მეტრისა და კილოგრამის ეტალონებზეა დაფუძნებული, კოსმოლოგიურ სისტემებში არსებული პარამეტრების გასაზომად მოუხერხებელია. საბედნიეროდ, ამ პრობლემების აღმოჩენამდე დიდი ხნით ადრე დამუშავდა **ნატურალური (ბუნებრივი) ერთეულები,** რომლებშიც გამოიყენება არა სპეციალურად დამზადებული ეტალონები, არამედ ბუნების ფუნდამენტური **ფიზიკური მუდმივები.**

ფუნდამენტური ფიზიკური მუდმივები ბუნების ფიზიკური კანონებისა და მატერიის თვისებების აღმწერ განტოლებებში შემავალ **მუდმივი სიდიდეებს** ეწოდება. ცნობილი ფუნდამენტური სიდიდეებია, **სინათლის c სიჩქარე** ($c = 299\ 792\ 458 \pm 1{,}2\ \frac{მ}{წმ} \approx 300\ 000\ \frac{მ}{წმ}$), **პლანკის** h

მუდმივა (h = 6,626 070 15 × 10^{-34} $\frac{\text{ჯოული}}{\text{წმ}}$) და ა. შ. სულ **26** ფუნდამენტური მუდმივა არსებობს, რომელთაგან სინათლის **c** სიჩქარე, გრავიტაციული **G** მუდმივა და პლანკის **h** მუდმივა ფიზიკის საფუძვლების აღწერისათვის საკმარის ბაზისს წარმოქმნის.

არსებობს რამდენიმე სახის ნატურალური ერთეულები, რომელთა დაწვრილებით განხილვა სცილდება ჩვენი სახელმძღვანელოს ფარგლებს. ყურადღებას შევაჩერებთ მხოლოდ ორ მათგანზე: **სტოინისეულ ერთეულებსა** და **პლანკისეულ ერთეულებზე.** იგი განპირობებულია იმ გარემოებით, რომ **სტოუნის ერთეულებმა** სათავე დაუდეს ნატურალური ერთეულების დამუშავების პროცესს, ხოლო **პლანკისეული ერთეულები** ფართოდ გამოიყენება ზოგადად ასტრონომიასა და, კერძოდ, კოსმოლოგიაში.

ისტორიულად პირველი **ნატურალური ერთეულები 1881** წელს შემოგვთავაზა ბრიტანელმა ფიზიკოსმა, მეცნიერებაში ტერმინ „**ელექტრონის**“ შემოტანმა **ჯორჯ ჯონსტონ სტონმა** (*ინგ. George Johnstone Stoney*, 1826—1911), რომლებსაც ავტორის საპატივცემლოდ **სტოუნის ერთეულები** ეწოდა. მასში ძირითადი ერთეულების მისაღებად **სტოუნმა** გამოიყენა ფუნდამენტური მუდმივები: $\boldsymbol{c}$ -სინათლის სიჩქარე ვაკუუმში, **G** - გრავიტაციული მუდმივა, $\boldsymbol{k_e} = \frac{\mathbf{1}}{\mathbf{4\pi\varepsilon_0}}$ - კულონის მუდმივა და $\boldsymbol{e}$ - ელემენტარული მუხტი.

მეორე **ნატურალური ერთეულები 1899** წელს დაამუშავა გერმანელმა ფიზიკოსმა **მაქს კარლ ერნსტ ლუდვიგ პლანკმა** (*გერმ. Max Karl Ernst Ludwig Planck*; *1858-1947*), რისთვისაც გამოიყენა ფუნდამენტური მუდმივები:

- **დირაკის მუდმივა ħ**:

$$\hbar \equiv \frac{h}{2\pi} = 1,\ 054\ 571\ 800\ (13)\ \times 10^{-34}\ \text{ჯ}\ \times \text{წმ} =$$
$$= 6,585\ 119\ 514\ (40)\ \times 10^{-16}\ \text{ელ.ვ.} \times \text{წმ},$$

სადაც **h = 6,626 070 15·10^{-34}** კგ· მ^2·წმ^{-1} **- პლანკის მუდმივა;** აქ და ქვემოთ არსებულ ფორმულებში ფრჩხილებში ჩასმული ციფრები აღნიშნავს უკანასკნელი ორი თანრიგის განუსაზღვრელობას (საშუალოკვადრატულ გადახრას);

- **სინათლის სიჩქარე c** = 299 792 458 მ/წმ (≈ 3 × 10^8 მ/წმ);
- **გრავიტაციული მუდმივა G** = 6,67430 (15) × 10^{-11} მ^3 წმ^{-2} კგ^{-1};
- **ბოლცმანის მუდმივა** $\boldsymbol{k_B}$ (ან **k**) = 1,380 649 × 10^{-23} ჯ/კელ.

ზემოთ მოყვანილი მუდმივებიდან მიღება პლანკისეული ერთეულები, რომლებიც იყოფა ძირითად და ნაწარმოებ ერთეულებად.

პლანკისეული ძირითადი ერთეულებია:

● პლანკისეული სიგრძე:

$$\boldsymbol{l_P} = \frac{1}{k}\sqrt{\frac{\hbar G}{c^3}} \approx 1,\ 616\ 225\ (18) \times 10^{-35} \text{ მეტრი};$$

● პლანკისეული მასა:

$$\mathbf{m_P} = \sqrt{\frac{\hbar c}{G}} \approx 2{,}176\ 434\ (24) \times 10^{-8} \text{ კგ};$$

● პლანკისეული დრო:

$$\mathbf{t_P} = \sqrt{\frac{\hbar G}{c^5}} \approx 5{,}391\ 247\ (60) \times 10^{-44} \text{ წამი};$$

● პლანკისეული ტემპერატურა:

$$\mathbf{T_P} = \frac{1}{k}\sqrt{\frac{\hbar c^5}{G}} \approx 1,\ 416\ 784 \times 10^{32} \text{ კელვინი}.$$

პლანკისეული ნაწარმოები ერთეულებია:

● ფართობი: $\boldsymbol{l_P^2} = \frac{\hbar G}{c^3} \approx \mathbf{2{,}6121 \times 10^{-70}}$ მ^2;

● მოცულობა: $\boldsymbol{l_P^3} = \left(\frac{\hbar G}{c^3}\right)^3 = \sqrt{\frac{(\hbar G)^3}{c^9}} \approx \mathbf{4{,}2217 \times 10^{-103}}$ მ^3;

● ენერგია: $\mathbf{E_P} = \mathbf{m_P}\ \mathbf{c^2} = \frac{\hbar}{t_P} = \sqrt{\frac{\hbar c^5}{G}} \approx \mathbf{1{,}9561 \times 10^9}$ ჯოული;

● ძალა: $\boldsymbol{F_P} = \frac{E_P}{l_P} = \frac{\hbar}{l_P\, t_P} = \frac{c^4}{G} = \mathbf{1{,}2103 \times 10^{44}}$ ნიუტონი;

● სიმჭიდროვე (სიმკვრივე)**:**

$$\boldsymbol{\rho_P} = \frac{\mathbf{m_P}}{\mathbf{t_P^3}} = \frac{\hbar\, t_P}{l_P^5} = \frac{c^5}{\hbar c^2} \approx \mathbf{5{,}1550 \times 10^{96}} \text{ კგ/მ}^3 ;$$

პლანკისრული $\mathbf{t_P}$ დრო არის სინათლის სიჩქარით მოძრავი ფოტონის მიერ პლანკისეული $\boldsymbol{l_P}$ სიგრძის გასავლელად საჭირო დრო. ეს დროის ყველაზე უმნიშვნელო ზომა - „დროის კვანტია“, რომელსაც შეიძლება რაიმე აზრი ჰქონდეს.

პლანკისეული $\boldsymbol{l_P}$ სიგრძე შეესაბამება მანძილს, რომლითაც შეიძლება ორი წერტილი ერთმანეთს მიუახლოვდეს, ე. ი. ამ მანძილებით დაშორებული წერტილებისაგან შემდგარი სივრცე უწყვეტ სივრცედ აღიქმება. **პლანკისეული $\boldsymbol{l_P}$ სიგრძე**, და მასთან დაკავშირებული **პლანკისეული t_P დრო**, განსაზღვრავს მასშტაბებს, რომლის ფარგლებში თანამედროვე ფიზიკური თეორიები კარგავს თავიანთ ჭეშმარიტობას: ფარდობითობის ზოგადი თეორიით ნაკარნახევი **სივრცე-დროის**

გეომეტრია პლანკისეული სიგრძის ტოლ და უფრო ნაკლებ მანძილებზე, კვანტური ეფექტების გამო, აზრს კარგავს; დღემდე ჩამოუყალიბებელია ფარდობითობის თეორიისა და კვანტური მექანიკის - კვანტური გრავიტაციის - გამაერთიანებელი თეორია.

პლანკისეული ρ_P სიმჭიდროვე დაახლოებით ერთ ატომურ ბირთვში შეკუმშულ მზის $\mathbf{10^{23}}$ მასას შეესაბამება, ხოლო **პლანკისეული $\mathbf{T_P}$ ტემპერატურის** მქონე სისტემისათვის ნებისმიერი ენერგიის გადაცემა სისტემის ტემპერატურას კი არ გაზრდის, არამედ შეამცირებს და სისტემა შემდგომი გახურების ნაცვლად გაგრილდება. კოსმოლოგიის თანამდროვე წარმოდგენებით ესაა სამჭიდროვე და ტემპერატურა, რომელიც სამყაროს ჰქონდა დიდი აფეთქების პირველ (პლანკისეულ) მომენტში.

ფიზიკურ კოსმოლოგია ჩვენ მიერ დაკვირვებადი სამყაროს ისტორიის ყველაზე ადრეულ ეპოქად, რომლის შესახებადაც რაღაც თეორიული ვარაუდების გამოთქმაა შესაძლებელი, მიიჩნევს **13,82** მილიარდი წლის წინათ არსებულ **პლანკისეულ ეპოქას**. იგი გრძელდებოდა პირველი პლანკისეული დროის განმავლობაში, **0**-დან $\mathbf{t_P} \approx 5{,}391\ 247\ (60) \times 10^{-44}$ წამამდე. ამ ეპოქაში სამყაროს ენერგია უდრიდა $\mathbf{E_P} \approx \mathbf{1{,}22} \times \mathbf{10^{19}}$ გიგაელექტრონ-ვოლტს, ტემპერატურა - $\mathbf{T_P} \approx 1{,}416\ 784 \times 10^{32}$ კელვინს (ცელსიუსით დაახლოებით 1000 ტრილიონ გრადუსს), სიმჭიდროვე - $\boldsymbol{\rho_P} \approx \mathbf{5{,}1550} \times \mathbf{10^{96}}$ კგ/მ^3-ს, ხოლო სამყაროს რადიუსის სიგრძე $\boldsymbol{l_P} \approx 1{,}616\ 225\ (18) \times 10^{-35}$ მეტრის ტოლი იყო.

1.6. 3. სამყაროს „სამშენებლო მასალა“

კაცობრიობას უძველესი დროიდან აინტერესებდა ამოეცნო სამყაროს აგების საიდმულო და დაედგინა ამ მიზნისთვის გამოყენებული და „სამშენებლო აგურების“ რაობა.

ჯერ კიდევ ჩვენს წელთაღრიცხვამდე **400** წლით ადრე ბერძენმა ფილოსოფოსმა დემოკრიტემ (*ძვ. ბერძ.* Δημόκριτος; *≈ 460 – 370 ჩვ. წ.აღ.*) ასეთ „სამშენებლო მასალად“ მიიჩნია **ატომი** (*ძვ. ბერძ* ***ἄτομος*** **– *„განუყოფელი“***), რომელსაც იგი მატერიის უმცირეს განუყოფელ ნაწილაკად თვლიდა. მატერიალურ ობიექტს, რომლის შემადგენელ ნაწილებად დაშლა შეუძლებელია, **ელემენტალური ნაწილაკი** ეწოდება. ასეთ ნაწილაკად **ატომი** დიდი ხნის განმავლობაში ითვლებოდა.

ატომის ჭეშმარიტი სტრუქტურის დადგენა მოხერხდა სამი ინგლისელი ფიზიკოსის მოპოვებული აღმოჩენების მეშვეობით. კერძოდ,

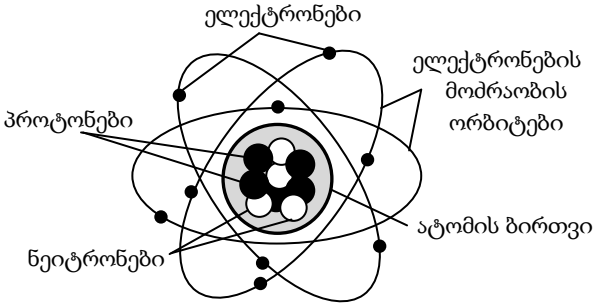
1897 წელს **ჯოზეფ ჯონ ტომსონის** (ინგ. *Joseph John Thomson; 1856-1940*) მიერ ელექტრონის, **1919** წელს **ერნსტ რეზერფორდის** (*ინგ. Ernest Rutherford; 1871-1937*) მიერ - **პროტონისა** და **1932** წელს **ჯეიმს ჩედვიკის** (ინგ. *James Chadwick; 1891-1974*) მიერ - **ნეიტრონის** აღმოჩენის შემდეგ ატომის სტრუქტურამ მიიღო **6.1** ნახაზზე მოყვანილი სახე. მის თანახმად ატომი შედგება სამი ძირითადი ნაწილაკისაგან: ელექტრონების, პროტონებისა და ნეიტრონებისაგან. პროტონები და ნეიტრონები წარმოქმნის **ატომის ბირთვს**, რომლის გარშემოც გარკვეულ ორბიტებზე მოძრაობს **ელექტრონები**.

ელექტრონებს, პროტონებსა და ნეიტრონებს, რომელთა ზომები ატომის ზომებზე ნაკლებია, **სუბატომური ნაწილაკები** ჰქვია. მათგან პროტონებსა და ნეიტრონებს, რომლებიც ატომის ბირთვშია გაერთიანებული, დამატებით **ნუკლონებსაც** (*ლათ. Nucleus – „ბირთვი“*) უწოდებენ. მოკლედ დავახასიათოთ თითოეული მათგანი.

პროტონები (*ძვ. ბერძ. πρῶτος – „პირველი“*) **1,67 × 10^{-27}** კგ მასის სუბატომური ნაწილაკებია, რომელიც დამუხტულია **e = +1,6022·10^{-19} კულონი** სიდიდის ტოლი მუხტით. მუხტის ეს სიდიდე ერთეულადაა მიღებული და პირობითად თვლიან, რომ პროტონი +1 მუხტის მატარებელ ნაწილაკია. იგი შედის ატომის ბირთვის შემადგენლობაში და ატომის იდენტობის განსაზღვრაში გადამწყვეტ როლს თამაშობს. თითოეული ატომის ბირთვში არსებული უნიკალური რაოდენობის პროტონებით წარმოიქმნება კონკრეტული ქიმიური ელემენტი. სხვადასხვა რაოდენობის პროტენების მქონე ატომი სხვადასხვა ქიმიურ ელემენტს წარმოქმნის.

ელექტრონები 9,1 × 10^{-31} კგ მასის მქონე სუბატომური ნაწილაკია, რომელიც დამუხტულია **e = - 1,6022·10^{-19}** კულონის ტოლი მუხტით. მისი სიდიდე ემთხვევა პროტონის მუხტის სიდიდეს და მისგან მხოლოდ ნიშნით განსხვავდება. მუხტის ეს სიდიდეც ერთეულადაა მიღებული და ელექტრონი პირობითად -1 მუხტის მატარებელ ნაწილაკად ითვლება. ელექტრონები პროტონებზე მსუბუქი ნაწილაკებია და არ შედის ატომის ბირთვში. ისინი განსაზღვრავს ატომის ზომებსა და მის ქიმიურ-ფიზიკურ მახასიათებლებს. ატომში პროტონების რაოდენობა ელექტრონების რაოდენობის ტოლია, რის გამოც მათი ჯამური მუხტი ნულის ტოლია და ატომი ელექტრულად ნეიტრალურია. ატომიდან ელექტრონის გაძევებისას პროტონების რაოდენობა გადააჭარბებს ელექტრონების რაოდენობას და ატომი დადებითად

დაიმუხტება, ანუ **იონად** გადაიქცევა. ამით ქიმიური ელემენტის სახე უცვლელი დარჩება, მაგრამ შეიცვლება მისი თვისებები.



ნახ. 1. 6. *ატომის პლანეტარული მოდელი*

ნეიტრონების მასა ემთხვევა პროტონების მასას, მაგრამ მათგან განსხვავებით ისინი ნეიტრალური (უმუხტო) ნაწილაკებია. ნეიტრონების ძირითადი დანიშნულებაა ერთმანეთთან მჭიდროდ დააკავშიროს, თუ შეიძლება ასე ვთქვათ, „შეაწებოს“ პროტონები: ისინი ხომ ელექტული ძალების მოქმედებით ერთმანეთს განიზიდავს (როგორც ერთნაირი მუხტიანი ნაწილაკები) და, ნეიტრონების არარსებობის შემთხვევაში, სხვადასხვა მხარეს გაფრინდებოდა. ნეიტრონები პროტონებს ბირთვში უკვე არაელექტრული ძალის მეშვეობით აკავებს. ამ ძალას **ძლიერი ბირთვული ძალა** ეწოდება. მის შესახებ შედარებით დაწვრილებით ქვემოთ ვისაუბრებთ.

ატომში არსებული ყველა ნაწილაკი თანაბარმნიშვნელოვანია. მათი ერთობლობით წარმოიქმნება თვისებების უნიკალური ნაკრები, რომელიც, როგორც მიკროსკოპულ, ისე მაკროსკოპულ დონეზე, განსაზღვრავს **მატერიის** მახასიათებლებსა და ქცევას. ამ ერთობლიობით იბადება **სინერგია**, რომელიც **ატომებს** მატერიის **სამშენებლო ბლოკებად** და, აქედან გამომდინარე, ყოველივე არსებულის, საფუძვლად გარდაქმნის.

ფიზიკოსებმა თეორიულად და ექსპერიმენტულადაც დაადგინეს, რომ ბუნებაში არსებობს ატომში შემავალი ნაწილაკების ანტინაწილაკები: **ანტიპროტონები**, **ანტიელექტრონები** (რომლებსაც **პოზიტრონები** ეწოდება) და **ანტინეიტრონები**. ნაწილაკისა და მისი ანტინაწილაკის სკალარულ და ვექტორულ სიდიდეების აბსოლუტური

მნიშვნელობები ერთმანეთის ემთხვევა, ვექტორულ სიდიდეებს კი აქვს ურთიერთსაპირისპირო მიმართულებები.

ნაწილაკების ერთობლიობით წარმოქმნლ სუბსტანცის **მატერია,** ხოლო ანტინაწილაკებით წარმოქმნილ სუბსტანციას - **ანტიმატერი**ა ეწოდება.

ნაწილაკებისა და ანტინაწილაკების, აგრეთვე მატერიისა და ანტიმატერიის შეჯახებისას ხდება ხდება **ანგილაცია** (*ლათ. Annihilatio – „სრული მოსპობა*) – მათი **მასების** ფოტონებად გადაქცევა და ჰაერში გაფანტვა; ასეთივე პროცესი ხდება მატერიისა და ანტიმატერიის შეჯახებისას.

ატომის სტრუქტურის (იხ. **ნახ. 1.6**) დადგენის შემდეგ ფიზიკოსებმა მთელი ყურადღება ატომში არსებულ სუბატომურ ნაწილაკებზე გადაიტანეს და დაადგინეს, რომ ატომგულში არსებული **სუბატომებიდან ბუნებრივად წარმოიშვება ახალი სუბატომური ნაწილაკები**. მოკლედ განვიხილოთ მათი წარმოშობის პროცესი.

ატომებისაგან აგებული მატერიაზე პერმანენტულად ზემოქმედებს **კოსმოსური სხივები**, რომლებიც დიდი ენერგიის მქონე სწრაფი ნაწილაკებისაგან (ძირითადად, **პროტონებისაგან**) შემდგარი ნაკადებია. ამ ნაწილაკებით უწყვეტად „იბომბება“ მატერიის შემადგენელ ატომთა ბირთვები. მკვლევრებმა დაადგინეს, რომ $\mathbf{E = mc^2}$ ფორმულის თანახმად ამ დროს სწრაფი ნაწილაკების კინეტიკური ენერგია **მასად** გარდაიქმნება. ამის შედეგად ატომის გულში ახალი სუბატომური ელემენტები წარმოიშობა.

კოსმოსური სხივების მსგავსი სხივების ხელოვნურად შესაქმნელად მეცნიერებმა ააგეს **„ელემენტალური ნაწილაკების ამაჩქარებლები**“. მათზე არსებულ ღილაკზე თითის დაჭერით წარმოიშობა მაღალი ენერგიის მქონე სწრაფი ნაწილაკთა მართვადი ნაკადები. მოკლე დროში მალე **ორასზე მეტი** ასეთი ამაჩქარებელი გაჩნდა და ზემოთ აღნიშნულმა **ექსპერიმენტების ჩატარებამ სისტემური ხასიათი მიიღო.** მათი მეშვეობით მალე რამდენიმე ასეული ახალი სუბატომური ნაწილაკი ფორმირდა. პროტონისაგან განსხვავებით, ყველა მათგანი არასტაბილური აღმოჩნდა: ისინი წარმოშობისთანავე (დაახლოებით $\mathbf{10^{-22} - 10^{-23}}$ წამის ან უფრო ნაკლებ დროის განმავლობაში) სხვა ნაწილაკებად იშლება. ეს დრო ბირთვის ფარგლებში მათი გადანაცვლებისა და ბირთვულ პროცესებში მონაწილეობის მიღებისათვის სრულიად საკმარისია. ასე რომ, **ატომის ბირთვი** პროტონისა და ნე-

იტრონის უბრალო „საცხოვრისიდან“ რამდენიმე ასეულობით **სუბატომური ნაწილაკების წარმოშობისა და მათი დინამიკურ ურთიერთზემოქმედებების არენადაც გადაიქცა**.

ახლად წარმოქმნილი არასტაბულური ნაწილაკები, პროტონებისა და ნეიტრონებსავით, „მძიმე“ ნაწილაკებია და მათი მასები მნიშვნელოვნად აღემატება ელექტრონის მასას. მათთან ერთად პროტონები და ნეიტრონები წარმოქმნის ბირთვში შემავალი ნაწილაკების ჯგუფს, რომელსაც **ადრონები** (*ლათ. Hadros - „დიდი“*) ეწოდა. ეს ჯგუფი საკმაოდ მრავალრიცხოვანია და რამდენიმე ასეულ წევრს შეიცავს. ამათგან სტაბილურია მხოლოდ **პროტონები** და **ნეიტრონები** (მათ სტაბილურ ადრონებს უწოდებენ). დანარჩენი ადრონები არასტაბილურებია.

არამდგრადი ადრონების კლასიფიცირება მათი დაშლის შედეგად მიღებული პროდუქტების შედგენილობის მიხედვითა ხდება: ადრონებს, რომლთა დაშლის შედეგად მიღებულ პროდუქტები პროტონს შეიცავს, **ბარიონები** (*ბერძ. barys – „მძიმე“*), ხოლო რომელიც არ შეიცავს - **მეზონები** (*ბერძ. mesos „საშუალო“*) ეწოდება. პროტონსა და ნეიტრონს **ბარიონებად** მიიჩნევენ. **მეზონების** მასას პროტონისა და ნეიტრონის მასების შუალედური სიდიდე აქვს.

ნაწილაკები, მიუხედავად სივრცული განცალკევებისა, ერთმანეთზე ზემოქმედებს. სივრცეში ურთიერთზემოქმედების ძალები გადააქვს **ბოზონებად** წოდებულ ნაწილაკებს. მეზონები **ბოზონების ჯგუფს მიეკუთვნება**. ნაწილაკებს შორის არსებული მიზიდვები და განზიდვები ამ ნაწილაკებს შორის **ბოზონების** გაცვლის შედეგია

1964 წელს ორმა ამერიკელმა ფიზიკოსმა **მიურეი გელ-მანმა** (*ინგ. Murray Gell-Mann; 1929-2019*) და **ჯორჯ ცვეიგმა** (*ინგ.George Zweig; 1993*) აღმოაჩინეს, რომ **ადრონები** აგებულია ელემენტალური ნაწილაკებისაგან. **ცვეიგმა** მათ „**ace**“ (*ინგ. „პირველი კლასი“, „მთავარი კოზირი“*) უწოდა. **გელ-მანმა** ამისათვის ირლანდიელი მწერლის **ჯეიმს ჯოისის** (*ინგ. James Augustine Aloysius Joyce; 1882-1941*) რომან „**Finnegans Wake**“-ში არსებული ფრაზიდან - „**სამი კვარკი** მისტერ მარკისათვის**!**“ („Three **quarks** for Muster Mark!“) - აიღო სიტყვა „**კვარკი**“ (**quark**), რომელიც **თოლიას ხმამიბაძვაა**. მან აღმოჩენილ ელემენტარულ ნაწილაკებს **კვარკები** უწოდა. იგი მეტად ორიგინალური აღმოჩნდა, რადგან პროტონი და ნეიტრონი ზუსტად სამ-სამი ასეთი ნაწილაკასგან აღმოჩნდა აგებული! ამან მისტერ მარკისათვის შეთავაზებული **სამი კვარკის** ფასი

საკმაოდ გაზარდა: **მარკი** ნაჩუქარი სამი კვარკისაგან სრულფასოვანი პროტონის ან ნეიტრონი აგებას შეძლებდა! დღეისთვის ტერმინი „**კვარკი**“ ფართოდ გამოიყენება ადრონების ასაგებად გამოყენებული ელემენტალური ნაწილაკების სახელწოდებად.

კვარკებს დამოუკიდებლად (ადრონის გარეთ) არსებობა არ შეუძლია. მათი არსებობის ფაქტის შესახებ მხოლოდ **ადრონის** თვისებების მიხედვით შეგვიძლია ვიმსჯელოთ. ამ ფენომებს **კვარკების თვითდატყვევება** ან **თვითჩამწყვდევა** ეწოდება. მის საჩვენებლად წარმოვიდგინოთ, რომ გვაქვს ელასტიკური ზონარი (რეზინი), რომლის ბოლოები კვარკებია. სისტემიდან გამოცალკევებისათვის მათზე მოვდოთ გარკვეული ენერგია და მისი სიდიდე თანდათან გავზარდოთ. სიდიდის გარკვეულ სიდიდეზე, რეზინი სადღაც შუაში გაწყდება, მაგრამ აღნიშნულ კვარკებს არ გამოცალკევდება.

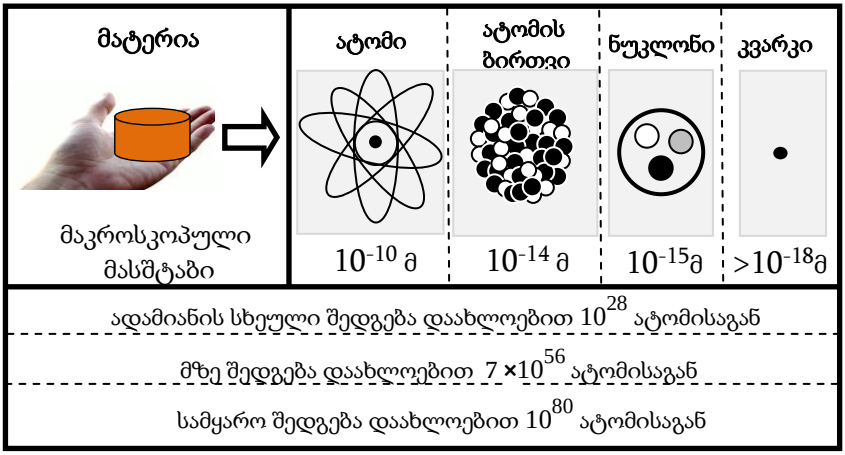
ბუნებაში სულ **ექვსი კვარკი** არსებობს. ისინი ორ-ორადაა დაწყვილებული და ე. წ. **დიბლეტებს** წარმოქნის. **პირველ დიბლეტში** შემავალ კვარკებს **ზედა *u*** (*ინგ. up*) და **ქვედა *d*** (*ინგ. down*) კვარკებს უწოდებენ. ისინი ყველაზე მუბუქი კვარკებია. **მეორე დიბლეტში** შემავალი კვარკების სახელებია სახელებია **მოხიბლული *c*** (*ინგ. Charmed*) და **უცნაური *s*** (*ინგ. strange*). **მესამე დიბლეტში** შემავალი კვარკებს უწოდებენ **ჭეშმარიტს - *t*** (*ინგ. truth*) და **ლამაზს *b*** (*ინგ. beauty*).

პროტონს წარმოქმნის კვარკების ***uud***, ნეიტრონს კი - ***udd*** კომბინაცია, ამიტომ ***u***-ს **პროტონულ**, ***d***-ს კი - **ნეიტრონულ** კვარკს უწოდებენ. ***u***-სა და ***d***-ს მუხტები შესაბამისად $+\frac{2}{3}$-სა და $-\frac{1}{3}$-ს უდრის. რადგან **პროტონი** ორი ***u*** და ერთი ***d***, ხოლო **ნეიტრონი** - ერთი ***u*** და ორი ***d*** კვარკისაგან შედგება, ამიტომ მათი ჯამური მუხტი შესაბამისად **1**-ისა და **0**-ის ტოლი გამოდის.

ფიზიკოსების აბსოლუტური უმრავლესობა მატერიის შემდგომ დეტალიზებაზე უარს აცხადებს და კვარკებს მიიჩნევენ იმ უმცირეს „აგურებად“, რომლებითაცაა აგებული მატერია. **კვარკი** უსტრუქტურო ელემენტალურ ნაწილაკად ითვლება, რომლიდანაცაა აგებული მატერია; ამიტომ მათ პროტონებისა და ნეიტრონების წარმომქმნელ **ფუნდამენტურ ნაწილაკებს** უწოდებენ.

ატომის ზომებზე ნაკლები ზომის ნაწილაკებს, როგორც აღვნიშნეთ, **სუბატომური ნაწილაკები** ეწოდება. ასეთი ნაწილაკებია **ელექტრონები**, **პროტონები** და **ნეიტრონები**. ამ ნაწილაკებიდან ელექ-

ტრონების შემდგომი დანაწევრება არ შეიძლება მათ **ლეპტონები** (*ბერძ. Λεπτός – „მსუბუქი“*) ეწოდება. კვარკების მსგავსად ბუნებაში **ექვსი ლეპტონი** არსებობს. ისინი სამ თაობადაა დაყოფილი. პირველი თაობას მიეკუთვნება **ელექტრონი** (e^-) და **ელექტრონისეული ნეიტრინო** (ν_e); მეორე თაობას - **მიუონი** (μ^-) და **მიუონისეული ნეიტრინო** (ν_μ), მესამე თაობას კი **ტაუ-ლეპტონი** (τ^-) და **ტაუ-ნეიტრინო** (ν_τ).



ნახ. 1. 7. *მატერია და ელემენტარული ნაწილაკები*

ატომის ბირთვში შემავალ სტაბილურ ნუკლონებს - **პროტონებსა** და **ნეიტრონებს** და დანარჩენ არასტაბილურ ნუკლონებს - **სუბბირთვულ ნაწილაკები** ეწოდება.

ლეპტონები და კვარკები უსტრუქტურო ნაწილაკებად ითვლება და მას **ელემენტალური ნაწილაკები** ეწოდება. რადგან მათი მეშვეობითაა აგებული სამყარო, ამიტომ მათ **ფუნდამენტურ ნაწილაკები** დაარქვეს. ზოგიერთი ნაწილაკის ზომები **1.7**. ნახაზზეა მოცემული. რეალური მატერიალური ობიექტების სირთულე შეიძლება მისი ფორმირებისათვის საჭირო ფუნდამენტური ნაწილაკების რაოდენობით შეფასდეს. მეტი თვალსაჩინოებისათვის ხშირად მათ ნაცვლად ყველასათვის ცნობილი **ატომებს** იყენებენ. ადამიანის სხეული **10^{28}**, მზე - **7×10^{56}**, ხოლო მთელი სამყარო - **10^{80}** რაოდენობის ატომისაგან შედგება (იხ. **ნახ. 1.7**).

1.6.4. ფუნდამენტური ძალების ცნება

ზემოთ განხილული **კვარკები** და **ლეპტონები** სამყაროს ასაგებად გამოყენებული ძირითადი **სამშენებლო მასალაა**. მათ შორის არსებული ურთიერზემოქმედებით წარმოიქმნება მატერია და, მაშასადამე, სამყარო. ამიტომ გასააზრებელია ამ ურთიერთზემოქმედებების განმაპირობებელი **ძალების** როგორც რაობა, ისე ფუნდამენტურ ნაწილაკებზე მათი ზემოქმედების მექანიზმი. მით უფრო, რომ **კლასიკური ფიზიკის** მიერ ფორმულირებული ძალის განსაზღვრება თვისებრივად განსხვავდება ელემენტალურ ნაწილაკებს შორის მოქმედი ძალის განსაზღვრისაგან. ამ განსხვავების ნათლად ჩვენებისათვის განვიხილოთ შემდეგი მაგალითი:

დავუშვათ, რომ მდინარეში ერთმანეთის საპირისპიროდ მიცურავს ორი ნავი და ნავები როდესაც ერთმანეთს გაუსწორდება, ერთ-ერთის მენავე მეორეს შამპანურიან ბოთლს გადაუგდებს. ბოთლის გადაგდებისას წარმოიშობა **ძალის იმპულსი,** რომელიც პირველი მენავის ნავს ერთ-ერთ მხარეზე გადახრის. ბოთლის დაჭერისას **მეორე მენავეს** ეს იმპულსი გადაეცემა და იგი, ნიუტონის მექანიკური კანონის თანახმად, მის ნავს საპირისპირო მხარეზე გადახრის.

ზემოთ, ფაქტობრივად, ჩვენ მოვიყვანეთ მაკროსხეულებს (ნავებს) შორის **ურთიერთზემოქმედების ძალის გადაცემის მაგალითი**. ამ ძალის გადამტანად **მაკროსხეული**, კერძოდ, **შამპანურის ბოთლი**, იყო გამოყენებული.

სრული ანალოგია არსებობს მიკროსხეულთა სამყაროშიც. ფუნდამენტაურ ნაწილაკებს შორის ოთხი სახის ურთიერთზემოქმედება არსებობს. ესენია: **ძლიერი ბირთვული** (მათი მეშეობითაა კვარკები შეკავებული ნაწილაკების შიგნით), **ელექტრომაგნიტური**, **სუსტი ბირთვული** (ისინი იწვევს ზოგიერთი ფორმის ბირთვულ დაშლას) და **გრავიტაციული ზემოქმედება**, რომელთა დროსაც იმავე სახელწოდების ძალები წარმოიშობა. სამყაროში უამრავი სახის ძალები არსებობს, მაგრამ ნებისმიერი მათგანი ზემოთ აღნიშნული ოთხი ძალიდან ერთ-ერთ მათგანზე დაიყვანება. ამიტომ ამ უკანასკნელებს **ფუნდამენტური ძალებს** უწოდებენ. მათი გადამტანებად გამოიყენება **მიკროსხეულები** - ელემენტარული ნაწილაკები. ისინი ფუნდამენტური ძალებისათვის იმავე ფუნქციას ასრულებს, რა ფუნქციაც ნავების შემთხვევაში **მაკროსხეულმა**, კერძოდ, **შამპანურის ბოთლმა** შეასრულა.

ამ ელემენტარულ ნაწილაკებს ინდოელი ფიზიკოსის **შატენდრანტ ბოზეს** (*ინგ. Satyendra Nath Bose; 1894-1974*) საპატივცემლოდ **ბოზონები** უწოდეს. მაშასადამე, **ბოზონები** ფუნდამენტურ ნაწილაკებს შორის ფუნდამენტური ძალების გადამტანი ელემენტარული ნაწილაკებია.

მიკროსამყაროში არსებობული ოთხი ფუნდამენტური ძალიდან თითოეულის გადასატანად არსებობს საკუთარი სახელის მქონე ინდივიდუალური ბოზონი. ძლიერი ბირთვული ზემოქმედების ძალის გადამტან ბოზონს **გლიუონი** (*ინგ. glue – „წებო“*), ელექტრომაგნიტური ძალის გადამტან ბოზონს - **ფოტონი** (*ძვ. ბერძ. φῶς –სინთლე, შუქი*), სუსტი ბირთვული ზემოქმედების ძალის გადამტან ბოზონებს **W**- და **Z-ბოზონები**, ხოლო გრავიტაციული ძალის გადამტან ბოზონს - **გრავიტონი** ეწოდება. ექსპერიმენტატორების მიერ დაფიქსირებულია თითოეული მათგანი, გარდა გრავიტონისა. ეს უკანასკნელი ჯერ-ჯერობით არაა დაფიქსირებული.

მოკლედ განვიხილოთ თითოეული ფუნდამენტური ძალა.

● **ელექტრომაგნიტური ძალა** (**ელექტრომაგნეტიზმი**) ეწოდება ძალას, რომლითაც ელექტრომაგნიტური ველი ზემოქმედებს დამუხტულ ნაწილაკებზე. ელექტრომაგნიტური ძალები, რომლებიც მოქმედებს ატომების შიგნით არსებულ პროტონებსა და ელექტრონებს შორის, აგრეთვე ატომებს შორის, ფაქტობრივად უზრუნველყოფს მატერიის მთლიანობას;

● **სუსტი ბირთვული ძალა** ერთ-ერთი ძალაა რომელიც ატომის შიგნით მოქმედებს პროტონებსა და ნეიტრონებს შორის. სუსტს იმიტომ უწოდებენ, რომ იგი 10^{13}-ჯერ ნაკლებია ძლიერ ბირთვულ ძალაზე და 10^{11}-ჯერ ნაკლებია ელექტრომაგნიტურ ძალაზე. იგი წამყვან როლს ასრულებს რადიოაქტიური ბეტა-დაშლასა და ნეიტრინოს წარმოქმნაში.

● **ძლიერი ბირთვული ძალა** მძლავრი, მაგრამ მოკლე მანძილებზე მოქმედი ძალაა, რომელის მეშვეობითას ატომის ბირთვ პროტონები და ნეიტრონები ვერ ტოვებს ატომის ბირთვს. იგი 10^{13}-ჯერ აღემატება სუსტ ბირთვულ ძალას და 10^{38}-ჯერ გრავიტაციას;

● **გრავიტაცია** (**გრავიტაციის ძალა**) ეწოდება ორ მასას შორის მიზიდულობის ძალას. იგი ყველაზე სუსტი ძალაა, რომლის სიდიდე იზრდება მასის ზრდასთან ერთად. იგი მნიშვნელოვან როლს თამაშ-

ობს გალაქტიკების, ვარსკვლავების, პლანეტებისა და შავი ხვრელების წარმოქმნაში

1.6. 5. დიდი აფეთქების ქრონოლოგია

13,82 მილიარდი წლის წინ მომხდარი **დიდი აფეთქების** შემდგომი განვითარების უწყვეტი პროცესი პირობითად **15** ეპოქადაა დაყოფილი ისინი ქვემოთ ქრონოლოგიის დაცვითაა განხილული. მათში **0**-ით აღნიშნულია დიდი აფეთქების მომენტი, საიდანაც სამყაროში დაიწყო დროის ათვლა.

1. პლანკის ეპოქა (ერა) - 0-იდან 10^{-43} წამამდე (ერთეულოვან პლანკისეულ დრომდე):

ესაა დროის აბსოლუტური საწყისისთან ყველაზე ახლომდებარე ეპოქა, რომელშიც შეღწევის უნარი აქვს თანამედროვე ფიზიკას. მის შესახებ ძალიან ცოტა რამაა ცნობილი. ფარდობითობის ზოგადი თეორიის თანახმად ამ ეპოქამდე არსებობდა გრავიტაციული სინგულარულობა (თუმცა კვანტურ ელემენტებს შეეძლო ისიც კი დაერღვია); ივარაუდება, რომ **ოთხივე ფუნდამენტურ ძალა** ისეთ **იდეალური სიმეტრიით შეიკრა ერთიან ფუნდამენტურ ძალად**, რომ იგი წამახულ წვერზე მდგარ და დიდი ხნით წონასწორობის შენარჩუნების უნარიან სიმეტრიულ სისტემას დაემსგავსა. სამყაროს სიგრძე 10^{-35} მ-ის (ერთეულოვან პლანკისეულ სიგრძის), ტემპერატურა კი ცელსიუსით - 10^{32} გრადუსის ტოლი იყო.

2. დიადი გაერთიანების ეპოქა - 10^{-43} წამიდან 10^{-36} წამამდე**:**

გრავიტაციის ძალა გამოეყო გაერთიანებულ დანარჩენ ფუნდამენტურ ძალებს; დაიწყო ყველაზე ადრეული ელემენტარული ნაწილაკებისა და ანტინაწილაკების ფორმირება.

3. ინფლაციური ეპოქა - 10^{-36} წამიდან 10^{-32} წამამდე**:**

მოცემული ეპოქა გამოიწვია ერთად შეკრულად დარჩენილი ფუნდამენტური ძალებიდან **ძლიერი ბირთვული ძალის** გამოყოფამ. დაიწყო წარმოუდგენლად დიდი სისწრაფით სამყაროს ექსპონენციალური გაფართოება, რომელსაც **კოსმოსური ინფლაცია** (*ლათ. inflatio – „გაბერვა“*) ეწოდა. ადრეული სამყაროს წრფივი ზომები, სულ მცირე, 10^{28}-ჯერ გაიზარდა და **10** სანტიმეტრის (დაახლოებით, გრეიფრუტის ზომის) ტოლი გახდა. დიადი გაერთიანების ეპოქიდან დარჩენილი

ელემენტარული ნაწილაკები (**კვარკ-გლიუონის** გავარვარებული მკვრივი **პლაზმა**) სამყაროში ძალიან თხელ ფენად გადანაწილდა.

4. ელექტროსუსტ ძალთა ეპოქა -10^{-36} წამიდან **10^{-12}** წამამდე**:**

ერთად შეკრულ სამ ფუნდამენტურ ძალას კიდევ ერთი - **ელექტროსუსტი ძალა** გამოეყო; ნაწილაკების ურთიერთზემოქმედებით წარმოიქმნა უამრავი ეგზოტიკური ნაწილაკები, რომელთა შორის იყო ***W***- , ***Z***- და ჰიგსის ბოზონები. მათგან მოკლედ მხოლოდ **ჰიგსის ბოზონზე** (*Higgs bosons*) შევჩერდებით, რომელიც ფუნდამენტური ნაწილაკების ოჯახის აუცილებელი წევრად ითვლება. იგი ანელებს ნაწილაკების სიჩქარეს და ფუნდამენტურ ნაწილაკებს მასას ანიჭებს. ამან თვისობრივად შეცვალა სამყარო, რომელიც მანამდე მხოლოდ სხივებისაგან შედგებდა და **მყარი მატერია** გააჩინა. იგი, რომელიც ბრიტანელმა ფიზიკოსმა **პიტერ უერ ჰიგსმა** (*ინგ. Peter Ware Higgs*; *1929-1924*) **1912** წელს აღმოაჩინა, არა მარტო მასას ანიჭებს ელემენტარულ ნაწილაკებს, არამედ, მეცნიერების აზრით, კოლაფსისაგან იცავს ჩვენს სამყაროს; ამიტომ მას მეტაფორულად **„ღმერთის ნაწილაკსაც**“ უწოდებენ.

5. კვარკების ეპოქა - 10^{-12} წამიდან **10^{-6}** წამამდე**:**

ამ ეპოქაში დიდი რაოდენობის კვარკები, ელექტრონები და ნეიტრინოები წარმოიშვა. სამყაროს ტემპერატურა **10×10^{15} C**-გრადუსამდე (*10 კვადრილიონ გრადუსამდე*) შემცირდა და ოთხივე ფუნდამენტურმა ძალა განცალკევდა. **კვარკებმა** და **ანტიკვარკებმა** ერთმანეთის **ანიგილირება** (განადგურება) დაიწყო; **ბარიოგენეზად** წოდებული პროცესის წყალობით, ყოველი მილიარდი **კვარკ-ანტიკვარკი** წყვილიდან ერთი კვარკი გადაურჩა ანიგილირებას. ამის წყალობით სამყაროში იმდენი **კვარკიმ** დარჩა. რომ მათი მეშვეობით სამყაროში წარმოქმნლმა მთელმა მატერიამ დღემდე მოაღწია!

6. ადრონული ეპოქა - 10^{-6} წამიდან **1**-წამამდე**:**

სამყაროს ტემპერატურა **10^{12} C**-გრადუსამდე (*ტრილიონ გრადუსამდე*) შემცირდა. ეს საკმარისი აღმოჩნდა იმისათვის, რომ კვარკები გაერთიანებულიყო და წარმოქმნილიყო **ადრონები** (ისეთი, როგორიცაა პროტონები და ნეიტრონები). ელექტრონები ადრონული ეპოქის ექსტრემალურ პირობებში **პროტონებთან** შეჯახებისას მათთან ერთიანდებოდა და წარმოიქმნებოდა **ნეიტრონები**; ამ დროს პროტონებიდან გამოიდევნებოდა მასის არმქონე **ნეიტრინოები**, რომლებიც კოსმოსში დღემდე განაგრძობს მოგზაურობას სიჩქარით, რომელიც სინათლის სიჩქარის ტოლია ან ოდნავ ჩამოუვარდება მას. ზოგიერთი

ნეიტრინი პროტონ-ელექტრონების წყვილებად ხელახლა ერთიანდება. ამ შემთხვევითი ან განმეორებითი გაერთიანების ერთადერთი მიზეზი შეიძლება იყოს **მუხტისა** და **ენერგიის** (**მასა-ენერგიის** ჩათვლით) რაოდენობის მუდმივობის შენარჩუნების კანონი.

7. ლეპტონური ეპოქა - 1 წამიდან **3** წუთამდე:

ლეპტონების ეპოქის ბოლოს, როდესაც ადრონებისა და ანტიადრონების უმრავლესობამ (და არა ყველამ) ერთმანეთის **ანიგილირება** მოახდინა, სამყაროს მასაში მადომინირებელ ნაწილაკებად **ლეპტონები** (როგორიცაა, **ელექტრონები)** და **ანტილეპტონები** (როგორებიცაა, **პოზიტრონები**) გადაიქცა. ერთმანეთთან შეჯახებისას ელექტრონები და პოზიტრონები ერთმანეთის ანიგულირებას ახდენს; თავისუფლდება **ფოტონების** სახის **ენერგია,** ხოლო შეჯახებული **ელექტრონები და პოზიტრონები**, თავის მხრივ, წარმოქმნის დიდი რაოდენობის **ელექტრონ-პოზიტერონის** წყვილებს.

8. ნუკლეოსინთეზის ეპოქა - 3-დან **20** წუთამდე:

ნუკლეოსინთეზი ეწოდება ბირთვული სინთეზის პროცესში შექმნილი პროტონებისა და ნეიტრონებისაგან ახალი ატომური ბირთვების შექმნის პროცესს. ამ ეპოქის დასაწყისში სამყაროს ტემპერატურა დაახლოებით $\mathbf{10^{12}}$ C (ანუ, მილიარდ) გრადუსამდე დაეცა, რის შედეგადაც შესაძლებელი გახდა დაწყებულიყო **ბირთვული რეაქციები.** მათი მეშვეობით ფორმირება დაიწყო ატომის ბირთვებმა; პროტონები და ნეიტრონები გაერთიანდა და წარმოიშვა მარტივი ელემენტების **(წყალბადის, ჰელიუმისა** და **ლითიუმის)** ბირთვები. დაახლოებით **20** წუთის შემდეგ სამყაროს ტემპერატურა და სიმკვრივე იმდენად შემცირდა, რომ ბირთვული რეაქციის გაგრძელება შეუძლებელი გახდა.

9. ფოტონური (რადიაციის დომინირების) **ეპოქა - 3** წუთიდან **240 000** წლამდე:

ამ ეპოქიდან დაიწყო სამყაროს თანდათანობით გაგრილების ხანგრძლივი პერიოდი. სამყარო სავსე იყო პლაზმით (გავარვარებული ატომური ბირთვებისა და ელექტრონებისაგან წარმოქმნილი „სუპით“). ლეპტონური ეპოქის ბოლოს ლეპტონებისა და ანტილეპტონების უმრავლესობის მიერ ერთმანეთის **ანიგილირების** შემდეგ სამყაროში დადგა **ფოტონების დომინირების** პერიოდი, რომლებმაც განაგრძეს პროტონებთან, ელექტრონებსა და ბირთვებთან ხშირი **ურთიერთზემოქმედება**.

10. რეკომბინაცია/გახსნის ეპოქა - 240 000-დან **300 000** წლამდე:

სამყაროს ტემპერატურა **3000^0 C**-მდე შემცირდა (დაახლოებით ასეთივე ტემპერტატურაა მზის ზედაპირზე) და გაგრძელდა სიმკვრივის დაცემა. ამის გამო წყალბადისა და ჰელიუმის იონიზებულმა ატომებმა დაიწყო ელექტრონების მიტაცება (ანუ, **რეკომბინაცია**) და, ამით საკუთარი ელექტრული მუხტების განეიტრალება. ატომებთან ელექტრონების საბოლოოდ შეკავშირების შემდეგ, სამყარო, ბოლოსდაბოლოს, შუქისათვის გამჭვირვალე გახდა. ამან ეს ეპოქა დღეისათვის **ყველაზე ადრეულ დაკვირვებად ეპოქად აქცია**. მან მანამდე ფოტონურ-ბარიონული გაუმჭვირვალე სითხეში ელექტრონებსა და პროტონებთან ურთიერთობით „დაკავებული“ **ფოტონებიც** გამოათავისუფლა (რაც „გახსნის“ სახელითაა ცნობილი), რომლებმაც კოსმოსში თავისუფალი „მოგზაურობა“ დაიწყო (მათ ჩვენ დღევანდელ **გამოსხივების კოსმოსურ ფონში** ვხედავთ). ეპოქის დასასრულს სამყარო შედგებოდა **ნისლისაგან**, რომელიც შეცავდა დაახლოებით **75%** წყალბადს, **25%** ჰელიუმსა და ლითიუმის კვალს.

11. ბნელი ეპოქა (ერა) - 300 000-დან **150** მილიონ წლამდე:

პირველი ატომების წარმოქმნიდან პირველ ვარსკვლავების გამოჩენამდე არსებულ პერიოდს **ბნელი ეპოქა** ეწოდება. სამყარო, ფოტონების არსებობის მიუხედავად, ჩაბნელებული იყო, რადგან სინათლის გამომსხივებელი ვარსკვლავები ჯერ კიდევ არ არსებობდა. მეტად დიფუზიური მატერიის დარჩენის გამო, ძალიან დაბალი დონის ენერგიისა და ძალიან დიდი დროითი მასშტაბის მქონე სამყაროში, აქტივობა მკვეთრად მინელდა. ამ პერიოდში რაიმე მნიშვნელოვანი არ ხდებოდა და სამყაროში „ბნელი მატერია“ ბატონობდა.

12. რეიონიზაციის ეპოქა – 150 მლნ წლიდან **1** მილიარდ წლამდე:

გრავიტაციული კოლაფსის (*გრავიტაციული ძალების ზემოქმედებით ობიექტების სწრაფი შეკუმშვის*) შედეგად, სამყაროში წარმოიქმნა პირველი კვაზარები. **კვაზარი** (*ინგ.„**quasi-stellar**“ - კვაზივარსკვლავი და „**radiosource**“ - რადიოწყარო*) ძალიან დიდი გამოსხივების უნარიანი მნათი კოსმოსური ობიექტია. მისი გამოსხივება, მაგალითად, **27** ტრილიონჯერ აღემატება ჩვენი უახლოესი ვარსკვლავის, მზის, გამოსხივებას. ამ გამოსხივებამ სამყაროში ხელახლა მოახდინა წყალბადის იონიზაცია, რომლის ნეიტრალიზაცია რეკომბინაციის პერიოდში (იხ. ზემოთ) განხორციელდა. აქედან დაწყებული, სამყაროს უმეტესი ნაწი-

ლი ნეიტრალური მდგომარეობიდან ხელახლა დაუბრუნდა იონიზებული პლაზმის მდგომარეობას.

13. ვარსკვლავების, პლანეტებისა და გალაქტიკების ფორმირების ეპოქა – 300-500 მილიონი წლიდან:

გრავიტაცია აძლიერებდა პირველადი აირის მცირე უთანაბრობას და სამყაროს სწრაფი გაფართოების მიუხედავად აირის ჯიბეები სულ უფრო და უფრო მკვრივი ხდებოდა. კოსმოსური აირის ეს პატარა მკვრივი ღრუბლები საკუთარი გრავიტაციის მოქმედების ძალით იკუმშებოდა და საკმაოდ ცხელდებოდა იმისათვის, რომ წყალბადის ატომებს შორის **ვარსკვლავების,** შემქმნელი ბირთვული სინთეზის რეაქციები დაწყებულიყო.

ვარსკვლავი საკუთარი კოლოსალური მასისაგან წარმოშობილი ვეებერთელა გრავიტაციული ძალების მეშვეობით წონასწორულ მდგომარეობაში შეკავებული აირისაგან (ძირითადად წყალბადისა და ჰელიუმისაგან) შემდგარ მნათ ცხელ და მასიურ სფეროს ეწოდება. გრავიტაცია ობიექტის ბირთვში **ბირთვულ სინთეზსაც** იწვევს, რომლის დროსაც სინათლე და სითბო გამოიყოფა. სინთეზის მიზეზია მაღალი წნევისა და ტემპერატურის პირობებში გრავიტაციის მიერ წყალბადის ძლიერ შეკუმშვა, რომელიც წყალბადის ორი ატომს ჰელიუმის ერთ ატომად გარდაქმნის. ამ დროს ვარსკვლავის ნათების გამომწვევი ვეებერთელა რაოდენობის ენერგია წარმოიქმნება. ჩვენთან უახლოესი ვარსკვლავია მზე.

ვარსკვლავების სინთეზირებას ბუნებრივად მოყვა პლანეტებისა და გალაქტიკების ფორმირების პროცესები.

პლანეტა ვარსკვლავის ირგვლივ მბრუნავი ობიექტია, რომელიც საკუთარ ორბიტაზე დომინირებს და მისგან განდევნის ანალოგური ზომის მქონე ყველა მახლობელ ობიექტს. საკუთარი მასა პლანეტისათვის იმისათვისაა საკმარისი, რომ მას სფერული ფორმა ჰქონდეს, მაგრამ არასაკმარისია პლანეტის შიგნით ბირთვული სინთეზის წარმოსაქმნელად. პლანეტები შეიძლება შედგებოდეს ქვის ქანებისაგან, როგორც ესაა დედამიწასა და მარსზე, ან აირისაგან, როგორც ეს იუპიტერსა და სატურნიზეა. მზის სისტემის გარეთ არსებულ პლანეტებს - **ეგზოპლანეტები** (*ძვ. ბერძ. ἔξω, exō — „გარეთ“, „გარედან“*) ეწოდება.

ვარსკვლავებსა და პლანეტებს შორის ძირითადი განსხვავება ისაა, რომ **ვარსკვლავები** შუქსა და სითბოს მათ ბირთვში მიმდინარე ბირთვული სინთეზის მეშვეობით თავად გამოიმუშავებს. ისინი ენერგი-

ას გამოყოფს სინათლისა და ელექტრომაგნიტური გამოსხივების სახით, რისი წყალობითაც მათი დანახვა ძალიან შორიდანაცაა შესაძლებელი. **პლანეტები**, პირიქით, სინათლეს ვერ გამოიმუშავებს. ისინი აირეკლავს იმ ვარსკვლავებიდან დაცემულ სინათლეს, რომელთა ირგვლივაც პლანეტები ბრუნავს. ამიტომ ჩვენ ეგზოპლანეტებს ისე შორიდან ვერ ვხედავთ, როგორც ვარსკვლავებს. მზე დაახლოებით მილიარდჯერ უფრო კაშკაშაა, ვიდრე მის ირგვლივ მოძრავი ეგზოპლანეტებიდან არეკლილი სინათლე.

ვარსკვლავები და პლანეტები სხვა პარამეტრებითაც განსხვავდება ერთმანეთისაგან. მათ ჩვენ არ განვიხილავთ. მხოლოდ ერთ მათგანზე, წარმოშობის მიხედვით განსხვავებზე, შევაჩერებთ თქვენს ყურაღებას. **ვარსკვლავები** წარმოიქმება აირისა და მტვრის ვეებერთელა ღრუბლებისაგან, რომლებიც სიმძიმის ძალის მოქმედებით იკუმშება, ცხელდება და მათში ბირთვული სინთეზი წარმოშობს. **პლანეტები** კი წარმოიქმება ვარსკვლავის შექმნის შემდეგ დარჩენილი გამოუყენებელი მატერიისაგან. აქედან გამომდინარე, **პირველად გაჩნდა ვარსკვლავები, და შემდეგ - მათ ირგვლივ მბრუნავი პლანეტები**.

პლანეტები წარმოიქმნა ე.წ. **აკრეციით** (*ლათ. Accrētiō – „მატება, გადიდება“*), რაც გარემომცველი სივრციდან მატერიის გრავიტაციული მიზიდვის მეშვეობით ციური სხეულის ზომის მატების პროცესს აღნიშნავს. იგი სხვა არაფერია, თუ არა გარემომცველი სივრციდან კოსმიურ სხეულზე (კერძოდ, პლანეტაზე) ვარსკვლავთა შექმნის შემდეგ დარჩენილი ნივთიერებების ჩამოცვენის პროცესი.

გალაქტიკა ეწოდება გრავიტაციულად დაკავშირებული ვარსკვლავების, პლანეტების, ვარსკვლავთშორისი აირისა და მტვრისაგან შემდგარ სისტემას. გალაკტიკებში შეიძლება გაერთიანებული იყოს რამდენიმე ათასიდან რამდენიმე ტრილიონამდე ვარსკვლავი. მსხვილი გალაქტიკების უმრავლესობათა ცენტრებში ზემასიური შავი ხვრელებია განთავსებული, რომელთაგანაც ზოგიერთის მასა დაახლოებით მილიარდჯერ აღემატება მზის მასას (≈**1,98 × 10^{30}** კგ-ს).

14. მზის სისტემის ფორმირების ეპოქა – 8,5-9 მილიარდი წელი:

მრავალი ადრეული თაობის ვარსკვლავთა ნამსხვრევებისაგან შეიქმნა გვიანი თაობის ვარსკვლავი - მზე. იგი და მის ირგვლივ არსებული მზისეული სისტემა ფორმირდა ≈ **4,5-5** მილიარდი წლის წინათ (დიდი აფეთქებიდან ≈ **8,5-9** მილიარდი წლის შემდეგ).

15. დღევანდელი ეპოქა - 13,82 მილიარდი წელი:

სამყაროს გაფართოება და ახალი ვარსკვლავების წარმოქმნა გრძელდება.

1.6. 6. დედამიწაზე სიცოცხლის გაჩენის პრობლემისათვის

1868 წელს ბერლინში დაარსებულ ჟურნალ „**Chemische Berichte**“-ში **1913** წელს გამოქვეყნდა ბიოიქიმიკოს **ვალტერ ლიობის (***გერმ. Valter Löb; 1872-1916*) სტატია. მასში ავტორმა აღწერა წყლიანი ამიაკისა და ნახშირორჟანგის ნარევზე მის მიერ განხორციელებული ელექტრული განმუხტვის პროცესი, რომლის შედეგადაც მან მიიღო **ამინომჟავა (**amino acetic acid) ანუ **გლიკოკოლი**; მაგრამ, რაც მთავარია, მეცნიერი სიცოცხლის წარმოშობისათვის **აუცილებელი შაქრისა** და **ფორმალდეჰიდის,** ანუ **ჭიანჭველალდეჰდის** (formaldehde ანუ formic aldehyde) წარმოქმნის მოწმე გახდა.

სამწუხაროდ, **ვალტერ ლიობი**, რომელიც გერმანელი მეცნიერისა და პოლიტიკური მოღვაწის **რუდოლფ ლუდვიგ კარლ ვირხოვის** (*გერ. Rudolf Ludwig Karl Virchow;1821-1902*) მიერ დაარსებულ საავადმყოფოში მუშაობდა**,** უინტერესო აღმოჩნდა სამეცნიერო საზოგადებისათვის. ამის გამო ჩატარებული უმნიშვნელოვანესი ექსპერიმენტი დიდი ხნის განმავლობაში უყურადღებოდ დარჩა. აღსანიშნავია, რომ **ლიობმა** თავის სტატიაში არ დაივიწყა ეხსენებინა მაშინდელ სერბიულ აკადემიაში მომუშავე **ს. მ. ლაზანიჩისა** და **მ. ს. იოვიშიჩის** გვარები, რომლებსაც ანალოგიური ექსპერიმენტის ჩატარება ჯერ კიდევ **1897** წელს ჰქონდათ ჩაფიქრებული, მაგრამ ვერ განახორციელეს სათანადო მეცნიერული ინსტრუმენტების არქონის გამო. ეს **ვალტერ ლიობის** მეცნიერულ სისპეტაკეზე მიგვითითებს.

ვალტერ ლიობის მიერ ზემოთ აღნიშნული სტატიის გამოქვეყნებიდან ზუსტად **50** წლის შემდეგ - **1952** წელს - მსგავსი ექსპერიმენტი, ნობელის პრემიის ლაურეატის **ჰაროლდ კლეიტონ იურის** (*ინგ. Harold Clayton Urey; 1893-1981*) ხელმძღვანელობით, ჩაატარა ჩიკაგოს უნივერსიტეტის სტუდენტმა **სტენლი ლოიდ მილერმა** (*ინგ. Stanley Lloyd Miller;1930-2007*). ლაბორატიულ პირობებში **მილერმა** შექმნა წყალბადური ატმოსფერო, რომელშიც მან ელექტრული განმუხტვებით მოახდინა ზემოქმედება წყლიანი მეთანისა და ამიაკისაგან შედგენილ ნარევზე. ამის შედეგად ფორმირებული იქნა ამინომჟავები, რომლებითაც პრიმიტული ცილების „აწყობაა“ შესაძლებელი. **მილერის ექსპე-**

რიმენტის მრავალჯერადად გამეორებით დაამტკიცდა შედეგის აბსოლუტური ჭეშმარიტობა. ნათელი გახდა, რომ **ორგანული ნივთიერებები (მათ შორის ამინომჟავები) თეორიულად შესაძლებელი იყო ახლადწარმოქმნილ დედამიწის პირობებში ფორმირებულიყო.** ექსპერიმენტი სწრაფად იქცა სენსაციად და **მილერმა** საყოველთაო აღიარება მოიპოვა.

ლიობისა და **მილერის** მიერ ერთმანეთისაგან დამოუკიდებლად ჩატარებული ექსპერიმენტებით ნათლად დამტკიცდა, რომ ახლადწარმოქმნილ დედამიწის არაორგანულ სამყაროში, ყოველგვარი დამხმარე „შემომქმედის“ მონაწილეობის გარეშე, შესაძლებელი იყო ფორმირებულიყო ცოცხალი ორგანიზმების უჯრედების ასაგებად საჭირო საკვანძო ელემენტები. ამ მკვლევრების მიერ ექსპერიმენტებში გამოყენებული ყველა არაორგანული ნაერთი ჭარბად მოიპოვებოდა ახლადწარმოქმნილ დედამიწაზე, ხოლო ექსპერიმენტებში გამოყენებული ელექტრული განმუხტვების როლი თავისუფლად შეეძლო ელვებს შეესრულებინა, რომლებითაც გაჯერებული იყო იმდროინდელი **დედამიწის ატმოსფერო**. ასე რომ, **ახლადწარმოქმნილი დედამიწა** საოცრებების შემქმნელ **ვეებერთელა ბუნებრივ ლაბორატორიას** შეგვიძლია შევადაროთ.

დედამიწა დაახლოებით **4,54** მილიარდი წლის წინათ, ანუ **დიდი აფეთქებიდან** დაახლოებით **9,28** მილიარდი წლის შემდეგ შეიქმნა. სწორედ ამ თარიღთან მიახლოებულ პერიოდში უნდა ჩამოყალიბებულიყო იგი ზემოთ აღნიშნულ ლაბორატორიად. დღევანდელი გადასახედიდან ძნელია დადგინდეს ჭეშმარიტებასთან მაქსიმალურად მიახლოებული მისი სტრუქტურული აგებულება. სანამ საქმის სპეციალისტებს აღნიშნული სტრუქტურის თავიანთი ვარიანტები არ შემოუთავაზებიათ, გავკადნიერდები და შევეცდები ჩემეული თვალსაზრისით ვაჩვენო ის ობიექტური რეალობა, რომლის ძალითაც დედამიწა იმ პერიოდში სრულიად მზად იყო ასეთ ლაბორატორიად გარდასაქმნელად. ქვემოთ მყვანილი მოსაზრების ჩამოსაყალიბებლად გამოვიყენე **§1.6.3**-ში მოცემული ფაქტობრივი მასალა.

კვარკების აღმოჩენის შემდეგ ფიზიკოსთა აბსოლუტურმა უმრავლესობამ მატერიის შემდგომ დეტალიზებაზე უარი განაცხადა და უმცირეს (უსტრუქტურო) ნაწილაკებად მათი მიჩნევის ჯენტლმენური შეთანხმება გააფორმეს. ასეთივე უსტრუქტურო ნაწილაკებად ითვლება **ლეპტონები**. ბუნებაში არსებობს კვარკებისა და ლეპტონების

სამ-სამი წყვილი, კერძოდ, ექვსი (***u*** და ***d;*** ***c***, და ***s***, ***t*** და ***b***) **კვარკი** და ექვსი (e^- და ν_e; μ^- და ν_μ ; τ^- და ν_τ) **ლეპტონი**. კვარკები და ლეპტონები მიჩნეულია ფუნდამენტურ ნაწილაკებად, რომლის მეშვეობითაც აიგება მატერია, ანუ ნებისმიერი სახის ნივთიერების ატომები. ამიტომ **კვარკებისა** და **ლეპტონების** ექვსეულები ქმნის ფუნქციურად სრულ სისტემებს. **ფუნქციურად სრული სისტემა** ვუწოდოთ სისტემას, რომელიც საკუთარ ფუნქციებს მხოლოდ მასში შემავალი ელემენტებით, გარედან სხვა ელემენტების შემოუტანლად შეუძლია. ასეთი სისტემებით (კვარკებისა და ლეპტონების ექვსეულებით) ახაგაზრდა დედამიწა დიდი აფეთქების შემდეგ მიმდინარე პროცესების მეშვეობით აღიჭურვა. **კვარკები და ლეპტონებია** ის „სამშენებლო მასალა“, ანუ, „სუბატომური აგურები“, რომლებითაც შესაძლებელია მატერიის (ნებისმიერი სახის ნივთიერების“) აგება; მაგრამ მარტო აგურებით ვერცერთი შენობა ვერ აშენდება: აუცილებელია მუშა ძალის არსებობაც. ასეთ მუშა ძალებად შეგვიძლია მივჩნიოთ ზემოთ განხილული **ოთხი ფუნდამენტური ძალა.** ისინი საკუთარ თავშივე შეიცავს საკუთარი ამოქმედებისათვის საჭირო ნაწილაკებს - **ბოზონებს**. ამიტომ უწოდებენ მათ არა უბრალოდ „ძალებს“, არამედ „ურთიერთქმედებებს“, რომელთა მეშვეობითაც ფუნდამენტური ნაწილაკები ერთმანეთს გადასცემენ ამ ძალებით შესასრულებელ ქმედებებს. კერძოდ, **ძლიერი ბირთვული ზემოქმედების ძალა** გადააქვს ბოზონ **გლიუონს,** ელექტრომაგნიტური ძალა - **ფოტონს,** სუსტი ბირთვული ძალა - **W**- და **Z**-**ბოზონებს**, ხოლო გრავიტაციული ძალა - **გრავიტონს.**

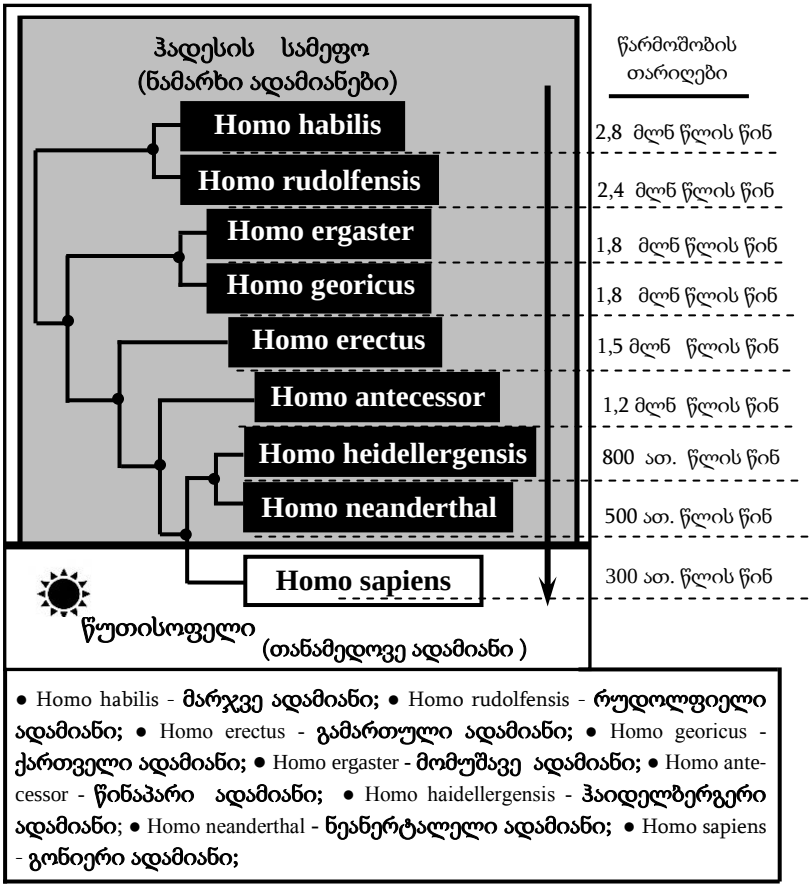
აღნიშნულ ლაბორატორიაში სინთეზირებული ატომებისაგან თავდაპირველად არაცოცხალი ბუნება იქნა რეალიზებული. მოგვიანებით კი, დაახლოებით **4,1 – 3,8** მილიარდი წლის წინათ (დიდი აფეთქებიდან **10,02 – 9,72** მილიარდი წლის გასვლის შემდეგ) დედამიწაზე **აბიოგენეზის** მეშვეობით წარმოიშვა ... სიცოცხლე.

აბიოგენეზი (***a*** - *უარყოფითი თავსართი,* ***bios*** *– „სიცოცხლე“,* ***genesis*** - *„წარმოქმნა“*) ეწოდება არაორგანული ნივთიერებებიდან, ფერმენტების მონაწილეობის გარეშე, სიცოცხლის წარმოშობის **ერთ-ერთ ძირითად ჰიპოთეზას**. იგი მრავალ მეცნიერულ კვლევებსა და პრაქტიკულ ექსპერიმენტებზეა დაფუძნებული, რომლებიც ზემოთ მაქსიმალურად შემოკლებული და გამარტივებული სახით გაგაცანით.

სიცოცხლის წარმოშობის მნიშვნელოვნად შემოკლებულ ქრონოლოგიას ასეთი სახე აქვს:

● **3 მილიარდი წლის წინ** გაჩნდა **ფოტოსინთეზის** უნარის მქონე ორგანიზმები;

● **540 მილიონი წლის წინ** - ფეხსახსრიანები - მწერები, ობობისმაგვარნი, კიბოსმაგვარნი და მთელი რიგი სხვა ჯგუფი;



ნახ.1.8. *დედამიწაზე ადამიანის ევოლუციის ზოგადი სურათი*

● **518 მილიონი წლის წინ** - თევზები;

● **475 მილიონი წლის წინ** - მიწისზედა მცენარეები;

● **300 მილიონი წლის წინ** - პირველი ქვეწარმავლები (რეპტილიები) და სინაპსიდები (ძუძუმწოვართა წინაპრები);

● **230 მილიონი წლის წინ** - პირველი დინოზავრები და ხვლიკისებული დინოზავრები, რომლებიც მოგვიანებით ფრინველთა კლასში გაერთიანდა

● **200 მილიონი წლის წინ** - პირველი ძუძუმწოვრები;

● **150 მილიონი წლის წინ** - პირველი ფრინველები;

● **130 მილიონი წლის წინ** - ყვავილოვანი მცენარეები;

● **66 მილიონი წლის წინ** - პრიმატები - **13-15** სმ-დან **175** სმ-მდე სიმაღლისა და განვითარებული ხუთთითიანი კიდურების მქონე მქონე ძუძუმწოვართა რიგი;

● **2,8 მილიონი წლის წინ** - ადამიანთა (Homo) გვარი.

● **500 ათასი წლის წინათ** ადამიანებმა მიიღეს თანამედროვე სახე (ანთროპოგენეზი).

● **40 ათასი წლის წინ** წინ გადაშენდა უკანასკნელი ნეანდერტალელები და დედამიწაზე **ჰომო საპიენსები** მარტოდმარტო დავრჩით.

2,8 მლიონი წლის წინ გაჩენილ ადამიანის პირველ სახეობის ნამარხი **ტანზანიის** ჩრდილოეთში არსებულ **ოლდუვაის** (*ინგ.* ***Olduvai Gorge***) ხეობაში **1960** წლის ნოემბერში ბრიტანელ-კენიელმა პალეოანთროპოლოგმა **ლუის ლიკიმ** (*ინგ. Louis Leakey; 1903-1972*) და მისმა მეუღლემ, ასევე ბრიტანელ-კენიელი ანთროპოლოგ-არქეოლოგმა, **მერი დუგლას ლიკიმ** (*ინგ. Mary Douglas Leakey; 1913-1996*) იპოვეს. მას **Homo habilis**, ანუ **მარჯვე ადამიანი** ეწოდა (**ნახ. 1.8**). ამის შემდეგ ადამი-ანთა ცამეტამდე სხვადასხვა სახეობათა ნაშთები იქნა აღმოჩენილი, რომელთაგანაც ცხრა სახეობის სახელები **1.8 ნახაზზეა** მოყვანილი. გამოვყოფთ **1991** წელს, მკვლევართა ჯგუფთან ერთად, ქართველი ანთროპოლოგ-არქეოლოგ **ოთარ ლორთქიფანიძის** მიერ **დმანისში** ნაპოვნ ადამიანთა ნაშთებს. მათი ჩონჩხის განმარხებული ნაწილები დაახლოებით **1,8** მილიონი წლის აღმოჩნდა. მეცნიერები თავდაპირველად ფიქრობდნენ, რომ მათი ნაპოვნი ქვედა ყბის ძვლები **Homo erectus**-ს ეკუთვნოდა, მაგრამ ზომებს შორის სხვაობამ ისინი იმაში დაარწმუნეს, რომ ნარჩენები მანამდე უცნობი სახეობის კუთვნილი უნდა ყოფილიყო. მისთვის სპეციალურად შემოიღეს ახალი ტერმინი - **Homo georicus** ანუ **ქართველი ადამიანი**. იგი **Homo habilis**-სა და **Homo erectus**-ს შორის არსებული გარდამავალი ადამიანის სახეობადაა მიჩნეული (იხ. **ნახ. 1.8**).

1.6. 7. „ვინც ფლობს სრულყოფილ მეტყველებას, ის ფლობს სამყაროს!“

მაღალგანვითარებულ პრიმატთა რიგის **ჰომინიდების** (*ლათ.Hominidae –„ადამიანისებრი“*) ოჯახის წევრებიდან მეტყველების უნარი მხოლოდ **ადამიანს** აქვს. იბადება კითხვა, ადამიანის თუ რომელ სახეს ერგო ეს უნარი და რა როლი ითამაშა მან მის ცხოვრებაში? ანთროპოლოგები დიდი ხნის განმავლობაში ვარაუდობდნენ, რომ ადამიანმა მეტყველების უნარი დაახლოებით **200 ათასი წლის წინ შეიძინა**. **2024** წლის დასაწყისში ბრიტანელმა ანთროპოლოგმა, რეადინგის უნივერსიტეტის (*ინგ. University of Reading*) პროფესორმა **სტივენ მაიტენმა** (*ინგ. Steven Mithen*) მოულოდნელად წამოაყენა დასაბუთებული ჰიპოთეზა, რომლის თანახმადაც ადამიანმა ეს უნარი შეიძინა ... **1,5 მილიონი წლის წინ!** ქვემოთ შემოგთავაზებთ ამ ჰიპოთეზის ჩვენებურ ინტერპრეტაციას, გიჩვენებთ, თუ რა სახის მეტყველებზეა მასში საუბარი, ვის მიერ და როდის მოხდა შემდგომში მისი სრულყოფა. გადმოსაცემი მასალის ნათლად გააზრებისათვის საჭიროდ ვთვლით გაგაცნოთ შემდეგი ორ დამხმარე ინფორმაცია.

1. *მეტყველების მოტორიკაზე პასუხისმგებელია თავის ტვინის ქერქის უბანი, რომელიც თავის ტვინის* ***შუბლისეულ წილშია*** *განთავსებული და* ***ბროკას არე*** *ეწოდება. მისი საშუალებით ადამიანი მართავს სახის, ენის, ხახის და ყბების კუნთებს. მისი მეშვეობით ადამიანი წარმოთქვამს მეტყველებისათვის აუცილებელ სიტყვებს;*

2. *ბგერითი მეტყველების ანალიზსა და სინთეზს ახდენს თავის ტვინის ქერქის მარცხენა ნახევარსფეროს (როგორც წესი) საფეთქლისეულ ზედა ხვეულაში განთავსებული უბანი, რომელსაც* ***ვერნიკეს არე*** *ეწოდება. იგი ადამიანს მოსაუბრისაგან მიღებული წინადადებების შინაარსის გაგების უნარს აძლევს.*

ადამიანმა რომ შეიძინოს მეტყველების უნარი, აუცილებელია მისივე თავის ტვინის ქერქში ჩამოყალიბდეს ***ბროკასა*** *და* ***ვერნიკეს*** *აღნიშნული არეები.*

პირველ ჰომინიდებს, რომლებიც **4-5 მილიონი წლის წინ** არსებობდნენ, და, რომელთა ოჯახში შედის **ადამიანი**, ჰქონდა ზრდასრული **შიმპანზის** თავის ტვინის, ანუ **400-500** სმ^3-ის, ტოლი მოცულობის **ტვინი**. იგი მას, დღეს არსებული შიმპანზის მსგავსად, მხოლოდ იმის საშუალებას მისცემდა, რომ სპეციფიკური ხმოვანი სიგნალის მეშვეობით თანამოძმეებისათვის მოულოდნელად წარმოშობილი საშიშროების ფაქტი ეცნობებინა; ლაპარაკის უნარის არსებობა პრაქტიკუ-

ლად გამოირიცხებოდა.

2,8 მილიონი წლის წინ წარმოშობილი პირველ ადამიან **Homo habilis**-ის („მარჯვე ადამიანის“) ტვინის მოცულობა **750** სმ^3-მდე გაიზარდა. ეს უთუოდ გააფართოებდა მის მიერ გადაცემულ შეტყობინებათა რეპერტუარს, მაგრამ მეტყველების უნარის ჩამოსაყალიბებლად აშკარად საკმარისი არ იქნებოდა. ტვინის შინაგანი სტრუქტურის ცვლილებები უნდა დაწყებულიყო, მაგრამ ეს ცვლილებები **ბროკასა** და **ვერნიკეს** მსგავსი სპეციფიკური არეების ფორმირებისათვის აშკარად არ იქნებოდა საკმარისი. შესაძლებელი იყო, რომ მეტყველებისათვის საჭირო წინაპირობები შექმნილიყო.

1,5 მილიონი წლის წინ წარმოშობილი **Homo erectus**-ის („გამართული ადამიანის“, იხ. ნახ. **1,8**) ტვინის მოცულობა დასაწყისში **800** სმ^3 იყო, რომელიც თანდათან გაიზარდა და, ბოლოს, **1300 სმ^3**-ს მიაღწია. ამან, აგრეთვე ზემოთ აღნიშნულმა წინაპირობების არსებობამ, გამართული სიარულის უფრო სრულყოფილ ფორმაზე გადასვლამ და თავის ქალის კონსტრუქციულმა სახეცვლილებებმა, ჯამურად, მის ტვინში წარმოშვა **ბროკასა** და **ვერნიკეს** მსგავსი სპეციფიკური არეები. მათი სტუქტურები, ალბათ, ჯერ კიდევ საჭიროებდა „კონსტრუქციულ დამუშავებას“; ამიტომ ისინი სრულყოფილ მეტყველების უნარს ნახტომისებურად ვერ ჩამოაყალიბებდა, მაგრამ პრიმიტული მეტყველების უნარის ჩამოყალიბებას თავისუფლად შეძლებდა, რაც ფაქტობრივად უნდა მომხდარიყო.

ზემოთაღნიშნულის გათვალისწინებით, აგრეთვე ყველასათვის ხელმისაწვდომი არქეოლოგიური, პალეოანატომიური, გენეტიკური, ნევროლოგიური და ლინგვისტურ მონაცემებზე დაყრდნობით, **2024** წლის დასაწყისში **მაიტენმა** წამოაყენა ჰიპოთეზა, რომლის თანახმადაც ადამიანს მეტყველების უნარი ჰქონდა არა **200 ათასი**, არამედ **1,5 მილიონი წლის წინ**. ამ დროს საუბარია **მეტყველების პრიმიტულ ფორმაზე**. ამ ფორმით ადამიანებს შეზღუდული საკომუნიკაციო უნარები შეეძლო შეეძინათ. მათ სპეციფიკურ კონტექსტებში რამდენიმე ათეული სხვადასხვა ბგერისა და ჟესტის გამოყენების საშუალება მიეცემოდა. ეს ადამიანებს გაუფართოებდა აწმყოში მუშაობის შესაძლებლობებს, მაგრამ მამავალში საკუთარი საქმიანობის დაგეგმვის საშუალებას ვერ მიცემდა. ადამიანებისათვის ეს შეზღუდული საკომუნიკაციო უნარებიც ძალიან დიდი მნიშვნელობის იყო. ისინი ადამიანებს ძალიან დაეხმარებოდა ეკოლოგიურ და კლიმატურ პირობებ-

თან შეგუების, შეძენილი ცოდნის დაგროვებისა და სამყაროს კოლონიზებასთან დაკავშირებული პრობლემების გადაწყვეტაში.

ახლა განვიხილოთ, თუ რომელი სახეობის ადამიანს და როდის შეიძლებოდა ჩამოყალიბებოდა **სრულყოფილი მეტყველების უნარი**. შევადგინოთ ადამიანთა იმ სახეობების სავარაუდო სია, რომლებსაც შეიძლებოდა ჩამოყალიბებოდა ეს უნარი. მასში, დიდი ალბათობით, მასში შევა **800** ათასი წლის წინ წარმოშობილი **ჰაიდელბერგელები**, **500** ათასი წლის წინ წარმოშობილი **ნეანდერტალელები** და **300** ათასი წლის წინ წარმოშობილი **საპიენსელები**, ანუ ჩვენ. საკუთარი თავის შემოწმება არ ღირს: ისედაც დარწმუნებულნი ვართ, რომ ჩვენ გაგვაჩნია ეს უნარი. ამიტომ სიიდან პირველად ჩვენი თავი უნდა ამოვიღოთ. მასში განსახილველად დარჩება **ჰაიდელბერგელები** და **ნეანდერ-ტალელები (ნახ. 1.8)**. მისგან შეგვიძლია **ჰაიდელბერგელებიც** ამოვიღოთ, რადგან ისინი ნეანდერტალელებს ისე შეერწყნენ, რომ მეცნიერები მათ **პროტონეანდერტელებსაც უწოდებენ**. ამიტომ მხოლოდ ნეანდერტალელების განხილვით შეგვიძლია შემოვიფარგლოთ. შემოვიფარგლოთ. დაგვრჩება ერთელემენტიანი ცხრილი, რომლის ძალითაც საჭიროა გამოირკვეს: შეეძლო თუ არა **ნეანდერტალელ** ადამიანს შეეძინა სრულყოფილი მეტყველების უნარი?

ზემოთ დასმულ კითხვაზე პასუხის გაცემა გაგვიადვილა ამერიკელმა კოგნისტივმა, **ბრაუნის კერძო კვლევითი უნივერსიტეტის** მეცნიერ-თანამშრომელმა **ფილიპ ლიბერმანმა** (*იხ. Philip Lieberman; 1934 - 2022*) და მისი კოლეგებმა. **ნეანდერტალელებისა** და **ადრეული ჰომინიდების** თავის ქალათა ფუძეების ფორმებისა და მეტყველებასთან დაკავშირებულ სტრუქტურათა ანატომიური განლაგებების გამოკვლევით დაადგინეს, რომ სახმო ტრაქტის მოწყობილობისა და მეტყველებითი შესაძლებლობების მიხედვით **ნეანდერტალელი ადამიანი** თანამედროვე ორი წლის ბავშვს უახლოვდება. ნებისმიერი სირთულის ტვინის შემთხვევაშიც ისინი მხოლოდ შეზღუდული რაოდენობის ბგერების წარმოთქმას შეძლებდნენ. მხოლოდ **საპიენსელებს** ჰქონდათ საჭირო რაოდენობის ბგერების ფორმირების, ანუ სრულყოფილი მეტყველების ათვისების უნარი.

საპიენსი დედამიწას დაახლოებით **300 ათასი წლის წინ** მოევლინა. იგი ამ უნარით ვერ დაიბადებოდა: იგი მას უნდა გამოემუშავებინა. დაისმის კითხვა, მან ეს **როდის მოახერხა**?

დღეს, პრობლემის წარმოშობიდან **300** ათასი წლის გასვლის შემდეგ, პრაქტიკულად შეუძლებელი უნდა იყოს ზემოთ დასმულ კითხვაზე მეტ-ნაკლებლად სწორი პასუხის გაცემა. ერთადერთი შესაძლო გზაა სუბიექტის მიერ შექმნილი და დღემდე მოღწეული ნაკეთობების „ასაკის“ დადგენით სუბიექტის საქმიანობის პერიოდის გამოთვლა და მიღებული მონაცემების მეშვეობით თავად სუბიექტის წარმოშობის მიახლოებითი თარიღის დადგენა. ამ მიმართულებით მეტად მნიშვნელოვანია ამერიკელ პალეოანთროპოლოგის, სტენფორდის უნივერსიტეტის პროფესორ **რიჩარდ კლაინის** (*ინგ. Richard G. Klein; 1941*) მიერ ჩატარებული კვლევის შედეგები. აღმოჩნდა, რომ დაახლოებით **50 ათასი წლის წინანდელ** არტეფაქტებში ნახტომისებურადაა გაზრდილი იმათი ნაირფეროვნება და მრავალ მათგანში აშკარად შეინიშნება მაღალი ხელოვნების ნიშნები. მან ყურადღება იმ ფაქტსაც მიაქცია, რომ ამ პერიოდშივე მნიშვნელოვნად იყო გაზრდილი მოსახლეობის ფარდობითი სიმჭიდროვე. ჩვენი აზრით, ეს უნდა მომხდარიყო მაშინდელ **ჰომო საპიენსელებში** სრულყოფილი მეტყველების არსებობის შედეგად. ამით შეგვიძლია დავასკვნათ, რომ საპიენსელებს სრულყოფილი მეტყველების უნარი, სულ მცირე, **50 ათასი წლის წინ** უნდა ჩამოყალიბდებოდათ; ამით ისინი ერთდროულად სამ, წარსულ, აწმყო და მომავალ, დროში აზროვნების უნარს შეიძენდნენ, რაც მნიშვნელოვნად გაზრდიდა მათ კოგნიტურ შესაძლებლობებს. **ნეანდერტალელები** კი უიმედოდ ჩაიკეტებოდნენ აწმყო დროში, რაც დაამუხრუჭებდა მათ კოგნიტურ განვითარებას. სათავე დაედებოდა **საპიენსელთა** რაოდენოს სწრაფი ზრდას. **ნეანდერტალელები** კი მილევით რეჟიმში არსებობის რეჟიმზე გადაერთვებოდნენ. ამის გამო მათ და დაახლოებით **40 ათასი წლის წინ** შეწყვეტეს არსებობა. ამ პერიოდიდან დედამიწას მთლიანად ჩვენ - **საპიენსელები** დავეპატრონებოდით: პლანეტაზე მარტოდმარტოდ დავრჩებოდით და ამ მარტოობას დღემდე განვაგრძობთ.

„**ვინც ფლობს ინფორმაციას, ის ფლობს სამყაროს**“, გვმოძღვრავს ებრაული წარმოშობის ბრიტანელი ბანკირი **ნათან მაიერ როტშილდი** (*ინგ. Nathan Mayer Rothschild*; *1777-1836*). ამ შესანიშნავი დარიგების პერეიფრაზირება - **„ვინც ფლობს სრულყოფოლ მეტყველებას, ის ფლობს სამყაროს**“ - ზუსტად მიესადაგება სამყაროში დღეს შექმნილ რეალობას: დღეს სამყაროს სრულყოფილი მეტყველების უნარის მქონე ერთადერთი სახეობის ადამიანები - **საპიენსელები** ვფლობთ. ეს ჩვენ

დიდ პასუხისმგებლობას გვაკისრებს: სამყაროს ბედზე ჩვენ ვაგებთ პასუხს! საჭიროა მანამდე შევინარჩუნოთ დედამიწაზე სიცოცხლისათვის საჭირო პირობები, სანამ რეალობად არ ვაქცევთ სხვა პლანეტაზე ჩვენი მომავალი თაობის აუცილებელი გადასახლების იდეას!

1.7. ჰომო საპიენსის თანამედროვე ადამიანად ჩამოყალიბება

ჰომინიდების ოჯახში პირველი ადამიანის **ჰომო ჰაბილისის** (*Homo habilis*) გაჩენისთანავე დაიწყო **თანამედროვე ადამიანად** მისი გარდაქმნა-ჩამოყალიბების გრძელი გზა. ოჯახში შემავალი სხვა წევრებისაგან განსხვავებით მას დაჰყვა **იარაღის კეთების უნარი**, რომელიც შემდგომში მემკვიდრეობით გადაეცემოდა დანარჩენი სახის ადამიანებს; ამიტომ ამერიკელმა განამათლებელმა და სახელმწიფო მოღვაწემ **ბენჯამინ ფრანკლინიმა** (ინგ. *Benjamin Franklin*;1706-1790) მათ იარაღის მკეთებელი ცხოველები უწოდა. იარაღებს ისინი მაშინ ადვილად მოსაპოვებელი ერთადერთი მასალის - **ქვის** - გამოყენებით აკეთებდნენ. რის გამოც გეოლოგებმა დედამიწაზე **ჰომო ჰაბლისის** წარმოქმნის შემდეგ დაწყებულ ხანას **პალეოლითის** (ბერძ. *παλαιός –„ძველი“ + λίθος –„ქვა“*), ანუ **ძველი ქვის ხანა** უწოდეს.

პირველი ადამიანები თავს **მითვისებითი მეურნეობით** (ნადირობითა და მომგროველობით) ირჩენდნენ. დიდი ხნის განმავლობაში ისინი სანადიროდ უხეშად დამუშავებული ქვის ნატეხებისაგან დამზადებულ პრიმიტულ იარაღებს იყენებდნენ და ამიტომ ისინი მეტად მწირი ნანადირევის მოპოვებას ახერხებდნენ.

წარმოშობიდან დაახლოებით **100 ათასი წლის შემდეგ** პირველმა ადამიანებმა **ცეცხლის გამოყენება** შეძლეს. ცეცხლი მათთვის არა მარტო კერძის მომზადებისა და გათბობისათვის საჭირო, არამედ უძლიერესი სამონადირეო საშუალებაც იყო. მან ადამიანებს **შედენითი (შემწყვდევითი)** სახის ნადირობაზე გადასვლის საშუალება მისცა. ჩირაღდნების ქნევით **შემდენელთა მწკრივი** ნახირს ჩასაფრების ადგილისკენ მიდევნიდნენ, სადაც მათ ჩასაფრებული შუბებიანი და კეტებიანი მონადრიეები ხვდებოდნენ. **არქოლოგიური მონაცემები** გვიამბობს შედენითი ნადირობის არაჩვეულებრივ ეფექტურობაზე. ფრა-

ნგი ანთროპოლოგისა და არქეოლოგის, თანამედროვე არქეოლოგიის ერთ-ერთი დამაარსებლის, **ლუი ლორან გაბრიელ დე მორტილეს** (ფრანგ. *Louis Laurent Gabriel de Mortillet; 1821 - 1898*) მიერ **სონა და ლუარას** (ფრანგ. *Saône-et-Loire*) დეპარტამენტში არსებულ კლდე **სოლიუტერეთან** (ფრანგ. *Solutré*) აღმოჩენილი იქნა **18 -15** ათასი წლის წინ მცხოვრები **ჰომინიდების** სადგომი. ამ სადგომზე ნაპოვნი იქნა მათ მიერ მიდენილი და შემდეგ გაწყვეტილი **10 000** ცხენის ჩონჩხი.

ზემოთ აღწერილი რეალობის თვისობრივ ცვლილებას საფუძველი ჩაეყარა დაახლოებით **11 700** წლის წინათ. ამ დროს დედამიწაზე დამთავრდა გამყინვარების მორიგი პერიოდი (ამ პერიოდების რაოდენობა ჯერ-ჯერობით განსაზღვრული არ არის) და დაიწყო **ჰოლოცენის** (*ჱრძ ὅλος - „მთელი“ + καινός – „ახალი“*) **ეპოქა**. ჩვენ დღეს ამ ეპოქაში განვაგრძობთ ცხოვრებას და იგი დედამიწაზე კიდევ **30 ათასი წელი იარსებებს**. ამ ეპოქის დადგომისას დედამიწაზე ადამიანთა სახეებიდან მარტო მარტო **ჰომო საპიენსები** (*Homo sapiens*) ცხოვრობდნენ. ისინი, თავის წინაპრებივით, **მითვისებითი მეურნეობით** განაგრძობდნენ თავის რჩენას.

ზემოთ აღნიშნულმა **შედენითმა ნადირობამ** მთლიანად გაანადგურა მრავალი სახის მსხვილი ცხოველები. ამის გამო საპიენსებს შეუმცირდათ ნადირობით ნაშოვნი სარჩოს რაოდენობა. გამოსავლის საპოვნელად მათ თავდაპირველად ნადირობის მეთოდების სრულყოფა დაიწყეს. ჯერ კიდევ დაახლოებით **13 ათასი წლის წინ**, ფრინველებსა და მცირე ცხოველებზე სანადიროდ, გამოიგონეს მშვილდისარი; „კავშირი შეკრეს“ **ძაღლების** წინაპრებთან - **ტურებთან** - და დაიწყეს ნადირობაში ურთიერთდახმარება; ვეშაპებსა და დიდ თევზებზე ნადირობისათვის შექმნეს სატყორცნი იარაღი **ჰარპუნი** (**ბარჯი**}, დაიწყეს ხის მორებისგან სათევზაო ნავების გამოთლა და ა. შ.

ზემოთ აღნიშნული ძალისხმევის მიუხედავად, **საპიენსები** თანდათან რწმუნდებოდნენ შეგროვებითი მეურნეობის უპერსპექტივობაში. ამას თან დაემთხვა **ჰოლოცენის ეპოქის** დადგომა, რომელმაც შეძლო მიწათმოქმედობისა და მეცხოველეობითვის ხელსაყრელი ბუნებრივი პირობები შექმნა. საპიენსებმა **მითვისებითი მეურნეობიდან** თანდათან **სამეწარმეო მეურნეობრიობაზე** - მიწათმოქმედებასა და მეცხოველეობაზე - გადასვლა დაიწყეს. ხელი მიჰყვეს მიწების დათესვასა და მოსავლის მოყვანას, აგრეთვე გარეული ცხოველების მოშინაურებას.

10 000 წლის წინ მკვეთრად ამაღლდა საპიენსების მიერ დამზადებული იარაღებს ხარისხი. გეოლოგები მიიჩნევენ, რომ ამ დროს დამთავრდა **პალეოლითი** (ძველი ქვის ხანა) და დაიწყო ახალი ქვის ხანა, ანუ **ნეოლითი** (ბერძ. *Νέος* - *„ახალი“* + *λίθος* - *„ქვა“*). ამ დროს **თვისებრივად შეიცვალა** ჰომო საპიენსების **საქმიანობის სახე.** ისინი **მითვისებითი მეურნეობიდან** გადავიდნენ **სამეწარმეო მეურნეობაზე.**

დიალექტიკური განვითარების კანონის თანახმად, ერთი თვისებრიობის მეორე თვისებრიობაზე გადასვლა მხოლოდ ნახტომისებურად, ანუ რევოლუციურადაა შესაძლებელი. მიღებული განზღვრების ძალით, **რევოლუცია** (*გვიანდ. ლათ. revolutio – „შემობრუნება“, „გადატრიალება“*) არის ერთი თვისებრივი მდგომარეობიდან მეორეში **დიალექტიკური, ნახტომისებური გადასვლა**, რომელიც რაოდენობრივი ცვლილებების თანდათანობითი დაგროვებითაა გამოწვეული.

საპიენსების მიერ მითვისებითი მეურნეობიდან სამეწარმეო მეურნეობაზე გადასვლას ბრიტანელ-ამერიკელმა მეცნიერმა, **XX** საუკუნის ერთ-ერთმა წამყვანმა არქეოლოგმა **ვერე გორდონ ჩაილდმა** (ინგ. *Vere Gordon Childe*; 1892-1957) **1923** წელს **ნეოლითური რევოლუცია** უწოდა. ვინაიდან ამ რევოლციის შემდეგ საპიენსელები **აგრარულ მეურნეობაზე** გადავიდნენ, ბევრი მეცნიერი მას **აგრარულ რევოლუციასაც** უწოდებს. ბოლო განმარტება რამდენადმე ცალმხრივად გვეჩვენება.

ზემოთ აღნიშნულის დასაბუთებლადე გავიხსენოთ, რომ მეურნეობის ახალ სახეზე საპიენსელები ნახტომისებურად არ გადასულან. ეს ეწინააღმდეგება **განვითარების დიალექტიკურ კანონს**, რომლის თანახმადაც თვისებრიობის რევოლუციურად შეცვლას წინ დიდი ხნით რაოდენობრივი ცვლილებები უნდა უძღოდეს. ეს ცვლილებები მათ მშვილდ-ისრის გამოგონებით ჯერ კიდევ **13 ათასი წლის წინ** დაიწყეს და შემდეგ არ შეუწყვეტიათ. ამ მიმართულებით ჩატარებული სამუშაოების ინტენსიურობა **ჰოლოცენის პერიოდის** დადგომისას, კერძოდ, **11 700 წლის წინ** გაიზარდა. ამ პერიოდმა შეცვალა **დიდი გამყინვარების პერიოდი**, როდესაც მეურნეობის ახალ სახეზე გადასვლა უაღრესად გაძნელებული იყო. სწორედ ჰოლოცენის პერიოდში შეიძლებოდა დაბადებოდა **ჰომო საპიენსს** მეურნეობის ფორმის შეცვლის იდეა.

ახალ მეურნეობაზე გადასვლა მდინარის ერთი ნაპირიდან მეორეზე გადასვლასავით ადვილი არ არის. ამისათვის დიდი რაოდენობის პრობლემებია დასაძლევი. აღვნიშნოთ ზოგიერთი მათგანი.

უპირველეს ყოვლისა **მითვისებით მეურენეობიდან** (ნადრობიდან და მომგროვებლობიდან) **სამეწარმეო მეურნეობაზე** (მიწათმოქმედებასა და მეცხოველეობაზე) გადასვლისას მნიშვნელობლად შეიცვალა კვების სახე. ეს აჩენს ავადმყოფობებს, რომელთა დასაძლევად საკმაოდ ხანგრძივი ადაპტაციის პერიოდია საჭირო. გარდა ამისა ჩნდება **ტანსაცმლის პრობლემა.** საპიენსელები ნანადირევი ცხოველების ტყავით იმოსებოდნენ, რაც მიწათმომქმედებაზე გადავლის შემდეგ ძნელი მოსასაპოვებელი იყო. ისინი იძულებულნი გახდნენ გრძელბოჭკოიანი მცენარეები, უპირველეს ყოვლისა, **სელი** ეთესათ და მიღებული მოსავლის გამოყენებით ხელი მიეყოთ რთვისა და ქსოვისათვის. ამგვარად, საპიენსელები იძულებულნი იყვნენ აეთვისებინათ დართვისა და ქსოვის სპეციალობები. **პრობლემური** გახდა **მარცვლეულის შენახვაც**, რომლებსაც თაგვთა „არმიები“ უმოწყალოდ ანადგურებდა. **საპიენსები** იძულებულნი გახდნენ **კერამიკა გამოეგონებინათ:** წნელისაგან **კალათების** წვნა დაიწყეს, რომლებსაც თიხით შემოგლესდნენ და შემდეგ კოცონზე გამოწვავდნენ. გამოწვის პროცესის გასაადვილებლად იძულებულნი გახდნენ გამოეგონებინათ სამეთუნეო ღუმელები. მაშასადამე, მათი ნაწილი ისინი იძულებულნი გახდნენ გამხდარიყვნენ - პირველი პროფესიონალური ხელოსნები - **მეთუნეები** - რომლებიც თემის ხარჯზე ცხოვრობდნენ და სარჩოს თემისაგან იღებდნენ. ე. ი. ჩაისახა ხელოსნობა.

ზემოთ აღნიშნულის გარდა, მიწათმოქმედებისა და მეცხოველოების განვითარებისათვის მეტად მნიშვნელოვანი გახდა საცხოვრისის პრობლემა. **მონადირეები** ნადავლის მოსაძიებლად განუწყვეტლივ მოძრაობდნენ და ხშირად ტყეში სახელდახელოდ აგებულ და ცხოველების ტყავით გადახურულ ქოხებში უხდებოდათ ღამის გათევა. სამეწარმეო მეურნეობით დაკავებულმა **საპიენსებმა** სპეციალურად აშენებულ სახლებში დაიწყეს ცხოვრება. პირველად ისინი სახლებს გამოუწვავი აგურებისაგან აშენებდნენ, შემდეგ დაიწყეს სამეთუნეო ღუმლებში მათი გამოწვა. გამომწვარი აგურები ძალიან ძვირი იყო და ისინი თავიანთ სახლებს გამომწვარი აგურებით მხოლოდ აპირკეთებდნენ. *სახლების ხანგამძლეობა დაბალი იყო და ისინი მალე იშლებოდა. ამაღლებულ ნასახლარებზე ადამიანები ახალ სახლებს აშენებდნენ. ამ პროცესების მრავალჯერადად გამეორებამ ადამიანთა დასახლებების ადგილებში* ***გორები*** *წარმოშვა. დღეს არსებული გორების უმრავლესობის წიაღში, დიდი ალბათობით, შესაძლებელია ადამიანთა საქმიანობის ნაშთები ინახებოდეს.*

მათი აღმოჩენა, ცნობილ ქართველი ისტორიკოსის ***ნიკო ბერძენიშვილის*** *(1895 - 1965) შეხედულებით, დღეს არსებული* ***გორების*** *მასობრივი არქეოლოგიური კვლევითაა შესაძლებელი [****80, 81****].*

ზემოთ აღნიშნული პერიოდის კიდევ ერთი აღმოჩენაა სპილენძის პირველი იარაღების დამზადება. შეიძლება პირველი **სპილენძი** სამეთუნეო ღუმელებში არსებული მადნისაგან შემთხვევით იქნა მიღებული, მაგრამ ამ აღმოჩენას საპიენსელთა ცხოვრებაზე შესამჩნევი გავლენა არ მოუხდენია. **სპილენძი** მათთვის იშვიათი ლითონი იყო და ამიტომ მას ისინი სამკაულებად იყენებდნენ. მოგვიანებით საპიენსმა აღმოაჩინა. რომ სპილენძზე ტყვიის დამატებით მიიღებოდა სპილენძზე მტკიცე **ბრინჯაო.** ეს უკანასკნელი სპილენზე ძვირი იყო და ამიტომ იგი ვერ გავრცელდა.

მიწათმოქმედებაზე გადასული **ჰომო საპიენსელებისათვის** საოცრად ძნელი აღმოჩნდა სათესი ნაკვეთების სასურველი ნაყოფიერების უზრუნველყოფის პრობლემის გადაწყვეტა. მიწათმოქმედებაზე გადასვლის **პირველ ეტაპზე** ისინი **სათოხნე მიწათმოქმედებას** იყენებდნენ. ამ დროს ნაკვეთი სწრაფად ღარიბდებოდა და ორი-სამი წლის შემდეგ იძულებული ხდებოდნენ ახალ ნაკვეთზე გადასულიყვნენ. **მეორე ეტაპზე** ისინი **ირიგაციულ მიწათმოქმედებაზე** გადავიდნენ. ამ დროს ნატანი შლამის ხარჯზე სწრაფად ხდებოდა ნიადაგის ნაყოფიერების აღდგენა და ახალ ნაკვეთზე გადასვლა არ უხდებოდათ. მეურნეობის ხასიათზე დამოკიდებულებით სწაფად იზრდებოდა მოსახლეობის სიჭარბე. **სამონადირეო მეურნეობისას** ერთ კვადრატულ კილომეტრზე დაახლოებით **0,05** საპიენსი **მოდიოდა; სათოხნე მიწათმოქმედებისას** ეს რაოდენობა **10**-მდე, **ხოლო ირიგაციულ მიწათმოქმედებისას** - **100**...**200**-მდე გაიზარდა.

მითვისებითი მეურნეობიდან **აგრარულ მეურნეობაზე** გადასვლა უბრალო გადაბიჯების შედეგად კი არა მომხდარა, არამედ იგი მთელი რიგი ეკონომიკური მიღწევების დაგროვების შედეგია. ჩვენი აზრით, მის სახელწოდებად **აგრარული რევოლუციის** ნაცვლად **ეკონომიკური რევოლუციის** გამოყენება უფრო სამართლიანი იქნებოდა. ამის დასასაბუთებლად ტერმინ „ეკონომიკის“ წარმოშობის ისტორიის გახსენება იქნება უპრიანი.

ეკონომიკა *(ძვ. ბერძ* <u>οἶκος</u> *– „სახლი, მეურნეობა“+* <u>νόμος</u> *- „წესი, კანონი“; პირდაპირი გაგებით „საოჯახო მეურნეობის გაძღოლის წესი“)* **საოჯახო მეურნეობის გაძღოლის წესების დასადგენად** წარმოიშვა. წერი-

ლობით დოკუმენტებში, იგი პირველად ჩვენს წელთაღრიცხვამდე **IV** საუკუნეში დაფიქსირდა. ბერძენი ისტორიკოსი და პოლიტიკური მოღვაწე, **ქსენოფონტე** (*ძვ.-ბერძ. Ξεν-οφῶν; დაახ. 430 - 356 ჩვენს წელთაღრიცხვამდე*), მას **„საბუნებისმეტყველო მეცნიერებად**“ მიიჩნევდა, ბერძენი ფილოსოფოსი **არისტოტელე** *(ძვ. ბერძ. 384-322 ჩვენს წელთაღრიცხვამდე)* კი მას მეურნეობის (ეკონომიკის) სწორად გაძღოლისათვის ყველაზე მნიშვნელოვან კრიტერიუმს უწოდებდა.

ზემოთ განხილულმა რევოლუციამ, ფაქტობრივად, აგრარულ მეურნეობაზე უფრო **მეოჯახეობასა** და მეოჯახეობისათვის საჭირო საქმიანობას დაუდო საფუძველი. ეს რევოლუცია ≈**10 000** წლის წინ მოხდა. მისი წყალობით, **ჯერ ერთი, საპიენსებმა** ხელით მატერიალური ნედლეულის დამუშავების გზით გარკვეული პროდუქტების დამზადება დაიწყეს, მომთაბარეობიდან - ბინადრულ ცხოვრებაზე გადავიდნენ და თანამედროვე ადამიანებად ჩამოყალიბდნენ: დედამიწას მოევლინა მისი უდიდებულესობა **თანამედროვე ადამიანი**, რომელსაც დღეს, უბრალოდ, **ადამიანს** ვუწოდებთ; **და მეორეც**, ამ რევოლუციით **ეკონომიკას**, ანუ (*ზემოთ მოტანილი განსაზღვრების თანახმად*) **საოჯახო მეურნეობას** დაედო საფუძველი. აღნიშნულიდან გამომდინარე, გამართლებული იქნება, თუ ტერმინ „**აგრარულ რევოლუციას**“ ტერმინ „**ეკონომიკური რევოლუციით**“ შევცვლით.

1.8. ჩეზარე ემილიანისეული ჰოლოცენის კალენდარი

ყველაზე გავრცელებულ და მთელი მსოფლიოს მომცველ კალენდრად დღეს **გრიგორიანული კალენდარი** ითვლება. იგი წლების აღრიცხვას ქრისტეს დაბადებიდან იწყებს, რომელსაც **ახალი წალთღრიცხვა** ეწოდება. მის შემოტანამდე არსებობდა **ძველ წელთაღრიცხვად** წოდებული სხვა წელთაღრიცხვაც. იგი დაიწყო **ათვლის გარკვეული წლიდან**, რომელიც ყოველი ახალი წლის დადგმისას ერთით კი არ იზრდებოდა, არამედ მცირდებოდა.

ძველი წელთაღრიცხვის ზემოთ ნახსენებ ათვლის წლად **google**-ში მოძიებული ინფორმაციით, რომელიც **2024** წლის **4** იანვრითაა დათა-

რიღებული, **5508** რიცხვია მითითებული. მაშასადამე, თანამედროვე კაცობრიობამ ქრისტემდე **5508** წელი იცხოვრა

იტალიელ-ამერიკელმა მიკროპალეონტოლოგმა **ჩეზარე ემილიანიმ** (იტალ. *Cesare Emiliani; 1922 - 1995*) **1993** წელს **ნეოლითური რევოლუციის** შემდგომ დამდგარ ერას **ჰოლოცენის ერა** ანუ **ადამიანთა ერა** (***Human Era* - *HE***) უწოდა. მან დედამიწაზე თანამედროვე კაცობრიობის ცხოვრების დაწყების ძველი, **5508** წელი, ახალი **10 000** წლით შეცვალა. ყოველი მომდევნო წლის დადგომისას იგი **1**-ით მცირდება და **0**-ამდე დაცემის შემდეგ (ე. ი. რიგით **10001-ე** წლის დადგომისას) იწყება **ქრისტეს** დაბადების **პირველი** წელი, ე. ი. დგება წელი **1**.

ასეთ კალენდარს **ჰოლოცენის კალენდარი** ეწოდა. იგი მეტად საინტერესო აღმოჩნდა და სწრაფად მოიპოვა მრავალი პრაქტიკული გამოყენება. კერძოდ, მან გააიოლა **გეოლოგიური, არქეოლოგიური, დენდროქრონოლოგიური, ანთროპოლოგიური და ისტორიული დათარიღებები**. რამდენადმე დაწვრილებით განვიხილავთ იგი.

ქვემოთ მოყვანილია ჩვენ მიერ შემოთავაზებული ორი უმარტივესი წესი, რომელიც საშუალებას მოგვცემს დღევანდელი **გრიგორიანული კალენდრის** მეშვეობით განსაზღვრულ ახალი და ძველი წელთაღრიცხვის წლებს შევუთანადოთ **ჰოლოცენის კალენდრით** განსაზღვრული წლები.

პირველი წესი: ახალი წელთაღრიცხვით წარმოდგენილი წელი რომ გარდავქმნათ **ჰოლოცენის კალენდრით** წარმოდგენილ წლად, მას უნდა დავუმატოთ **10000**. ამისათვის ახალი წელთაღრიცხვით წარმოდგენილი წელი თუ:

ა) ოთხციფრიანია, მას წინ უნდა მივუწეროთ 1;
ბ) სამციფრიანია, მას წინ უნდა მივუწეროთ 10;
დ) ორციფრიანია, მას წინ უნდა მივუწეროთ 100;
ე) ერთციფრიანია, მას წინ უნდა მივუწეროთ 1000

მაგალითები:

- ახალი წელთაღრიცხვის **1825**, **1948** და **2024** წლებს შეესაბამება ჰოლოცენის კალენდრის 1**1825**, 1**1948** და 1**2024** წლები.
- ახალი წელთაღრიცხვის **145**, **384** და **837** წლებს შეესაბამება ჰოლოცენის კალენდრის 10**145**, 10**384** და10**837** წლები;
- ახალი წელთაღრიცხვის **24, 37** და **65** წლებს შეესაბამება ჰლოცენის კალენდრის 100**24,** 100**37** და 100**65** წლები;

● ახალი წელთაღრიცხვის **3, 5** და **9** წლებს შეესაბამება ჰოლოცენის კალენდრის 1000**3,** 1000**5** და 1000**9** წლები.

მეორე წესი: ძველი წელთაღრიცხვით წარმოდგენილი წელი რომ გარდავქმნათ **ჰოლოცენის კალენდრით** წარმოდგენილ წლად, საჭიროა **10001**-ს გამოვაკლოთ ძველი წელთაღრიცხვით წარმოდგენილი წლის მნიშვნელობა.

მაგალითები:

● ძველი წელთაღრიცხვის **4714** წელს შეესაბამება ჰოლოცენის კალენდრის 10001 - 4714 = **5287** წელი;

● ძველი წელთაღრიცხვის **295** წელს შეესაბამება ჰოლოცენის კალენდრის 10001 - 295 = **9706** წელი;

● ძველი წელთაღრიცხვის **45** წელს შეესაბამება ჰოლოცენის კალენდრის 10001 - 45 = **9956** წელი;

● ძველი წელთაღრიცხვის **8** წელს შეესაბამება ჰოლოცენის კალენდრის 10001 - 8 = **9993** წელი;

არისტოტელე ცხოვრობდა **ძველი წელთაღრიცხვის 384 - 322** წლებში. მათზე თვალის ერთი გადავლებისას ძნელი გასააზრებელია, თუ რა დიდი ხნის წინ უცხოვრია მას; გარდა ამისა, გარდაცვალების ასაკის დასადგენად გარდაცვალების თარიღს კი არ უნდა გამოვაკლოთ დაბადების თარიღი, როგორცაა მიღებული, არამედ პირიქით - დაბადების თარიღს - გარდაცვალების თარიღი (**384** – **322** = **62**).

ჰოლოცენის წელთაღრიცხვის გამოყენებისას ეს ლაფსუსები სწორდება. ამ წელთაღრიცხვით **არისტოტელე** (1001-384) - (1001-322) = **9617 - 9679** წლებში იცხოვრებდა. მათზე თვალის გადავლებისთანავე ნათელი ხდება, თუ რა დიდი ხნის წინ უცხოვრია ამ ცნობილ ადამიანს. ამასთანავე, მისი გარდაცვალების ასაკს, როგორც წესია, გარდაცვალების თარიღიდან დაბადების თარიღის გამოკლებით დავადგენთ: **არისტოტელე 9679 – 9617 = 62** წლის ასაკში გარდაცვლილა.

ახალი წელთაღრიცხვით გამოსახული წლები ძალიან მარტივად, გარდაიქმნება **ჰოლოცენის წლებად**. მაგალითად, ახლანდელ **2024** წელს შეესაბამება 1**2024** წელი, ე. ი. მას მხოლოდ წინ **1**-იანი ემატება. ყოველივე ჩამოთვლილი **ჰოლოცოცენის კალენდრის** ღირსებებია. იგი აერთიანებს ძველ და ახალ წელთაღრიცხვებს და ამკვიდრებს **ერთიან ზოგადსაკაცობრიო წელთაღრიცხვას!**

1.9. გზა ხელოვნური ინტელექტისაკენ

სამეწარმეო მეურნეობას, რომელზედაც ადამიანი **ნეოლითური რევოლუციის** შემდეგ გადავიდა, **საოჯახო საქმიანობის** სახე ჰქონდა. ადამიანებიდან თანდათან გამოიყო ცალკეული საქმის ხელით კარგად შესრულებაში დახელოვნებული ადამიანები - **ხელოსნები**, რომლებმაც **სახელოსნოები** შექმნეს და მათში დამზადებული **პროდუქტების** (*ლათ. Productus –„ქმნილება“, „ნაწარმი“*) გასხვისება შემოსავლის წყაროდ აქციეს. სახელოსნოების ნიდაგზე აღმოცენდა დაქირავებული მუშაკების ხელით შრომაზე დაფუძნებული საწარმოები - **მანუფაქტურები** (*manus „ხელი“ + facere „კეთება“*), სადაც შრომა ცალკეულ ოპერაციებად იყო დანაწილებული. **ინგლისში** არსებულ მანუფაქტურებში სამუშაოების შესასრულებლად ორთქლის ძრავებზე მომუშავე მანქანების გამოყენებამ (*ანუ, რაც იგივეა, შრომისათვის ბუნებრივი კუნთოვანი ძალის ნაცვლად მისგან თვისებრივად განსხვავებული **ორთქლის ძალის** გამოყენებამ*), ნახტომისებურად გაზარდა დამზადებული პროდუქტებისა და ქალაქებში მცხოვრები მოსახლეობის რაოდენობა. იგი ინგლისში მოხდა, ამიტომ მას **ინგლისური სამრეწველი რევოლუცია** (***1760 – 1840*** წწ) ეწოდა.

შრომის ტექნიკური საშუალებების გამრავალფეროვნების კვალობაზე აღნიშნულ რევოლუციას სხვა რევოლუციები მოჰყვა და მათთვის რიგითობის ნომრების მინიჭება დაიწყეს. ინგლისურ სამრეწველო რევოლუციას **პირველი** სამრეწველო რევოლუცია უწოდეს. მის შემდეგ მომხდარმა რევოლუციებმა მიიღეს **მეორე**, **მესამე** და **მეოთხე** სამრეწველო რევოლუციების სახელწოდებები **google**-ი კი ახლო მომავალში **მეხუთე** სამრეწველო რევოლუციის დადგომასაც გვიანონსებს.

ზემოთ ჩამოთვლილი რევოლუციების განხილვა სცდება ჩვენი წიგნის ფარგლებს, მაგრამ **მესამე სამრეწველო რევოლუციაზე** ორიოდე სიტყვის თქმა მაინც აუცილებლად მიგვაჩნია.

1970-იან წლებში მომხდარმა **მესამე სამრეწველო რევოლუციამ** სათავე დაუდო ინფორმაციულ საუკუნეს (*The Information Age*). მის აღსანიშნავად მრავალი სინონიმია გამოყენებული, კერძოდ: კომპიუტერული საუკუნე (*Computer Age*), ციფრული საუკუნე (*Digital Age*), სილიკონის საუკუნე (*Silicon Age*), ახალი მედიის საუკუნე (*New Media Age*), ინტერნე-

ტის საუკუნე (*Internet Age*) და ციფრული რევოლუცია (*The Digital Revolution*).

მესამე სამრეწველო რევოლუციის მნიშვნელობაზე მის შემდგომ დამდგარი საუკუნის სახელწოდების სინონიმების რაოდენობაც ღაღადებს. ამ რევოლუციის წყალობით **მატერიალური ტექნოლოგიის** გვერდით მტკიცედ დამკვდრდა მისგან თვისებრივად განსხვავებული **ინფორმაციული ტექნოლოგია.**

მატერიალურ ტექნოლოგიის დროს მატერიალური ნედლეულის (*მაგალითად, ყურძნის*) გადამუშავებით ისე მატერიალური პროდუქტი (*ღვინო*) მიიღებოდა. **ინფორმაციული ტექნოლოგიის** დროს როგორც ნედლეულის, ასევე პროდუქტის როლს არამატერიალური ბუნების მქონე **ინფორმაცია** ასრულებს. ნედლეულად გამოიყენება **ძველი** (*დასამუშავებელი*), პროდუქტად კი - **ახალი** *(ძველი ინფორმაციის დამუშავების შედეგად მიღებული*) ინფორმაცია. ასეთი რამ წინა წლებში არ მომხდარა: დედამიწაზე პირველად შემოაბიჯა მატერიალური ტექნოლოგიისაგან თვისებრივად განსხვავებულმა **ინფორმაციულმა ტექნოლოგიამ**: **ინფორმაცია** საქონლად გადაიქცა.

ინფორმაციულმა რევოლუციამ XX საუკუნეში რადიკალურად შეცვალა ინფორმაციის გადაცემა-შენახვის ხერხები და ინსტრუმენტული საფუძვლები, მნიშვნელოვნად გაზარდა მოსახლეობის აქტიური ნაწილისათვის ხელმისაწვდომი ინფორმაციის მოცულობა, შექმნა კაცობრიობის ინტელექტუალური უნარების გაერთიანებისათვის საჭირო ტექნოლოგიური საფუძველი.

მატერიალური და ინფორმაციული ტექნოლოგიების ათვისებამ ადამიანის პოტენციური შესაძლებლობები ისეთ სიმაღლეზე აიყვანა, რომ იგი დამაჯერებლად დაადგა ხელოვნური ინტელექტის შექმნისაკენ მიმართულ გზას.

1.10. ხელოვნური ინტელექტის დაბადება და პერსპექტივები

„ მსგავსი ყველაი მსგავსსა შობს, ესე გრძენთაგან თქმულია..“
შოთა რუსთაველი

„ დადგა დრო, ***რკინამ აიდგა ენა!***
ის ითხოვს პასუხს, როგორც მკვლელობა,
და სასტიკია, როგორც გენა,
მისი სისწრაფის ულმობელობა.”
გალაკტიონ ტაბიძე

XIX საუკუნის დასაწყისში ინგლისში ვერ ხერდებოდა უშეცდომოდ „საზღვაო ალმანახის“ (*«Nautical Almanac»*) გამოცემა, რადგან მასში არსებული ცხრილების შემდგენელი ასამდე გამომთვლი რუტინული არითმეტიკული ოპერაციების შესრულებისას პერმანენტულად უშვებდნენ ელემენტალურ შეცდომებს. მან გადაწყვიტა აეგო ამ ოპერაციების უშეცდომოდ შემსრულებელი მანქანა, რომელიც ჩაანაცვლებდა გამომთვლელებს და მოხერხდებოდა აღნიშნული ალმანახის უშეცდომოდ გამოცემა.

აღნიშნული მანქანის აგების პროცესი თითქმის მთელი საუკუნე გაგრძელდა და **XX** საუკუნის **40**-იან წლებში **დაპროგრამებადი კალკულატორების** შექმნით დამთავრდა, რომლის ერთადერთი მუშა კვანძი იყო **არითმეტიკული მოწყობილობა** (იხ. **ნახ. 1.1**). მას უშეცდომად შეეძლო ნებისმიერი არითმეტიკული ამოცანის შესრულება. ამით ფაქტობრივად რეალიზებული იქნა **ჩარლზ ბებიჯის** იდეა, მაგრამ მანქანის ფუნქციური შესაძლებლობების გაფართოებაზე მუშაობა გაგრელდა. არითმეტიკულ მოწყობილობას დაემატა **ლოგიკური მოწყობილობა** და მანქანის მუშა კვანძის ფუნქციის შესრულება დაიწყო **არითმეტიკულ-ლოგიკურმა მოწყობილობამ** (იხ. **ნახ. 1.3**), რომელსაც მოგვიანებით **პროცესორი** ეწოდა. აღნიშნული ცვლილების შემდეგ დაპროგრამებადი კალკულატორი **კომპიუტერად** გარდაიქმნა, რომელსაც არითმეტიკულის გარდა, **ლოგიკური ოპერაციების** შესრულებაც შეუძლია.

კომპიუტერს აქვს საკუთარი (*ორობითი*) ენა და როგორც არითმეტიკული, ისე ლოგიკური ოპერაციების შესასრულებლად საჭირო ელექტრონული კვანძები.

ენა და ლოგიკა განუხრელადაა დაკავშირებული აზროვნებასთან. აზროვნება ენასთან ერთად აღმოცენდა და განვითარდა, იგი აზროვნების იარაღი და გარსია; **ლოგიკას** კი (*ძვ. ბერძ. λόγος - „სწავლება, მეცნიერება*) სწორად აზროვნების მეცნიერებასაც უწოდებენ. აღნიშნულიდან გამომდინარე, ადამიანმა კომპიუტერში შესასრულებლად გადაიტანა საკუთარ აზროვნებასთან უშუალოდ დაკავშირებული გარკვეული ოპერაციები. აღნიშნულის გამო კომპიუტერმა შეიძინა აზროვნებისათვის დამახასიათებელი ცალკეული ოპერაციების ავტომატურად შესრულების უნარი. ფილოსოფოსები **ინტელექტს** აზროვნების **დისპოზიციას** უწოდებენ (იხ. თავი 1.5). აზროვნებითი პროცესების ავტომატურად მარეალიზებელი ხელოვნური ელემენტების კომპიუტერში დისპოზიციამ (განლაგებამ) სათავე **ხელოვნურ ინტელექტს** დაუდო.

ერთ-ერთი განსაზღვრების თანახმად, **ხელოვნურ ინტელექტი** ეწოდება ისტორიულად მხოლოდ ადამიანისათვის დამახასიათებელი რთული ფუნქციების (*განსჯის, გადაწყვეტილების მიღების, პრობლემის გადაწყვეტის*) შესრულების უნარის მქონე **კომპიუტერულ სისტემებს**. ამიტომ **ხელოვნურ ინტელექტს** მეტაფორულად ადამიანს შევადარებ. **ხელოვნური ინტელექტი** ჰგავს ადამიანს, რომელიც:

1) ძალიან სწრაფად კითხულობს და წაკითხულს არასდროს არ ივიწყებს;

2) წაკითხულიდან მიღებული ინფორმაციის საფუძველზე მყისიერად გამოაქვს დასკვნა;

3) არასდროს არ იღლება;

4) სრული და ზუსტი მონაცემების მიღებისას არასდროს არ უშვებს შეცდომას.

ხელოვნური ინტელექტის პრობლემით პირველად ინგლისელი მათემატიკოსი, ლოგიკოსი და კრიპტოგრაფი **ალან ტიურინგი** (*ინგ. Alan Mathison Turing; 1912-1954*) დაინტერესდა. **1950** წელს მიერ გამოქვეყნებული სტატია „**Computing Machinery and Intelligence**“ (*„კომპიუტერული ტექნიკა და ინტელექტი“*) ამ სფეროში დღემდე რჩება ყველაზე ხშირად ციტირებად ნაშრომად.

ტიურინგის აზროვნებამ მნიშვნელოვნად გაუსწრო იმდროინდელი მეცნიერების აზროვნებას. თანამედროვე კომპიუტერის აგებამ-

დე **12** წლით ადრე, **1936** წელს, მან აღმოაჩინა თანამედროვე კომპიუტერული ტექნიკის ფუნდამენტური პრინციპი - **შენახვადი პროგრამების დაპროექტება**. კემბრიჯის სამეფო კოლეჯში სამეცნიერო ხარისხის მიღებიდან (**1934** წლიდან) მოკლე დროში, **1937** წელს მან გამოაქვეყნა სტატია „**On Computable Numbers, with an Application to the Entscheidungsproblem: A correction**“, რომელშიც მსოფლიოში პირველად აღწერა აბრტრაქტული ციფრული გამომთვლელი მანქანა, რომელსაც დღეს **ტიურინგის მანქანას** უწოდებენ. **ტიურინგის** იდეებზე დაყრდნობით შეიქმნა პრაქტიკულად ყველა თანამედროვე კომპიუტერი, რამაც სათავე დაუდო **ხელოვნურ ინტელექტს**. სამწუხაროდ, **ტიურინგი** ძალიან ახალგაზრდა გარდაიცვალა და ბოლომდე ვერ მოახდინა ყველა თავისი ჩანაფიქრის რეალიზება.

დაწყებული საქმის დასრულებისა და კომპენტენტენტური დასკვნის გამოსატანად, **1956** წელს **15** ივნისს, **დარტმუნტის კოლეჯში** (აშშ) ორთვიან საზაფხულო სემინარზე მიიწვიეს **10** ამერიკელი მკვლევარი, რომელთა საქმიანობა დაკავშირებული იყო მართვის თეორიის, ავტომატების თეორიის, ნეირონული ქსელების, თამაშების თეორიისა და ინტელექტის კვლევის საკთხებთან. სია ასეთი იყო:

1. ჯონ მაკარტი (*ინგ. John McCarthy; 1927- 2011*), კომპიუტერული მეცნიერების სპეციალისტი, ტერმინ „ხელოვნური ინტელექტის“ ავტორი, დაპროგრამების ენა **ლისპის** ავტორი, ფუნქციური დაპროგრამების ავტორი, ტიურინგის პრემიის ლაურეატი; **2. მარვინ ლი მინსკი** (*ინგ. Marvin Lee Minsky; 1927-2016*); **3. კლოდ ელვინ შენონი** (*ინგ. Claude Elwood Shannon; 1916,— 2001*), ინფორმაციის თეორიის დამფუძნებელი, ინფორმაციის უმცირესი ერთეულის აღსანიშნავად ტერმინ „ბიტის“ შემომტანი; **4. ნათანიელ როსჩესტერი** (*ინგ. Nathaniel Rochester; 1919 – 2001*), პირველი მასობრივი მეცნიერული კომპიუტერ *IBM 701*-ის მთავარი არქიტექტორი, პირველი **ასემბლერის** დამწერი; **5. არტურ სამუელი** (*ინგ. Arthur Samuel; 1901-1990*), კომპიუტერული თამაშების სფეროს პიონერი მისი *Checkersplaying* პროგრმა მსოფლიოში პირველი თვითშემსწავლელი პროგრამაა; **6. ალენ ნიუელი** (*ინგ. Arthur Samuel; 1927-1992*), მონაწილეობდა დაპროგრამების ენა IPL-ის შექმნაში; **7. ჰერბერტ ალეხანდერ საიმონი** (*ინგ. Herbert A. Simon; 1916- 2001*); **8. ტრენჩარდ მორე** (*ინგ. Trenchard More; 1930- 2019*); **9. რეი სოლომონოფი** (*ინგ. Ray Solomonoff; 12926- 2009*); ამერიკელი მათემატიკოსი, ალგორითმული ალბათობის გამომგონებე-

ლი, ხელოვნური ინტელექტის განშტოების - მანქანური სწავლების შემქმნელი; **10. ოლივერ სელფრიჯე** (*ინგ. Oliver Selfridge, 1926-2008*).

სემინარზე დადგინდა, რომ **კომპიუტერს** შეუძლია ადამიანური ინტელექტის ზუსტი მოდელირება. ამ უნარმა **ხელოვნური ინტელექტის** სახელწოდება მიიღო. **ვიკიპედიაში** სტატიების დასათვალიერებლად დამუშავებულ ვებ-ინტერფეის WikiWand-ზე (*"viki" + "ჯადოსნური ჯოხი"*) ასეთი განსაზღვრებაა მოცემული: „**ხელოვნური ინტელექტი** მაშინაა ფორმირებული, როდესაც კომპიუტერებსა და მანქანებს ყველაფრის გაგება, შექმნა და განხილვა ადამიანის დაუხმარებლად შეუძლია. მათ უნდა შეეძლოს ინფორმაციის აღმოჩენა, შეკრება და, საკუთარი ცოდნის საფუძველზე, გადაწყვეტილებები მიღება. იგი დამოუკიდებლად მოქმედ გონიერ რობოტს ჰგავს."

ხელოვნური ინტელექტის ზემოთ მოყვანილი განსაზღვრებამ შეიძლება მისი ყოვლისშემძლებლობის ყალბი წარმოდგენა შეგვიქმნას და, როგორც საშინელებათა ფილმებშია, მომავალში იგი დედამიწაზე მოსასვლელად გამზადებულ მკაცრ მბრძანებლად მოგვეჩენოს. **დორტმუნტის** სემინარში მონაწილე ზემოთ ჩამოთვლილ მკვლევრებს შეიძლება **ხელოვნური ინტელექტის მამები** ვუწოდოთ. მათ შორის განსაკუთებული ადგილი უკავია **ჯონ მაკარტის**, რომლის მოფიქრებულია ტერმინი „**ხელოვნური ინტელაქტი**". აღნიშნული სემინარი მისი დიდი ძალისხმევით იქნა მოწვეული და თავად იყო მისი ერთ-ერთი ხელმძღვანელიც. გაგაცნობთ სემინარის წინ **მაკარტის** მიერ გაკეთებულ განცხადებას, რომელშიც მან ჩამოაყალიბა **ხელოვნური ინტელექტის ძირითადი დანიშნულება**:

„ჩვენ გთავაზობთ **10** ადამიანის მონაწილეობით **2** თვის განმავლობაში გამოვიკვლიოთ ხელოვნური ინტელექტი. კვლევა იმ ვარაუდს ეფუძნება, რომ ინტელექტის ნებისმიერი თვისება შეიძლება ისე ზუსტად აღიწეროს, რომ მანქანამაც შეძლოს მისი სიმულირება. ჩვენ შევეცდებით გავიგოთ, თუ როგორ უნდა ვასწავლოთ მანქანას გამოიყენოს **ბუნებრივი ენები**, ჩამოაყალიბოს ამოცანის გადაწყვეტის აბსტრაქციები და კონცეფციები, რაც ახლა მხოლოდ ადამიანებს შეუძლია და **გავიუმჯობესოთ საკუთარი თავი**. ჩვენ ვთვლით, ამ პრობლემებიდან ერთი ან რამდენიმე მათგანის მიღწევა სრულიად შესაძლებელია, თუ სპეციალურად შერჩეული მეცნიერების ჯგუფი მთელი ზაფხულის განმავლობაში მათზე იმუშავებს".

დიდებული მეცნიერის მართლაც დიდებული განმარტებაა: ხელოვნური ინტელექტის შექმნის ერთადერთი მიზანი ადამიანის მიერ **საკუთარი თავის გასაუმჯობესება** და არა დედამიწაზე მკაცრი ხელოვნური დიქტატორის მოვლინებაა!

ინოვაციების - ორთქლის ძრავზე მომუშავე მანქნების, ელექტრული მანქანების, მაცივრების, ატომური ელექტროსადგურებისა და სხვათა - გამოჩენას ადამიანი სიხარულთან ერთად გარკვეული შიშითაც ხვდებოდნენ. მსგავს მოვლენასთან გვაქვს საქმე ხელოვნური ინტელექტის შემთხვევაშიც. მისი დაძლევის შესანიშნავ გზას გვიჩვენებს ცნობილი ბრიტანელი ფიზიკოს-თეორეტიკოსი, კოსმოლოგი და ასტროფიზიკოსი **სტივენ უილიამ ჰოკინგი** (*ინგ. Stephen William Hawking; 1942-2018*), როდესაც წერს: „კომპიუტერების ხელოვნური ინტელექტი უახლოესი **100** წლის განმავლობაში გადაასწრებს ადამიანურ ინტელექტს; როდესაც ეს მოხდება, იმაში უნდა ვიყოთ **დარწმუნებული**, რომ კომპიუტერისა და ადამიანის მიზნები ერთმანეთს კი არ დაეჯახება, არამედ დაემთხვევა“.

დედამიწა **≈4,5** მილიარდი წლისაა. **≈1,3** მილიარდი წლის შემდეგ ცოცხალი ორგანიზმების უმრავლესობისათვის იგი საცხოვრებლად არ ივარგებს: აუცილებლად მოგვიწევს სხვა პლანეტაზე გადასახლება. ჩვენ დედამიწაზე მარტოდმარტოდ მცხოვრები მოაზროვნე არსებები ვართ (*იხ. § 1.6.7*) და ამ პრობლემის გადაწყვეტაში დამხმარედ, **ხელოვნური ინტელექტის** გარდა, ვერავის ვერ ვიპოვით. ამიტომ უნდა გავამართლოთ **ჰოკინგის** იმედები და დავმეგრობდეთ ხელოვნური ინტელექტის მატარებელ კომპიუტერულ სისტემებთან. ამისათვის მათთან საურთიერთო ენების კარგად ცოდნა დაგვჭირდება. კომპიუტერულ ენებზე მოცემული წიგნის მეორე და მესამე ნაწილებში გესაუბრებით.

☆ ☆ ☆

მეორე და მესამე ნაწილებში შედარებით დაწვრილებით განვიხილავთ კომპიუტერულ ენებს. ახლა კი მოკლედ გაგაცნობთ დღეს თუ როგორ „ვესაუბრებით“ კომპიუტერს.

ადამიანები ერთმანეთთან საუბრისას იყენებენ სალაპარაკო ენებს, რომლებსაც **ბუნებრივ** ენები ეწოდება. მსოფლიოში გავრცელებულ ენების შესახებ ყველაზე ცნობილ ცნობარ „**Ethnologue: Languages of the World**“-დან ვიგებთ, რომ **2022** წლის მდგომარეობის მიხედვით

მსოფლიოში არსებობს **142** სხვადასხვა ენობრივ ჯგუფში შემავალი **7151** ბუნებრივი ენა. ასეთი ენებია ქართული, ინგლისური, გერმანული, ებრაული და ა. შ. ენები. ყოველი ენა შედგება გარკვეული რაოდენობის ასოებისაგან. მაგალითად, ქართულ ენაში გვაქვს **33**, ინგლისურსა და გერმანულში - **26**, ებრაულში (ივრითში) - **22** ასო.

კომპიუტერისათვის ადამიანმა შექმნა ორი ასოსაგან შემდგარი უმარტივესი ენა, რომელსაც **მანქანური ენა** ეწოდება. ამ ასოების აღნიშვნისათვის მან ორი არაბული ციფრი **0** და **1** გამოიყენა. მასში შემავალი ასოების რაოდენობის გამო მანქანურ ენას ხშირად **ორობით ენასაც** უწოდებენ.

ადამიანი ალგორითმულად მოაზროვნე არსებებაა: იგი რაიმე მიზნის მისაღწევად იყენებს სალაპარაკო ენის მეშვეობით ჩამოყალიბებულ ალგორითმს: მასში შემავალი მოქმედებების (ბრძანებების) რეალიზების მეშვეობით აღწევს დასახულ მიზანს.

ალგორითმი შეიძლება შედგეს როგორც ვერბალურად (სიტყვიერად), ასევე წერილობით. წერილობითი სახით შედგენილ ალგორითმის აღსანიშნავად გამოიყენება ტერმინი **პროგრამა** (*ბერძ. προ – „-თვის“, γράμμα - „ჩანაწერი“*), რაც ქართულად „წინასწარ ჩანაწერს“ ნიშნავს. ბუნებრივ ენაზე დაწერილი ალგორითმი ამ ენის მცოდნე ადამიანისათვის **სამუშაო პროგრამაა**. ანალოგიურად, მანქანურ ენაზე ჩაწერილი ალგორითმი კომპიუტერისათვის სამუშაო პროგრამა იქნება და მას **კომპიუტერული პროგრამაც** უწოდებენ.

ბუნებრივ (სალაპარაკო) ენაზე დაწერილი პროგრამის შესრულება, ამ ენის შინაარსის კონტექსტური (არაცალსახა) გაგების გამო, კომპიუტერს არ შეუძლია: იგი მხოლოდ მანქანურ ენაზე უნდა იყოს შედგენილი.

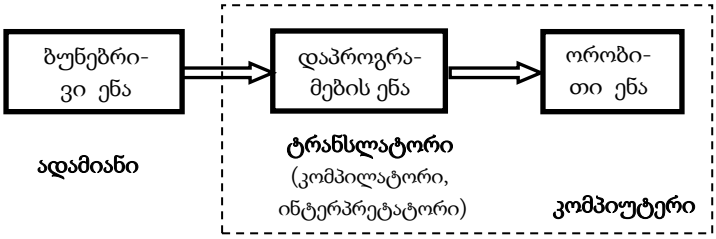
ბუნებრივ ენაზე დაწერილი პროგრამა მანქანურ ენაზე შეგვიძლია „ინფორმაციის გასაცვლელი სტანდარტული კოდის“ („***A**merican **S**tandard **C**ode for **I**nformation **I**ntercange*“ - ***ASCII***) გამოყენებით გადავიყვანოთ. იგი მეტად შრომატევადია და შემდეგი უფრო მარტივი გზა იქნა ნაპოვნი.

შეიქმნა ბუნებრივ ენასთან ძალიან დაახლოები და მასზე გაცილებით მარტივი **ფორმალური ენა**, რომელსაც **დაპროგრამების ენა** ეწოდება და აიგო დაპროგრამების ენიდან მანქანურ ენაზე მთარგმნელი მოწყობილობა **ტრანსლატერი**, რომლითაც დღეს აღჭურვილია თანამედროვე კომპიუტერი. იგი დაპროგრამების ენაზე დაწერილი პროგ-

რამას გარდაქმნის ორობით ენაზე დაწერილ პროგრამად, ანუ **კომპიუტერულ პროგრამად**.

განასხვავებენ ორი სახის ტრანსლატერს - კომპილატორსა და ინტერპრეტატორს. **კომპილატორი** დაპროგრამების ენაზე დაწერილ პროგრამას მთლიანად გარდაქმნის ორობით ენაზე დაწერილ პროგრამად და მერე გადასცემს მას კომპიუტერს შესასრულებლად. **ინტერპრეტატორი** პროგრამაში შემავალი თითოეული ბრძანების ორობით ენაზე გადაყვანის შემდეგ გადასცემს მას კომპიუტერს შესასრულებლად. მისი შესრულებისა და სათანადო შეტყობინების მიღების შემდეგ იგი გადადის მომდევნო ბრძანებაზე და მის მიმართ იმეორებს ზემოთ აღწერილ პროცედურებს.

კომპილატორი უფრო სწრაფია (იგი ბრძანებების ცალ-ცალკე გადაცემაზე არ ჰკარგავს დროს) ვიდრე ინტერპრეტატორი, სამაგიეროდ ინტერპრეტატორი შეუცვლელია კომპიუტერის **დიალოგურ** (**ინტერაქტიულ**) რეჟიმში მუშაობისას, როდესაც მოცემულ მომენტში გადასაცემი ბრძანებას განსაზღვრავს წინა ბრძანების შესრულებისას მიღებული შედეგის შინაარსი.



ნახ. 1.9. *პროგრამის გადაყვანა ბუნებრივი ენიდან ორობით ენაზე*

ტრანსლატორით აღჭურვილი კომპიუტერის მუშაობის ზემოთ აღწერილი პროცესის ბლოკური სქემა **1.9** ნახაზზეა მოყვანილი. ტრანსლატორის მეშვეობით ადამიანსა და კომპიუტერს შორის ურთიერთობის ზოგადი ალგორითმი ასეთია:

1. ადამიანი კომპიუტერზე შესასრულებელ პროგრამას ან უშუალოდ დაპროგრამების ენაზე ადგენს, ან მის შესადგენად იყენებს ბუნებრივ ენას და შემდეგ გადაჰყავს დაპროგრამების ენაზე;

2. დაპროგრამების ენაზე დაწერილი პროგრამა გადაეცემა კომიუტერში არსებულ ტრანსლატორს - კომპილატორს ან ინტერპრეტა-

ატორს. ამასთანავე, კომპიუტერის დიალოგურ (ინტერაქტიურ) რეჟიმში მუშაობისას იგა გადაეცემა ინტერპრეტატორს, საწინააღმდეგო შემთხვევაში კი - კომპილატორს.

3. ტრანსლატორი დაპროგრამების ენაზე დაწერილ პროგრამას გარდაქმნის მანქანურ (ორობით) ენაზე დაწერილ პროგრამად და კომპიუტერის სათანადო კვანძს გადასცემს შესასრულებლად;

4. მანქანურ ენაზე დაწერილ პროგრამას დამხმარე კვანძების მეშვეობით ასრულებს კომპიუტერის პროცესორი და მიღებული შედეგი მეხსიერების გავლით გადაეცემა კომპიუტეზე მომუშავე პირს.

1.11. „ინფორმატიკა“ თუ „კომპიუტერული მეცნიერება“? - „Gentlmen, let us COMPUTE !“

კომპიუტერული მოწყობილობებისა და პროგრამული უზრუნველყოფის შემსწავლელი დისციპლინის სახელწოდებისათვის ტერმინი „**ინფორმატიკა**“ გერმანელმა კიბერნეტიკოსმა **კარლ შტეინბუხმა** (*გერმ. Karl Steinbuch; 1917-2005*) შემოიტანა **1957** წელს გამოქვეყნებულ სტატიაში „Informatik: Automatische Informationsverarbeitung“ *(„ინფორმატიკა: ინფორმაციის ავტომატური დამუშავება“)*. იგი მან მიიღო ტერმინების „ინფორმატიკისა“ და „ავტომატიკის“ შემდეგნაირად შერწყმით.

ინფორმ~~ატიკა~~ + ~~ავტომ~~**ატიკა** = **ინფორმატიკა**.

ტერმინი „**ინფორმატიკა**“ ევროპელი სპეციალისტებმა მოიწონეს და იგი ევროპის ქვეყნებში (მათ შორისა, ჩვენთანაც) დაინერგა. ამ ტერმინის თანახმად კომპიუტერი ინფორმაციის ავტომატურად დამამუშავებელ მანქანად მიიჩნევა, რაც, როგორც ქვემოთ დავინახავთ, მთლად ზუსტი არ არის. ზოგადად კი ტერმინის მოწყვლადობას მის სტრუქტურაში „**ინფორმაციის**“ ცნების არსებობა იწვევს, რომელიც მეცნიერებაში ერთ-ერთ ყველაზე მეტად საკამათო ცნებაა.

მიუხედავად იმისა, რომ ტერმინი „**ინფორმაცია**“ მყარად შემოვიდა როგორც ყოველდღიურ ცხოვრებაში, ისევე თანამედროვე მეცნიერების და ტექნიკის სხვადასხვა დარგში, მისი ზუსტი მეცნიერული განსაზღვრება დღემდე მაინც ვერ მოხერხდა. ამიტომ ინფო-რმაციის განსაზღვრების ნაცვლად **ინფორმაციის ცნებას** იყენებენ. გა-

ნსაზღვრებისგან ცნება იმით განსხვავდება, რომ მეცნიერებისა და ტექნიკის სხვადასხვა სფეროში (მათემატიკაში, ფიზიკაში, ბიოლოგიაში, სოციოლოგიაში და ა. შ.) **ცნების** ფორმულირებას ისე ახდენენ, რომ მისი შინაარსი მაქსიმალურად დაემთხვეს კონკრეტული დისციპლინის საგანსა და გადასაწყვეტ ამოცანებს.

სპეციალისტებს, რომლებიც შეისწავლიან ინფორმაციის სტრუქტურასა და თვისებებს, აგრეთვე ასრულებენ მისი შეკრების, დამუშავების, შენახვის, გავრცელებისა და გამოყენების ოპერაციებს, ინფორმაციის ასეთი ცალმხრივი განმარტებები არ აკმაყოფილებს. მათთვის განსაკუთრებით მიუღებელია ინფორმაციის აზრობრივი, სემანტიკური შინაარსის იგნორირება. გარდა ამისა, მათ სჭირდებათ ერთმანეთისაგან მკვეთრად გამიჯნონ ხშირად **სინონებად მიჩნეული** ისეთი ტერმინები, როგორიცაა, „ცნობები“, „მონაცემები“, „ინფორმაცია“ და „ცოდნა“, დააგინონ მათ შორის არსებული კავშირები. ხშირად ისინი იძულებულნი არიან აღნიშნული ტერმინები ერთმანეთის გამოყენებით განმარტონ, რაც მანკიერ ლოგიკურ წრეებს წარმოქმნის. ეს კარგად ჩანს ცნებების „**მონაცემებისა**“ და „**ინფორმაციის**“ განმარტებებიდან, რომლებიც საბჭოთა და რუსმა მეცნიერმა, ცნობილმა ინფორმატიკოსმა, **რუდჟერო გილიარევსკიმ** (*რუს. Гиляревский Руджеро Сергеевич; 1929. 95 წლის*) ჩამოაყალიბა [**82, 83**]:

მონაცემები ფაქტების, იდეებისა და ცნობების დედაარსია, რომელებიც ნიშნური (სიმბოლური) სახითაა წარმოდგენილი. წარმოდგენის ასეთი ფორმა მისი გადაცემის, დამუშავებისა და ინტერპრეტირების საშუალებას იძლევა. **ინფორმაცია** ადამიანის მიერ მონაცემებისათვის მიწერილი აზრია; ამას იგი მონაცემებში ფაქტების, იდეებისა და შეტყობინებების განთავსების მისთვის ცნობილი წესების საფუძველზე ახდენს.

ინფორმაციის ასეთი გაგება თავად ამ ტერმინის ეტიმოლოგიას (*ლათ. Informatio – „ცნობა“, „განმარტება“, „გადმოცემა“*) შეესაბამება. სტრუქტურირებული, ანუ მიზეზშედეგობრივი და სხვა მიმართებებით დაკავშირებული და სისტემატიზებელი ინფორმაცია **ცოდნად** გარდაიქმნება.

კომპიუტერები **მონაცემებს** გადაამუშავებენ. მოყვანილი განმარტებებიდან ჩანს, რომ მონაცემები მხოლოდ მაშინ გარდაიქმნება **ინფორმაციად**, როდესაც მას ადამიანი აღიქვამს და მოახდენს მის ინტერპრეტირებას. სწორედ ამ შემთხვევაში გარდაიქმნება „თავისთავად

არსებული ინფორმაცია“ – „ჩვენ კუთვნილ ინფორმაციად“. მონაცემები, **გილიარევსკის** აზრით, გარკვეულწილად, წერილობითი შეტყობინების მსგავსია, რომელიც წერა-კითხვის მცოდნე ადამიანს გადასცემს გარკვეულ ცნობებს, მაგრამ გაურკვეველი რჩება წერა-კითხვის უცოდინარი ადამიანისათვის.

ამგვარად, ინფორმაცია მონაცემების პოტენციური თვისებაა. იგი შეიძლება რეალიზდეს ზოგიერთი ადამიანებისათვის, ზოგიერთებისათვის კი არა. **მანქანური** დამუშავების ობიექტი **მონაცემებია** და არა **ინფორმაცია**, რადგან მანქანას არ შეუძლია მისი ინტერპრეტირება, ესე იგი, ინფორმაციად მათი გარდაქმნა. ამას განაპირობებს ის გარემოება, რომ, ადამიანისაგან განსხვავებით, **მანქანას** არ გააჩნია სამყაროს შესახებ ცოდნის აუცილებელი მარაგი და **მას არ შეუძლია აზროვნება**. ასეთია ცნებებ „მონაცემებსა“ და „ინფორმაციას“ შორის არსებული დიალექტიკური კავშირი.

ინფორმაციის შემსწავლელი დისციპლინა კომპიუტერის აგებამდე ჩამოყალიბდა და მას „**ინფორმატიკა**“ ეწოდება. იგი ფართოდ გამოიყენებოდა და დღედაც გამოიყენება **საბიბლიოთეკო საქმიანობაში**. მისი ძირითადი დანიშნულება იყო ინფორმაციათა უკიდეგანო სივრცეში საჭირო ინფორმაციის უსწრაფესად მოძიება და მისი დამუშავების პროცესის მაქსიმალურად გამარტივება.

აღნიშნულის გამო კომპიუტერული მოწყობილობებისა და პროგრამული უზრუნველყოფის შემსწავლელი დისციპლინის სახელად ტერმინ „**ინფორმატიკის**“ გამოყენება რამდენადმე არაკორექტულია.

ტერმინი „**კომპიუტერული მეცნიერება**“ („**Computer science**“) პირველად აშშ-ში გამომავალ ჟურნალ „Communications of the ACM“-ში **1959** წელს გამოქვეყნებულ ერთ-ერთ სტატიაში გამოჩნდა. მასში ავტორმა წამოაყენა წინადადება, რომ ამერიკაში გახსნილიყო **1921** წელს დაარსებული **ჰარვარდის ბიზნეს-სკოლის მსგავსი** უმაღლესი სკოლა, რომელშიც ახალგაზრდები შეისწავლიდნენ **კომპიუტერული მეცნიერებას** (**Computer science**). მოულოდნელად გამოჩენილმა ამ ტერმინმა ჯერ ამერიკაში, ხოლო შემდეგ - ინგლისურენოვან სახელმწიფოებში მოღვაწე სპეციალისტების საყოველთაო მოწონება დაიმსახურა და იგი იქ დღემდე გამოიყენებენ ტერმინ „ინფორმატკის“ ნაცვლად.

ტერმინი „**კომპიუტერული მეცნიერება**“ თავისუფალია ტერმინ „ინფორმატიკისათვის“ დამახასიათებელი ზემოთ აღნიშნული ხარვეზისაგან და ნათლად გამოხატავს კომპიუტერის დანიშნულებას, რომ-

ელიც ხატოვნად ჩამოაყალიბა **ლაიბნიცმა** თავის ცნობილ გამოთქმაში (იხ. § **1.3.2**): „**Gentlmen, let us COMPUTE!**“ ასე რომ, კითხვაზე: **„ინფორმატიკა“ თუ „კომპიუტერული მეცნიერება?“** შეგვიძლია თამამად ვუპასუხოთ: „**კომპიუტერული** (გამოთვლითი) **მეცნიერება**, ბატონებო!“

1.12. ხელოვნური ინტელექტისაკენ მიმავალ გზაზე დატოვებული ქართული ნაკვალევი

ჩარლზ ბებიჯის მიერ **1834** წელს დამუშავებულ დაპროგრამებად კალკულატორში (იხ. ნახ. 1. 1) არსებული არითმეტიკული მოწყობილობის მთავარი ფუნქცია იყო შეესრულებინა პროგრამით გათვალისწინებული **არითმეტიკულ ოპერაციათა მიმდევრობები**. დანარჩენი კვანძების (*შეტანა-გამოტანის მოწყობილობები, მეხსიერება და მართვის მოწყობილობა*) დანიშნულება იყო არითმეტიკულ მოწყობილობას დახმარებოდა მასზე დაკისრებული მოვალეობის შესრულებაში. ამიტომ **არითმეტიკული მოწყობილობა** შეიძლება **ოსტატს** *(ლათ. Magister – „უფროსი, მთავარი, მმართველი, ზედამხედველი, ხელმძღვანელი“ [84, 85, 86])*, დანარჩენი კვანძები კი - **ოსტატის დამხმარე შეგირდებს** შევადაროთ.

პირველი მექანიკური არითმეტიკული მოწყობილობა (მანქანა) **XVII** საუკუნეში გამოიგონეს. **XX** საუკუნის **20**-იან წლებამდე ბევრი მისი ნაირსახეობა შეიქმნა, მაგრამ თითოეული მათგანი მექანიკურ მოწყობილობებად რჩებოდა. თავის დროზე სწორედ ამან შეუშალა ხელი **ბებიჯს** მიეღწია დასახული მიზნისათვის - აეგო კომპიუტერი.

ასეთი ერთფეროვნება **1928** წლის **16** იანვარს დაირღვა, როდესაც ცნობილმა ფრანგმა მათემატიკოსმა **მორის დ'ოკანმა** (*ფრ. Philbert Maurice d'Ocagne, 1862 – 1938*) საფრანგეთის აკადემიაში შეკრებილ მეცნიერნებს წარუდგინა ახალგაზრდა ქართველი მათემატიკოსის **გიორგი ნიკოლაძის (1888 -1931)** მიერ გამოგონებული **არითმეტიკული მანქანა**. გამომსვლელის თანახმად, იგი იყო ... **პირველი ასეთი მანქანა** [**77**]!

აღნიშნული მანქანის მოქმედი მაკეტი გამოფენილი იყო მოსკოვის პოლიტექნიკურ მუზეუმში, საიდანაც იგი **1936** წელს დაიკარგა. საქართველოს ტექნიკური უნივერსიტეტის **ინფორმატიკისა და მარ-**

თვის სისტემების ფაკულტეტის თანამშრომლებმა მანქანის ზუსტი მოქმედი ასლი აღადგინეს. აღდგენაში შეტანილი წვლილისათვის, ფაკულტეტის დღისადმი მიძღვნილ ღონისძიებაში, რომელიც **1923** წლის ივნისში გაიმართა, ასლის ერთი ეგზემპლარი **უნივრსიტეტის რექტორს** გადაეცა. ახლა იგი უნივერსიტეტის ადმინისტრაციულ კორპუსშია გამოფენილი [**78**, **79**].

ელექტრული არითმეტიკული მანქანის გამოყენება მნიშვნელოვნად გაამარტივებდა კომპიუტერის შექმნის პროცესს და შესაძლებელს გახდიდა წარმატებით დასრულებულიყო ბებიჯის მიერ დაწყებული საქმე.

სამწუხაროდ, **გიორგი ნიკოლაძეს** არ დასცალდა ამ მიმართულებით მუშაობის გაგრძელება - ფილტვების ანთებით იგი უდროოდ (**43** წლის ასაკში) გარდაიცვალა და მისი დაწყებული საქმე დროებით შეჩერდა. ამის გამო კიდევ რამდენიმე წლით გადაიდო ბებიჯის იდეის რეალიზაცია.

მეორე მსოფლიო ომის დაწყების შემდეგ არსებულმა სიტუაციამ გააქტიურა კომპიუტერის შექმნის პრობლემა. მოწინავე ქვეყნებში გაიცა მისი გადაწყვეტის მითითებები და გამოიყო საჭირო დაფინანსებები. ძალისხმევამ შედეგი გამოიღო და წინა საუკუნის **40**-იან წლებში შეიქმნა პირველი ელექტრონული კომპიუტერი. ამან **ხელოვნური ინტელექტის** შექმნის ოცნება რეალური გახადა. აღნიშნულის გამო, **გ. ნიკოლაძის** ელექტრული არითმეტიკული მოწყობილობა შეიძლება **ხელოვნური ინტელექტისაკენ მიმავალ გზაზე ქართველი მეცნიერის მიერ დატოვებულ ნაკვალევს** შევადაროთ!

წიგნის თავდაპირველი ვარიანტისათვის მოცემული პარაგრაფის დამატების იდეა თავაზიანად შემომთავაზა ამ წიგნის რედაქტორმა, ბატონმა **ბადრი ცხადაძემ**. ამან საშუალება მომცა მკითხველთა ფართო წრისთვის შედარებით ფართოდ გამეცნო ზემოთ აღნიშნული მეტად საინტერესო გამოგონება. მისივე რჩევით, მკითხველებს გავაცანი ბატონ **ნიკო ბერძენიშვილის** ერთ-ერთი უმნიშვნელოვანესი მოსაზრება, რომელიც 88-89 გვერდებზე განსხვავებული შრიფტითაა დაბეჭდილი. აღნიშნულის გამო დიდი მადლობა მინდა გადავხადო ბატონ **ბადრი ცხადაძეს.**

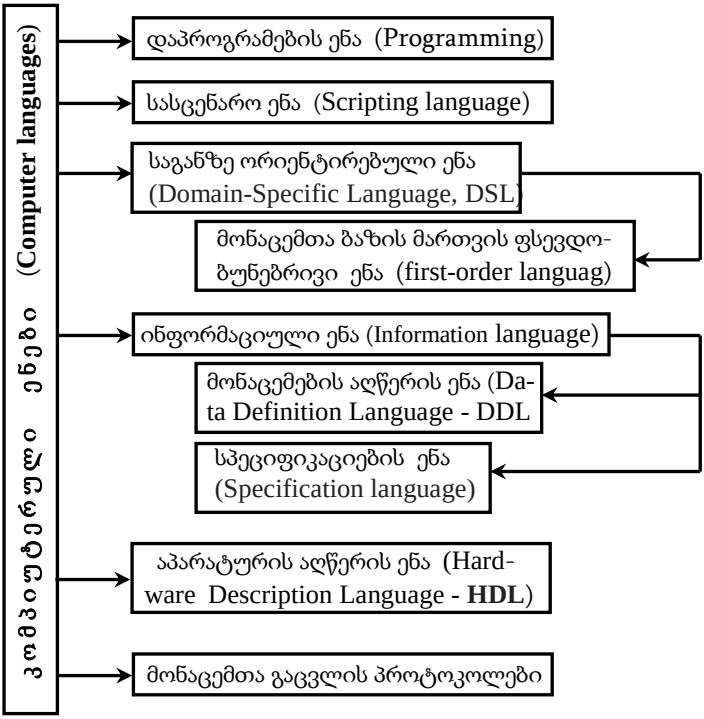
მეორე ნაწილი

კომპიუტერული ენების ცნება და მათი კლასიფიცირების პრობლემა

2.1. კომპიუტერული ენები

კომპიუტერული ენები (Computer language) კომპიუტერულ ტექნიკასთან ასოცირებულ ენებს ეწოდება. ძალიან ხშირად ეს ცნება თანხვდება **დაპროგრამების ენას**, მაგრამ მათ შორის ტოლობის დასმა არ შეიძლება.

ნებისმიერი სხვა ენის მსგავსად **კომპიუტერული ენა** მაშინ ჩნდება, როდესაც საჭირო ხდება ერთი წყაროდან ინფორმაცია გადაეცეს მეორე წყაროს. **დაპროგრამების ენა** დამპროგრამებელსა და კომპიუტერს შორის ინფორმაციის გადაცემას უწყობს ხელს. მისგან განსხავავებით, მაგალითად, **მონიშვნის ენები** განსაზღვრავს დოკუმენტების (ეკრანული წარმოდგენის) სტრუქტურას, რომელიც გასაგებია როგორც კომპიუტერისათვის, ისე ადამიანებისათვისაც და ა. შ. კომპიუტერულ ენას ხშირად კომპიუტერთან ამა თუ იმ სახის ურთიერთობით დაკავებულ ადამიანთა ჯგუფის წევრებს შორის გავრცელებულ **სლენგთან** (სიტყვების ნაკრებთან ან არსებული სიტყვების ახალ მნიშვნელობებთან, ჟარგონთან) აიგივებენ. კომპიუტერული ენების კლასიფიკაცია მოცემულია **ნახ. 1**-ზე. ნახაზიდან ჩანს, რომ დაპ;როგრამების ენა, რომელზედაც ვსაუბრობდით წინა თავში, კომპიუტერული ენების ერთ-ერთი ნაირსახეობაა. მოკლედ განვიხილოთ ცხრილში მოყვანილი ენები.



ნახ. 1.1. კომპიუტერული ენების კლასიფიკაცია

დაპროგრამების ენა ეწოდება კომპიუტერული პროგრამების ჩასაწერად განკუთვნილ ენას. იგი განსაზღვრავს ლექსიკური, სინტაქსური და სემანტიკური წესების იმ ნაკრებს, რომელთა მეშვეობითაც ჩამოყალიბებულია ზოგადად შემსრულებლისა და, კერძოდ, კომპიუტერის მიერ შესასრულებელი პროგრამისა მოქმედებების გარეგნული სახე.

სასცენარო ენა (ინგ. scripting language; „script“ - სცენარი) არის სისტემის მიერ შესასრულებელი მოქმედებების მოკლე სცენარების აღმწერი ენა. მას ინგლისური ტერმინოლოგიის გამოყენებით **სკრიპტულ ენასაც** უწოდებენ.

სასცენარო Tcl ენის შემქმნელი ამერიკელი მეცნიერის **ჯონ უსტერჰაუტის** (John Ousterhout. **1954**) თანახმად მაღალი დონის ენები შეიძლება დაიყოს სისტემური დაპროგრამების ენებად (system programming

languages) და სასცენარო ენებად. ამ უკანასკნელ ენებს იგი **შემაწებებელ ენებს** (glue languages) ან **სისტემური ინტეგრაციის ენებსაც** უწოდებდა. სცენარები ჩვეულებრივ ინტერპრეტირდება და არ კომპილირდება.

საგანზე ორიენტირებული ენა (domain-specific language, DSL) სპეციალიზებულია კონკრეტულ სფეროში გამოუენებისათვის (მისი საპირისპირო ენაა **საერთო გამოყენების ენა**, რომელიც სფეროთა ფართო სპექტრისათვის შეიძლება გამოვიყენოთ და რომელიც არ ითვალისწინებს ცოდნის კონკრეტულ სფეროთა თავისებურებებს). ასეთი ენის აგებულება და/ან მისი მონაცემების სტრუქტურა ასახავს ასახავს მის მიერ გადასაწყვეტი ამოცანების სპეციფიკას.

მკაცრად თუ ვიმსჯელებთ, დაპროგრამების ენების დაყოფა საერთო დანიშნულელებისა და საგანზე ორიენტირებულ ენებად ძალიან პირობითია, განსაკუთრებით თუ გავითვალისწინებთ, რომ ფორმალურად ნებისმიერი ფაილი ან ფაილების ფორმატი ენას წარმოადგენს. არსებობს საერთო დანიშნულების უამრავი ენა, რომლებიც საგანზე ორიენტირებულ ენად გამოიყენება განსაზღვრული ამოცანების გადასაწყვეტად და, პირიქით, არსებობს საერთო ენებად გამოყენებადი საგანზე ორიენტირებული ენები.

მონაცემთა ბაზის მართვის ფსევდობუნებრივი ენა (first-order languag) არის ენა, რომლის კონსტრუქცია წინასწარგანზრახულად ბუნებრივი (ინგლისური, გერმანული და ა. შ.) ენის მსგავსად არის რეალიზებული. ისინი გამოუცდელი მომხმარებლებზეა გათვლილი. ზოგიერთ ფსევდობუნებრივ (მაგალითად, SQL) ენაში ბუნებრივ ენას მხოლოდ უმარტივესი კონსტრუქციები ჩამოგავს, ხოლო რთულ მოთხოვნებს აშკარად „კომპიუტერული“ სახე აქვს. უმარტივესია უმრავლესი ფსევდობუნებრივი ენების სინტაქსი, რაც ენის განხილვის პროგრამის ადვილად დაწერის საშუალებას გვაძლევს.

ინფორმაციული ენა (information language) ინფორმაციის დამამუშავებელ სხვადასხვა სასტემებში გამოიყენება. ინფორმაციული ენებისაგან უნდა განასხვავოთ დაპროგრამების ენები, მანქანური ენები და მეცნიერების ფორმალიზებული ენები. ჩვეულებრივად განასხვავებენ:

• **ინფორმაციულ-ლოგიკურ ენას,** რომელიც გამოიყენება ინფორმაციულ-ლოგიკური სისტემებისათვის. უპირველეს ყოვლისა, ასეთი ენებია ცოდნის წარმოდგენისათვის განკუთვნილი ენები (მაგალითად, SC, SCR, SCL) და მონაცემთა ბაზების ენები (მაგალითად, SQL);

• **ინფორმაციულ-საძიებო ენას**, რომელიც არის ნიშნური სისტემა, რომელიც გამოიყენება ტექსტების (დოკუმენტების) ან მათი ნაწილების ძირითადი აზრობრივი აღწერისათვის, აგრეთვე ინფორმაციული ძიების რეალიზაციის მიზნით ინფორმაციული შეკითხვების აზრობრივი შინაარსის გამოსახატავად. ინფორმაციულ-საძიებო ენის მაგალითია **ბიბლიოგრაფიული აღწერის ენა**, რომელიც განკუთვნილია ტექსტების იდენტიფიცირებისათვის და გამოიყენება ალფაბეტურ კატალოგებში, კარტოტეკებსა და ბიბლიოგრაფიულ მაჩვენებლებში. მის შემადგენლობაში შედის ბიბლიოგრაფიული ელემენტები (ავტორთა გვარები, სათაურები, დაწესებულებებისა და პერიოდული გამოცემების დასახელებები და ა. შ.). ინფორმაციულ-საძიებო ენის სხვა მაგალითებია საძიებო Yandex ან Google სისტემებთან მიმართ-ვის ენები.

ინფორმაციული ენა ინფორმაციის დამამუშავებელ სხვადასხვა სისტემაში გამოიყენება. ინფორმაციული ენებისაგან უნდა განასხვავოთ **დაპროგრამების ენები, მანქანური ენები** და **მეცნიერების ფორმალიზებული ენები**. ჩვეულებრივად განასხვავებენ:

• **ინფორმაციულ-ლოგიკურ ენას,** რომელიც გამოიყენება ინფორმაციულ-ლოგიკური სისტემებისათვის. უპირველეს ყოვლისა, ასეთი ენებია ცოდნის წარმოდგენისათვის განკუთვნილი ენები (მაგალთად, SC, SCR, SCL) და მონაცემთა ბაზების ენები (მაგალითად, SQL);

• **ინფორმაციულ-საძიებო ენას**, რომელიც არის ნიშნური სისტემა, რომელიც გამოიყენება ტექსტების (დოკუმენტების) ან მათი ნაწილების ძირითადი აზრობრივი აღწერისათვის, აგრეთვე ინფორმაციული ძიების რეალიზაციის მიზნით ინფორმაციული შეკითხვების აზრობრივი შინაარსის გამოსახატავად. ინფორმაციულ-საძიებო ენის მაგალითია **ბიბლიოგრაფიული აღწერის ენა**, რომელიც განკუთვნილია ტექსტების იდენტიფიცირებისათვის და გამოიყენება ალფაბეტურ კატალოგებში, კართოტეკებსა და ბიბლიოგრაფიულ მაჩვენებლებში. მის შემადგენლობაში შედის - ბიბლიოგრაფიული ელემენტები (ავ-

ტორთა გვარები, სათაურები, დაწესებულებებისა და პერიოდული გამოცემების დასახელებები და ა. შ.). ინფორმაციულ-საძიებო ენის სხვა მაგალითებია საძიებო Yandex ან Google სისტემებთან მიმართვის ენები.

ლოგიკურ და საძიებო ენები ერთმანეთისაგან პრინციპულად არ განსხვავდება, რამდენადაც მრავალი ინფორმაციული ენა შეგვიძლია გამოვიყენოთ როგორც ერთ, ისევე მეორე სისტემაში. ნებისმიერი ინფორმაციული ენამ უნდა უზრუნველყოს ინფორმაციის ცალსახად ჩაწერა და შემდგომში გარკვეული სისრულითა და სიზუსტით მისი გამოცნობა. ინფორმაციულ-ლოგიკურმა ენამ აღნიშნული გარდა, უნდა უზრუნველყოს ლოგიკური დასკვნის ფორმალიზება.

მონაცემათა აღწერის ენები (Data Definition Language - DDL) ეწოდება მონაცემების ბაზათა სტრუქტურების აღწერისათვის გამოყენებული ენების ოჯახს. მიმდინარე მომენტში მონაცემათა აღწერის ყველაზე პოპულარული ენაა მონაცემების ბაზათა მართვის რელაციურ სისტემაში მონაცემების პოვნისა და მათზე მანიპულირებისათვის გამოყენებული SQL ენა.

მონაცემათა აღწერის ენათა ფუნქციები განისაზღვრება წინადადებაში არსებული პირველი სიტყვით, რომელსაც ხშირად **შეკითხვას** უწოდებენ და ყოველთვის ზმნას წარმოადგენს. SQL-ის შემთხვევაში ასეთი ზმნებია: „create” („შექმენი“), „alter” („შეცვალე“), “drop” (“გააძევე“). ეს შეკითხვები ანუ ბრძანებები ხშირად SQL-ის ბრძანებებთანაცაა შერეული, რომლის გამოც მონაცემათა აღწერის ენები განცალკევებულ კომპიუტერულ ენას არ წარმოადგენს.

მონაცემთა აღწერის ენების ოჯახში შემავალი ენებია სპეციფიკაციების ენა და სტილების კასკადური ცხრილები. მოკლედ განვსაზღვროთ ისინი.

სპეციფიკაციების ენა (Specification language) ფორმალური ენაა, რომელიც განკუთვნილია მონაცემთა სტრუქტურის, კავშირების, თვისებებისა და მათი გარდაქმნების დეკლარაციული აღწერისათვის; აქტიური ენებისაგან განსხვავებით, იგი ცხადად არ მოიხსენიებს, თუ რა თანამიმდევრობით უნდა შესრულდეს შესასრულებელი მოქმედებები და მონაცემთა რა კონკრეტული მნიშვნელობები უნდა იყოს გამოყენებული.

კონკრეტული პროგრამების შესრულების დროს გამოყენებული დაპროგრამების ენებისაგან განსხვავებით სპეციფიკაციების ენები გამოიყენება სისტემური ანალიზისა ჩატარებისა და მოთხოვნების გასაანალიზებლად, შესაქმნელი პროგრამული სისტემების არქიტექტურის დასამუშავებლად და პროგრამული უზრუნველყოვის ფორმალური ვერიფიცირებისათვის.

სტილების კასკადური ცხრილები (ინგ. hardware description language) არის მონიშვნების (უფრო ხშირად HTML ან XHTML) ენის გამოყენებით დაწერილი ფორმალური ენა, რომელიც განკუთვნილია დოკუმენტის (ვებ-გვერდის) გარეგანი სახის აღწერისათვის. იგი შეიძლება გამოვიყენოთ ნებისმიერი XML-დოკუმენტისთვისაც.

აპარატურის აღწერის ენა (hardware description language - **HDL**) არის ელექტრონული სქემების, უფრო ხშირად ციფრული ელექტრონული სქემების, სტრუქტურისა და ქცევის აღწერისათვის გამოყენებული სპეციალიზებული კომპიუტერული ენა.

აპარატურის აღმწერი ენები გარეგნულად წააგავს დაპროგრამების ისეთ ენებს, როგორებიცაა C ან Pascal. ამ ენებზე დაწერილი პროგრამებიც შედგება სტრუქტურების მმართველი გამოსახულებებისა და ოპერატორებისდაგან. დაპროგრამების ჩვეულებრივ ენასა და აპარატურის აღმწერ ენას შორის მნიშვნელოვანი განმასხვავებელი ნიშანია აპარატურის აღწერის ენაში დროის კონცეფციის ცხადი ჩართვა.

აპარატურის აღმწერი ენები **ავტომატიზებული დაპროექტების სისტემის** (Computer-aided design - **CAD**) განუყოფელი ნაწილია, განსაკუთერებით ისეთი რთული სქემების შემთხვევაში, როგორიცაა სპეციალიზებული ინტეგრალური სქემები, მიკროპროცესორები და დაპროგრამებადი ლოგიკური სქემები.

პრაქტიკულად ხშირად გამოყენებადი ამ ტიპის ენებია **Verilog** და **VHDL**. არსებობს აგრეთვე რამდენიმე ათეული ალტერნატიული ენებიც.

მონაცემების გადაცემის პროტოკოლი არის ლოგიკური დონის ინტერფეისის წესების ან შეთანხმებების გარკვეული ნაკრები, რომელიც განსაზღვრავს განსხვავებულ პროგრამებს შორის მონაცემების გაცვლას. ეს წესები დასახავს შეტყობინებების გადაცემის და შეცდომების დამუშავების გადაცემის ერთგვაროვან ხერხს.

სასიგნალო პროტოკოლი გამოიყენება შეერთებების მართვისათვის, მაგალითად, კავშირგაბმულობის დამყარებას, გადამისამართებას და გაწყვეტას. პროტოკოლების მაგალითებია **RTSP**, **SIP**. მონაცემების გადასაცემად გამოიყენება **RTP**-ს მსგავსი პროტოკოლები.

ქსელური პროტოკოლი არის იმ წესებისა და მოქმედებების (მოქმედებათა თანამიმდევრობის) ნაკრები, რომელიც საშუალებას გვაძლევს ერთმანეთთან შევაერთოთ ქსელში ჩართული ორი და მეტი რაოდენობის მოწყობილობები და მოვახდინოთ მათ შორის მონაცემების გაცვლა.

სხვადასხვა პროტოკოლები ხშირად კონკრეტული ტიპის კავშირის მხოლოდ სხვადასხვა მხარეს აღწერს.

ინტერნეტისათვის განსაზღვრავს:

- ახალ პროტოკოლებს - **ინტერნეტის საინჟინრო საბჭო** (Internet Engineering Task Force, **IETF**);

- დანარჩენ პროტოკოლებს - **ელექტროტექნიკისა და ელექტრონიკის ინჟინერთა ინსტიტუტი** (Institute of Electrical and Electronics Engineers - **IEEE**) ან **სტანდარტიზაციის საერთაშორისო ორგანიზაცია** (International Organization for Standardization, **ISO**);

- ტელესაკომუნიკაციო პროტოკოლებსა და ფორმატებს - **ტელეფონიასა და ტელეგრაფიაში საერთაშორისო საკონსულტაციო კომიტეტი** (International Telecommunication Union - Telecommunication sector - ITU-T).

ქსელური პროტოკოლების კლასიფიკაციის ყველაზე გავრცელებული სისტემაა **ღია სისტემების ურთიერთზემოქმედების მოდელი** (The Open Systems Interconnection model - **OSI**), რომლის შესაბამისად პროტოკოლები თავისი დანიშნულების მიხედვით ფიზიკური (ელექტროტექნიკური ან სხვა სიგნალების ფორმირებისა და გამორკვევის) დონიდან დაწყებული გამოყენებითი (დანართებით ინფორმაციის გადაცემისათვის საჭირო პროგრამული დანართების ინტერფეისის) დონით დამთავრებული, იყოფა 7 დონედ.

ქსელური პროტოკოლები მუშაობის წესებს განუსაზღვრავს ქსელზე მიერთებულ კომპიუტერებს. გარკვეული დონის პროტოკოლი განსაზღვრავს კავშირგაბმულობის ერთ-ერთ რომელიმე ტექნიკურ წესს. დღეისათვის ქსელური პროტოკოლებისათვის გამოიყენება **ღია სისტემების ურთიერთზემოქმედების მოდელი** (**OSI**).

2.2. დაპროგრამების ენათა პარადიგმები

2.2.1. დაპროგრამების პარადიგმის არსი

დაპროგრამის პარადიგმა არის პროგრამის კონსტრუირების განმსაზღვრელი პრინციპების, მეთოდებისა და ცნებების ერთობლიობა.

თომას კუნის მიხედვით **მეცნიერული ფილოსოფიის პარადიგმა** არის მდგრადი მეცნიერული შეხედულებების სისტემა, რომლის ჩარჩოებში წარმოებს კვლევები.

პარადიგმა განისაზღვრება:

- გამოთვლითი მოდელით;
- ბაზისური პროგრამული ერთეულებით;
- აბსტრაქციის დაყოფის მეთოდებით.

დაპროგრამების ენა არაა აუცილებელი იყენებდეს მხოლოდ ერთ პარადიგმას. რამდენიმე პარადიგმის მეშვეობით ფორმირებულ ენებს **მულტიპარადიგმული ენები** ეწოდება. ასეთი ენების შემქმნელები იზიარებენ თვალთახედვას, რომელის მიხედვითაც არცერთი პარადიგმა არ შეიძლება ერთნაირად ეფგექტური იყოს ყველა ამოცანისათვის, რის გამოც დამპროგრამებელს ცალკე აღებული თითოეული ამოცანისათვის დაპროგრამების უკეთესი სტილის ამორჩევის უფლება უნდა ჰქონდეს

2.2.2. ტერმინის წარმოშობის ისტორიიდან

სამეცნიერი-ტექნიკურ სფეროში ტერმინ პარადიგმისათვის თანამედროვე მნიშვნელობის მინიჭებაში, როგორც ლიტერატურიდანაა ცნობილი, გადამწყვეტი როლი ითამაშა **თომას კუნმ**ა და მისმა წიგნმა **„სამეცნიერო რევოლუციების სტრუქტურა**“. **კუნი** პარადიგმებს უწოდებდა სამეცნიერო შეხედულებების დამკვიდრებულ შეხედულებებს, რომლის ჩარჩოებში წარმოებს კვლევები. **კუნის** თანახმად სამეცნიერო დისციპლინის განვითარების პროცესში ერთი პარადიგმა შე-

იძლება შეიცვალოს სხვა პარადიგმით. ამის შემდეგ ძველი პარადიგმამ გარკვეული დროის განმავლობაში შეიძლება მაინც განაგრძოს არსებობა. უფრო მეტიც, იმ მიზეზით, რომ მისი მრავალი მხარდაჭერი ამა თუ იმ მიზეზით ახალ პარადიგმის ჩარჩოებში მუშაობისათვის გადაწყობისათვის უუნარონი აღმოჩდებიან, ძველმა პარადიგამამ შეიძლება განაგრძოს განვითარება.

ტერმინი „**დაპროგრამების პარადიგმა**“ ტიურინგის პრემიის მინიჭებისას წაკითხულ ლექციაში პირველად ახსენა **რობერტ ფლოიდმა** (Robert W Floyd, 1936 – 2001). იგი აღნიშნავდა, რომ დაპროგრამებაშიც შეგვიძლია შევნიშნოთ **კუნის პარადიგმის** მსგავსი მოვლენა. მაგრამ **მისგან** განსხვავებით, **დაპროგრამების პარადიგმები ერთმანეთს კი არ გამორიცხავს**, არამედ ავსებს ერთმანეთს. დაპროგრამების ხელოვნების პროგრესი ზოგადად პარადიგმების მუდმივ გამოგონებასა და სრულყოფას მოითხოვს. ამიტომ თავისი ხელოვნების სრულყოფისათვის დამპროგრამებელი პერმანენტულად უნდა აფართოვებდეს პარადიგმების საკუთარ რეპერტუარს.

ამგვარად, სამეცნიერო სამყაროში არსებული პარადიგმებისაგან განსხვავებით, რომელიც **კუნმა** აღწერა, **რობერტ ფლოიდის** აზრით, დაპროგრამების პარადიგმებმა შეიძლება ერთმანეთთან შერწყმის მეოხებით გაამდიდროს დამპროგრამებლის ინსტრუმენტარიუმი.

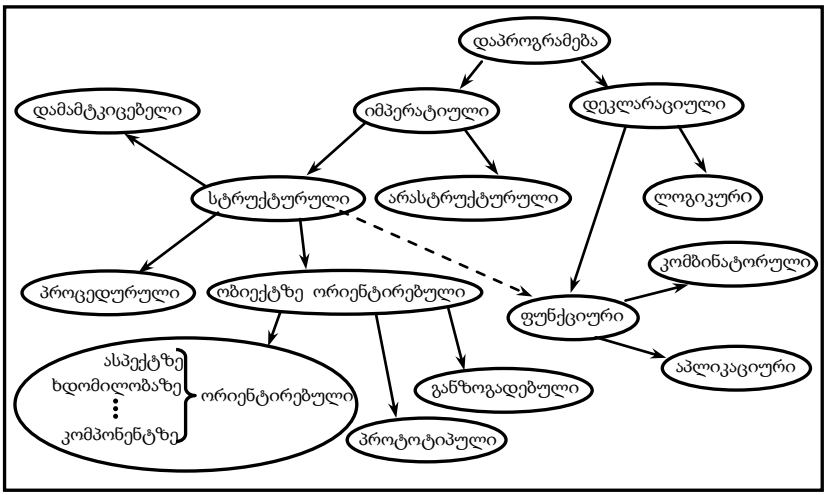
მნიშვნელოვანია აღინიშნოს, რომ დაპროგრამების პარადიგმა დაპროგრამების ენით ცალსხად არ განისაზღვრება. დაპროგრამების პრაქტიკულად ყველა ენაში ამა თუ იმ ზომითაა გამოყენებული სხვადასხვა პარადიგმა. მაგალითად, დაპროგრამის ბევრისთვის ნაცნობ **C** ენას, რომელიც არაა ობიექტზე ორიენტირებული ენა, შეუძლია იმუშაოს **ობიექტზე ორიენტირებული დაპროგრამების** პრინციპების შესაბამისად, ოღონდ ეს მთელ რიგ სიძნელებთანაა დაკავშირებული. **ფუნქციური დაპროგრამება** შეგვიძლია გამოვიყენოთ ფუნქციების შემცველ ნებისმიერ იმპერატიულ ენაზე მუშაობის დროს (ამისათვის საკმარისია არ გამოვიყენოთ მინიჭების ოპერაცია) და ა. შ.

2.2.3. დაპროგრამების პარადიგმათა გალერეა

არსებულ ლიტერატურაში დაპროგრამების **პარადიგმის** საერთოდ მიღებული განმარტების ნაცვლად გვხვდება მისი სხვადასვა ინ-

ტერპრეტაცია. მაგალითად, სხვადასხვა ავტორი ამტკიცებს, რომ **პარადიგმა** არის:

- მიდგომა პროგრამების შექმნისადმი;
- პროგრამის დაწერის სტილი;
- დამპროგრამებლის განზრახულობათა აღწერა;
- აზროვნება პროგრამის შესახებ;
- გამოთვლითი მეთოდი;
- პროგრამის შესრულების დროს კომპიუტერის მიერ გამოთვლების ორგანიზების განმსაზღვრელი და ა. შ.



ნახ. 1. *დაპროგრამების პარადიგმები*

ჩვენ დაპროგრამების პარადიგმად მივიჩნევთ პროგრამის დაწერის სტილს, რომელიც ალგორითმის ფორმალიზების ხერხს გვთავაზობს.

თავიდანვე უნდა შევთანხმდეთ, რომ დაპროგრამების პარადიგმასა და დაპროგრამების ენას შორის არ არსებობს ურთიერთცალსახა დამოკიდებულება: კონკრეტული პარადიგმა შეიძლება განივთებული იყოს თვისობრივად განსხვავებულ დაპროგრამების რამდენიმე ენაში ან დაპროგრამების მოცემულ ენაში გამოყენებული იყოს სხვადასხვა პარადიგმა, ე. ი. იგი მისი დამუშავების პროცესში გამოყენებული იყოს სხვადასხვა სტილი.

ნახაზ 1-ზე მოყვანილია დაპროგრამების დღეისათვის არსებული სხვადასხვა პარადიგმის, ანუ სტილების იერარქიული კლასიფიკაცია. ნახაზიდან ჩანს, რომ არსებობს დაპროგრამების იმპერატიული, დეკლარაციული, სტრუქტურული და ა. შ. პარადიგმები (ანუ, სტილები). მოკლედ დავახასიათოთ ისინი.

იმპერატიული დაპროგრამება არის დაპროგრამების პარადიგმა რომლისთვისაც დამახასიათებელია შემდეგი:

- პროგრამის საწყის კოდში ჩაიწერება ინსტრუქციები (ბრძანებები);
- ინსტრუქციები უნდა შესრულდეს მიმდევრობით;
- წინა ინსტრუქციების შესრულების შედეგად მიღებული მონაცემები მომდევნო ინსტრუქციებს შეუძლია წაიკითხოს მეხსიერებაში;
- ინსტრუქციების შესრულების შედეგად მიღებული მონაცემები შეიძლება ჩაიწეროს მეხსიერებაში.

იმპერატიული პროგრამა ბუნებრივი (სალაპარაკო) ენის ბრძანებითი კილოთი გამოთქმულ ბრძანებებს ჰგავს (ინგ. **imperative** - „ბრძანება“, „ბრძნებითი ბრუნვა“): იგი პროცესორის მიერ შესასრულებელი ბრძანებების მიმდევრობაა.

იმპერატიული მიდგომის დროს კოდის შესადგენად (დეკლარაციული პარადიგმის კუთვნილი ფუნქციური მუდგომისაგან განსხვავებით) ფართოდ გამოიყენება **მინიჭების ოპერაცია**. მინიჭების ოპერატორების არსებობა ზრდის გამოთვლების მოდელის სირთულეს და იმპერატიულ პროგრამებს ისეთი შეცდომებისადმი მოწყვლადს ხდის, რომლებიც არ წარმოიშობა ფუნქციური მიდგომის დროს.

იმპერატიული ენებისათვის დამახასიათებელი ძირითადი ნიშნებია:

- სახელდებული ცვლადების გამოყენება;
- მინიჭების ოპერატორის გამოყენება;
- შედგენილი გამოსახულებების გამოყენება;
- ქვეპროგრამების გამოყენება და ა.შ.

კოდის შედგენისადმი **იმპერატიული მიდგომის** დროს **(ნახ. 1**-ზე მითითებული **ფუნქციური დაპროგრამებისაგან** განსხვავებით**)** ფართოდ გამოიყენება ე. წ. **მინიჭების ოპერატორები**; მათი არსებობა ართულებს გამოთვლების მოდელის სირთულეს და ამის გამო **იმპერატიული პროგრამები** მიდრეკილი ხდება ისეთი სპეციფიკური შეცდო-

მებისაკენ, რომლებიც არ წარმოიშობა **ფუნქციური პროგრამების** დროს.

არასტრუქტურული დაპროგრამება არის პარადიგმა, რომელშიც დაშვებულია უპირობო გადასვლის (უმეტეს შემთხვევაში - **GO-TO**) ბრძანების ცხადი სახით გამოყენება. არასტრუქტურირებული ენების ტიპური წარმომდგენლებია **ბეისიკისა** და **ფორტრანის ადრეული ვერსიები**. მოცემული სტილის წარმოშობა განაპირობა კოდების გამოყენებით შედგენილი **მანქანური პროგრამის** შესრულების თავისებურებების გათვალისწინებამ და მემკვიდრეობითაა მიღებული ასემბლერის ენაზე დაწერილი პროგრამებიდან, რადგან მათში გადასვლის ბრძანების არსებობა აუცილებელია. სამწუხაროდ, ასეთი დაპროგრამების მაღალი დონის ენებში ასეთი მიდგომის გამოყენება განაპირობებს მრავალ სერიოზული ნაკლოვანებებს: ზემოთ და ქვემოთ უამრავი გადასვლების არსებობის გამო პროგრამა „სპაგეტივით“ იხლართება; ამის გამო რთულდება როგორც მათი მოდიფიცირება, ასევე მათთვის თანხლები დოკუმენტაციის მომზადება. არასტრუქტურული მიდგომა ფაქტურად არ გვაძლევს დიდი პროგრამების დამუშავების საშუალებას. ადრე დაპროგრამების სწავლებაში საბაზისოდ არასტრუქტურირებული (ძირითადად, ბეისიკის) ენის გამოყენება შემდეგ ძალიან ართულებდა დაპროგრამების უფრო თანამედროვე სტილზე გადასვლას.

ცნობილი ჰოლანდიელი მეცნიერი **ედსგერ დეიკსტრა** (*Edsger Wybe Dijkstra. 1930 – 2002*) აღნიშნავდა, რომ „სტუდენტებს, რომლებმაც ადრე შეისწავლეს ბეისიკი, პრაქტიკულად შეუძლებელია ასწავლოთ კარგი დაპროგრამება. როგორც პოტენციურმა დამპროგრამებლებმა, შეუქცევადი გონებრივი დეგრადაცია განიცადეს.“

არასტრუქტურირებული დაპროგრამებისათვის დამახასიათებელია შემდეგი თვისებები:

- სტრიქონები, როგორც წესი, ინომრება;
- პროგრამის თითოეული სტრიქონიდან დასაშვებია ნებისმიერ სტრიქონზე გადასვლა.
- მეტად რთულია პროგრამაში **რეკურსიის** რეალიზაცია.

სტრუქტურული დაპროგრამება არის დაპროგრამების პარადიგმა, რომელსაც საფუძვლად უდევს პროგრამის წარმოდგენა ბლოკების იერარქიული სტრუქტურიის სახით. იგი კონცეპტუალიზებული იქნა **1960**-იანი წლების ბოლოსა და **1970**-იანი წლების დასაწყისში პროგრამების სტრუქტურულად ორგანიზების შესაძლებლობის დამასაბუთებელ **ბიომი-იაკოპინის** თეორემისა და **ედსგერ დეიკსტრის** ნაშრომის „**ოპერატორი goto დამაზიანებელია**“ (**Goto considered harmful**) საფუძველზე. აღნიშნული თეორემის თანახმად ნებისმიერ ალგორითმს შეიძლება მიეცეს სამი სტრუქტურული ბლოკის შემცველი სტრუქტურის სახე, რომელთაგანაც ერთ-ერთი ახდენების მიდევნების (**sequence**), მეორე - განშტოების (**selectio**), ხოლო მესამე გამეორებების, ანუ ციკლის (**iteration**) პროცესის ინიცირებას.

მოცემული პარადიგმის შესაბამისად ოპერატორ **goto**-ს გამოუყენებლად აგებული ნებისმიერი პროგრამა შედგება სამი საბაზისო კონსტრუქციისაგან: მიდევნებისაგან, განშტოებისა და ციკლისაგან. გარდა ამისა, მასში გამოიყენება ქვეპროგრამები. ამასთანავე ბიჯურად აიგება „ზემოდან ქვემოთ“ მეთოდის გამოყენებით.

სტრუქტურული დაპროგრამების მეთოდოლოგია გამოჩნდა, როგორც კომპიუერებზე გადასაწყვეტი ამოცანების სირთულის ზრდისა და შესაბამისად პროგრამული უზრუნველყოფის გართულების საპასუხოდ. **1970**-იან წლებში პროგრამათა მოცულობებმა და სირთულემ ისეთ დონეს მიაღწია, რომ პროგრამების ტრადიციულმა (არასტრუქტურირებულმა) დამუშავებამ უკვე ვერ დააკმაყოფილა პრაქტიკის მოთხოვნები. ძალიან რთული გახდა პროგრამების ნორმალური თანმხლები დოკუმენტების შედგენა. აღნიშნულის გამო დღის წესრიგში დადგა პროგრამების დამუშავების პროცესისა და სტრუქტურის სისტემატიზება.

სტრუქტურული დაპროგრამების რეალიზება მას შემდეგ გახდა შესაძლებელი, როდესაც წინა საუკუნის **60**-იანი წლების დასასრულსა და **70**-იანი წლების დასაწყისში დამუშავებული იქნა **სტრუქტურული სტილი**, რომელსაც საფუძვლად დაედო შემდეგი ორი იდეა:

- ამოცანა დაიყოფა დიდი რაოდენობის მცირე ქვეამოცანებად, რომელთაგანაც თითოეული მათგანის გადაწყვეტა საკუთარი პროცედურის ან ფუნქციის საშუალებით ხდება (ამას ეწოდება **დეკომპოზიცია**). ამ დროს პროგრამა პროექტდება **ზემოდან ქვემოთ**: თავდა-

პირველად განისაზღვრება პროგრამის გადაწყვეტისათვის აუცილებელი მოდულები, მათი შესასვლელები და გამოსასვლელები; მხოლოდ ამის შემდეგ იწყება ამ მოდულების დამუშავება. ასეთი მიდგომა ლოკალურ სახელებან ცვლადებთან ერთად საშუალებას გვაძლევს პროექტი გადავწყვიტოთ დიდი რაოდენობის დამპროგრამებლების ძალების ჩართულობით.

- როგორც დაამტკიცა **ე. დეიკსტრმა**, ნებისმიერი ალგორითმი შეიზლება რეალიზდეს მხოლოდ სამი მმართველი კონსტრუქციის - მიმდევრობითი შესრულების (მიდევნების), განშტოებისა და ციკლის - გამოყენებით. ეს სათანადო ოპერატორების არსებობის დროს გარემოება საშუალებას იძლევა, ენიდან გამოვრიცხოთ გადასვლის ბრძანება.

პროგრამული უზრუნველყოფის სტრუქტურული დამუშავების მეთოდოლოგია მიჩნეული იქნა **„70-იანი წლების უძლიერეს ფორმალიზაციად**“. ცნობილი ფრანგი ინფორმატიკოსის **ბერტრან მაიერის** (*Bertrand Meyer, 1950*) აზრით, „დაპროგრამებაზე **დეიკსტრის** მიერ დაწყებულმა რევოლუციურმა შეხედულებებმა მიგვიყვანა სტრუქტურულ დაპროგრამებად წოდებულ მოძრაობამდე, რომელმაც შემოგვთავაზა პროგრამების კონსტრუირების სისტემატური, რაციონალური მიდგომა. სტრუქტურული დაპროგრამება გახდა ყოველივე იმის საფუძველი, რაც მოხდა დაპროგრამების მეთოდოლოგიაში ობიექტზე ორიენტირებული დაპროგრამების ჩათვლით“.

სტრუქტურული დაპროგრამების პრინციპები პირველად რეალიზებული იქნა დაპროგრამების ენა **ალგოლში**, მაგრამ ყველაზე დიდი პოპულარობა მოიხვეჭა შვეიცარიელი მეცნიერის **ნიკოლაუს ვირტის** (*Niklaus Emil Wirth. 1934*) მიერ შექმნილმა **პასკალმა**. იგი შეიძლება ჩავთვალოთ **სანიმუშო ენად**, რომელიც დღემდე ინარჩუნებს პოპულარობას (მაგალითად, ფირმა **Imprise**-ის მიერ დამუშავებული ერთ-ერთი ვერსიის **Delphi**-ის სახით).

დამტკიცებითი დაპროგრამების მათემატიკური საფუძველია **სტრუქტურირებული ალგორითმების** მათემატიკური სემანტიკა, რომელიც ფორმულირებულია **ფლოიდის, დეიკსტრას, ერშოვისა**, **კაიმინის** შრომებში, აგრეთვე **კაიმინის** ავტორობით გამოცემულ საბაზისო სახელმძღვანელოებში. ამ უკანასკნელებში აღწერილია ანალი-

ზის მეთოდები, რომლებიც გამოიყენება **ბეისიკის, პასკალის, C**-ს, **JavaScript**-ისა და სხვა ენებზე შედგენილი **სტრუქტურირებული ალგორითმებისა და პროგრამების** სისწორის შესამოწმებლად.

ინფორმატიკის საბაზისო სახელმძღვანელოებში სტრუქტურირებული ალგორითმების სემანტიკა ახსნილია ელემენტალური მათემატიკის ენასა და იმ მათემატიკის ენაზე, რომელიც შეისწავლება უმაღლესი სასწავლებლების ტექნიკურ, მათემატიკურ და ეკონომიკურ ფაკულტეტებზე.

ალგორითმებსა და პროგრამევში შეცდომების არარსებობის შესახებ დასკვნის გამოტანისათვის საკმარისია არსებობდეს ტესტი, რომლითაც განსახილველ მასალაში ამოვარდნის ან მტყუნებების არსებობის შემთხვევაში მივიღებთ არასწორ შედეგს.

პროგრამებში შეცდომების მოძებნისა და გასწორების ოპერაციებს ასრულებს კომპიუტერი. პროგრამების სისწორის დასადასტურებლად საჭიროა ნაჩვენები იყოს, რომ მონაცემების ყველა დასაშვები მნიშვნელობების დროს მიღებული იქნება სწორი შედეგი. ასეთი მტკიცებულება მონაცემების ნებისმიერი დასაშვები მნიშვნელობის დროს ფორმულირებული შედეგების ამომწურავი ანალიზის მეშვეობით მიიღება.

პროგრამების შემოწმობისადმი ორი, კერძოდ, პრაგმატული და მტკიცებითი, მიდგომა არსებობს.

პრაგმაგატული მიდგომის დროს კომპიუტერზე პროგრამები ტესტირების გზით მოწმდება.

ტესტირება არის ტესტების გარკვეული ნაკრების დახმარებით კომპიუტერზე პროგრამების შემოწმება. ნათელია, რომ ტესტირება საშუალებას არ გვაძლევს პროგრამები შევასრულოთ მონაცემების ყველა ნაკრებისათვის აღნიშნულიდან გამომდინარე ზოგადად **ტესტირებას არ შეუძლია და ვერც გვაძლევს პროგრამებში შეცდომების არარსებობის სრულ გარანტიას.**

პროგრამების გამართვა არის შეცდომების ძებნისა და მისი გასწორების პროცესების ორგანიზება კომპიუტერზე. ვინაიდან პროგრამების გამართვის დროს შეცდომები იძებნება ტესტების დახმარებით, ამიტომ **გამართვასაც არ შეუძლია** მოგვცეს პროგრამაში ყველა შესაძლო შეცდომის აღმოჩენისა და გასწორების გარანტია და ვერც გვაძლევს მას.

ამავე მიზეზით არაა ნათელი, კომპიუტერზე შეცდომების ძიებისა და გასწორების პროცესი თუ როდის შეიძლება მივიჩნიოთ დამთავრებულად. ხოლო იმის თქმა, რომ გამომჟღავნებული და გასწორებულია ყველა შეცდომა, არავის არ შეუძლია.

ამგვარად, **პრაგმანტული მიდგომის** დროს შეიძლება მივიღოთ ისეთი პროგრამები, რომლებშიც შეცდომები იარსებებს გამართვის პროცესის „დამთავრების“ შემდეგაც, ამიტომ ესტაფეტა უნდა გადაეცეს **მტკიცებით მიდგომას**, რომლის რეალიზება ხდება **დამტკიცებითი დაპროგრამების** დროს.

პროცედურული (ალგორითმული) დაპროგრამება არის კომპიუტერის **ფონ ნეიმანისეული** არქიტექტურის ასახვა. **პროცედურულ ენაზე დაწერილი პროგრამა წარმოადგენს ამოცანის გადაწყვეტის ალგორითმში შემავალი ბრძანებების მიმდევრობას.** პროცედურული ენის ძირითადი იდეს თანახმად მეხსიერება გამოიყენება მონაცემების შესანახად. მისი ძირითადი ოპერატორია (ბრძანებაა) მინიჭების ოპერატორი. პროგრამა გარდაქმნის მეხსიერების შიგთავსს და ამის შედეგად საწყისი ანუ შესასვლელი მდგომარეობიდან დასკვითი მდგომარეობის ანუ შედეგის მიღებას განაპირობებს.

პროცედურული (**ალგორითმული**) **დაპროგრამება** ფაქტურად არის იმპერატიულ ენაზე დაპროგრამება, რომელშიც მიმდევრობითად შესრულებადი ოპერატორები თავად ენის მექანიზმების მეშვეობით შეიძლება კოდის უფრო დიდ მთლიანურ ერთეულად - **ქვეპროგრამად** გაერთიანდეს.

პროცედურული დაპროგრამება წარმოადგენს **1940**-იან წლებში **ფონ ნეიმანის** მიერ შემოთავაზებული ტრადიციული კომპიუტერების არქიტექტურაა ასახული მის თეორიულ მოდელს წარმოადგენს **ტიურინგის მანქანად** წოდებული აბსტრაქტული მანქანა.

დაპროგრამების პროცედურული ენა დამპროგრამებელს შესაძლებლობას აძლევს ამოცანის გადაწყვეტის პროცესი განსაზღვროს მის მიერ გადასადგმელი თითოეული ნაბიჯი.

დაპროგრამების ასეთი ენების თავისებურებაა ის, რომ ამოცანა დაიყოფა ბიჯებად და მისი გადაწყვეტა ნაბიჯ-ნაბიჯ ხდება. იყენებს რა პროცედურულ ენას, ალგორითმული ნაბიჯების მიმდევრობითად

შესასრულებლად დამპროგრამებელი თავად განსაზღვრავს ენისეულ კონსტრუქციებს.

პროცედურული ენების მაგალითებია: **Algol 60**, **Algol 68**, **Basic** (Visual Basic-ის გამოჩენამდე), **Kobol**, **Fortran**, **Modila-2**, **Pascal**, **Ada** და სხვები.

ობიექტზე ორიენტირებული დაპროგრამება. კომპიუტერზე რთული ამოცანების გადასაწყვეტად გარდა მისი გამოთვლითი სიმძლავრის გაზრდისა საჭიროა ეფექტურად იყოს დაწერილი რთული პროგრამები. ამ მეორე მოთხოვნის შესასრულებლად ნორვეგიელმა მეცნიერებმა, ტიურინგის პრემიისა ლაურეატებმა **კრისტან ნიგაარდმა** (*ნორ. Kristen Nygaard, 1926-2001*), და **ოლეიოჰან დალიმ** (ნორ. *OlepJohan Dahl, 1931-2002*) დაამუშავეს სპეციალური სახის დაპროგრამების საფუძვლები და მისი გამოყენებით შექმნეს დაპროგრამების ახალი ენა - **სიმულა**. შემოთავაზებულ დაპროგრამებას მოგვიანებით ამერიკელმა მეცნიერმა, ასავე ტიურინგის პრემიის ლაურეატმა, **ალან კეიმ** (*იწგ. **Alan Kay**. 1940*) **ობიექტზე ორიენტირებული დაპროგრამა** უწოდა. ასეთი სახელწოდება იმან განაპირობა, რომ ამ დაპროგრამების ძირითადი იდეა რეალური სამყაროს ობიექტების ქცევებისა და ურთიერთმოქმედებების იმიტირების უნარანი პროგრამული სტრუქტურების შექმნა იყო (ე. ი. პროგრამაში უნდა შექმნილიყო **რეალური ობოექტების ვირტუალური ანალოგები**). დაპროგრამების ენა **სიმულა** გვთავაზობდა ასეთი მიდგომის მოხერხებულად და სწრაფად ხორცის შესხმისათვის საჭირო საშუალებებს. იგი კარგი რეპუტაციით სარგებლობდა აკადემიურ წრეებში, მაგრამ მთელი რიგი მიზეზების გამო კომერციული პროგრამული უზრუნველყოფის დამმუშავებლების სიმპათიები ვერ დაიმსახურა. ამის მიუხედავად ობიექტზე ორიენტირებული დაპროგრამების ძირითადი იდეა და შესაძლებლობები ძალიან ხიბლავდა დამპროგრამებლებს ძირითად ბირთვს. ამის გამო მოგვიანებით დამუავებული იქნა ობიექტზე ორიენტირებული დაპროგრამების ისეთი ენები, როგორიცაა **SmallTalk** (1980), **C++** (1983), **Eifel** (1986), **Object Paskal** (1986), **Oberron-2** (1991), **Java** (1991), **Visual Basic** (1991) და **Delphi** (1995). ზოგიერთი მათგანი დაპროგრამებაში სამრეწ-ველო სტანდარტებად გადაიქცა, რაც გვაიძულებს მეტ-ნაკლებად დაწ-

ვრილებით გაგაცნოთ ობიექტზე ორიენტირებული დაპროგრამების საკვანძო დებულებები.

მარტივი ცნებების მეშვეობით რთული მოვლენების გააზრებისა და გამოსახატავად ყოველდღიურ ცხოვრებაში ჩვენ ხშირად ვიყენებთ „ეკონომიური“ აზროვნების ისეთ ტიპურ ხერხებს, როგორიცაა **აბსტაჰირება**, **განზოგადება** და **კლასიფიცირება.** კერძოდ, **აბსტრაჰირებას** ვიყენებთ არაარსებითი დეტალების უკუგდებისათვის, **განზოგადებას** - ზოგადი არსებითი ნიშნების გამოსაყოფად, ხოლო ხოლო **კლასიფიცირებას** - მოვლენებს შორის არსებული კავშირების გაცნობიერებისა და მათ შორის არსებული მსგავსების ხარისხის დასადგენად. მათი გამოყენება გვეხმარება თავი გავართვათ განსახილველი მოვლენების სირთულეებს,

ანალოგურად, დაპროგრამების საკუთარი სირთულეების დასაძლევად ობიექტზე ორიენტირებული დაპროგრამების ენებმაც შემოგვთავაზეს ისეთი სპეციფიკური საშუალებები, როგორიცაა **ობიექტები და კლასები**, **ინკაფსულირება, მემკვიდრეობა და პოლიმორფიზმი.**

ობიექტები და კლასები არის მონაცემებისა და მათი დამუშავების ალგორითმების გამაერთიანებელი სპეციალური სტრუქტურები, **ინკაფსულირება** მალავს ობიექტების ფუნქციონირების წვრილმანებს, ხოლო **მემკვიდრეობა** ახალი კლასების შექმნის „შემოკლების“ ხერხია, ხოლო **იზომორფიზმი** ერთი ფუნქციის რამდენიმე რეალიზაციის გამოყენების საშუალებას გვაძლევს. მოკლედ განვიხილოთ ისინი. წინასწარ შევნიშნავთ, რომ ობიექტზე ორიენტირებულ დაპროგრამემებაში არაა დამკვიდრებული ცალსახად განსაზღვრული ტერმინოლოგია და მის სხვადასხვა ენაში ერთი და იმავე ცნების აღნიშვნისათვის სხვადასხვა ტერმინია გამოყენებული.

1) ობიექტები ეწოდება მონაცემებისა და ამ მონაცემების დამუშავებისათვის გამოყენებული ალგორითმების შემდგარ განსაკუთრებულ პროგრამულ ერთეულებს. ობიექტში შემავალ **მონაცემებს** უწოდებენ თვისებებს (ატრიბუტებს, ველებს ან წევრებს), ხოლო **ალგორთმებს** - მეთოდებს (სერვისებს, ოპერაციებს ან ფუნქცია-წევრებს).

2) კლასები მონაცემების ობიექტური ტიპებია. იმის მსგავსად, რომ მთელი რიცხვები შეიძლება რომელიმე მთელრიცხვული (მაგ. **intege** ან **byte**) **ტიპის** იყოს, ასევე ობიექტებიც შეიძლება რომელიმე

ტიპის იყოს, ოღონდ აქ ტერმინ „ტიპის“ ნაცვლად აქ გამოიყენება ტერმინი „კლასი“.

ზოგიერთ ენებში (**C++, Java**) ობიექტებს **კლასის ეგზემპლარებს** (instances-ს) უწოდებენ.

ტერმინების „კლასებისა“ და „ობიექტების“ შემოტანა საშუალებას გვაძლევს მონაცემების დამუშავების დროს მონაცემებსა და ფუნქციებს შორის ლოგიკური (აზრობრივი) შესაბამისობის (მონაცემების გამოყენების სისწორის) შემოწმება ტრივიალურ ამოცანად გარდავქმნათ და იგი კომპიუტერს (კერძოდ, კომპილატორს) მივანდოთ.

• **ინკაფსულირება** („კაფსულში ჩასმა“, დამალვა, დაფარვა) - კლასის შინაგანი სტრუქტურის შესახებ ინფორმაციის კონტროლირებად დამალვას ნიშნავს. კლასში შეიძლება ობიექტების მიერ მხოლოდ საკუთარი სამუშაოს შესასრულებლად გამოყენებული ველები და მეთოდები (მაგალითად, ბუფერი დინამიკურ მეხსიერებაში, მუშა მონაცემებიან ფაილი, ამ ფაილთან მუშაობისათვის საჭირეო მეთოდები და ა. შ.). სახიფათოა ასეთი ველის შეცვლა, ან გარედან მეთოდების გამოძახება - ამან შეიძლება დაარღვიოს მისი მუშა მდგომარეობა. ობიექტების უსაფრთხოების უზრუნველყოფისათვის მსგავსი ველები და მეთოდები საჭიროა დაიმალოს - აიკრძალოს გარედან მათთვის მიმართვა.

„სირთულესთან ბრძოლის“ პოზიციიდან ინკაფსულირება საშუალებას გვაძლევს ობიექტებთან მუშაობის სისწორის კონტროლის ნაწილი კომპილატორს (კომპიუტერს) გადავცეთ.

ობიექტზე ორიენტირებული დაპროგრამების სხვადასხვა ენა ველებისა და მეთოდების ინკაფსულირების სხვადასხვა (სრული არარსებობიდან დაწყებული - ყველა ველის ავტომატურ დამალვამდე დამთავრებული) შესაძლებლობას გვთავაზობს. მაგალითად, ობიექტზე ორიენტირებული დაპროგრამების ისეთ ენებში, როგორიცაა **c++, Java**, **Delfi**, **Eifel** და ა. შ. გათვალისწინებულია ველებისა და მეთოდების ინკაფსულირების **public** (საჯარო), **protected** (დაცული) **და private** (პრივატული, კონფიდენციალური) დონეები.

საჯარო (public) დონის დროს ობიექტების საჯარო ველებსა და ობიექტებთან მიმართვისათვის არავითარი შეზღუდვა არ არსებობს. **დაცული (protected) დონის დროს** დაცულ ველებსა და მეთოდებთან პირდაპირი მიმართვა შეიძლება მხოლოდ მოცემული კლასის მეთოდებიდან და შვილობილი კლასებიდან. **პრივატული, კონფიდენცი-**

ური (private) დონის დროს პრივატულ ველებსა და მეთოდებთან პირდაპირი მიმართვა შეიძლება მხოლოდ და მხოლოდ მოცემული კლასის მეთოდებიდან.

3) მემკვიდრეობა ობიექტზე ორიენტირებული დაპროგრამების ცენტრალური ცნებაა. იგი არსებული კლასებიდან ახალი კლასების მიღების საშუალებას გვაძლევს. ამ დროს საჭიროა ჩამოვწეროთ არსებულ და ახალ კლასებს შორის არსებული განსხვავებები და არ აღვწეროთ თანხვდომი ელემენტები. ახალ კლასს ეწოდება **შთამომავალი** (ნაწარმოები, შვილობილი) **კლასი** ან **ქვეკლასი**, ხოლო საწყის კლასს - **წინაპარი** (ბაზისური, მშობლისეული) **კლასი** ან **სუპერკლასი**.

განმეორებადი აღწერების არარსებობა ამცირებს პროგრამის ზომას. წინაპარ კლასში გაცხადებული ყველა ველი და მეთოდი ავტომატურად გადადის შთამომავალ კლასში და მათ **მემკვიდრეობითი** (inherited) **ელემენტები** ეწოდება.

აუცილებლობის შემთხვევაში შესაძლებელია ნებისმიერი მშობლისეული მეთოდი განისაზღვროს, ე. ი. შთამომავალი კლასის იმავე სახელწოდების მეთოდის გამოძახების დროს სხვა ალგორითმი შესრულდეს.

ობიექტზე ორიენტირებული დაპროგრამების ზოგიერთ ენაში შესაძლებელია **მრავლობითი მემკვიდრეობა**. ამ დროს წარმოებული კლასი საკუთარ თვისებებსა და მეთოდებს მემკვიდრეობით რამდენიმე კლასიდან ერთდროულად იღებს. ამ დროს ზოგჯერ შესაძლებელია შეიქმნას კონფლიქტური სიტუაცია, რაც ართულებს დაპროგრამების ენას და, განსაკუთრებით, კომპილატორს. ამიტომ ობიექტზე ორიენტირებული დაპროგრამების ზოგიერთ ენაში უბრალოდ აკრძალულია მრავლობითი მემკვიდრეობა, თუმცა შესაძლებელია მათი იმიტირება,

ყველა წინაპარი და შთამომავალი კლასების სიმრავლეს **კლასების იერარქია** ეწოდება. მისი აგება აუცილებელია იმისათვის, რომ ობიექტზე ორიენტირებული დაპროგრამების მექანიზმების უმრავლესობას შეეძლოს მოახდინოს საკუთარი უპირატესობის რეალიზაცია.

- **პოლიმორფიზმი** *(ბერძ. „**πολὺ**“ – „მრავალი“ და „**μορφή**“ — ფორმა)* ობიექტზე ორიენტირებული დაპროგრამების ყველაზე მნიშვნელოვან პრინციპად მიიჩნევა. იგი სწორედ პოლიმორფიზმის საშუალებით იქცა ისეთ მძლავრ და მაღალი კლასის საქმედ, რომელიც სხვადასხვა ტიპის მონაცემთან მუშაობისათვის ერთი და იგივე კოდის მრავალ-

ჯერ დაწერის აუცილებლობის ნაცვლად ამ კოდის დუბლირებას გვთავაზობს.

პოლიმორფიზმი ყველაზე მარტივად შეგვიძლია „ფუნქციის (მეთოდის) მიერ სხვადასხვა ტიპის მონაცემების გამოყენების უნარის“ სახით განვსზღვროთ. თითქოს ყველაფერი ძალიან მარტივია, მაგრამ ამ სიმარტივე-შია დამალული მთელი სირთულე! თუ ჩავუფიქრდებით, მივხვდებით, რომ თითოეული **ფუნქცია სხვადასხვა ტიპის მონაცემს სხვადასხვანაირად იყენებს**. არის ორი სახის პოლიმორფიზმი: ***ad-hoc*** პოლიმორფიზმი და პარამეტრული პოლიმორფიზმი. ინგლისური ტერმინი „***ad-hoc***“ ქართულად ითარგმნება, როგორც **„ამ შემთხვევისათვის**“, ამიტომ ***ad-hoc*** პოლიმორფიზმს პირობითად **კერძო სახის პოლიმორფიზმი,** ხოლო პარამეტრულ პოლიმორფიზმს ჭეშმარიტი ანუ უნივერსალური პოლიმორფიზმი ვუწოდოთ.

Ad-hoc პოლიმორფიზმი ბანალური სახისაა. იგი ორ ეტაპისაგან შედგება. **პირველ ეტაპზე** ხდება ტიპების მიხედვით მონაცემების დაჯგუფება, რომელთაგანაც თითოეული ჯგუფში შედის ერთნაირი ტიპის მონაცემები. დავუშვათ, რომ ჯგუფების რაოდენობა ***N***-ის ტოლია. **მეორე ეტაპზე** მზადდება ერთნაირი სახელწოდების, მაგრამ განსხვავებული პარამეტრებიანი ***N*** რაოდენობის ფუნქცია. ამგვარად, ერთნაირი სახელწოდების მიუხედავად მათ აქვთ **განსახვავებული პროგრამული ტანი.** თითოეულ მათგანს უნდა მიეწოდოს კონკრეტულ ჯგუფში შემავალი მონაცემები. ისინი მიწოდებულ მონაცემებს სხვადასხვანაირად გადაამუშავებს.

პარამეტრული, ანუ ნამდვილი პოლიმორფიზმის დროს გვაქვს ერთნაირი პროგრამული ტანიანი ფუნქციები და მათ შეუძლია გადაამუშაოს სხვადასხვა ტიპის მონაცემი. იმის ახსნა, თუ როგორ ხდება ეს დაინტერესებულ პირებს შეუძლიათ გაეცნონ სპეციალიზებულ ლიტერატურაში. ჩვენ კი თქვენ ყურადღებას შევაჩერებთ პოლიმორფიზმის სარგებლიანობაზე. ამას განსაზღვრავი ის საოცარი გარემოება, რომ პროგრამაში არსებული ყველა ***if*** ოპერატორი შესაძლებელია შეიცვალოს პოლიმორფიზმებით; მართლაც, ერთი შტო შეიძლება წავიდეს რომელიმე მემკვიდრის კლასისკენ, ხოლო მეორე (***else***) შტო - მეორე მემკვიდრის ბაზისური კლასისკენ. შტოების არარსებობის შემთხვევაში რჩება ცარიელი ადგილი - ფუნქცია არაფერს არ აკეთებს. ეს მთელი ხარისხით ამცირებს პროგრამის ზომას. პოლიმორფიზმი წარმოადგენს დახლართული, რთული და ძნელად მხარდასაჭერი კო-

დის თავიდან აცილების ხერხს. სწორედ პოლიმორფიზმის დახმარებით შეგვიძლია ვუზრუნველყოთ მოქნილობა და თავიდან ავიცილოთ კოდის გადამეტტვირთულობა

თანამედროვე დაპროგრამებითი ინჟინერინგმა იმ მდგომარეობას მიაღწია, რომ მას თავისუფლად შეუძლია უარი თქვას მემკვიდრეობაზე. სამწუხაროდ, თუ არ იქნება მიმკვიდრეობა, მაშინ არ იქნება ნამდვილი პოლიმორფიზმი. მაგრამ თუ პატერნ დეკორატორიდან დაწყებული ყველა არსებულ პატერნს გადავხედავთ, დავინახავ, რომ ისინი პოლომორფიზმის გამოყენებითაა აგებული. ინფორმატიკაში ტერმინ „**პატერნით**“(ინგ. pattern – «შაბლონი, „ნიმუში“) კომპიუტერული პროგრამების დაწერის ეფექტურ ხერხი აღინიშნება. პატერნი „**დეკორატორი**“ წარმოადგენს დაპროექტების სტრუქტურულ პატერნს; იგი საშუალებას გვაძლევს სასარგებლო სახვევებში გახვევის გზით ობიექტებს დინამიკურად ახალი ფუნქციურობა დავუმატოთ. თუ ჩვენ კოდში არაა მემკვიდრეობა, მაშინ არც პოლიმორფიზმია და ჩვენი კოდიც არ იქნება ობიექტზე ორიენტირებული დაპროგრამების კოდი; ამის შედეგად საკუთარი პროგრამის შუა ნაწილში მივიღებთ პროცედურულ კოდს. ეს ძალიან ცუდია! ვიმედოვნებთ, რომ ისტორიის კიდევ ერთი წრის შემდეგ ჩვენ ხელახლა დავუბრუნდებით მდიდრულ ობიექტებს.

დეკლარაციული დაპროგრამება არის დაპროგრამების პარადიგმა, რომელშიც მოცემულია ამოცანის გადაწყვეტის **სპეციფიკაცია**, ე. ი. აღწერილია თუ რას წარმოადგენს პრობლემა და როგორი შედეგის მიღებაა მოსალოდნელი მისი გადაწყვეტით.

შეგახსენეთ, რომ ტერმინი „სპეციფიკაცია“ (ლათ. „species + facio” = „საქმე“ + „ვაკეთებ“) ნიშნავს მოთხოვნების შემცველ დოკუმენტს.

დეკლარაციული დაპროგრამების საწინააღმდეგო პარადიგმაა იმპერატიული დაპროგრამება, რომელშიც დეტალიზაციის ამა თუ იმ დონეზე აღიწერება შედეგის მისაღებად თუ როგორ უნდა იქნეს გადაწყვეტილი პრობლემა. ზოგადად და მთლიანობაში დეკლარაციული დაპროგრამება მიმართულია ადამიანიდან მანქანისაკენ, მაშინ, როდესაც, იმპერატიული დაპროგრამება - მანქანიდან ადამიანისაკენ. როგორც შედეგი, დეკლარაციულ პროგრამებში არ გამოიყენება მდგომარეობის ცნება, კერძოდ, იგი არ შეიცავს მითითებითი გამ-

ჭვირვალების უზრუნველმყოფ ცვლადებსა და მინიჭების ოპერატორებს.

მითითებითი გამჭვირვალება კომპიუტერულ პროგრამათა ნაწილების თვისებაა. გამოსახულება ითვლება დამოწმებითად გამჭვირვალედ, თუ იგი პროგრამის ქცევის შეუცვლელად შეიძლება შესაბამისი მნიშვნელობით შეიცვალოს. ასეთი ფუნქციის გამოთვლის შედეგად არგუმენტების ერთი და იმავე მნიშვნელობების დროს ვიღებთ ერთსა და იმავე მნიშვნელობებს (შედეგებს). მათ სუფთა ფუნქციებსაც უწოდებენ. დაპროგრამების ენებში **სუფთა ფუნქციად** ითვლება თანამდევი ეფექტების არმქონე დეტერმინირებული ფუნქცია. დეტერმინირებული (ლათ. determinans – „განმსაზღვრელი“) ნიშნავს განსაზღვრებადობას.

დეკლარაციული დაპროგრამების ქვესახეებად მიჩნევენ **ფუნქციურ და ლოგიკურ დაპროგრამებებს**, მიუხედავად იმისა, რომ ამ ენებზე დაწერილი პროგრამები არც თუ ისე იშვიათად შეიცავს ალგორითმულ მდგენელებსაც.

„წმინდა დეკლარაციული“ კომპიუტერული, მაგალითად, **SQL** და **HTML**, ენები **ტიურინგის მიხედვით სრული** არ არის, რამდენადაც დეკლარაციული აღწერის მიხედვთ შემსრულებელი კოდის წარმოშობა თეორიულად ყოველთვის შეუძლებელია. ამის გამო ხშირად საკამათო ხდება ტერმინ „დეკლარაციული დაპროგრამების“ ტერმინის კორექტურობა.

ტიურინგის მიხედვით სისრულე გამოთვლადობის თეორიაში არის **შემსრულებლის** მახასიათებელი და აღნიშნული შემსრულებლის საშუალებით ნებისმიერი გამოთვლადი ფუნქციის რეალიზების შესაძლებლობადობას გამოხატავს.

ფუნქციური დაპროგრამება არის დაპროგრამების პარადიგმა, რომელშიც გამოთვლების პროცესად მიიჩნევა მათემატიკური ფუნქციების მნიშვნელობის გამოთვლის პროცესი (მათემატიკური ფუნქციისაგან განსხვავებით **პროცედურულ დაპროგრამებაში** ფუნქციად გამოიყენება არა მათემატიკური ფუნქცია, არამედ **ქვეპროგრამა**). დაფუძნებულია **λ**-აღრიცხვაზე. ეს უკანასკნელი არის გამოთვლადობის ცნების ფორმალიზებისა და ანალიზისათვის ამერიკელი მათემატიკოსის **ალონზო ჩერჩის** (*Alonzo Church. 1903- 1995*) მიერ დამუშავებული ფორმალური სისტემა.

ფუნქციური დაპროგრამება იმპერატიული დაპროგრამების პარადიგმის საპირისპიროა, რომელიც გამოთვლების პროცესს მდგომარეობის მიმდევრობითი ცვლილების გამოთვლის სახით აღწერს. აუცილებლობის დროს, ფუნქციურ დაპროგრამებაში გამოთვლითი პროცესის მიმდევრობითი მდგომარეობების მთელი ერთობლიობა ცხადი, მაგალითად, **სიის** სახით წარმოიდგინება.

იმპერატიულ დაპროგრამებაში არსებული „ფუნქციის“ ცნება იმით განსხვავდება **მათემატიკური ფუნქციისაგან**, რომ **იმპერატიული ფუნქცია** დამოკიდებულია არა მარტო არგუმენტებზე, არამედ ფუნქციისადმი გარეგან ცვლადებზეც და, გარდა ამისა, მისთვის დამახასიათებელია **თანამდები ეფექტების** არსებობა, რომლებიც ცვლის გარე ცვლადების მდგომარეობებს.

ამგვარად, **იმპერატიულ დაპროგრამებაში** ერთნაირი პარამეტრებიანი ერთი და იმავე ფუნქციას თუ ალგორითმის შესრულების სხვადასხვა ეტაპზე გამოვიძახებთ, მაშინ გამოსასვლელზე შეიძლება მივიღოთ განსხვავებული მონაცემები: ეს გამოწვეულია ფუნქციაზე ცვლადების მდგომარეობის გავლენით.

რაც შეეხება **ფუნქციურ ენას**, ამ შემთხვევაში ერთსა და იმავე არგუმენტებზე დამოკიდებული ფუნქციის გამოძახების დროს გამოსასვლელზე ყოველთვის ერთნაირ შედეგს მივიღებთ: გამოსასვლელი მონაცემები მხოლოდ შესასვლელ მონაცემებზეა დამოკიდებული. ეს ფუნქციურ ენებზე დაწერილი პროგრამების შემსრულებელ გარემოს საშუალებას აძლევს მოახდინოს მოახდინოს ფუნქციათა შედეგების **კეშირება** და ისინი არ გამოიძახოს ალგორითმის მიერ გასაზღვრული თანამიმდევრობით და დააპარალელოს დამპროგრამებლის მიერ რაიმე დამატებითი მოქმედებების ჩაუტარებლად. ეს უზრუნველყოფს თანმდევი ეფექტების არმქონე ფუნქციების, ე. წ. **სუფთა ფუნქციების**, არსებობას.

აპლიკაციური დაპროგრამება არის დეკლარაციული დაპრო-
-გრამების ერთ-ერთი სახე. მისი გამოყენებით დაწერილი პროგრამებში ერთ ობიექტი სისტემატიურად გამოიყენება მეორე ობიექტის მიმართებით. აღნიშნულის შედეგად წარმოიქმნება ახალი ობიექტი, რომელმაც შეიძლება შეასრულოს როგორც ფუნქციის, აგრეთვე არგუმენტის და ა.შ. როლი. ეს ჩანაწერს მათემატიკურად ცხადს ხდის. ჩვენ შეგვიძლია ფუნქციები გამოსახულებების სახით ჩავწეროთ. ამის შე-

დეგად მივიღებთ **გამოსახულება-ფუნქციებს**, ანუ **ფუნქციურ ელემენტებს**. ეს უკანასკნელები კი შეგვიძლია გამოვიყენოთ სხვა ფუნქციების როგორც არგუმენტებად, ისე მნიშვნელობებადაც.

კომბინატორული დაპროგრამება (ანუ **function-level programing**) არის დაპროგრამების პარადიგმა, რომელიც იყენებს **კომბინატორულ ლოგიკის** პრინციპებს. ეს იმას ნიშნავს, რომ იგი არ საჭიროებს ფუნქციის (პროგრამის) მიერ განსაზღვრული არგუმენტების ცხადად ხსენებას და რომელიც ცვლადების ნაცვლად იყენებს **კომბინატორებსა** და **კომპოზიციებს.** იგი **ფუნქციური დაპროგრამების** სახესხვაობას წარმოადგენს, ოღონდ მისი ძირითადი მიმართულებისაგან განსხვავებით, არ გამოიყენებს **λ-აბსტრაქციას**.

კომბინატორული ლოგიკა არის მათემატიკური ლოგიკის მიმართულება, რომელიც დაკავებულია ფორმალური ლოგიკური სისტემებისა და აღრიცხვების ფუნდამენტური ცნებებითა და მეთოდებით.

მოცემული პარადიგმის შემომტანი და პოპულიზატორია ამერიკელი მათემატიკოსი **ჯონ ბეკუსი** (*John Warner Backus. 1924 – 2007*), რომელმაც ამსტერდამელ კოლეგებთან ერთად დაამუშავა ***FL*** ენა.

ლოგიკური დაპროგრამება არის დაპროგრამების პარადიგმა, რომელიც დაფუძნებულია თეორემების ავტომატურ დამტკიცებასა და დისკრეტული მათემატიკის იმ განყოფილებაზე, რომელიც შეისწავლის ინფორმაციის ლოგიკურად მიღების პრინციპებს. ლოგიკური დაპროგრამება დაფუძნებულია მათემატიკური ლოგიკის თეორიასა და აპარატზე, რომელიც იყენებს რეზოლუციების მათემატიკურ პრინციპებს. შევნიშნავთ, რომ **რეზოლუციების წესი** ეს არის დასკვნის მიღების წესი, რომელიც წინაღმდეგობის ძიების მეშვეობით მიმართულია თეორემების დამტკიცების მეთოდისაკენ. ლოგიკური დაპროგრამების ყველაზე ცნობილი ენაა **Prolog.**

ლოგიკური დაპროგრამება წარმოადგენს **დეკლარაციული დაპროგრამების** კერძო შემთხვევას, რამდენადაც დამპროგრამებელი იძლევა მხოლოდ ფორმულების ნაკრებს, ხოლო გამოთვლების ორგანიზების შესახებ გადაწყვეტილებებს იღებს კომპილატორი.

განზოგადებელი დაპროგრამება (**generic programming**) არის დაპროგრამების პარადიგმა, რომლის მიზანია ალგორითმების ისეთი ფორმით შედგენა, რომ სხვადასხვა ტიპის მონაცემებისათვის იყოს შესაძლებელი მათი გამოყენება. დავუშვათ, რომ გვსურს დავამუშაოთ ნებისმიერი ტიპის **a** და **b** არგუმენტებიდან უდიდესი არგუმენტის განმსაზღვრელი **maximum (a, b)** ფუნქციების ოჯახის ოჯახის დამუშავება შემთხვევებისათვის, როდესაც **a** და **b** არგუმენტებად შეიძლება გამოყენებული იქნება ნებისმიერი ტიპის ცვლადები.

დასმული ამოცანის ტრივიალურად გადაწყვეტის დროს საჭიროა **a** და **b** არგუმენტების მნიშვნელობებად ავიღოთ კონკრეტული ტიპის ცვლადები და მათთვის გადავწყვიტოთ დასმული ამოცანა. შემდეგ მიმდევრობით ვცვალოთ აღნიშნული არგუმენტების მნიშვნელობებად გამოყენებული ცვლადების ტიპები და თითოეული შემთხვევისათვის გადავწყვიტოთ დასმული ამოცანა.

ნათელია, რომ დასმული ამოცანის ტრივიალურად გადაწყვეტა შრომატევადია და საჭიროებს გამარტივებას. ამ მიზნით **განზოგადებული დაპროგრამების პარადიგმაში** მოხდენილია აბსტრაქტული ტიპის მონაცემების განზოგადება.

აბსტრაქტული ტიპის მონცემები არის მონაცემებისა და მასზე შესასრულებელი ოპერაციების ნაკრები. ე. ი. ერთი მიზნის მისაღწევად საჭირო მონაცემებისა და მეთოდების ნაკრები.

აბსტრაქტული ტიპის მონაცემების განზოგადების შედეგად მონაცემთა ტიპები წარმოიდგინება **ინტერფეისების სახით,** რომლებიც დამპროგრამებლა „უმალავს“, თუ რა ტიპის მონაცმთან აქვს მას საქმე (რამდენადსაც მომავალში შეიძლება შეიცვალოს მისი ტიპი). დამპროგრამებელი აბსტრაქტული ტიპის მონაცემებთან მუშაობს არა უშუალოდ, არამრდ ინტერფეისის მეშვეობით. ობიექტზე ორიენტირებულ დაპროგრამებაში ასეთ მიდგომას **ინკაფსულაცია** ეწოდება.

მაშასადამე, მონაცემების ცალკეული ტიპების აღწერის ნაცვლად განზოგადებულ დაპროგრამებაში აღიწერება **საერთო ინტერფეისისა** და **სემანტიკური ქცევის** (***semantic behavior***) მქონე ტიპების ოჯახი. ინტერფეისისა და სემანტიკური ქცევის აღმწერი მოთხოვნების ნაკრებს ეწოდება **კონცეფცია** (***concept***). ამგვარად, **განზოგადებული სტილით დაწერილი ალგორითმი** შეგვიძლია გამოვიყენოთ საკუთარი

კონცეფციების დამაკმაყოფილებელი ნებისმიერი ტიპებისათვის. ასეთ შესაძლებლობას **პოლიმორფიზმი** ეწოდება.

განზოგადებული დაპროგრამების შესაძლებლობები **ჯენერიკების** (ინგ. generic– *„გვაროვნული“.* არა უფრო სპეციფიკური, არამედ უფრო ზოგადი ფუნქციების) სახით პირველად **1970**-იან წლებში გამოჩნდა დაპროგრამების ენებში **Ada**, **Clu** ხოლო შემდეგ - ობიექტზე ორიენტირებულ ენებში **C++, Python, Java**. **Object Pascal, D, Eiffel, .NET** პლატფორმის ენებში.

ობიექტზე ორიენტირებული დაპროგრამება მუდმივად ვითარდება და წარმოშობს ახალ პარადიგმებს. ასეთი პარადიგმებია:

- **ასპექტებზე ორიენტირებული დაპროგრამება**;
- **სუბიექტებზე ორიენტირებული დაპროგრამება**;
- **აგენტზე ორიენტირებული დაპროგრამება**.

ასპექტზე ორიენტირებული დაფუძნებულია ფუნქციურობის დაყოფის იდეაზეა დაფუძნებული. ასეთი დაყოფა აუმჯობესებს პროგრამის მოდულებად დაყოფას, მისი მეთოდოლოგია შემოთვაზებული იქნა კვლევითი ცენტრ **Xerox PARC**-ის ცენტრის ინჟინრთა ჯგუფის მიერ, რომელსაც ხელმძღვანელობდა **გრეგორ ლიჩალესი** (*Gregor Kiczales*). მათ მიერვე იქნა დამუშავებული ასპექტზე ორიენტირებული გაფართოება დაპროგრამების ენისთვის **Java.** ასპექტზე ორიენტირებული დაპროგრამება ავსებს ობიექტზე ორიენტირებულ დაპროგრამებას, ამდიდრებს მას სხვა სახის მოდულურობით, რომელიც საშუალებას იძლევა ერთ მოდულში მოახდინოს „გამჭოლი“ ლოგიკის მარეალიზებელი კოდის ლოკალიზება. ასეთ მოდულებს **ასპექტებს** უწოდებენ. ასპექტზე ორიენტირებული კოდის გამოცალკევების ხარჯზე მარტივდება „გამჭოლ“ მიმართებებთან მუშაობა.

კომპილაციის დროს ასპექტები შეიძლება სისტემაში შეიცვლოს, ჩაისვას, გაძევდეს ისინი სისტემიდან და, უფრო მეტიც, განმეორებით იქნას გამოყენებული.

კომპონენტზე ორიენტირებული დაპროგრამება (***componentoriented programming, COP***) არის დაპროგრამების პარადიგმა, რომელიც არსებითად ეყრდნობა კომპონენტია ცნებას. **კომპონენტა** პროგრამის საწყისი კოდის დამოუკიდებელი მოდულია, რომელიც განკუთვნილია განმეორებითი გამოყენებისათვის. იგი საერთო ნიშნის მიხედვით გაერთიანებული ენური კონსტრუქციების (მაგალითად, ობიექტ-

ზე ოეიენტირებულ დაპროგრამებაში არსებული „კლასების“) სახით არის რეალიზებული და ორგანიზებულია გარკვეული წესებისა და შეზღუდვების შესაბამისად

ხდომილობაზე ორიენტირებული დაპროგრამება დაფუძნებულია ხდომილობებზე (კლავიშზე დაჭერაზე, მაუსზე დაწკაპუნებაზე). საპასუხოდ **Windows**-ში გენერირდება სათანადო შეტყობინება, რომელიც ეგზავნება პროგრამის შესაბამის ფანჯარას.

ხდომილობაზე ორიენტირებულ დაპროგრამებაში პროგრამის სტრუქტურა ასეთია: პროგრამის მთავარი ნაწილი წარმოადგენს ერთ უსასრულო ბლოკს, რომელიც გამოიკითხავს **Windows**-ს, თვალყურს ადევნებს ახალი შეტყობინების გამოჩენას. მისი აღმოჩენის დროს გამოიძახება ამ შეტყობინების დამამუშავებელი პროგრამა. გამოკითხვის ციკლი გაგრძელდება მუშაობის დამთავრების შესახებ შეტყობინების მიღებამდე.

ხდომილობა შეიძლება იყოს სამომხმარებლო, სისტემური და პროგრამული (თავად პროგრამის მიერ გენერირებული).

დეკლარაციული დაპროგრამება არის დაპროგრამების პარადიგმა, რომელშიც მოცემულია ამოცანის გადაწყვეტის **სპეციფიკაცია**, ე. ი. აღწერილია თუ რას წარმოადგენს პრობლემა და როგორი შედეგის მიღებაა მოსალოდნელი მისი გადაწყვეტით.

შეგახსენეთ, რომ ტერმინი „სპეციფიკაცია“ (ლათ. „species + facio” = „საქმე“ + „ვაკეთებ“) ნიშნავს მოთხოვნების შემცველ დოკუმენტს.

დეკლარაციული დაპროგრამების საწინააღმდეგო პარადიგმაა იმპერატიული დაპროგრამება, რომელშიც დეტალიზაციის ამა თუ იმ დონეზე აღიწერება შედეგის მისაღებად თუ როგორ უნდა იქნეს გადაწყვეტილი პრობლემა. ზოგადად და მთლიანობაში დეკლარაციული დაპროგრამება მიმართულია ადამიანიდან მანქანისაკენ, მაშინ, როდესაც, იმპერატიული დაპროგრამება - მანქანიდან ადამიანისაკენ. როგორც შედეგი, დეკლარაციულ პროგრამებში არ გამოიყენება მდგომარეობის ცნება, კერძოდ, იგი არ შეიცავს მითითებითი გამჭვირვალების უზრუნველმყოფ ცვლადებსა და მინიჭების ოპერატორებს.

მითითებითი გამჭვირვალება კომპიუტერულ პროგრამათა ნაწილების თვისებაა. გამოსახულება ითვლება დამოწმებითად გამჭვირ-

ვალედ, თუ იგი პროგრამის ქცევის შეუცვლელად შეიძლება შესაბამისი მნიშვნელობით შეიცვალოს. ასეთი ფუნქციის გამოთვლის შედეგად არგუმენტების ერთსა და იმავე მნიშვნელობების დროს ვიღებთ ერთსა და იმავე მნიშვნელობებს (შედეგებს). მათ სუფთა ფუნქციებსაც უწოდებენ. დაპროგრამების ენებში **სუფთა ფუნქციად** ითვლება თანამდევი ეფექტების არმქონე დეტერმინირებული ფუნქცია. დეტერმინირებული (ლათ. determinans – „განმსაზღვრელი“) ნიშნავს განსაზღვრებადობას.

დეკლარაციული დაპროგრამების ქვესახეებად მიჩნევენ **ფუნქციურ და ლოგიკურ დაპროგრამებებს**, მიუხედავად იმისა, რომ ამ ენებზე დაწერილი პროგრამები არც თუ ისე იშვიათად შეიცავს ალგორითმულ მდგენელებსაც.

„წმინდა დეკლარაციული“ კომპიუტერული, მაგალითად, **SQL** და **HTML**, ენები **ტიურინგის მიხედვით სრული** არ არის, რამდენადაც დეკლარაციული აღწერის მიხედვთ შემსრულებელი კოდის წარმოშობა თეორიულად ყოველთვის შეუძლებელია. ამის გამო ხშირად საკამათო ხდება ტერმინ „დეკლარაციული დაპროგრამების“ ტერმინის კორექტურობა.

შევნიშნავთ, რომ **ტიურინგის მიხედვით სისრულე** გამოთვლადობის თეორიაში არის **შემსრულებლის** მახასიათებელი და აღნიშნული შემსრულებლის საშუალებით ნებისმიერი გამოთვლადი ფუნქციის რეალიზების შესაძლებლობადობას გამოხატავს. იგი ატარებს ინგლისელი მათემატიკოსის **ალან ტიურინგის** (*Alan Mathison Turing; 1912-1954*) სახელს, რომელმაც დაამუშავა **ტიურინგის მანქანად** წოდებული აბსტრაქტული შემსრულებელი და განსაზღვრა მისი მეშვეობით გამოთვლადი ფუნქციების სიმრავლე. **ტიურინგის პრემია** კი დღეისთვის ინფორმატიკის სფეროში ყველაზე პრესტიჟულ პრემიად ითვლება.

ფუნქციური დაპროგრამება არის დაპროგრამების პარადიგმა, რომელშიც გამოთვლების პროცესად მიიჩნევა მათემატიკური ფუნქციების მნიშვნელობის გამოთვლის პროცესი (მათემატიკური ფუნქციისაგან განსხვავებით **პროცედურულ დაპროგრამებაში** ფუნქციად გამოიყენება არა მათემატიკური ფუნქცია, არამედ **ქვეპროგრამა**). დაფუძნებულია λ-აღრიცხვაზე. ეს უკანასკნელი არის გამოთვლადობის ცნების ფორმალიზებისა და ანალიზისათვის ამერიკელი მათემატი-

კოსის **ალონზო ჩერჩის** (*Alonzo Church. 1903- 1995*) მიერ დამუშავებული ფორმალმური სისტემა.

ფუნქციური დაპროგრამება იმპერატიული დაპროგრამების პარადიგმის საპირისპიროა, რომელიც გამოთვლების პროცესს მდგომარეობის მიმდევრობითი ცვლილების გამოთვლის სახით აღწერს. აუცილებლობის დროს, ფუნქციურ დაპროგრამებაში გამოთვლითი პროცესის მიმდევრობითი მდგომარეობების მთელი ერთობლიობა ცხადი, მაგალითად, **სიის** სახით წარმოიდგინება.

იმპერატიულ დაპროგრამებაში არსებული „ფუნქციის“ ცნება იმით განსხვავდება **მათემატიკური ფუნქციისაგან**, რომ **იმპერატიული ფუნქცია** დამოკიდებულია არა მარტო არგუმენტებზე, არამედ ფუნქციისადმი გარეგან ცვლადებზეც და, გარდა ამისა, მისთვის დამახასიათებელია **თანამდები ეფექტების** არსებობა, რომლებიც ცვლის გარე ცვლადების მდგომარეობებს.

ამგვარად, **იმპერატიულ დაპროგრამებაში** ერთნაირი პარამეტრებიანი ერთსა და იმავე ფუნქციას თუ ალგორითმის შესრულების სხვადასხვა ეტაპზე გამოვიძახებთ, მაშინ გამოსასვლელზე შეიძლება მივიღოთ განსხვავებული მონაცემები: ეს გამოწვეულია ფუნქციაზე ცვლადების მდგომარეობის გავლენით.

რაც შეეხება **ფუნქციურ ენას**, ამ შემთხვევაში ერთსა და იმავე არგუმენტებზე დამოკიდებული ფუნქციის გამოძახების დროს გამოსასვლელზე ყოველთვის ერთნაირ შედეგს მივიღებთ: გამოსასვლელი მონაცემები მხოლოდ შესასვლელ მონაცემებზეა დამოკიდებული. ეს ფუნქციურ ენებზე დაწერილი პროგრამების შემსრულებელ გარემოს საშუალებას აძლევს მოახდინოს მოახდინოს ფუნქციათა შედეგების **კეშირება** და ისინი არ გამოიძახოს ალგორითმის მიერ გასაზღვრული თანამიმდევრობით და დააპარალელოს დამპროგრამებლის მიერ რაიმე დამატებითი მოქმედებების ჩაუტარებლად. ეს უზრუნველყოფს თანამდევი ეფექტების არმქონე ფუნქციების, ე.წ. **სუფთა ფუნქციების**, არსებობას.

აპლიკაციური დაპროგრამება არის დეკლარაციული დაპრო-გრამების ერთ-ერთი სახე. მისი გამოყენებით დაწერილი პროგრამებში ერთ ობიექტი სისტემატურად გამოიყენება მეორე ობიექტის მიმართებით. აღნიშნულის შედეგად წარმოიქმნება ახალი ობიექტი, რომელმაც შეიძლება შეასრულოს როგორც ფუნქციის, ისე არგუმენტის

და ა. შ. როლი. ეს ჩანაწერს მათემატიკურად ცხადს ხდის. ჩვენ შეგვიძლია ფუნქციები გამოსახულებების სახით ჩავწეროთ. ამის შედეგად მივიღებთ **გამოსახულება-ფუნქციებს**, ანუ **ფუნქციურ ელემენტებს**. ეს უკანასკნელები კი შეგვიძლია გამოვიყენოთ სხვა ფუნქციების როგორც არგუმენტებად, ისე მნიშვნელობებადაც.

კომბინატორული დაპროგრამება (ანუ **function-level programing**) არის დაპროგრამების პარადიგმა, რომელიც იყენებს **კომბინატორულ ლოგიკის** პრინციპებს. ეს იმას ნიშნავს, რომ იგი არ საჭიროებს ფუნქციის (პროგრამის) მიერ განსაზღვრული არგუმენტების ცხადად ხსენებას და რომელიც ცვლადების ნაცვლად იყენებს **კომბინატორებსა** და **კომპოზიციებს.** იგი **ფუნქციური დაპროგრამების** სახესხვაობას წარმოადგენს, ოღონდ მისი ძირითადი მიმართულებისაგან განსხვავებით, არ გამოიყენებს **λ-აბსტრაქციას**.

კომბინატორული ლოგიკა არის მათემატიკური ლოგიკის მიმართულება, რომელიც დაკავებულია ფორმალური ლოგიკური სისტემებისა და აღრიცხვების ფუნდამენტური ცნებებითა და მეთოდებით.

მოცემული პარადიგმის შემომტანი და პოპულიზატორია ამერიკელი მათემატიკოსი **ჯონ ბეკუსი** (*John Warner Backus. 1924 – 2007*), რომელმაც ამსტერდამელ კოლეგებთან ერთად დაამუშავა ***FL*** ენა.

ლოგიკური დაპროგრამება არის დაპროგრამების პარადიგმა, რომელიც დაფუძნებულია თეორემების ავტომატურ დამტკიცებასა და დისკრეტული მათემატიკის იმ განყოფილებაზე, რომელიც შეისწავლის ინფორმაციის ლოგიკურად მიღების პრინციპებს. ლოგიკური დაპროგრამება დაფუძნებულია მათემატიკური ლოგიკის თეორიასა და აპარატზე, რომელიც იყენებს რეზოლუციების მათემატიკურ პრინციპებს. შევნიშნავთ, რომ **რეზოლუციების წესი** ეს არის დასკვნის მიღების წესი, რომელიც წინააღმდეგობის ძიების მეშვეობით მიმართულია თეორემების დამტკიცების მეთოდისაკენ. ლოგიკური დაპროგრამების ყველაზე ცნობილი ენაა **Prolog**.

ლოგიკური დაპროგრამება წარმოადგენს **დეკლარაციული დაპროგრამების** კერძო შემთხვევას, რამდენადაც დამპროგრამებელი იძლევა მხოლოდ ფორმულების ნაკრებს, ხოლო გამოთვლების ორგანიზების შესახებ გადაწყვეტილებებს იღებს კომპილატორი.

განზოგადებელი დაპროგრამება (**generic programming**) არის დაპროგრამების პარადიგმა, რომლის მიზანია ალგორითმების ისეთი ფორმით შედგენა, რომ სხვადასხვა ტიპის მონაცემებისათვის იყოს შესაძლებელი მათი გამოყენება. დავუშვათ, რომ გვსურს დავამუშაოთ ნებისმიერი ტიპის **a** და **b** არგუმენტებიდან უდიდესი არგუმენტის განმსაზღვრელი **maximum (a, b)** ფუნქციების ოჯახის დამუშავება შემთხვევებისათვის, როდესაც **a** და **b** არგუმენტებად შეიძლება გამოყენებული იქნება ნებისმიერი ტიპის ცვლადები.

დასმული ამოცანის ტრივიალურად გადაწყვეტის დროს საჭიროა **a** და **b** არგუმენტების მნიშვნელობებად ავიღოთ კონკრეტული ტიპის ცვლადები და მათთვის გადავწყვიტოთ დასმული ამოცანა. შემდეგ მიმდევრობით ვცვალოთ აღნიშნული არგუმენტების მნიშვნელობებად გამოყენებული ცვლადების ტიპები და თითოეული შემთხვევისათვის გადავწყვიტოთ დასმული ამოცანი.

ნათელია, რომ დასმული ამოცანის ტრივიალურად გადაწყვეტა შრომატევადია და საჭიროებს გამარტივებას. ამ მიზნით **განზოგადებული დაპროგრამების პარადიგმაში** მოხდენილია აბსტრაქტული ტიპის მონაცემების განზოგადება.

აბსტრაქტული ტიპის მონცემები არის მონაცემებისა და მასზე შესასრულებელი ოპერაციების ნაკრები. ე. ი. ერთი მიზნის მისაღწევად საჭირო მონაცემებისა და მეთოდების ნაკრები.

აბსტრაქტული ტიპის მონაცემების განზოგადების შედეგად მონაცემთა ტიპები წარმოიდგინება **ინტერფეისების სახით,** რომლებიც დამპროგრამებლა „უმალავს“, თუ რა ტიპის მონაცმთან აქვს მას საქმე (რამდენადაც მომავალში შეიძლება შეიცვალოს მისი ტიპი). დამპროგრამებელი აბეტრაქტული ტიპის მონაცემებთან მუშაობს არა უშუალოდ, არამრდ ინტერფეისის მეშვეობით. ობიექტზე ორიენტირებულ დაპროგრამებაში ასეთ მიდგომას **ინკაფსულაცია** ეწოდება.

მაშასადამე მონაცემების ცალკეული ტიპების აღწერის ნაცვლად განზოგადებულ დაპროგრამებაში აღიწერება **საერთო ინტერფეისისა** და **სემანტიკური ქცევის** (***semantic behavior***) მქონე ტიპების ოჯახი. ინტერფეისისა და სემანტიკური ქცევის აღმწერი მოთხოვნების ნაკრებს ეწოდება **კონცეფცია** (***concept***). ამგვარად, **განზოგადებული სტილით დაწერილი ალგორითმი** შეგვიძლია გამოვიყენოთ საკუთარი

კონცეფციების დამაკმაყოფილებელი ნებისმიერი ტიპებისათვის. ასეთ შესაძლებლობას **პოლიმორფიზმი** ეწოდება.

განზოგადებული დაპროგრამების შესაძლებლობები **ჯენერიკების** (ინგ. generic– *„გვაროვნული“.* არა უფრო სპეციფიკური, არამედ უფრო ზოგადი ფუნქციების) სახით პირველად **1970**-იან წლებში გამოჩნდა დაპროგრამების ენებში **Ada**, **Clu** ხოლო შემდეგ - ობიექტზე ორიენტირებულ ენებში **C++, Python, Java**. **Object Pascal, D, Eiffel, .NET** პლატფორმის ენებში.

2.3. დაპროგრამების ენათა კალეიდესკოპი

დაპროგრამება დღეს არსებული ციფრული ეპოქის საფუძველია. ირგვლივ რომ ვხედავთ თანამედროვე ციფრული ტექნოლოგიების განსაცვიფრებელ მიღწევებს, ყოველვის უნდა გვახსოვდეს ბრძნული გამონათქვამი: „ხილს რომ ვიხილებთ, მებაღე ვიკითხოთ და ბაღსაც მივხედოთო“. ზემოთ აღნიშნულ წარმატებებში ლომის წილი მიუძღვის **დაპროგრამების ენებს**, რომლებიც კულისებიდან უჩინარი ძაფებით ეწევა ინფორმაციულ ტექნოლოგიებს ახალ და ახალ მწვერვალებისაკენ. მათი შექმნისათვის ძალისხმევა არ დაუკლიათ ინფორმატიკის მრავალ სპეციალისტს და, სულ მცირე, იმას მაინც იმსახურებენ, რომ მათ სახელებს არ ვივიწყებდეთ მადლიერი შთამომავლობა. ამიტომ ყოველთვის ვცდილობ, ისინი მოვიხსენიო ჩემს მოკრძალებულ ნაშრომებში.

მიმზიდველი აღმოჩნდა დაპროგრამების ენების ევოლუციური განვითარების ისტორია. თავდაპირველად ძნელი წარმოსადგენი იყო, რომ დღისთვის ფორმირებულ ტექნოლოგიური საზოგადოებისაკენ მიმართულ გზას საფუძველს ჩაუყრიდა **XIX** საუკუნის შუა პერიოდში **ადა ლავლეისის** მიერ წამოწყებული სიახლე - პირველი მანქანური ალგორითმი. საზოგადოებამ წარმატებით გაკვალა ადრეული მანქანური კოდებიდან ადვილად წაკითხვადი ურთულესი კოდებისაკენ მიმავალი ერთი შეხედვით მეტად ხორკლიანი და ურთულესი გზა. რა თქმა უნდა, ამ გზის გაყვანის პროცესი არ დასრულებულა. ინ-

ფორმაციული ტექნოლოგიების უმსხვილესი ვებ-სერვის **Git-Hu**-ის ვერსიით **XXI** საუკუნის დასაწყისში **საუკეთესო ენების ათეული** ასე გამოიყურება:

1. **Python**;
2. **Java;**
3. **JavaScript;**
4. **C# ;**
5. **C და C++;**
6. PHP;
7. **R;**
8. **Objective-C;**
9. **Swift;**
10. **MATLAB.**

სხვა ენებთან ერთად ისინი განხილულია ცხრილ **1**-ში. მასში არაა მოყვანილი **R** და **Matlab** ენები. მოკლედ ისინიც განვიხილოთ.

Matlab („Matrix Laboratory“) არის ტექნიკური გამოთვლებისათვის განკუთვნილი გამოყენებითი **პროგრამების პაკეტი.** მას იყენებს მილიონზე მეტი ინჟინერი და მეცნიერ-მუშაკი; მუშაობს ყველა თანამედროვე ოპერაციულ სისტემაზე. **დაპროგრამების ენის სახით** იგი **1970**-იანი წლების ბოლოს დაამუშავა ამერიკელმა მათემატიკოსმა და დამპროგრამებელმა **კლევ მოლერმა** (Cleve Barry Moler. 1939), რომელსაც მუშაობდა **ნიუ მექსიკის** უნივერსიტეტის კომპიუტერული მეცნიერების ფაკულტეტის დეკანად. **1984** წელს მან კომპანიონთან ერთად დააფუძნა მათემატიკური გამოთვლებისა და იმიტაციური მოდელირების დამუშავებაზე სპეციალიზებული ამერიკული კომპანია **MathWorks** და მოახდინა მის მიერ შექმნილი პროგრამული პროდუქტის კომერციალიზება.

R ენა **1993** წელს დაამუშავეს ოკლენდის (ახალი ზელანდია) უნივერსიტეტის სტატისტიკური უნივერსიტეტის პროფესორებმა **როს აიჰაკამ** (Ross Ihaka. 1954) **და რობერეტ ჯენტლემანმა** (Robert Geentleman. 1959). არის დაპროგრამების მულტიდაპროგრამებადი, მაინტერპრეტირებელი ენა. ითვლება **სტანდარტულ ენად** სტატისტიკური მონაცემების დასამუშავებლად.

ცხრ. 1. დაპროგრამების საკვანძო ენების წარმოქმნის ქრონოლოგია

დამუშავების თარიღი	დაპროგრამების ენა	შენიშვნა
1	*2*	*3*
1852	**ადა ლავლეისის მანქანური ენა**	დაამუშავა **ადა ლავლეისმა** (1815-1852) ბებიჯის ანალიზური მანქანისათვის. ამით ჩაეყარა საფუძველი დაპროგრამების ენების დამუშავებას
1943-1948	**Plankalkü**	დაამუშავა გრამანელმა **კონრად ცუზემ** (Konrad Ernst Otto Zuse. 1910-1995). Pla-Plankalkül ქართულად ნიშნავს „გეგმის გამოთვლას“.
1947- 1948	**ასემბლერის ენა**	დაამუშავეს ბრიტანელებმა **კეტლინ ბუტმა** (Kathleen Boot. 1922. 98 წლის) და **დევიდ უილერმა** (David Wheeler. 1927 – 2004). -დაპროგრამების დაბალი დონის ენა, რომელმაც გაამარტივა მანქანური კოდი.
1949	**Short Code** („მოკლე კოდი“)	შემოგვთავაზა ამერიკელმა **ჯონ მოუჩლიმ** (John William Mauchly. 1907-1980), მისი რეალიზება **BINAC** და **UNIVAC** კომპიუტერებზე მოახდინა **უილიამ შმიტმა** (William Schmidt). არის დაპროგრამების მაღალი დონის პირველი ენა, რომელიც ოდესმე დამუშავებულა კომპიუტერებისათვის.
1952	**Autocode** (ავტოკოდირება)	Mark 1-თვისდაამუშავა ბრიტანელმა **ალიკ ედვარდს გლენიმ** (Alick Edwards Glennie. 1925-2003). გამოიყენება დაპროგრამების ენათა ოჯახის სახელწოდებადაც. არის ისტორიაში პირველი მაკომპილირებელი ენა, რაც იმას ნიშნავს, რომ **კომპილატორად** წოდებული პროგრამით უშუალოდ გარდაიქმნება მანქანურ კოდად.

გაგრძელება იხ, მომდევნო გვერდზე

ცხრ. 1. დაპროგრამების საკვანძო ენების წარმოქმნის ქრონოლოგია (გაგრძელება)

1	2	3
1957	**Fortran**	დაამუშავა ამერიკელმა **ჯონ ბეკუსმა** (John Warner Backus. 1924-2007). **FOR**mula +**TRAN**slation არის დაპროგრაების მაღალი დონის უძველესი ენა, იმპერატიული, პროცედურული, მოდულური, ობიექტზე ორიენტირების ელემენტებიანი, მაკომპილირებლი. სამეცნიერო მათემატიკური და სტატიკური გამოთვლებისათვის.
1958	**Algol**	დაამუშავა **ამერიკელი და ევროპელი მეცნიერების ერთობლივმა კომიტეტმა. Algo**rithmik +**L**nguage . პროცედურული, იმპერატიული, სტრუქტურული. მან მისცა იმპულსი დაპროგრამების ისეთი უმნიშვნელოვანესი ენების შექმნას, როგორებიცაა **Pascal**, **C**, **C++** და **Java.**
1958	**Lisp** (სიების დადამამუშავებელი)	დაამუშავა ამერიკელმა **ჯონ მაკარტიმ** (John McCarthy 1927 – 2011). მულტიპარადიგმული, ფუნქციური, პროცედურული, მაინტერპრეტირებელი. თავდაპირველად განკუთვნილი იყო ხელოვნური ინტელექტისათვის. არის დაპროგრამები უძველესი ენა, რომელიც დღესაც შეგვიძლია გამოვიყენოთ Ruby-ისა და Python-ის ნაცვლად. გამოიყენებს ცნობილი კომპანიები Acceleration, Boeing და Genworks
1959	**COBOL**	დაამუშავებას ხელმძღვანელობდა ამერიკელი **გრეის მიურეი ჰოფერი** (Grace Brewster Murray Hopper. 1906 – 1992). პროცედურული, იმპერატიული, ობიექტზე ორიენტირებული. ბიზნესისთვის შექმნილი, საფუძვლად უდევს საკრედიტო ბარათების, ბანკომატების, სატელეფონო და ა.შ. პროცესორს. საბანკო სისტემებში დღემდე გამოყენებადი უძველესი ენა.
1964	**Basic**	დაამუშავე დარტმუნდის კოლეჯის მასწავლებლებმა, ამერიკელმა **თომას კურცმა** (Thomas Eugene Kurtz. 1929; 93 წლის) და უნგრელმა **ჯონ კემენიმ** (John Kemeny. 1926-1992). ალგორითმული, მოგვიანებით პროცედრული, უფრო მოგვიანებით - ობიექტზე ორიენტირებული. სრულყოფაში მონაწილეობდნენ Microsoft-ის დამფუძნებლები **ბილ გეიტსი** და **პოლ ალენი**.

გაგრძელება იხ. მომდევნო გვერდზე

ცხრ. 1. დაპროგრამების საკვანძო ენების წარმოქმნის ქრონოლოგია (გაგრძელება)

1	*2*	*3*
1970	**Pascal**	კომპიუტერული დაპროგრამების სწავლების მიზნით დაამუშავა შვეიცარიელმა **ნიკოლაუს ვირტმა** (Niklaus Emil Witrth. 1944). იმპერატიული, სტრუქტურული, მაკომპილირებელი. თავისი საქმიანობის საწყის ეტაპზე ფირმა Apple მას უპირატესობას აძლევდა სიმარტივისა და სიმძლავრის გამო. დღეს ითვლება სასწავლო ენად.
1972	**Smalltal**k	დაამუშავეს ამერიკელმა მეცნიერებმა **ალან კეიმ** (Alan Kay. 1940), **ადელ გოლდბერგმა** (Adele Goldberg. 1945) და **დანიელ ინგალსმა** (Daniiel Hnry Holmes Ingalls. 1944). დაპროგრამების ობიექტზე ორიენტირებული ენა. დამპროგრამებელს აძლევს კოდის სწრაფად შეცვლის შესაძლებლობას. მასში რეალიზებული ბევრი იდეა გამოყენებულია ენებში Python, Java და Ruby. საკუთარ ტექნიკურ სტეკებში მას იყენებს ცნობილი ფირმები Leafly, Logitech და CrowdStrike.
1972	C	დაამუშავა ამერიკელმა მეცნიერმა **დენის რიჩიმ** (Denif MacAlistar Ritche. 1941-2011). პროცედურული, მაკომპილირებელი, საერთო დანიშნულების. სახელწოდება იმ მიზეზით მიიღო, რომ მის დამუშავებამდე არსებობდა ენები „A“ და „B“. მისგან მიღებული იქნა მრავალი ცნობილი ენა, მათ შორის C#, Java, JavsScript, Perl, PHP, Python და სხვები. მას დღემდე იყენებს ცნობილი კომპანიები Google, Fasebook და Apple.
1972	SQL (თავიდან: SEQUEL)	დაამუშავეს ამერიკელებმა **რეიმონდ ბოისემ** (Raymond Boyse 1947-1974) და **დონალდ ჩემბერლენმა** (Donald Chamberlin). დეკლარაციული. გამოიყენება მონაცემთა ბაზაში ინფორმაციის დათვალიერებისა და შეცვლისათვის, აბრევიატურა SQL ნიშნავს „სტრუქტურირებული მიმართვების ენას“. იყენებს მრავალი კომპანია, მათ შორის Microsof და Accenture.

გაგრძელება იხ. მომდევნო გვერდზე

ცხრ. 1. დაპროგრამების საკვანძო ენების წარმოქმნის ქრონოლოგია (გაგრძელება)

1	*2*	*3*
1980 – - 1981	**Ada**	**აშშ**-ს თავდაცვის სამინისტროს დაკვეთით დაამუშავა ჯგუფმა, რომელსაც ხელმძღვანელობდა ფრანგი მეცნიერი **ჟან დავიდ იშბია** (Jean David Ichbih. 1940-2007). დაპროგრამების ფართო სპექტრის, მულტიპარადიგმული, იმპერატიული, ობიექტზე ორიენტირებული ენა. იგი გაფართოებულია იმ დროისათვის ისეთ პოპულარულ ენებში არსებული ნოვაციებით, როგორიცაა Pascal. დღეს გამოიყენება გერმანიის, ბელგიისა და ავსტრალიის საჰაერო მოძრაობის მართვის სისტემებში.
1983	**C++**	დაამუშავა დანიელმა **ბიარნ სტროუსტრუპმა** (Bjame Stroustrup. 1950). დაპროგრამების პროცედურული, ობიექტზე ორიენტირებული, განზოგადებული ენა. 1986 წელს შევიდა 10 საუკეთესო ენების ჩამონათვალში. 2003 წელს მიიღო დიდების დარბაზის სტატუსი, ფართოდ გამოიყენება ოპერაციული სისტემების დასამუშავებლად, ჩაშენებად სისტემებში, სხვადასხვა მაღალმწარმოებლური პროდუქტების შესაქმნელად.
1983	**Objict-ive-C**	დაამუშავა ამერიკელმა **ბრედ კოკსმა** (Brad Cox. 1944-2021). დაპროგრამების მაკომპილირებელი, ობიექტზე ორიენტირებულიენა. აგებულია დაპროგრამება ენების C-სა და Smalltalk-ის საფუძველზე. იყენებს კორპორაცია Apple თავისი ოპერაციული macOS სისტემისათვის.
1987	**Perl**	დაამუშავა ამერიკელმა **ლარი უოლმა** (Larri Wall. 1954). არის დაპროგრამების მაღალი დონის უნივერსალური ენა. თავდაპირველად შეიქმნა, როგორც ტექსტის რედაქტირებისათვის განკუთვნილი ენა. დღეს იგი მრავალი ისეთი სხვადასხვა მიზნებისათვის გამოიყენება, როგორებიცაა CDI სტანდარტი, მონაცემთა ბაზების მართვა, ქსელური და გრაფიკული დაპროგრამება.

გაგრძელება იხ. მომდევნო გვერდზე

ცხრ. 1. დაპროგრამების საკვანძო ენების წარმოქმნის ქრონოლოგია (გაგრძელება)

1	*2*	*3*
1990	**Haskell**	დამუშავებულია **8** ინფორმატიკოსისაგან შემდგარი ჯგუფის მიერ, რომლის შემადგენლობაში შედიოდნენ შვედი მეცნიერები **ლენარტ აუგუსტსონი** (Lennart Augustron) და **ჯონ ჰიუზი** (John hughes. 1958), ამერიკელი მეცნიერი **პოლ ჰუდაკი** (Paul Raymond Hudak. 1952-2015) და სხვები. სახელი მიენიჭა ამერიკელი მათემატიკოსისა და ლოგიკოსის **ჰასკელ ბრუკს კარის** (Haskell Bruks Curry. 1900-1982) საპატივცემლოდ. დაპროგრამების სტანდარტიზებული წმინდა ფუნქციური საერთო დანიშნულების ენა. წარმატებით გამოიყენება ისეთ დარგებში, სადაც საქმე გვაქვს რთულ გამოთვლებთან, კომპიუტერული თამაშების შესაქმნელად და ა.შ.
1991	**Python**	დამუშავებულია ჰოლანდიელი **გვიდო ვან როსუმის** (Gvido van Rossum. 1956) მიერ. დაპროგრამების მულტიპარადიგმული ენა. არის ობიექტზე ორიენტირებული, სტრუქტურული,განზოგადებული, ფუნქციური ენა. ასპექტზე ორიენტირებული დაპროგრამების ბაზისური მხარდაჭერა რეალიზებულია მეტადაპროგრამების ხარჯზე. დღემდე რჩება მსოფლიოში ერთ-ერთ უპოპულარეს ენად. იყენებს კომპანიები Google, Yahoo და Spotify . სახელწოდება მიიღო ცირკის ხელმძღვანელის **მონტი პითონის** (Monty pitjon) საპატივცემლოდ, რომლის წარმოდგენა მას ძალიან მოეწონა.
1991	**Visual Basic**	დაამუშავა კომპანიამ **Microsoft.** დაპროგრამების პროცედურული, ობიექტზე ორიენტირებული, კომპონენტზე ორიენტირებული, ხდომილურად ორირნტირებული ენა. დამპროგრამებელს საშუალებას აძლევს კოდის წინასწარ მონიშნული ფრაგმენტების ამორჩევისა და შეცვლისათვის გამოიყენოს გრაფიკული ინტერფეისის მეშვეობით **გადათრევის სტილი.** დღეს იგი ფართოდ არ გამოიყენება. Word-ში, Excel-სა და Access-ში მას იყენებს კომპანია Microsoft.

გაგრძელება იხ. მომდევნო გვერდზე

ცხრ. 1. დაპროგრამების საკვანძო ენების წარმოქმნის ქრონოლოგია (გაგრძელება)

1	*2*	*3*
1992	**Ruby** (ლალი)	დაამუშავა იაპონელმა დამპროგრამებელმა **იუკიხიტო მაცუმოტომ** (松本行弘). დაპროგრამების ობიექტზე ორიორიენტირებული, დინამიკური, რეფლექსიური, მაინტერპრეტირებელი ენა. სინტაქსის თავისებურებით უახლოვდება ენებს Perli და Eiffel, ხოლო ობიექტზე ორიენტირებული მიდგომით - ენა Smalltalk-ს. გარდა ამისა, ზოგიერთი თვისება მიღებული აქვს ენებიდან Python, Lisp, Dylan Klu. იყენებს ცნობილი კომპანიები Twitter, Hulu და Groupon.
1995	**Java**	ინტერაქტიული ტელევიზიის პროექტისათვის დაამუშავა კანადელმა **ჯეიმს გოსლინგმა** (Jamas Gosling. 1955.). დაპროგრამების მკაცრად ტიპიზებული ობიექტზე ორიენტირებული ენა. აქვს **კროსპლატფორმირების**, ანუ რამდენიმე აპარატურულ პლატფორმასან ან ოპერაციულ სისტემასთან მუშაობის უნარი. უცვლელად შედის დაპროგრამების საუკეთესო ენათა სიაში. გამოიყენება კომპიუტერებიდან დაწყებული და საპარკე მთვლელებისაგან დამთავრებული, ყველგან.
1995	**PHP**. „Personal Homo Page“ - პირადი საშინაო გვერდი	დაამუშავა დანიელმა **რასმუს ლერდორფმა** (Rasmus Lerdorf. 1968). დაპროგრამების საერთო დანიშნულების სკრიპტული ენა. ინტენსიურად გამოიყენებოდა **ვებ**-დანართებისათვიუსდ. დღეს გამოიყენება **ჰოსტინგ-პროვაიდერების** უმრავლესობა; ლიდეროზს დინამიკურ ვებ-სა**საიტების** შესაქმნელად გამოყენებულ ენებს შორის.
1995	**Java Script**	დაამუშავა ამერიკელმა **ბრენდან ეიჰმა** (Brendan Eich. 1961). დაპროგრამების მულტიპარადიგმული, ობიექტზე ორიენტირებული (პროტოტიპული), განზოგადებული, ფუნქციური, ასპექტზე ორიენტირებული ენა. ძირითადად გამოიყენება დინამიკური ვებ-საიტებისათვის, PDF დოკუმენტებისათვის ვებ-ბრაუზერებისა და კომპიუტერის სამუშაო მაგიდაზე ვიდეოსიუჟეტების დასამუშავებლად.

გაგრძელება იხ. მომდევნო გვერდზე

ცხრ. 1. დაპროგრამების საკვანძო ენების წარმოქმნის ქრონოლოგია (გაგრძელება)

1	*2*	*3*
2000	**C#**	დაამუშავა ინჟინერთა ჯგუფმა ჯგუფმა დანიელი **ანდერ ჰეილსბერგის** (Andres Hejisberg. 1960) ხელმღვანელობით იმისათვის, რომ C++-ის გამოთვლითი შესაძლებლობები გაეერთიანებინა Visual Basic-ის სიმარტივესთან. არის დაპროგრამების მულტიპარადიგმული, ობიექტზე ორიენტირებული, განზოგადებული, პროცედურული, ფუნქციური, რეფლექტური ენა. გამოყენებულია **Microsoft**-ის თითქმის ყველა პროგრამულ პროდუქტში.
2003	**Scala**	დაამუშავა გერმანელმა მეცნიერმა **მარტინ ოდერსკიმ** (Martin Odersky. 1943) . დაპროგრამების მულტიპარადიგმული, ფუნქციური, ობიექტზე ორიენტირებული, იმპერატიული ენა. მასში გაერთიანებულია მათემატიკური ფუნქციური და ობიექტზე ორიენტირებული დაპროგრამებები. შეთავსებადია დაპროგრამების ენასთან Java. ამის გამო სასარგებლოა მისი გამოყენება Android-ის ქვეშ დაპროგრამებისათვის. მას იყენებს Linkedin, Twiter. Foursquare, Netflix და სხვა კომპანიები.
2003	**Groovy** (საუკეთესო, გამორჩეული)	დაამუშავა **ჯეიმს სტრაჩნმა** (James Strachan). დაპროგრამების ობიექტზე ორიენტირებული, იმპერატიული, სცენარული, ფუნქციური, მულტიპარადიგმული, მაინტერპრეტირებელი ენა. დამუშავებულია Java-ს პლატფორმისათვის, როგორც დაპროგრამების Pyyhon, Ruby და Smalltalk ენების შესაძლებლობების მქონე ენა. შეგვიძლია გამოვიყენოთ ნებისმიერ Java-სპროექტში ან როგორც სკრიპტული (სცენარული) ენა. ამ ენამ დაასრულა 1998 წელს ფორმირებული ფორმალურ Java Communite Process - JCP პროცესში, რომელიც დაინტერესებულ პირებს საშუალებას აძლევს მიიღოს მონაწილეობა Java ენის მომავალი ვერსიების ფორმირებაში. გამოიყენებს კომპანიები Starbucks, Transferwise და Craftbase.

დასასრული იხ. მომდევნო გვერდზე

ცხრ. 1. დაპროგრამების საკვანძო ენების წარმოქმნის ქრონოლოგია (დასასრული)

1	*2*	*3*
2009	**Goland (GO)**	კომპანია Google-ს შიგნით დაამუშავეს შვეიცარიელმა მეცნიერებმა **რობერტ გრიზმერმა** (Robert Griesemer. 1964), **რობერტ პაიკმა** (Robert C. Pike, 1956) და კომპიუტერული მეცნიერების პიონერმა **კენეთ (კენ) ლეინ ტომპსონმა** (Kenneth Lane Thompson). დაპროგრამების მრავალნაკადური, იმპერატიული, სტრუქტურირებული, მაკომპილირებული ენა. შეიქმნა იმ პრობლემების გადასაწყვეტად, რომლებიც წარმოიშობოდა პროგრამული სისტემების სიდიდის გამო. სიმარტივისა და თანამედროვე სტრუქტურის გამო დიდი პოპულარობა მოიხვეჭა ისეთ უმსხვილეს კომპანიებში, როგორიცაა Google, Uber, Twitch და Dropbo.
2014	**Swift**	დაამუშავა ამერიკელმა ინჟინერ-დამპროგრამებელმა **კრის ლატნერმა** (Chris Lattner. 1978) . დაპროგრამების მულტიპარადიგმული (ობიექტზე ორიენტირებული, პროტოკოლზე ორიენტირებული, ფუნქციური, იმპერატიული) ენა. შექმნის მიზანი იყო დაპროგრამების C, C++ და Objektive-C ენების შემცვლელი ენის ფორმირება, რომელიც აღნიშნულ ენებზე მარტივი იქნებოდა, რაც შეამცირებდა პროგრამებში შეცდომების დაშვების ალბათობას. არის უნივერსალური ენა: შესაძლებელია გამოვიყენოთ სამაგიდო, ასევე მობილურ კომპიუტერებში და ღრუბლოვანი დაპროგრამების დროს.

კაცობრიობამ **XIX** საუკუნის შუა პერიოდიდან დაწყებული დღემდე დაპროგრამების უამრავი ენა დაამუშავა. **ცხრილ 1-ში** მოყვანილია დაპროგრამების საკვანძო ენების დამუშავების ქრონოლოგიური თანამიმდევრობა და მათი მოკლე დახასიათებები. ვიმედოვნეთ, რომ იგი დაგეხმარებათ დაპროგრამების ენების დღეს არსებული ოკეანის წარმოქმნის პროცესის გააზრებაში.

მესამე ნაწილი

დაპროგრამების სასწავლო ენები

3.1. დაპროგრამების ენა **Paskal** (პასკალი)

3.1.1. ენა Paskal-ის ისტორიიდან

არსებობს დაპროგრამების უამრავი ენა. მათ შორის ფართოდ გამოყენებადი ენების რაოდენობა გაცილებით, თითქმის მთელი ხარისხით, ნაკლებია. იყო დრო, როდესაც **Pascal** იყო პოპულარული, როგორც არა მარტო როგორც სასწავლო, არამედ ისეთი პროგრამა, რომელიც პრაქტიკული ამოცანების გადაწყვეტისათვის გამოიყენებოდა.

Pascal ნ. ვირტმა წინა საუკუნის **60**-იან წლებში სტუდენტებისათვის დაპროგრამების სწავლების მიზნით შექმნა. მასში რეალიზებული პროგრესული იდეებისგამო იგი პრაქტიკოს დამპროგრამებლებშიც გავრცელდა. ამ ენაზე დაიწყეს არა მარტო გამოყენებითი, არამედ სისტემური პროგრამების შედგენაც.

დაპროგრამების ენის დამუშავება გულისხმობს არა მარტო მის აღწერას, არამედ ამ ენაზე დაწერილი პროგრამის **მანქანურ ენაზე მთარგმნელი ტრანსლატორის დამუშავებასაც**. 70-იანი წლებიდან დაიწყო **Pascal**-ის აქტიური განვითარება, მან შეიძინა ახალი შესა-

ძლებლობები. ბუნებრივია, რომ მისთვის შეიქმნა სხვადასხვა სახის **ტრანსლატორები** და **დამუშავების გარემო**.

Pascal-ის ტრანსლატორები ჰქონდა სხვადასხვა ტიპის კომპიუტერის უმრავლესობას. ტრანსლატორების შექმნის სპეციალური მეთოდების არსებობამ გაამარტივა მათი დამუშავება და ხელი შეუწყო ენის ფართო გავრცელებას. ტრანსლატორებს შეეძლო მოეხდინათ **კოდის ოპტიმიზაცია**, რაც ეფექტური პროგრამების შექმნა გახადა შესაძლებელი. ეს გახდა სისტემური დაპროგრამების ენად **Pascal**-ის გამოყენების ერთ-ერთი მიზეზი.

80-იან წლებში მასიურად დაიწყო **კომპიუტერული მეცნიერების** (**ინფორმატიკს**) გავრცელება, სკოლებში გაჩნდა ახალი საგანი „**ინფორმატიკა**“. თავისი გავრცელების გამო **Pascal** გადაიქცა სასწავლო ენად, მისი გამოყენება დაიწყო როგორც სკოლებში, ასევე უმაღლეს სასწავლებლებში. ამის შედეგად შეიქმნა ამ **Pascal**-ის შესასწავლად განკუთვნილი უამრავი სასწავლო სახელმძღვანელო.

Pascal რომ დარჩენილიყო მარტივ ენად, **ნ. ვირტმა** მასში შემოიტანა გარკვეული შეზღუდვები. მაგალითად, **Pascal** გვასწავლის, თუ როგორ შეინახება მონაცემები კომპიუტერის მეხსიერებაში, მაგ-რამ მათთან მუშაობაზე თავად ენაში არსებობს მთელი რიგი შეზღუდვა.

რადგანაც თავის დროზე **Pascal** საკმაოდ ფართოდ გამოიყენებოდა პრაქტიკული ამოცანების გადასაწყვეტად, ამიტომ მის საფუძველზე ჩამოყალიბდა ენა **Object Pascal**, რომელიც დღეს გამოიყენება დამუშავების **Delphi** და **Lazarus** გარემოში.

3.1.2. ენა Paskal-ის ლექსიკონი

ჩვენ მიერ ყოველდღიურად გამოყენებული ნებისმიერ ბუნებრივ ენას აქვს საკუთარი ალფაბეტა და ლექსიკონი, ე. ი. დასაშვები ასოები ნაკრები და დასაშვები სიტყვების ნაკრები. **რაღაც მათი მსგავსი** აქვს ფორმალურ, მათ შორის დაპროგრამის, ენებსაც ოღონდ ნაცვლად ტერმინ „ასოებისა“ აქ იყენებენ ტერმინს „სიმბოლოებს“.

- ოპერაციების ნიშნები და ოპერაციათა შედგენილ აღნიშვნებში გამოყენებული ნიშნები: + - * / : = < >.

• შემზღუდველები: . , () [] {} : ; ' ".

დაპროგრამება ენა **Pascal**-ში გამოიყენება **სიმბოლოთა შემდეგი ნაკრებები**:

• ინგლისური ენის და სხვა ნაციონალური, მათ შორის, ქართული ენის, ასოები. ასოები შეიძლება იყოს როგორც ნუსხური, ისე მთავრული.

• არაბული ციფრები: 0 1 2 3 4 5 6 7 8 9.

• სპეციალური სიმბოლოები: $ @ # & amp ; ^ _ ~ %.

დაპროგრამებაში ერთ-ერთი საკვანძო ცნებათაგანია ცვლადის შესახებ ცნება. **ცვლადი** კომპიუტერის ფიზიკური მეხსიერების მცირე უბნის სახით შეგვიძლია წარმოვიდგინოთ. მეხსიერების ასეთ უჯრედებში პროგრამაში გამოყენებული მონაცემები შეინახება. მათთვის მიმართვისათვის გამოიყენება **ცვლადის სახელები**.

ცვლადის სახელები უნდა მოიგონოს დამპროგრამებელმა და ისინი არ არის ენის ლექსიკონის ნაწილი. პროგრამის დამმუშავებლის მიერ მოგონებულ აღნიშვნებს დაპროგრამებაში **იდენტიფიკატორები** ეწოდება. ამგვარად, ცვლადების სახელებიც იდენტიფიკატორებია. ოღონდ იდენტიფიკატორებს არა მარტო ცვლადების სახელებია. დამპროგრასმებელი სახელებს არქმევს მუდმივებსაც, პროგრამებსაც, მოდულებსაც, ფუნქციებსა და პროცედურებსაც. ამიტომ ყველა მათგანიც იდენტიფიკატორებს მიეკუთვნება.

იდენტიფიკატორები ჩაიწერება **Pascal**-ში დასაშვები სიმბოლოებით და ისინი უნდა აკმაყოფილებდეს შემდეგ მოთხოვნებს:

• სახელების შემადგენლობაში დასაშვებია მხოლოდ ინგლისური ასოების, ციფრებისა და ხაზგასმის სიმბოლოების გამოყენება;

• სახელი შეიძლება იწყებოდეს მხოლოდ ინგლისური ასოთი ან ხაზგასმის სიმბოლოთი და არავითარ შემთხვევაში ციფრით.

უნდა აღინიშნოს, რომ **Pascal**-ის იდენტიფიკატორებში ერთმანეთისაგან არ განსხვავდება ნუსხური და მთავრული ასოები. ამგვარად, ერთი და იმავე საქხელად ითვლება სახელები number, Number და NUMBER. **დაპროგრამების მრავალ სხვა ენებში ეს ასე არ არის.**

სალექსიკონო ნაკრები დაპროგრამების თითქმის ყველა ენაში შემდეგ სამ ჯგუფად იყოფა:

• დარეზერვებულ, ანუ საკვანძო სიტყვებად;

• წინასწარგანსაზღვრულ სახელებად;

• დამპროგრამებლის მიერ მოგონილ იდენტიფიკატორებად.

*ცხრ. 1 . **Pascal**-ის საკვანძო სიტყვები*

absolute - აბსოლუტური	**label** - ჭდე
and - ლოგიკური „და“	**library** - ბიბლიოთეკა
begin - ბლოკის დასაწყისი	**mod** - გაყოფა მოდულის მიხედვით
asm - ასემბლერი	**nil** - არარსებობა
array - მასივი	**not** - ლოგიკური „არა“
case - ვარიანტი	**or** - ლოგიკური „ან“
const - კონსტანტა	**of** - -დან
constructor - კონსტრუქტორი	**object** - ობიექტი
div - მთელრიცხვული გაყოფა	**packed** - ჩალაგებული
goto - გადასვლა -ზე	**procedure** - პროცედურა
do - შესრულდეს	**program** - პროგრამა
downTo - შემცირდეს -მდე	**record** - ჩაწერა
destructor - დესტრუქტორი	**repeat** - გამეორდეს
else –1) სხვაგვარად; 2) საწინააღმდეგო შემთხვევაში	**set** - სიმრავლე
	shl - თანრიგების მარცხნივ ძვრა
end - ბლოკის დასასრული	**shr** - თანრიგების მარჯვნივ ძვრა
exports - ექსპორტი	**string** - სტრიქონი
external - გარეგანი	**then** - მაშინ
file - ფაილი	**to** - 1) გაიზარდოს. 2)გაზრდა
for - 1)-თვის; 2)იმისთვის, რომ	**type** - ტიპი
forward - წინსწრებით	**unit** - მოდული
function - ფუნქცია	**until** - -მდე
if - თუ	**uses** - გამოყენებული იქნეს
implementation - რეალიზაცია	**var** - ცვლადი
in - ში, შედის -ში	**while** – 1) სანამ; 2) მაშინ როდესაც
inline - ძირითადი	**with** - -თან
interrupt - შეწყვეტა	**xor** – 2-ის მოდულით შეკრება
nterface - ინტერფეისი	

საკვანძო სიტყვებად დაპროგრამების ენებში მიიჩნევა სიტყვები, რომლებსაც აქვს ერთადერთი, ერთხელ და სამუდამოდ მისთვის მინიჭებული მნიშვნელობა. პროგრამაში დაუშვებელია იდენტიფიკატორების სახელების ჩანაწერები ემთხვეოდეს საკვანძო სიტყვების ჩანაწერებს. მაგალითად, ცვლადს დაუშვებელია დავარქვათ სახელი **begin**, რადგან ეს სიტყვა თავად **Pascal**-ში გამოიყენება, ე. ი. მისი საკვანძო სიტყვაა.

ცხრილ 1-ში მოყვანილია დაპროგრამების ენა **Pascal**-ში არსებული საკვანძო სიტყვები და მათი ქართული მნიშვნელობები.

წინასწარ განსაზღვრულ ანუ **სტანდარტულ სახელებს** ენაშიც აქვს წინასწარ მოცემული აზრი. დამპროგრამებელი ცვლადს თუ ასეთივე სახელს მიანიჭებს, მაშინ შეცდომები არ წარმოიშვება, მაგრამ უმჯობესია ასე არ მოიქცეს, რადგან ამ დროს სიტყვის ძველი, წინასწარ განსაზღვრული მნიშვნელობა შეიძლება დაიკარგოს, თითქოსდა წაიშალოს. მაგალითად, **Pascal**-ში წინასწარ-განსაზღვრული სახელებია Integer, Writeln და სხვ.

ათობითი რიცხვები ყოველთვის იწყება ციფრით, რომლის წინაც შეიძლება იდგეს რიცხვის ნიშნის + ან - ნიშანი.

ნამდვილი რიცხვები ორი ფორმით ჩაიწერება. კერძოდ, ფიქსირებული წერტილიანი ფორმატის ან მცურავი წერტილიანი ფორმატის სახით.

ფიქსირებული წერტილიანი ფორმატის დროს ცალსახადაა მითითებული ათობითი ათობითი წერტილის ადგილი, მაგლითად, **6.483, -27.48, +23.0.**

მცურავ წერტილიან ფორმატში გამოიყენება ათობითი ხარისხი, რომელიც აღინიშნება მთავრული ან ნუსხური **E** ასოთი, რომლის შემდეგაც იწერება ხარისხის მაჩვენებელი მთელი რიცხვი, მაგალითად, **7e12, 2.57e-5, 1,6E+5**.

ენა **Pascal**-ში მაქსიმალური დასაშვები რიცხვი უდრის **2 147 483 547**-ს (იგი შედის წინასწარ განსაზღვრულ მთელ **MaxLongInt** კონსტანტაში). ამ რიცხვის მისაღებად საკმარისია შევასრულოთ გამოსახულება:

```
writeln (maxlongint);
```

Pascal-ში გამოყენებული მთელი რიცხვები მოთავსებული უნდა იყოს **-2 147-483 648**-დან **+2 147 483 647**-მდე დიაპაზონში.

Pascal-ში წინასწარ განსაზღვრული მთელი კონსტანტა **MaxInt** თავის თავში შეიცავს **32 747** მნიშვნელობას:

```
writeln (maxint);
```

ენა **Pascal**-იდან მთარგმნელი გავრცელებული ტრანსლატორების უმრავლესობა საშუალებას გვაძლევს ოპერირება მოვახდინოთ ნამ-

დვილ რიცხვებზე, რომელთა სიდიდე **38** ხარისხამდეა, მაგრამ არსებობს ისეთი ტრანსლატორებიც, რომელთა მეშვეობითაც შეგვიძლია ოპერირება მოვახდინოთ ნამდვილ რიცხვებზე, რომელთა სიდიდე **67** ხარისხსაც აღწევს.

მთელი რიცხვები შეგვიძლია გამოვსახოთ თვლის არა მარტო ათობითი, არამედ თექვსმეტობითი სისტემების მეშვეობით. თექვსმეტობითი რიცხვის პირველი სიმბოლოა $ ნიშანი.

ენა **Pascal**-ში **ტექსტურ ლიტერალს** (**სტრიქონს**) უწოდებენ აპოსტროფებს შორის მდგარ ნებისმიერი დასაშვები სიმბოლოების მიმდევრობას (მაგალითად 'Hello World!'). სტრიქონის სიმბოლოდ თუ საჭიროა გამოვიყენოთ აპოსტროფი, მაშინ ერთმანეთის მიმდევრობით ორი აპოსტროფი უნდა ჩავწეროთ:

```
writeln ('Don''t do it');
```

სტრიქონი შეგვიძლია გამოვსახოთ მიმდევრობის სახით, რომელიც წარმოქმნილია სიმბოლოებისაგან #, რომელსაც მოყვება საჭირო სიმბოლოს ციფრული კოდი (მაგალითად, ჩანაწერი #72#73#33 ეკვივალენტურია 'HI!' სტრიქონის). სტრიქონულ მონაცემებში ნუსხური და მთავრული ასოები ერთმანეთისაგან განსხვავდება.

პრობელები ითვლება დამყოფებად. ნებისმიერი სახელის, რიცხვის, საკვანძო სიტყვის წინ უნდა იდგეს სულ მცირე ერთი დამყოფი მაინც, თუმცა მათი რაოდენობა შეიძლება რაგინდ უფრო მეტიც იყოს. **დაუშვებელია** სახელის, რიცხვის ან საკვანძო სიტყვის შიგნით ერთი სიმბოლო გამოვყოთ დანარჩენი სიმბოლოებისაგან.

3.1.3 . კომპილაცია

Pascal-ზე დაწერილი პროგრამის წინასწარი **კომპილირება** უნდა მოვახდინოთ. **კომპილირება** ნიშნავს **Pascal**-ის ენაზე დაწერილი **საწყისი პროგრამის** გარდაქმნას **ობიექტურ პროგრამად**, ანუ. კომპიუტერის ენაზე, ან მასთან დაახლოებულ ენაზე, დაწერილ პროგრამად. პროგრამის ამუშავების დროს გამოთვლები სრულდება ობიექტური პროგრამის და არა საწყისი პროგრამის მიხედვით.

კომპილაციის შემდეგ ვიღებთ პროგრამის **ორ ვერსიას**. ერთ-ერთი დაწერილია **Pascal**-ის ან მასთან ახლო მდებარე (მსგავს), ხოლო მეორე - კომპიუტერის ენაზე. **კომპიუტერის ენაზე დაწერილი**

პროგრამას თუ გამოვიტანთ კომპიუტერის ეკრანზე, მაშინ ჩვენ დავინახავთ გაუგებარ „სიტყვებსა“ და კაუჭებს.

Pascal-ის და სხვა **კომპილირებად ენაზე** დაწერილი პროგრამები გაცილებით უფრო სწრაფად სრულდება ვიდრე **ინტერპრეტირებად** (მაგალითად, ბეისიკის) **ენაზე** დაწერილი პროგრამები. ეს იმასთანაა დაკავშირებული, რომ კომპიუტერისა ენასთან დაახლოებულ ენაზე (ან უშუალოდ კომპიუტერის ბრძანებების გამოყენებით) დაწერილი **ობიექტური კოდი** სწრაფად სრულდება, მაშინ, როდესაც **ინტერპრეტირებად ენებზე** დაწერილი პროგრამის ინსტრუქციების შესასრულებლად საჭიროა საწყისი კოდი უშუალოდ კომპიუტერის ენაზე გადაითარგმნოს. კომპილირებული პროგრამის შესრულების დროს სიჩქარის ამაღლების საფასურია **დროის დანაკარგები**, რომლებიც აუცილებელია კომპილირებისათვის და მასთან დაკავშირებული მოუხერხებლობის დასაძლევად. თუმცა აქვე უნდა აღვნიშნოთ, რომ სისტემათა უმრავლესობაში ობიექტური პროგრამების შენახვაა შესაძლებელი, ამიტომ ისინი შესაძლებელია რეკომპილირების გარეშეც შესრულდეს.

თავდაპირველად საწყისი პროგრამა უნდა დაწეროს დამპროგრამებელმა, მოახდინოს მისი რედაქტირება და შემდეგ მისთვის სასურველი სახელით შეინახოს დისკზე.

შემდეგ ნაბიჯზე კომპიუტერის მეხსიერებაში უნდა ჩაიტვირთოს კომპილატორი და პროგრამის საწყისი ტექსტი. კომპიუტერი „წაკითხვის“ გზით გადააქცევს მას **ობიექტურ კოდად** და ასევე დისკზე შეინახავს.

ბოლო ნაბიჯზე ხდება პროგრამის შესრულება. **კომპიუტერულ** (ოპერატიულ) **მეხსიერებაში** სწორედ **სწორედ ობიექტური (შესრულებადი) კოდი ჩაიტვირთება.** შესაძლებელია შესრულებადი კოდი შეტანილი იქნეს (input) კლავიატურიდან და შედეგები გამოტანილი იქნეს (output) მონიტორის ეკრანზე. ეს არის მონაცემების შეტანა-გამოტანის საკმაოდ გავრცელებული სქემა, რომელიც **Pascal**-ში სტანდარტულ სქემად ითვლება; მაგრამ ეს არ არის შეტანა-გამოტანის ერთადერთი სქემა. შეიძლება ენა ჯერ კიდევ მაშინ იქნეს დამუშავებული, როდესაც ფაილები ინახებოდა მაგნიტურ ლენტაზე, შეტანა მოხდეს პერფობარათებით, ხოლო გამოტანა - საბეჭდი მოწყობილობით.

3.1.4. პროგრამის სტრუქტურა

პასკალის ენაზე დაწერილი პროგრამა შედგება სათაურისაგან და ბლოკისაგან.

პროგრამის სათაური

სათაურში მიეთითება პროგრამის სახელი და პარამეტრების სია. მისი ზოგადი სახე ასეთია:

```
program n (input, output, x, y, …);
```

აქ **n** არის პროგრამის სახელი, **input** - შეტანის ფაილი, **output** - გამოტანის ფაილი, **x, y** - პროგრამაში გამოყენებული გარე ფაილები.

სათაური შეიძლება არ შეიცავდეს პროგრამის სახელს, ან იყოს უპარამეტრო.

ბლოკი

პროგრამის ბლოკი შედგება ერთმანეთისადმი მკაცრად განსაზღვრული შემდეგი თანამიმდევრობით განთავსებული ექვსი განყოფილებისაგან:

1. ჭდეების (label-ების) განყოფილებისაგან;
2. მუდმივების (const-ების) განყოფილებისაგან;
3. ტიპების (type-ების) განყოფილებისაგან;
4. ცვლადების (var-ების) განყოფილებისაგან;
5. პროცედურებისა და ფუნქციების განყოფილებისაგან;
6. მოქმედებების (ოპერატორების) განყოფილებისაგან.

მოქმედებების (ოპერატორების) განყოფილება პროგრამაში ყოველთვის უნდა არსებობდეს, დანარჩენი განყოფილებები კი შეიძლება არსებობდეს ან არარსებობდეს.

ოთხი პირველი განყოფილებიდან ნებისმიერი მათგანი იწყება შესაბამისი საკვანძო (შესაბამისად, label, const, type, var) სიტყვით, რომელიც განყოფილების თავში ერთხელაა ჩაწერილი და შემდგომი ინფორმაციისაგან გამოყოფილია მხოლოდ პრობელით, ან სტრიქონის ბოლოთი, ან კომენტარით. მოკლედ განვიხილოთ თითოეული განყოფილება.

▌ჭდეების განყოფილება (label).

ნებისმიერი შესრულებადი ოპერატორი შეიძლება აღიჭურვოს **ჭდით** - არაუმეტეს 4 ციფრისაგან შემდგარი მთელი დადებითი რიცხვით. პროგრამაში არსებული ყველა ჭდე **label** განყოფილებაში უნდა აღიწეროს. აღნიშნული **განყოფილების ზოგადი სახე** ასეთია:

```
label  l1, l 2, l 3 ...;
```

აქ **11, 12, 13** არის ჭდეები. **მაგალითი**:

```
label 5, 10, 100;
```

ჭდე ოპერატორისაგან ორი წერტილითაა გამოყოფილი.

მაგალითი. დავუშვათ, რომ a := b გამოსახულებას აქვს ჭდე 20. მაშინ ეს ოპერატორი ასე გამოიყურება:

```
20: a := b;
```

▌კონსტანტების განყოფილება (**const**).

პროგრამაში თუ გამოიყენება საკმაო დიდი ჩანაწერის მქონე კონსტანტები (მაგალითად **8** ნიშნის მქონე **π** კონსტანტა), ან საცვლელი კონსტანტები (პროგრამის ვარიანტის მოსაცემად), მაშინ ასეთი კონსატანტები ჩვეულებრივ აღინიშნება გარკვეული სახელით და ისინი აღიწერება **const** განყოფილებაში, ხოლო თავად პროგრამაში გამოიყენება კონსტანტას მხოლოდ სახელი. ეს ამაღლებს პროგრამის თვალსაჩინოებას, მოსახერხებელია ასეთი პროგრამის გამართვა და მასში ცვლილებების შეტანა. კონსტანტების განყოფილების **ზოგადი სახე** ასეთია:

```
const a1 = c1; a2 = c2, …
```

აქ **a1, a2** არის კონსტანტათა სახელები, - **c1, c2** - კონსტანტათა მნიშვნელობები. მაგალითი: const pi = 3.14’ c = 5.7483;

```
const pi = 3.14; c = 5.7483;
```

▌ტიპების განყოფილება (type).

სტანდარტული ტიპისაგან განსხვავებული ტიპის პროგრამაში შეტანისას იგი უნდა აღიწეროს განყოფილებაში type:

```
type t1 = ტიპის სახე;
     t2 = ტიპის სახე;
```

სადაც **t1** და **t2** არის შესატანი რიცხვების იდენტიფიკატორები.

შემდეგ ტიპი გამოიყენება ცვლადების გამოცხადების დროს. არასტანდარტული ტიპების გამოყენების მაგალითი:

```
const
            len=40;
type
            year=1930---2010
             names=string[len];
var
             empl: names;
             y: year;
```

Pascal-ის ენაში ტიპების აღწერის განყოფილებას დიდი მნიშვნელობა აქვს. პროგრამაში ტიპების გამოუყენებლობის დროს შეიძლება წარმოიშვას ტიპების შეუთავსებლობა მაშინაც კი, როდესაც ისინი ერთნაირადაა აღწერილი.

▌ ცვლადების განყოფილება (var).

დავუშვათ, რომ პროგრამაში გვხვდება **v11, v12**,...; ყველა მათგანი ასე უნდა აღიწეროს:

```
var v11, v12,...: type1;
    v21, v22,…: type2; …
```

აქ **v11, v12,…** არის ცვლადების სახელები; **v11, v12,…** ცვლადების ტიპია type1, ხოლო v21, v22,… ცვლადების ტიპი კი type2.

მაგალითი: var k, i, j: integer; a, b: real;

თითოეული ცვლადი პროგრამაში მათი გამოყენებამდე უნდა აღიწეროს და იგი უნდა მიეკუთვნებოდეს ერთ რომელიმე ტიპს. განყოფილებათა (**const, type, var…**) დასახელებები მხოლოდ ერთხელ მიეთითება. მაგალითი:

```
var  a: real;
       b: real;
```

ამგვარად, **var** განყოფილებაში შემოიტანება თითოეული ცვლადის სახელი და მიეთითება თუ რა ტიპისაა იგი. ცვლადის ტიპი შეიძლება ორი ხერხით განვსაზღვროთ: მივუთითოთ ტიპის სახელი (მაგალითად, real, color და ა. შ), ან აღვწეროთ თავად ტიპი, მაგალითად: array[1..16] of char.

▌**პროცედურებისა და ფუნქციების განყოფილება**.

ამ განყოფილებაში არსებობს სამომხმარებლო პროცედურებისა და ფუნქციების სათაურები და სხეულები.

▌**მოქმედებების (ოპერატორების) განყოფილება.**

პროგრამის ეს ნაწილი იწყება სიტყვით **begin** და მთავრდება სიტყვით **end**, რომლის შემდეგ წერტილი უნდა იყოს დასმული. მოქმედებების განყოფილება არის ოპერატორებისაგან შემდგარი პროგრამის შესრულებადი ნაწილი.

3.1.5. პროგრამის პუნქტუაცია

Pascal-ის ენაზე დაწერილ პროგრამებში დაცულია შემდეგი პუნქტუაცია:

- სათაური მთავრდება წერტილ-მძიმეთი.
- ნებისმიერ განცხადებაში თითოეული სია მთავრდება წერტილ-მძიმეთი.
- ოპერატორები ერთმანეთისაგან წერტილ-მძიმეებითაა გამოყოფილი.

სიტყვები **begin** და **end** ოპერატორები არ არის - ისინი პუნქტუაციის ნიშნებია. სიტყვა **begin** ასრულებს მარცხენა, ხოლო **end** - მარჯვენა ფრჩხილის როლს. რადგანაც ისინი თავადაა პუნქტუაციის ნიშნები, ამიტომ **begin**-ის შემდეგ მძიმის, ხოლო **end**-ის შემდეგ წერტილ-მძიმის დასმა არაა აუცილებელი. პასკალის ენაზე დაწერილ პროგრამაში სიტყვები **begin** და **end** უპირატესად **შედგენილი ოპერატორების** წარმოსაქმნელად გამოიყენება. **შედგენილი ოპერატორი** შეგვიძლია პროგრამის ნებისმიერ ადგილზე გამოვიყენოთ, სადაც შეგვეძლო მარტივი ოპერატორის გამოყენება. შედგენილი ოპერატორის მაგალითია:

```
begin
   t := a;
   a := b;
   b := t
end;
```

ზოგადად სიტყვები სხვა ოპერატორებშიც პუნქტუაციის როლში გამოდის.

```
if ab > bd thn
else
   write (‘no’);
```

სიტყვები **if, then, else** პუნქტუაციის ფუნქციებს ოპერატორის შიგნით ასრულებს.

ოპერატორები პუნქტუაციის ნიშნებითაა დაყოფილი, ამიტომ კომპილატორის თვალსაზრისით მნიშვნელობა არა აქვს ფურცლის თუ რომელ ადგილზე იქნება განთავსებული პროგრამა, საკმარისია მხოლოდ შემდეგი ორი წესი იყოს დაცული:

1. სიტყვები ერთმანეთისაგან განცალკევებით უნდა დავწეროთ;

2. სიტყვა არ უნდა გავწყვიტოთ პრობელებით ან ახალ სტრიქონზე გადატანით.

ყოველივე დანარჩენი, კერძოდ, თუ როგორ იქნება პროგრამა ფურცელზე განთავსებული, კომპილატორისათვის სულერთია, მაგრამ ეს სულერთი არ არის დამპროგრამებლისათვის. სტრიქონის დასაწყისში ტექსტის (ნაწერის) მარჯვნივ შეწევების შესახებ ძალიან განსხვავებული შეხედულებები არსებობს, მაგრამ ყველა იმაზე თანხმდება, რომ აღნიშნული შეწევების დანიშნულებაა პროგრამის სტრუქტურა მაქსი-მალურად თვალსაჩინო გახადოს.

program, const, var, begin, end (იხ. ცხრ.1) სიტყვებსა და სხვა მრავალ მსგავს სიტყვებს **დარეზერვებული სიტყვები** ეწოდება. არ შეიძლება დარეზერვებული სიტყვების დაგრძელება (მაგალითად, **constant** იქნება შეცდომა) და შემოკლება (მაგ. შეცდომაა **prog**).

Pascal-ის ენაზე დაწერილ პროგრამულ კოდში შეიძლება გამოვიყენოთ როგორც მთავრული, ისე ნუსხური ასოები. უფ

რო მეტიც, დასაშვებია ასეთი ასოების მონაცვლეობაც.

3.1.6. Paskal-ის ოპერატორები

Pascal-ის ენაზე **ოპერატორებად** იგულისხმება მხოლოდ **მოქმედებების აღწერა**. ოპერატორები ერთმანეთისაგან მხოლოდ წერტილ-მძიმეებითაა გაყოფილი. ოპერატორი თუ დგას **end, until** ან **else** სიტყვების წინ, მაშინ მის შემდეგ წერტილ-მძიმე არ უნდა დავსვათ. რამდენადმე დაწვრილებით განვიხილოთ მინიჭების და შედგენილი ოპერატორები.

მინიჭების ოპერატორის ზოგადი სახე ასეთია:

```
v := a;
```

აქ **v** არის ცვლადი, **a** - გამოსახულება, ხოლო **: =** - მინიჭების ოპერაცია. გამოსახულება **a** შეიძლება შეიცავდეს კონსტანტებს, ცვლადებს, ფუნქციათა სათაურებს, ოპერაციის ნიშნებს და ფრჩხილებს.

მაგალითი: f := 3 * C + 2 * sin(x);

გამოსახულების სახე ცალსახად განსაზღვრავს მისი გამოთვლის თანამიმდევრობას: მოქმედებები მარცხნიდან მარჯვნივ სრულდება შემდეგი პრიორიტეტულობის დაცვით:

1. not;
2. *, /, div, mod, and;
3. +, -, or;
4. =, <, >, <>, <=, >=, in.

ფრჩხილებში არსებული მოქმედებები ფრჩხილში არსებულ მოქმედებებზე ადრე სრულდება.

მინიჭება დასაშვებია ნებისმიერი ტიპის ცვლადებისათვის, გარდა ფაილის ტიპის ცვლადისა.

v := a ოპერაციაში ერთი და იმავე ტიპის უნდა იყოს ცვლადი და გამოსახულება, ხოლო ინტერვალური ტიპის შემთხვევაში ერთსა და იმავე ტიპის უნდა იყოს მნიშვნელობათა ქვესიმრავლეები.

აქვე უნდა შევნიშნოთ, რომ **real** ტიპის ცვლადს შეგვიძლია მივანიჭოთ **integer** ტიპის გამოსახულება, მაგრამ დაუშვებელია integer ტიპის ცვლადისათვის **real** ტიპის გამოსახულების მინიჭება.

შედგენილი ოპერატორი. პასკალის ენაში შესაძლებელია რამდენიმე ოპერატორის ერთ შედგენილ ოპერატორად გაერთიანება.

იგი იწყება საკვანძო **begin** სიტყვით და მთავრდება **end** სიტყვით. მათ შორის უნდა მოვათავსოთ შემადგენელი ოპერატორები, რომლებიც საკუთარი თანამიმდევრობის შესაბამისად შესრულდება. **end**-ის შემდეგ უნდა დავსვათ წერტილ-მძიმე, ხოლო **begin**-ის შემდეგ - მხოლოდ პრობელები (ან კომენტარები). მაგალითად:

```
begin
        i  := 5;
        k := i / 7
end;
```

სიტყვები **begin** და end ასრულებს ოპერატორული ფრჩხილების როლს. თავად პროგრამის სხეულსაც აქვს შედგენილი ოპერატორის სახე. პროგრამის ბოლო end-ის შემდეგ უნდა დავსვათ წერტილი. დაუშვებელია შედგენილი ოპერატორის გარედან მართვა გადავცეთ ამ შედგენილი ოპერატორის შიგნით.

3.1.7. Paskal-ის გამოსახულებები

მინიჭების ოპერატორებში შეგვიძლია გამოვიყენოთ არითმეტიკული გამოსახულებები. მაგალითად, განვიხილოთ მინიჭების შემდეგი ორი **მაგალითი**:

```
num := (d + n) / 15;
sq := trunk(num) + 2;
```

ფრჩხილები უზრუნველყოფს გამოთვლებისათვის აუცილებელი თანამიმდევრობის დაცვას. პირველ მაგალითში თუ ფრჩხილებს გამოვტოვებთ, მაშინ მივიღებთ:

```
num := d + n / 15;
```

ამ შემთხვევაში ჯერ შესრულდება გაყოფა, რომელსაც აქვს უფრო მაღალი პრიორიტეტი. არითმეტიკულ გამოსახულებებ-ში მაღალი პრიორიტი აქვს გამრავლებასა (*) და გაყოფას (/), ხოლო დაბალი პრიორიტეტი - შეკრებასა და გამოკლებას.

მინიჭების მეორე მოყვანილ მაგალითში ხდება მთელი რიცხვის მინიჭება. **trunc** ფუნქცია გვაძლევს მთელ შედეგს, ხოლო რიცხვი 2 ჩაწერილია ათობითი წერტილის გარეშე; ამგვარად, ორივე შესაკრები ჯამში გვაძლევს მთელ მნიშვნელობას. ზოგადად, როდესაც გამოსახულების ყველა წევრი მთელია, მაშინ თავად გამოსახულებაც იღებს მთელ მნიშვნელობას.

არსებობს ზემოთ ჩამოყალიბებულ წესისაგან ერთი გამონაკლისი: გაყოფისას (/ ნიშნის გამოყენებით) ყოველთვის მიიღება ნამდვილი შედეგი:

```
8.5 / 2 = 4.25
4 / 2 = 4.0
```

მთლიანად გაყოფა (განაყოფისა და ნაშთის პოვნა) შეიძლება შევასრულოთ **div** და **mod** ოპერაციების დახმარებით.

გამოსახულება შეიძლება საკუთარ თავში შეიძლება მოიცავდეს მთელ და ნამდვილ წევრებს. თუნდაც ერთი ნამდვილი წევრის ან / ნიშნის არსებობა იმას იწვევს, რომ განაყოფი იქნება ნამდვილი. **trunc** და **round** ფუნქციები შეგვიძლია გამოვიყენოთ ნამდვილი რიცხვის გარდასაქმნელად მთელ რიცხვად.

sqr ახდენს არგუმენტის (რომელიც ჩაწერილია ფრჩხილებში) კვადრატში ახარისხებას. პასკალში არ არსებობს ნებისმიერ ხარისხში აყვანის ოპერატორი. **ახარისხება აქ ხდება ლოგარითმების გამოყენებით.** მათემატიკური a^x გამოსახულების ნაცვლად Pascal-ში შეგვიძლია დავწეროთ exp(ln(a) * x).

<, >= -ს მსგავსი ნიშნებიც თამაშობს ოპერაციების როლს. ასეთი ოპერაციების შემცველი გამოსახულებები იღებს ლოგიკურ მნიშვნელობებს და ამიტომ მათ ლოგიკურ გამოსახულებებს უწოდებენ. ლოგიკური გამოსახულებების შემადგენლობაში შეიძლება შედიოდეს ლოგიკური **not** (არა), **and** (და), **or** (ან) ოპერაციები. ასეთ ლოგიკურ გამოსახულებებს **რთული ლოგიკური გამოსახულებები** ეწოდება.

3.1.8. შეტანა-გამოტანის ოპერატორები

კომპიუტერული პროგრამები დაამუშავებს სხვადასხვა მონაცემს და შემდეგ შეცვლილი ფორმით მათ ან ნაცვლად გამოიტანს სხვა მონაცემებს.

აღნიშნულიდან გამომდინარე დაპროგრამების ნებისმიერ ენას უნდა ჰქონდეს მონაცემების როგორც შეტანის, ისე გამოტანისთვის საჭირო ინსტრუმენტები. პასკალში მონაცემები შეიტანება **read()** და **readln()** ოპერატორების, ხოლო გამოიტანება - **write() და writeln()** პროცედურების მეშვეობით. **ln დაბოლოებიან პროცედურებს** საკუთარი ამოცანის შესრულების შემდეგ **მაჩვენებელი გადაყავს ახალ სტრიქონზე**.

მონაცემები პროგრამაში, საიდან ან რისი დახმარებით შეიტანება? ჩვეულებრივ, ეს ხდება კლავიატურის მეშვეობით, ან გარკვეული ფაილებიდან.

მონაცემები შეიძლება გატანილი იქნეს მონიტორის ეკრანზე, ფაილში, პრინტერზე და ა. შ.

შეტანის სტანდარტული მოწყობილობაა კლავიატურა, ხოლო გამოტანის კლასიკური მოწყობილობა - მონიტორი. ტერმინი „სტანდარტული“ იმას ნიშნავს, რომ თუ რაიმე დამატებითი არ არის მითითებული, მაშინ მონაცემები შეიტანება კლავიატურიდან და გამოიტანება მონიტორზე. ამას უწოდებენ „**უტყვი თანხმობით მუშაობის რეჟიმს**“. კლავიატურისა და მონიტორის ერთობლიობას **კონსოლს** უწოდებენ. ამგვარად, **მონაცემების შეტანა-გამოტანის სტანდარტული მოწყობილობაა კონსოლი**.

ეკრანზე მონაცემების გამოტანა. ფორმატიზებული გამოტანა

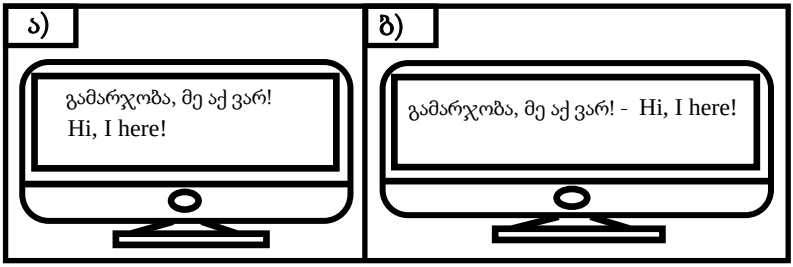
დაპროგრამების ენა **Pascal**-ში ეკრანზე ან ფაილში მონაცემები გამოიტანება **`write()`** და **`writeln()`** პროცედურების მეშვეობით. ჩვენ გან-ვიხილავთ მხოლოდ ეკრანზე გამოტანას.

დავუშვათ, რომ საჭიროა ეკრანზე გამოისახოს ორი ფრაზა, მაგალითად, „გამარჯობა, მე აქ ვარ!“ და „Hi, I here!”. თუ გვინდა, რომ თითოეული მათგანი ახალი სტრიქონიდან იწყებოდეს, მაშინ უნდა

გამოვიყენოთ ოპერატორი **writeln()** და პროგრამას ექნება **ლისტინგ 1**-ზე არსებული, მასზე მოყვანილი პროგრამის დამუშავების შემდეგ მონიტორის ეკრანზე გამოტანილი მონაცემები **ნახ,1,ა**-ზეა მოყვანილი.

```
begin
        writele (‘ გამარჯობა, მე აქ ვარ! ‘ );
        writele (‘ Hi, I here! ‘)
end;
```

ლისტინგი 1.



***ნახ. 1** მონაცემების გამოტანა* **writeln** (ა) *და* **write** (ბ) *ოპერატორებით.*

გამოსატანი ფრაზების ახალი სტრიქონებიდან დაწყება თუ საჭირო არ არის, მაშინ უნდა გამოვიყენოთ ოპერატორი ოპერატორი **write()** და პროგრამას ექნება **ლისტინგ 2**-ზე მოყვანილი სახე.

```
begin
        write (‘ გამარჯობა, მე აქ ვარ! ‘ );
        write (‘ Hi, I here! ‘)
end;
```

ლისტინგი 2

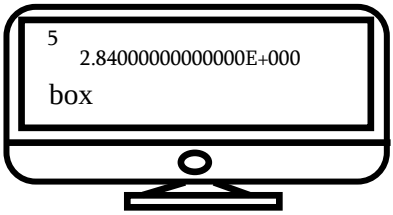
ამ პროგრამის დამუშავების შემდეგ მონიტორის ეკრანზე გამოტანილი მონაცემები **ნახ.1,ბ-ზე** არის მოყვანილი.

write() ხშირად მაშინ გამოიყენება, როდესაც შეტყობინება გამოტანილი უნდა იყოს ეკრანზე და შემდეგ გვიწევს კურსორის გადაუყვანლად მონაცემების დალოდება. მაგალითად, როდესაც ეკრანზე გამოგვყავს ბრძანება „შეიტანე მონაცემები“ და ახალ სტრიქონზე კურსორის გადაუყვანლად ველოდებით შეტანას.

განვიხილოთ კიდევ ერთი მაგალითი. დავუშვათ, რომ კომპიუტერის მეხსიერებაში შენახულია გამოსატანი მონაცემები. პროგრამიდან მათ მივმართავთ **num, fl** და **st** ცვლადებით დახმარებით. ეკრანზე მათი მნიშვნელობები შეგვიძლია სხვადასხვაგვარად გამოვიტანოთ. კერძოდ, **ლისტინგ 3**-ზე მოყვანილ პროგრამ გამოყენების დროს ეკრანზე გამოტანილი მონაცემები **ნახ. 2-ზეა** ნაჩვენები.

```
var
      num:  integer;
      fl: real;
      st: string
begin
      num := 5;
      fl := 2.84;
      st := 'box';
      writeln (num );
      writeln (fl );
      writeln (' box ');
readln
end;
```

ლისტინგი 3



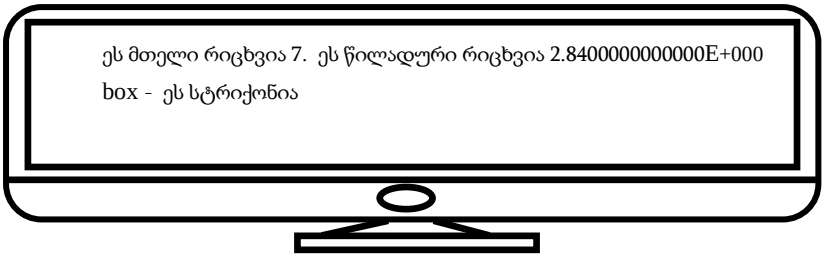
ნახ. 2. მონაცემების გამოტანა **num, fl** და **st** ცვლადებით

განვიხილოთ **ლისტინგ 4**-ზე მოყვანილი პროგრამა

```
var
      num:  integer;
      fl: real;
      st: string
begin
      num := 5;
      fl := 2.84;
      st := 'box';
      writeln ('ეს მთელი რიცხვია', num,'ეს წილადური რიცხვია', fl);
      write (st, '- ეს სტრიქონია')
readln
end;
```

ლისტინგი 4

ამ პროგრამის დამუშავების შემდეგ მონიტორის ეკრანი მიიღებს **ნახ.3-ზე** ნაჩვენებ სახეს. ამ ნახაზიდან ჩანს ჩანს, რომ გამოტანის პროცედურები (როგორც **write()**, ასევე **writeln()**) საშუალებას გვაძლევს გამოსატანი ინფორმაცია სხვადასხვა (სტრიქონი-კონსტანტა და ცვლადები) კომპონენტებისაგან ავაწყოთ.



ნახ. 3 *სხვადასხვა კომპონენტებით გამოსატანი ინფორმაციის აწყობა*

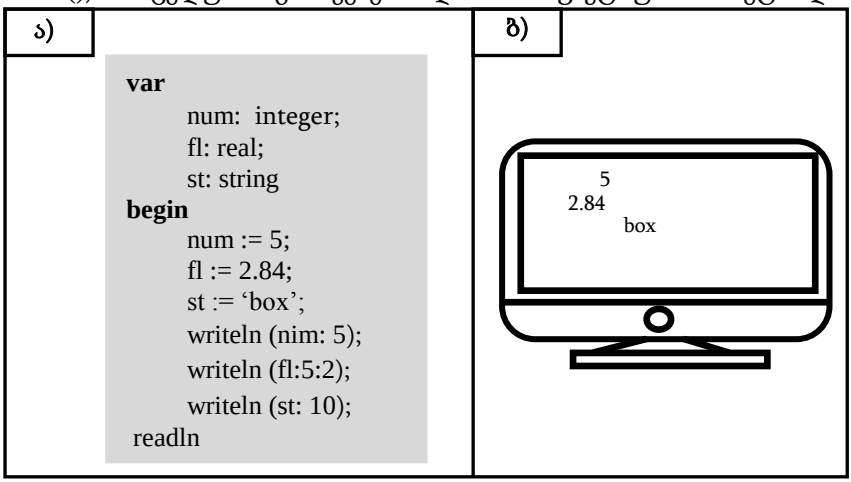
ახლა გავეცნოთ მონაცემების სპეციალიზებულ გამოტანას, რომელიც ცნობილია **მონაცემების ფორმატირებული გამოტანის** სახელწოდებით. ამისათვის მონაცემების გამოტანისათვის უნდა გამოვიყენოთ **ნახ. 4,ა** ნახაზზე მოყვანილი. პროგრამა, მისი რეალიზების შედეგად მონიტორის ეკრანი მიიღებს **ნახ.4,ბ** ნახაზზე ნაჩვენებ სახეს. მასზეა ნაჩვენები მონაცემების გამოტანის ფორმატიზებული სახე. ფორმატიზებული გამოტანის დროს პროგრამაში გამოსატანი მნიშვნელობისათვის მითითებულია ველის სიგანე (ნიშნების ადგილთა რაოდენობა). ნამდვილი (წილადური) რიცხვის გამოტანის დროს მეორე რიცხვთან დასმული ორი წერტილის იქით მითითებულია მძიმის შემდეგ არსებული ნიშნების რაოდენობა.

ნამდვილი რიცხვებისათვის ფორმატირებას თუ არ განვახორციელებთ, მაშინ ისინი ისე გამოისახება, როგორც ეს კონკრეტული კომპიუტერისათვისაა დამახასიათებელი. თუ მივუთითებთ მხოლოდ ნიშნების ადგილების რაოდენობას, მაგრამ არ დავაფიქსირებთ წილადურ ნაწილს, მაშინ რიცხვები ექსპონენციალირი ფორმით გამოიტანება.

მონაცემების შეტანა

დაპროგრამების ენა პასკალში მონაცემები შეიტანება read() და readln() პროცედურებით. მონაცემები შეიტანება კლავიატურიდან ან ფაილიდან. ჩვენ მონაცემების მხოლოდ კლავიატურიდან შეტანას განვიხილავთ.

მონაცემები შეტანის შემდეგ თავსდება მეხსერების უჯრედებში, რომლებთანაც შეღწევა ცვლადების მექანიზმის დახმარებითაა უზრუნველყოფილი. ამიტომ პასკალზე დაწერილ პროგრამაში read() (ან readln()) პროცედურის გამოყენების დროს მას ფაქტიურ პარამეტრად



ნახ.4. *მონაცემების ფორმატიზებული გამოტანა*

(არგუმენტად) გადაეცემა შესატან მონაცემებთან დაკავშირებული ცვლადის სახელი. შემდეგ ეს მონაცემები შეიძლება გამოიყენოს პროგრამამ ან იგი უბრალოდ გამოტანილი იქნეს ეკრანზე **(ნახ. 5,ა,ბ**).

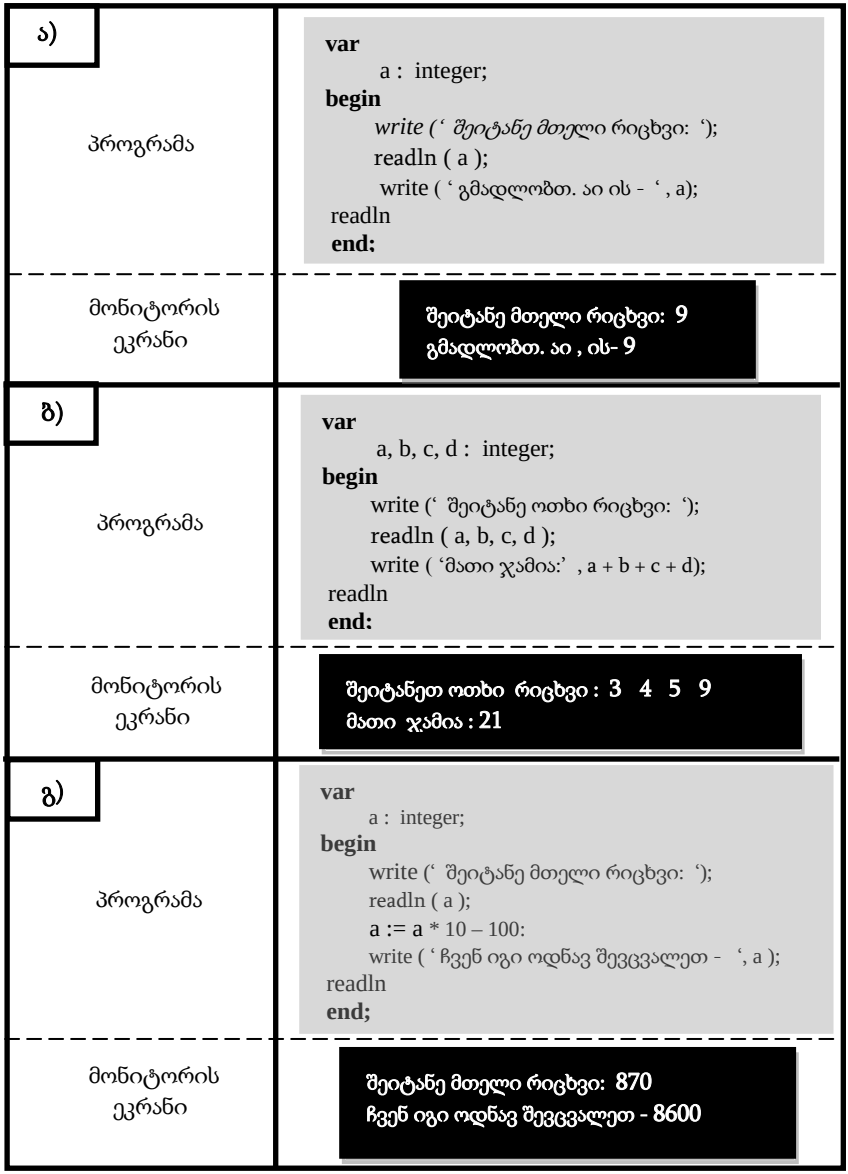
შეტანის პროცედურაში შეგვიძლია გადავცეთ არა ერთი პარამეტრი, არამედ პარამეტრების სიმრავლე (ნ**ახ. 5,გ**).

მონაცემები read() ოპერატით შეტანისაგან განსხვავებით readln() ოპერატორის რამდენჯერმე გამოყენების დროს მისი თითოეული გამოყენების შემდეგ თუ თითი არ დავჭირეთ Enter ღილაკს, მომდევნო readln() ოპერატორი არ ამოქმედდება.

3.1.9. ცვლადები და კონსტანტები

ცვლადები

ნებისმიერი პროგრამა ახდენს მონაცემების (ინფორმაციის, ობიექტის) დამუშავებას. პროგრამის მიერ გამოსაყენებელი მონაცემები შენახულია (უფრო ხშირად, ოპერატიულ) მესხიერებაში. პროგრამამ უნ-



ნახ. 5. *მონაცემების გამოტანა დაპროგრამების ენა პასკალში*

და იცოდეს მათი ადგილმდებარეობა, მათ მიერ დაკავებული მეხსიერების მოცულობა, მათი ინტერპრეტირების ფორმა (მაგალითად, ისინი წარმოდგენილი უნდა იყოს რიცხვების თუ სტრიქონის სახით). მეხსიერების უბნებთან პროგრამის შესაღწევად არსებობს ცვლადების მექანიზმი.

ცვლადები პროგრამის დასაწყისშია აღწერილი და ამით ჩვენთვის ნათელია, თუ როგორ მონაცემებთან იმუშავებს პროგრამა და მესიერების

თუ რა მოცულობას დაიკავებს ისინი. ეს იმას არ ნიშნავს, რომ მეხსიერების ამ უჯრებში მოთავსდება კონკრეტული მნიშვნელობები (მონაცემები). მეხსიერების დარეზერვების მომენტში მათში ნებისმიერი რამ შეიძლება იყოს მოთავსებული.

პროგრამის შესრულების პროცესში მეხსიერების უჯრედებში შეიძლება მოთავსდეს კონკრეტული მნიშვნელობები, ამოღებული იქნეს მათგან, დამუშავდეს და ხელახლა ჩაიწეროს მათში. ჩვენ მათ შეგვიძლია პროგრამის მეშვეობით მივმართოთ იმ სახელებით, რომლებიც პროგრამის დასაწყისში აღწერის დროს მიენიჭათ.

ცვლადების სახელებად შეგვიძლია გამოვიყენოთ ინგლისური ასოებისა და ციფრების თითქმის ნებისმიერი უპრობელი ნაკრები. ცვლადის სახელი არ უნდა დაემთხვეს იმ დაპროგრამების ენაში ბრძანების სახელად გამოყენებულ რომელიმე სიტყვას. ცვლადი სახელი არ უნდა იწყებოდეს ციფრით ან სპეციალური სიმბოლოთი. ცვლადების ადვილი აღქმადობისათვის საჭიროა დავიცვათ რამდენიმე წესი. პროგრამა თუ მარტივი მაგალითი არ არის, მაშინ სახელებად საჭიროა გამოვიყენოთ აზრიანი სიტყვები ან მათი შემოკლებები. სასურველია, ცვლადების სახელები ძალიან გრძელი არ იყოს.

პასკალში ნუსხური და მთავრული ასოები ერთმანეთისაგან არ განსხვავდება.

ცვლადების აღწერისას უნდა სახელებთან ერთად უნდა მივუთითოთ მათი ტიპები. ცვლადის ტიპი გვატყობინებს, თუ რა მოცულობის მეხსიერება უნდა გამოვყოთ და რა სახის მონაცემები შეგვიძლია შევინახოთ მასში. მასში კი ყოველთვის ინახება თვლის ორობითი სისტემაში წარმოდგენილი ნებისმიერი სახის მონაცემი. კერძოდ, მთელი ან ნამდვილი რიცხვი, სიმბოლო, სტრიქონი, მასივი, ჩანაწერი და ა.შ. მაშასადამე ცვლადის ტიპით განისაზღვრება, თუ რა სახით შეგვიძლია შევინახოთ ის, რაც დაკავშირებულია აღწერილ ცვლადთან.

განვიხილოთ **ლისტინგ 5-ზე** მოყვანილი პროგრამა.

```
var
   int 1, int2: integer;
   r: real;
   ch: char;
begin
    write(‘ ტიპი integer”: ‘);
    readlin(int1);
    write(‘ ტიპი real: ‘);
    readlin(r);
    write(ტიპი char: ‘);
    readlin(ch);

    int2:=int mod 10;
    int1:=int div 10;
    r:=r + 0.5;
    ch:=chr(ord(ch)+1);
    writeln(int1:5,int2:3,r:7:2,ch:3);
end
```

ლისტინგი 5

მოცემული პროგრამით სხვადასხვა ტიპის სამი ცვლადის შეიტანება და მათზე მათზე სრულდება გარკვეული ოპერაციები. **integer**-ის ტიპის ცვლადს შეგვიძლია მივანიჭოთ -32768-დან +32767-მდე დიაპაზონში არსებული მთელი რიცხვები. მათთვის პასკალის ენაში გამოიყოფა 2 ბიტი (16 ბიტი). ეს ნიშნავს, რომ 2^{16} = 65536 რაოდენობის მნიშვნელობის (დადებითი, უარყოფითი რიცხვები და ნული) შენახვა შეგვიძლია. ამ დიაპაზონში int1 და int2 ცვლადებს შეუძლია ნებისმიერი მნიშვნელობა მიიღოს. ამ ტიპის დროს ცვლადი მნიშვნელობის შენახ-ვის მცდელობისას წარმოიშობა შეცდომა.

პროგრამის მუშაობის მაგალითი **ლისტინგ 6**-ზეა მოყვანილი.

```
Type integer: 32
Type real: 5.34
Tupe char: A
    3  2  5.84  B
```

ლისტინგი 6

მაშასადამე, ცვლადები დაკავშირებულია მეხსიერების უბნებთან, რომელთა შიგთავსები პროგრამის შესრულების დროს გარკვეულ ფარგლებში შეიძლება შეიცვალოს.

პასკალის ადრეული ვერსიის კომპილატორებში შეზღუდული იყო ცვლადის სახელის სიგრძე. სახელში გაითვალისწინებოდა მხოლოდ პირველი რვა ასო. ამიტომ ისეთ სახელებს, როგორიცაა **variable1** და **variable2** კომპილატორი ერთსა და იმავე სიტყვად აღიქვამდა, რადგან მათი პირველი რვა სიმბოლო ერთმანეთს ემთხვევა.

Pascal-ის თანამედროვე ვერსიებში ასეთი შეზღუდვა არ არსებობს და მათი კომპოლატორები ცვლადებისა და სხვა იდენტიფიკატორების სახელებში რვაზე მეტ სიმბოლოს ითვალისწინებს.

კონსტანტები

ერთი შეხედვით პრობლემა არ უნდა იყოს პროგრამაში რომელიმე კონკრეტული რიცხვის მუდმივი გამოყენება. შეგვიძლია აღვწეროთ ცვლადი, მივანიჭოთ მას ამ რიცხვის მნიშვნელობა და ეს შემდეგ პროგრამაში არ შევცვალოთ. სამწუხაროდ ასეთი მიდგომა ყოველთვის არაა მოსახერხებელი (შეიძლება შემთხვევით შეიცვალოს). ამიტომ დაპროგრამების ენებში მონაცემების შესანახად ცვლადების გარდა არსებობს კონსტანტები.

კონსტანას მთავარი უპირატესობაა ის, რომ იგი პროგრამის დასაწყისში აღიწერება, იქვე მიენიჭება მნიშვნელობა და პროგრამის შესრულებისას იგი არ იცვლება. კოდის გამართვის დროს დამპროგრამებელი თუ გადაწყვეტს კონსტანტას შეუცვალოს მნიშვნელობა, მან აღწერაში უნდა ჩაწეროს სასურველი მნიშვნელობა, ხოლო კოდის რედაქტირება მას არ დასჭირდება. დამპროგრამებელი სხვადასხვა მნიშვნელობების ხშირად გამოიყენებას გადაწყვეტს, მაშინ მან ისინი უნდა აღწეროს ცვლადების განყოფილებამდე განსათავსებელ კონსტანტების განყოფილებაში (*ლისტინგი 7*).

```
const
        კონსტ1 = მნიშვნელობა;
        კონსტ2 = მნიშვნელობა;
```

ლისტინგი 7

კონსტანტების მნიშვნელობებია შეიძლება გამოვიყენოთ პასკალში გამოყენებული მრავალი სხვადასხვა ტიპის მონაცემი. განვიხილოთ **ლისტინგ 8**-ზე მოყვანილი პროგრამა:

მასში გამოიყენება მნიშვნელობა **10**-ის მქონე **n** კონსტანტა. პროგრამა **1**-დან **10**-მდე დიაპაზონში ათი რიცხვის ჯამს გამოითვლის. ჩვენ თუ მოვინდომებთ **20**-მდე რიცხვის ჯამი გამოვითვალოთ, საკმარისია პროგრამის დასაწყისში შევცვალოთ კონსტანტას მნიშვნელობა. კონსტანტა რომ არ გამოგვეყენებინა, მაშინ მოგვიხდებოდა გადაგვეხედა მთელი პროგრამისათვის და მნიშვნელობა **10** შეგვეცვალა მნიშვნელობა **20**-ით. დიდი პროგრამის დროს ადვილად შეგვეძლო დაგვეშვა შეცდომა: ვერ გვეპოვნა საჭირო ცვლადი ან შეგვეცვალა სულ სხვა მნიშვნელობა.

```
const n=10;
var
   i: byte;
   sum: word;
begin
   sum: 0;
   for i:=1 to n do
      sum:=sim+i;

   writeli('Sum of',n,'numbers= ', sum);
end
```

ლისტინგი 8

ტიპიზებული კონსტანტები

პასკალის ენაში **ჩვეულებრივი კონსტანტების** გარდა გამოიყენება ე.წ. **ტიპიზებული კონსტანტები**. მათ უკავია საშუალედო მდებარეობა ცვლადებსა და კონსტანტებს შორის. ისინი მნიშვნელობებს იღებს აღწერის დროს (როგორც ჩვეულებრივი კონსტანტები), მაგრამ შეუძლია მნიშვნელობა შეიცვალოს პროგრამის სხეულში (როგორც ცვლადებმა). ტიპიზებული კონსტანტები აღიწერება კონსტანტების განყოფილებაში **(ლისტინგი 9**)

```
const
        კონსტ1 = მნიშვნელობა;
        კონსტ2 = მნიშვნელობა;
```

ლისტინგი 9

მაგალითი ნაჩვენებია **ლისტინგ 10-ზე** .

```
const
        nums: integer = 10;
```

ლისტინგი 10.

3.1.10. მონაცემების ტიპები Paskal-ში

დაპროგრამების რომელიმე ენაზე დაწერილი ნებისმიერი პროგრამა მონაცემების დამუშავებისთვისაა განკუთვნილი. მონაცემების სახეებია რიცხვები, ტექსტები, გრაფიკული გამოსახულებები, ბგერები და ა. შ. ერთმანეთისაგან განასხვავებენ საწყის და მათი დამუშავების გზით მიღებულ შედეგობრივ მონაცემებს.

მონაცემები შეინახება კომპიუტერის მეხსიერებაში. პროგრამა მათ მიმართავს მეხსიერების იმ უბნებთან დაკავშირებული ცვლადების სახელების მეშვეობით, სადაც მონაცემებია შენახული.

ცვლადების აღწერა ხდება პროგრამის ძირითადი კოდის დაწყებამდე. იქ მიეთითება ცვლადების სახელები და მათში შენახული მონაცემთა ტიპები.

დაპროგრამების ენა პასკალში მრავალი სხვადასხვა ტიპის მონაცემებია არსებობს. გარდა ამისა თავად დამპროგრამებელს შეუძლია შემოიტანოს ახალი ტიპის მონაცემები.

ცვლადის ტიპით განისაზღვრება თუ როგორი მონაცემები შეიძლება შევინახოთ მეხსიერების მასთან დაკავშირებულ უჯრედში.

integer-ის ტიპის ცვლადებმა შეიძლება მიიღოს -**32768**-დან **32767**-მდე დიაპაზონში არსებული მთელი მნიშვნელობები. **Pascal**-ში არსებობს სხვა ტიპის მთელი რიცხვებიც (**byte, longint**).

real-ის ტიპის ცვლადებში შეინახება ნამდვილი (წილადური) რიცხვები.

ბულის (ლოგიკური) ტიპის (boolesan) ცვლადმა შეიძლება მიიღოს მხოლოდ ორი მნიშვნელობა - **true** (1, ჭეშმარიტი) და **false** (0, ყალბი).

სიმბოლური ტიპის (**char**) ცვლადი იღებს სიმბოლოთა მოწყარიგებულ მიმდევრობაში შემავალ მნიშვნელობებს.

ინტერვალური ტიპი განსაზღვრება მომხმარებლის მიერ და იგი ყალიბდება მხოლო რიგობითი ტიპებისაგან. იგი წერმოადგენს კონკრეტულ დიაპაზონში არსებული მნიშვნელობების ქვესიმრავლეს.

შეგვიძლია შევქმნათ **საკუთარი ტიპის მონაცემებიც**. ამისათვის უნდა ჩამოვთვალოთ ის მნიშვნელობები, რომელთა მიღების უფლება აქვს მოცემული ტიპის ცვლადს. ასეთი ტიპის ცვლადებს **ჩამოთვლადი ტიპის მონაცემებიც** ეწოდება.

ყველა ზემოთ ჩამოთვლილი მონაცემებს ეწოდება **მარტივი მონაცემები**. მათგან განსხვავებით არსებობს **რთული** (**სტრუქტურილებული**) მონაცემებიც, რომლებიც აიგება მარტივი ტიპის მონაცემებც. ესენია მასივები, სტრიქონები, ჩანაწერები, სიმრავლეები და ფაილები. განვსაზღროთ ისინი.

მასივი ეწოდება ფიქსირებული რაოდენობის ერთი რომელიმე ტიპის ცვლადებისაგან შემდგარ სტუქტურას, რომელიც მეხსიერებაში ერთიან ადგილს იკავებს.

სტრიქონი ეწოდება **255**-ზე არაუმეტესი რაოდენობის კომპონენტების მიმდევრობას. რაოდენობის **255**-მდე შეზღუდვა **pascal**-ისთვისაა დამახასიათებელი.

ჩანაწერი არის ფიქსირებული რაოდენობის **ველებად** წოდებული კომპონენტებისაგან შემდგარ სტრუქტურა.

სიმრავლე ეწოდება ნებისმიერი რაოდენობის, ოღონდ ერთსა და იმავე ჩამოთვლადი ტიპის, ელემენტების ერთობლიობას.

ფაილები Pascal-ისათვის არის ერთპიპური მონაცემების მიმდევრობა, რომლებიც შენახულია გარე მეხსიერებაში (მაგალითად, ხისტ დისკზე).

ისეთი ტიპის მონაცემები, როგორიცაა **მაჩვენებელი**, კომპიუტერის მეხსიერებაში მონაცემების დინამიკურად შენახვასთან არის დაკავშირებული. ხშირად სტატიკურ მონაცემების გამოყენებაზე უფრო უკეთესია დინამიკური მონაცემების გამოყენება.

3.1.10.1. მთელი ტიპის ცვლადები

დაპროგრამების პასკალის ენაში არსებობს შემდეგი ხუთი ტიპის მთელი ცვლადები: **shorting, integer, longint, byte** და **word. ცხრ. 2**-ში მოცემულია ამ ცვლადების მნიშვნელობათა დასაშვები დიაპაზონები.

მთელი ტიპის ცვლადებს მხოლოდ მთელი მნიშვნელობების მიღება შეუძლია. ისინი აღწერილია **ლისტინგ 11**-ზე:

```
a, b, c: integer;
```

ლისტინგი 11.

აქ a, b, c, …არის ცვლდების სახელები, **integer** - ცვლადების ტიპი. ცვლადების ასეთი აღწერის შეხვედრის დროს ტრანსლატორი იმახსოვრებს, რომ ამ ცვლადებს მხოლოდ მთელი მნიშვნელობების მიღება შეუძლია და ამის შესაბამისად წარმოქმნის პროგრამის ბრძანებას.

ცხრ. 2 პასკალში მთელი ცვლადების ტიპები

ტიპი	დასაშვებ მნიშვნელობათა დიაპაზონი	გამოყოფილი მეხსიერება, ბიტები
shorting	-128 ... 127	1
integer	-32 768 ... ... 32 767	2
longint	-2 147 483 648 ... ... 2 147 483 497	4
byte	0 ... 255	1
word	0 ... 65 535	2

ცხრილ 3-ში მოცემულია მთელი ტიპის მონაცემებზე ჩასატარებელი ოპერაციები, რომელთა შესრულების შედეგადაც ხელახლა მთელი ტიპის მონაცემებს ვიღებთ.

მთელი ტიპის ოპერანდებზე ოპერაციები სწორად მხოლოდ იმ შემთხვევაში სრულდება, როდესაც შედეგისა და თითოეულ ოპერანდის მნიშვნელობა არანაკლებია მნიშვნელობათა დიაპაზონის მარცხენა განაპირა მნიშვნელობაზე და არ აღემატება ამ დიაპაზონის მარჯვენა განაპირა მნიშვნელობას. მაგალითად, პასკალში არსებობს კონსტანტა **maxint,** რომელიც მოიცავს მაქსიმალურ დასაშვებ მნიშვნელობას **in-**

teger ტიპისათვის. მაშინ ოპერაციის შესრულების დროს პროგრამაში დაცული უნდა იყოს შემდეგი პირობები:

ცხრ. 3. მთელ რიცხვებზე ჩასატარებელი ოპერაციები, რომელთა შედეგადაც მიიღება მთელი ტიპის მნიშვნელობები

ოპერაციის ნიშანი	ოპერაცია
+	შეკრება
-	გამოკლება
*	გამრავლება
div	მთელრიცხვული გაყოფა (ნაშთი უკუიგდება). გაყოფა დაუმრგვალებლად (განაყოფის მთელი ნაწილი)
mod	მოდულის მიხედვით გაყოფა (განაყოფის ნაშთის გამოყოფა). გაყოფით მიღებული ნაშთი a mod b = a ((a – divb)* b)

ცხრ. 4. მიმართებათა ოპერაციები

ოპერაციის ნიშანი	ოპერაცია
=	ტოლია
<>	ტოლია
<=	მეტია ან ტოლია
>	მეტია
>=	ნაკლებია ან ტოლია
<	ნაკლებია

(a ოპერაცია b) <= maxint,

a <= maxint, b <= maxint.

მთელ და მრავალ სხვა ტიპის ცვლადებზე დასაშვებია მიმართებების (შედარებების) ოპერაციების შესრულება (**ცხრ. 4**). ასეთი ოპერაციების შედეგები მიეკუთვნება Boolean ტიპს, და შეუძლია ორი მნიშვნელობიდან ერთ-ერთი მიიღოს: ან **true** (ჭეშმარიტი), ან **false** (ყალბი).

მთელი ტიპის ცვლადები შეგვიძლია გამოვიყენოთ დაპროგრამება ენა **Paskal**-ის მთელი რიგი სტანდარტული ფუნქციათა **(ცხრ. 5**) ფაქტურ პარამეტრებად.

ფუნქცია **random** დააბრუნებს თანაბრად განაწილებულ შემთხვევით მთელ რიცხვს, თუ მას გადვცემთ მთელ არგუმენტს. პროგრამის ხელახლა ამუშავების შემდეგ იგი დააბრუნებს იმავე მნიშვნელობას. ამის თავიდან ასაცილებლად პროგრამის დასაწყისში უნდა გამოვიძახოთ უპარამეტრო პროცედურა **randomize**.

inc და **dec** პროცედურებს შეიძლება ჰქონდეს მთელი ტიპის თითო-თითო ან ორ-ორი პარამეტრი.

ორი პარამეტრის არსებობის დროს პირველი პარამეტრის მნიშვნელობა მეორე პარამეტრის სიდიდის ტოლი სიდიდით იზრდება (**inc**-ის შემთხვევაში) ან მცირდება (**dec**-ის შემთხვევაში). **მ**აგალითად: inc(x,2) = x+2.

ერთი პარამეტრის არსებობის დროს მისი მნიშვნელობა ერთი ერთეულით იზრდება (**inc**-ის შემთხვევაში) ან მცირდება (**dec**-ის შემთხვევაში).

მაგალითი: dec(x) =x-1.

ასეთი პროცედურები Free Pascal-ში არ არის.

ცხრ. 5. paskal-ის სტანდარტული ფუნქციები, რომლებიც მთელური ტიპის არგუმენტებისათვის გამოიყენება

ფუნქცია	შედეგის ტიპი	შესრულების შედეგი
abs(x	მთელი	x-ის მოდული (x-ის აბსოლუტური მნიშვნელობა)
sqr(x)	მთელი	x-ის კვადრატი
succ(x)	მთელი	შემდეგი მნიშვნელობა x (x +1)
pred(x)	მთელი	წინა მნიშვნელობა x (x -1)
random(x)	მთელი	0...x-1 ინტერვალიდან შემთხვევითი მთელი რიცხვი
sin(x)	ნამდვილი	სინუს x(კუთხე რადიანებში)
cos(x)	ნამდვილი	კოსინუს x (კუთხე რადიანებში)
arctan(x)	ნამდვილი	არკტანგეს x (კუთხე რადიანებში)
lnx	ნამდვილი	x-ის ნატურალური ლოგარითმი
expx	ნამდვილი	x-ის ექსპონენტა
sqrt(x)	ნამდვილი	კვადრატული ფესვი x-იდან
add(x)	ლოგიკური	მნიშვნელობა true, თუ x კენტი რიცხვია; false – თუ ლუწია

შემდეგი ფუნქციები არგუმენტებად იღებს ნამდვილი და აბრუნებს მთელი ტიპის რიცხვებს:

- trunc(x) - წერტილის შემდეგ ათობით ნიშნების უკუგდება;
- round(x) - მთელამდე დამრგვალება.

მაგალითები:

1. დავუშვათ, რომ a=17, b = 5. მაშინ:

a div b = 3, a mod b = 2; sqr(b) = 16.

2. დავუშვათ, რომ x = 4.7389. მაშინ:

trunc(x) =4, round(x) = 5.

3. 4*43 გამოსახულება გვაძლევს მთელი ტიპის შედეგს, ხოლო 4*43.0 გამოსახულება - ნამდვილი ტიპის შედეგს, რადგან ერთ-ერთი მამრავლი არის ნამდვილი რიცხვი.

3.1.10.2. ნამდვილი ტიპის ცვლადები

ნამდვილი რიცხვების წარმოსადგენად დაპროგრამების ენა **pascal**-ში არსებობს შემდეგი ტიპის ცვლადები: **shortint, integer, longint, byte** და **word.** ცხრილ **6**-ში მოცემულია ამ ცვლადების მნიშვნელობათა დასაშვები დიაპაზონები.

ნამდვილი რიცხვების წარმოსადგენად დაპროგრამების ენა **pascal**-ში არსებობს რამდენიმე ტიპის ცვლადი, ოღონდ ყველაზე ხშირად მათ წარმოსადგენად გამოიყენება **Real**-ის ტიპის ცვლადები. მონაცემების ყველა ტიპი მოყვანილია ცხრილ **6**-ში.

ცხრ. 6. *პასკალში ნამდვილი ცვლადების ტიპები*

ტ ი პ ი	დ ი ა პ ა ზ ო ნ ი	ციფრების რაოდენობა	მეხსიერება, ბაიტი
Single	2.9E-39...1.7E38	11 - 12	6
Single (ერთმაგი სიზუსტის)	1.5E-45...3.4E38	7 - 8	4
Double (ორმაგი სიზუსტის)	5.0E-324...1.7E308	15 - 16	8
Extended (ამაღლებული სიზუსტის)	1.9E-4951...1.1E4932	19 - 20	10
Comp* (რთული)	(-2E+63)+1...2E+63)-1	19 - 20	8

* - *მხოლოდ მთელი მნიშვნელობა*

ციფრების რაოდენობა განსაზღვრავს მეხსიერებაში ნამდვილი რიცხვის შენახვის სიზუსტეს. მაგალითად, **Real**-ისათვის მანტისას თანრიგიანობამ არ უნდა გადააჭარბოს რვა ათობით ციფრს. **Comp ტიპი შეიცავს** მხოლოდ მთელ მნიშვნელობებს (იხ. ცხრ. **7**), რომლებიც გამოთვლებში ნამდვილი ცვლადის როლს ასრულებს.

ნამდვილ რიცხვებზე შეიძლება შესრულდეს შეკრების (+), გამოკლების (-), გამრავლებისა (*) და გაყოფის (/) ოპერაციები. ამ ოპერაციების შედეგად მიიღება ნამდვილი რიცხვები. ერთ-ერთი ოპერანდი მაინც თუ არის ნამდვილი, ოპერაციის შედეგი ნამდვილი იქნება.

ორი მთელი ოპერანდის გაყოფის (/) ოპერაციის შედეგაგაც შეიძლება მივიღოთ ნამდვილი რიცხვი. მაგალითად, 8/2=4.0; 7/2=3.5.

ნამდვილ რიცხვებზე ისევეა დასაშვები დასაშვები მიმართებების (შედარებების) ოპერაციების შესრულება, როგორც მთელ რიცხვებზე.

სტანდარტული **abs(x)** (**x-ის** მოდულის) ფუნქციიდან მიიღება მთელი შედეგი, თუ **x** არგუმენტი მთელი რიცხვია, და ნამდვილი შედეგი, თუ აღნიშნული არგუმენტი ნამდვილი რიცხვია.

შემდეგი ფუნქციები ნამდვილ შედეგს გვაძლევს როგორც ნამდვილი, ისე მთელი აქგუმენტის შემთხვევაში:

sin(x) – სინუს x (x რადიანებში);

cos(x) – კოსინუს x (x რადიანებში);

ln(x) – x-ის ნატურალური ლოგარითმი;

exp(x) – x-ის ექსპონენტა;

sqrt(x) – კვადრატული ფესვი x-დან;

arctan(x) – არკტანგენს x (x რადიანებში);

ნამდვილი მნიშვნელობის სახით არგუმენტის მთელ ნაწილს გვაძლევს int ფუნქცია, ხოლო წილადურ ნაწილს - frac ფუნქცია.

მთელი ტიპის შედეგს გვაძლევს trunc და **round ფუნქციები. პირველი მათგანი არგუმენტიდან წაკვეთავს წილადურ ნაწილს, ხოლო მეორე მათგანი არგუმენტს ამრგვალებს უახლოეს მთელ რიცხვამდე.**

უარგუმენტო ფუნქცია გვაძლევს **0**-დან **1**-მდე არსებულ თანაბრად განაწილებულ შემთხვევით რიცხვს.

არგუმენტების არმქონე **pi** ფუნქცია გვაძლევს პითაგორას რიცხვს.

ნამდვილი ტიპის ცვლადები და კონსტანტები არ შეიძლება გამოვიყენოთ;

ა) **pred, succ, ord** ფუნქციებში;
ბ) მასივების ინდექსებად;
გ) ჭდეებად მართვის გადაცემის ოპერატორებში.

3.1.10.3. ბულის ტიპის ცვლადები (Boolean)

ბულის ტიპის ცვლადები ეწოდება ცვლადებს, რომლებიც იღებს ორ, კერძოდ **true** (ჭეშმარიტ) ან **false** (ყალბ) მნიშვნელობას. ეს ცვლადები ასე არის მოწესრიგებული: **false** < **true.**

ბულის ტიპის ცვლადებზე **and, or, not** ოპერაციების ჩატარების შედეგად ასევე ბულის ტიპის ცვლადები მიიღება. განვიხილოთ ეს ოპერაციები.

- **and ოპერაცია** (ლოგიკური გამრავლება, გადაკვეთა, **და**-ოპერაცია). **a and b** გამოსახულება true მნიშვნელობას იღებს მაშინ და მხოლოდ მაშინ, როდესაც ასეთივე მნიშვნელობები აქვს ორივე (**a** და **b**) არგუმენტს. არგუმენტების ყველა დანარჩენი მნიშვნელობების დროს გამოსახულება იღებს მნიშვნელობას **false** (**ლისტინგი 12**).

```
true and true = true;
true and false = false;
false and false = false.
```

ლისტინგი 12.

- **or ოპერაცია** (ლოგიკური შეკრება, გაერთიანება, **ან**-ოპერაცია). **a or b** გამოსახულება **false** მნიშვნელობას იღებს მაშინ და მხოლოდ მაშინ, როდესაც ასეთივე მნიშვნელობა აქვს ორივე (**a** და **b**) არგუმენტს. არგუმენტების ყველა დანარჩენი მნიშვნელობების დროს გამოსახულება იღებს მნიშვნელობას **true** (**ლისტინგი 13**).

```
true or true = true;
true or false = true;
false or false = false.
```

ლისტინგი 13.

```
not true = false;
not false = true.
```

ლისტინგი 14.

- **not ოპერაცია** (უარყოფა, **არა**-ოპერაცია). not a გამოსახულებას აქვს **a** არგუმენტის საწინააღმდეგო მნიშვნელობა **(ლისტინგი 14**).

არსებობს ბულის შემდეგი სტანდარტული ფუნქციები:
odd(x) = true, თუ **x** კენტია (**x** არის მთელი);
eoln(x) = true, თუ შეგვხდა ტექსტური **x** ფაილის სტრიქონის ბოლო;
eof(x) = true, თუ შეგვხდა **x** ფაილის ბოლო.
დანარჩენ შემთხვევებში ეს ფუნქციები ღებულებს მნიშვნელობას **false**;

3.1.10.4. სიმბოლური ტიპის ცვლადები (Char)

char ტიპის ცვლადს შეუძლია მნიშვნელობა მიიღოს სიმბოლოების გარკვეული მოწესრიგებული მიმდევრობისაგან. ამ ტიპის ცვლადი იკავებს **1** ბაიტს და იღებს **ASCII** კოდის (ინფორმაციის გასაცვლელი ამერიკული სტანდარტული კოდის) **256** მნიშვნელობიდან ერთ-ერთ მნიშვნელობას. სიმბოლოები მათი კოდის შესაბამისადაა მოწესრიგებული, ამიტომ სიმბოლური ტიპის მონაცემებისათვის შეიძლება გამოყენებული იქნეს მიმართების ოპერაციებები.

პროგრამაში სიმბოლოს ნაცვლად შეგვიძლია გამოვიყენოთ #-ისა და კოდირებული სიმბოლოს ნომრისაგან შემდგარი მისი კოდი (მაგალითად, #51). ჩვეულებრივ სიმბოლოები, რომელთა ეკრანზე წარმოდგენაა შესაძლებელი, აპოსტროფებში ჩასმული ცხადი სახით ჩაიწერება (მაგალითად,'A', 'b', '*'). **ორი სტანდარტული ფუნქცია** საშუალებას გვაძლევს მოცემულ მიმდევრობას შევუთანადოთ მთელი არაუარყოფითი რიცხვების სიმრავლე (სიმბოლოთა მიმდევრობის რიგითი ნომრები). ამ ფუნქციებს ეწოდება **გარდაქმნის ფუნაციები.** ესენია:

- **ord(ch),** რომელიც გვაძლევს სიმბოლოს ნომერს (დანომვრა მოხდენილია ნულიდან);
- **chr(i),** რომელსაც სიმბოლოების ცხრილიდან გამოაქვს i-ური სიმბოლო.

მაგალითები:

1. ord('H')-ს ტრანსლატორის მიერ გამოყენებული ყველა სიმბოლოს მიმდევრობაში გამოაქვს სიმბოლო *H*-ის ნომერი;

2. chr(15)-ს გამოაქვს ამ მიმდევრობის მე-15 სიმბოლო.

გარდა ამისა, სიმბოლური ცვლადებისათვის გამოიყენება შემდეგი ფუნქციები:

pred(ch) – გვიბრუნებს წინა სიმბოლოს;

succ(ch) – გამოაქვს მომდევნო სიმბოლო;

upcase(ch) – ლათინური ალფაბეტის ნუსხურ ასოს გარდაქმნის მთავრულ ასოდ.

გარდა ამისა, შეგვიძლია გამოვიყენოთ inc და dec პროცედურები.

3.1.10.5. ტიპების ცხადად გარდაქმნა

მთელ რიგ შემთხვევაში **Pascal** -ში ავტომატურად ხდება ერთი ტიპის მონაცემიდან მეორე ტიპის მონაცემზე (მთელი მონაცემიდან - ნამდვილ მონაცემზე, სიმბოლური მონაცემიდან - სტრიქონულ მონაცემზე) ავტომატური გადასვლა. არსებობს, აგრეთვე, ტიპების გარდამქმნელი ფუნქციები (**ord, chr, trunc, round**). გარდა ამისა, პასკალში შესაძლებელია ტიპების ცხადად გარდაქმნა **(მონაცემების რეტიპიზაცია**). ტიპის ცხადად გარდაქნისათვის საჭიროა ისევე გამოვიყენოთ, როგორც გამოიყენება ფუნქციის სახელი. ამ შემთხვევაში პარამეტრად უნდა მივუთითოთ გარდასაქმნელი ცვლადის სახელი.

```
var
    x: real;
    sign: integer;
begin
    readln(x);
    sign:byte(x>0) – byte(x<0);
    writeln(sign);
end
```

ლისტინგი 15.

შესაძლებელია ნებისმიერი ტიპი გარდავქმნათ სხვა ტიპად, ოღონდ უნდა დავიცვათ მოთხოვნა: მინიჭების ოპერატორში მარცხენა ცვლადმა მეხსიერებაში უნდა დაიკავოს იმდენივე ბაიტი, რამდენი ბაიტიც უკავია გარდასაქმნელ მნიშვნელობას.

განვიხილოთ მაგალითი, რომელშიც გამოიყენება ტიპების ცხადი გარდაქმნა.გამოვითვალოთ შემდეგი ფუნქციის მნიშვნელობა:

$$sign(x) = \begin{cases} 1, \text{ როდესაც } x > 0 \\ 0, \text{ როდესაც } x = 0 \\ -1, \text{ როდესაც } x < 0 \end{cases}$$

პროგრამაში (**ლისტინგი 15**):

- დადებითი x-ის დროს x>0 გამოსხულება იღებს true მნიშვნელობას, ხოლო x<0 გამოსახულება - false მნიშვნელობას. ამის შედეგად მიიღება, რომ byte(x>0) = 1, byte(x<0)=0, ხოლო sign=1.
- უარყოფითი x-ის დროს byte(x>0)=0, byte(x<0)=1, sign=-1;
- x-ის ნულოვანი მნიშვნელობის დროს sign=0.

3.1.10.6. ჩამოთვლადი ტიპის ცვლადები

პროგრამაში შეგვიძლია შევიტანოთ ისეთი ცვლადი, რომლის ტიპი არ ემთხვევა არცერთ სტანდარტული ტიპის ცვლადს. მისი მოცემისათვის ტიპის გამოცხადების დროს საჭიროა ჩამოვთვალოთ აღნიშნული ცვლადის შესაძლო მნიშვნელობები. მათი მიღება შეეძლება ანალოგური (ე. ი. ახლად შემოტანილი) ტიპის ნებისმიერ სხვა ცვლადსაც, რომელსაც გამოვიყენებთ პროგრამაში შემდგომ. ჩამოთვლადი ტიპის ცვლადის აღწერის ზოგადი სახე **ლისტინგ 16**-ზეა მოყვანილი.

```
tipe
        nm = (word1, word2, … , vordN);
var
        w:nm
```

ლისტინგი 16.

აქ **nm** არის ნებისმიერი ტიპის იდენტიფიკატორი; **word1, word2** … **wordN** - კონკრეტული მნიშვნელობები, რომლებიც შეუძლია მიიღოს ახლად შემოტანილი **nm** ტიპის ნებისმიერმა ცვლადმა. ითვლება, რომ მოცემული ტიპის მნიშვნელობები არის მოწესრიგებული, ე. ი. ტიპის აღწერას იმავდროულად შემოაქვს: **word1** < **word2** < … < <**wordN**. რიგითი მნიშვნელობები აითვლება 0-იდან.

ჩამოთვლადი ტიპის ცვლადბზე შეიძლება გამოვიყენოთ ord, pred ფუნქციები და **inc**, **dec** ოპერაციები.

ერთსა და იმავე ტიპის სნებისმიერი ცვლადებისათვის შეიძლება გამოვიყენოთ =, <>, <= , >=, <, > მიმართებები (თანაფარდობები).

ჩამოთვლადი ტიპის ცვლადებისათვის დამახასიათებელი თავისებურებაა ის, რომ მათი მნიშვნელობები არ შეიძლება შევიტანოთ კლავიატურიდან და გამოვიტანოთ ეკრანზე (მაგრამ შეგვიძლია გამოვიყენოთ **ტიპიზებულ ფაილებთან**).

ჩამოთვლადი ტიპის ცვლადის მაგალითია:

color = (red, yellow, green, blue);

აქ ითვლება, რომ ცვლად **color**-მა შეუძლია მიიღოს ჩამოთვლილი მნიშვნელობებიდან ((წითელი, ყვითელი, მწვანე ან ლურჯი ფერებიდან) ნებისმიერი მნიშვნელობა და, ამასთანავე დაცულია შემდეგი მიმართება: **red < yellow <green <blue**.

დავუშვათ, რომ **x** არის ჩვენს მიერ ახლად შემოტანილი ტიპის ცვლადი და განვსაზღვროდ **succ(x), pred(x)** და **ord(x)** ფუნქციები.

- **succ(x)** ფუნქცია **x** ელემენტისათვის განსაზღვრავს იმ მოწესრიგებულ მიმდევრობას, რომელსაც იგი ეკუთვნის და გვაძლევს აღნიშნულ მიმდევრობაში მის შემდეგ მდგარ ცვლადს. დავუშვათ, რომ ასოები მოყვანილია ალფაბეტასეული მიმდევრობით. მაშინ succ(A) = succ(B), succ(B) = succ(C) და ა. შ. ზემოთ განხილული მაგალითისათვის succ(yellow) = green.
- **pred(x)** ფუნქცია **x** ელემენტისათვის განსაზღვრავს იმ მოწესრიგებულ მიმდევრობას, რომელსაც იგი ეკუთვნის და გვაძლევს აღნიშნულ მიმდევრობაში მის წინ მდგარ ცვლადს. მაგ. pred(yellow) = red.
- **ord(x)** ფუნქცია **x** ელემენტისათვის განსაზღვრავს იმ მოწესრიგებულ მიმდევრობას, რომელსაც იგი ეკუთვნის და გვაძლევს აღნიშნულ მიმდევრობაში იმ ადგილის ნომერს, რომელზედაც იგი დგას, მაგალითად ord(red) = 0; ord(yellow) = 1, ord(green) = 2.

3.1.10.7. დიაპაზონური ანუ ინტერვალური ტიპის ცვლადები

სკალარული (ჩამოთვლადი) ტიპის ცვლადისათვის შეგვიძლია მივუთითოთ იმ მნიშვნელობათა გარკვეული ქვესიმრავლე, რომე-

ლიც შეიძლება მიიღოს ცვლადმა. ზოგადად, ეს ასე შეგვიძლია ჩავწეროთ, როგორც ეს **ლისტინგ 17**-ზეა ნაჩვენები.

```
a:min ..max
```

ლისტინგი 17.

აქ a არის ინტერვალური ცვლადი, **min** - ქვესიმრავლის (დიაპაზონის) მარცხენა, ხოლო **max** - მარჯვენა საზღვარი. დიაპაზონის საზღვრები გაყოფილია ორი წერტილით; **min** საზღვარი ყოველთვის ნაკლები უნდა იყოს **max** საზღვარზე.

min და **max** კონსტანტები ერთსა და იმავე ტიპის უნდა იყოს. ისი-ნი განსაზღვრავს **a** ცვლადის ბაზისურ ტიპს. მაგალითად, თუ საზღვრებად გამოყენებულია არის **integer** ტიპის მთელი რიცხვები, მაშინ **a** ცვლადის გამოეყოფა მეხსიერების ისეთივე მოცულობა, როგორიც უნდა გამოეყოს integer ტიპს. ოღონდ, **a** ცვლადს მხოლოდ ისეთი მნიშვნელობის მიღება შეუძლია, რომელიც მოთავსებულია დიაპაზონის ფარგლებში. განვიხილოთ **მაგალითები.**

• დავუშვათ, რომ **k** ცვლადმა უნდა მიიღოს -560..560 დიაპაზონში არსებული მნიშვნელობა. მაშინ იგი ასე უნდა აღვწეროთ: **k**:-560..560. ამ დროს **k** ცვლადის ბაზისური ტიპია **integer**, რამდენადაც დიაპაზონის საზღვრებად გამოყენებულია მთელი -560 და 560 კონსტანტები;

```
type color=(red, yellow, green, blue);
var d:red..green;
begin
    d:=red;
    writeln(d);
    d:=yellow;
    writeln(d);
    d:=green;
    writeln(d);
readin
end
```

ლისტინგი 18.

• **d** ცვლადს თუ შეუძლია **red, yellow, green** მნიშვნელობებიდან მიიღოს ერთ-რთი მნიშვნელობა, მაშინ იგი უნდა წარმოვადგინოთ ასე: **b: red..green (ლისტინგი 18**). ცხადია, რომ **d**- ის ბაზისური ტიპი იქნება **color** (იხილეთ წინა პარაგრაფი):

• დავუშვათ, რომ **g** არის ცვლადი, რომელიც იღებს რომელიმე დაწესებულების თანამშრომელთა დაბადების წლების მნიშვნელობებს. გონივრული გადაწყვეტილება იქნება, თუ **g** ცვლადის მნიშვნელობათა დიაპაზონს თუ ასე აღვწერთ: **g:** 1930…2000**.**

3.1.11. ლოგიკური გამოსახულებები და ლოგიკური ოპერაციები

3.1.11 .1. მარტივი ლოგიკური გამოსახულებები

იმისათვის, რომ პროგრამა წრფივი არ იყოს (ე. ი. სიტუაციაზე დამოკიდებულებით მასში სრულდებოდეს სხვადასხვა ინსტრუქცია), დაპროგრამების ენებში გამოიყენება ლოგიკური გამოსახულებები, რომელთა მნიშვნელობები შეიძლება იყოს ჭეშმარიტი (true) ყალბი (false). ლოგიკური გამოსახულების მნიშვნელობა ჩვეულებრივ პროგრამის შესრულების ორი გზიდან ირჩევს ერთ-ერთს.

მარტივი ლოგიკური გამოსახულება მიიღება ორ ოპერანდს (მნიშვნელობას) შორის მიმართების (შედარების) ოპერაციის ჩატარების შედეგად. ქვემოთ მოყვანილ მაგალითებში ოპერანდების როლს ასრულებს **x** და **y** ცვლადები. ზოგადად ოპერანდებად შეგვიძლია გამოვიყენოთ რიცხვები, სიმბოლოები და სხვა ტიპის მონაცემები, რომელთა ურთიერთშედარება არის შესაძლებელი. ოღონდ რეკომენდებული არ არის ერთმანეთს შევადაროთ ნამდვილი რიცხვები. ამას განაპირობებს კომპიუტერის მეხსიერებაში მათი შენახვის თავისებურებები.

პასკალში გათვალისწინებულია მიმართების (შედარების) შემდეგი ექვსი ოპერატორი:

• ნაკლებობა: **x < y**;

• მეტობა: **x > y**;

• ტოლობა: **x = y**;

• უტოლობა: **x <> y**;

- ნაკლებობა ან ტოლობა: **x <= y**;
- მეტობა ან ნაკლებობა: **x <= y**;

3.1.11.2. ბულის ტიპის გამოსახულებები

ლოგიკური გამოსახულების შედეგად ყოველთვის მიიღება ბულისეული (ლოგიკური) მნიშვნელობა. **ბულის ტიპის მონაცემს (boolean)** ყოველთვის მხოლოდ ორი მნიშვნელობის (**true** ან **false**) მიღება

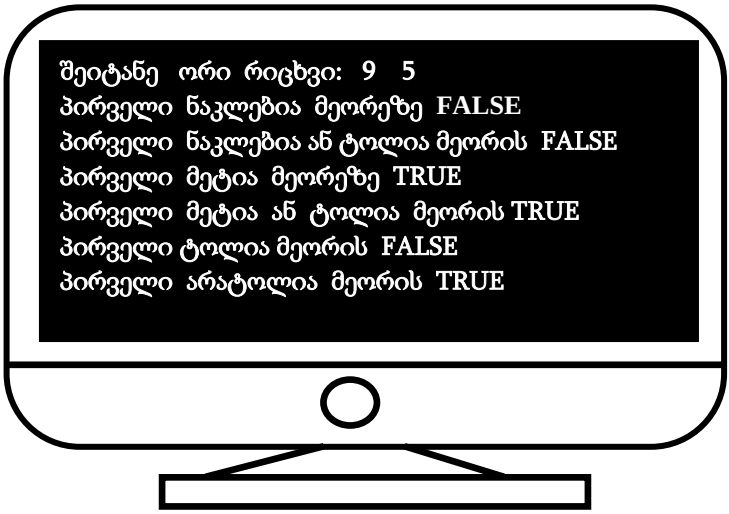
```
war
        x, y: integer:
        b1, b2, b3, b4, b5, b6: Boolean;
begin
    write (‘შეიტანე ორი რიცხვი:    ‘);
    readln (x, y);

   b1 := x < y;
   b2 := x <= y;
   b3 := x > y;
   b4 := x >= y;
   b5 := x = y;
   b6 := x <> y;

    writeln (‘ პირველი ნაკლებია მეორეზე ‘, b1);
    writeln (‘ პირველი ნაკლებია ან ტოლია მეორეზე ‘, b2);
    writeln (‘ პირველი მეტია მეორეზე ‘, b3);
    writeln ( ‘ პირველი მეტია ან ტოლია მეორის ‘, b4);
    writeln (‘ პირველი ტოლია მეორის ‘, b5);
    writeln (‘ პირველი არატოლია მეორის ‘, b6);
readln
end
```

ლისტინგი 19.

შეუძლია. ეს სიდიდეები მოწესრიგებულია როგორც **false < true**. ეს ნიშნავს, რომ ბულის ტიპის მონაცემები მიმართების (შედარების) არა

მარტო შედეგია, არამედ მათ შეუძლია შეასრულოს მიმართების ოპერაციის ოპერანდების როლიც. ბულის ტიპის ოპერანდების მიმართ შეგვიძლია გამოვიყენოთ ფუნქციები **ord, succ, pred** და პროცედურები **inc и dec.** ბულის (**boolean**) ტიპის მნიშვნელობა კომპიუტერის მეხსიერებაში იკავებს **1** ბაიტის ტოლ მნიშვნელობას.

ლისტინგ 19-ზე მოყვანილ მაგალითში ბულისეულ ექვს ცვლადს ენიჭება მარტივი ლოგიკური გამოსახულებების მნიშვნელობები. ასეთ ცვლადებში შენახული მნიშვნელობები შემდეგ გამოიტანება ეკრანზე (**ნახ. 6**).



ნახ. 6. b_i, $i = 1,2,\ldots,6$ *ცვლადებში შენახული მნიშვნელობების გამოტანა მონიტორის ეკრანზე*

boolean ტიპის გარდა პასკალში შემოტანილია კიდევ სამი ბულისეული ტიპის ცვლადი:

- **bytebool**, რომელიც მეხსიერებაში იკავებს 1 ბაიტის ტოლ მოცულობას;

- **wordbool** - იკავებს 2 ბაიტის ტოლ მოცულობას;

- **longbool** - იკავებს 4 ბაიტის ტოლ მოცულობას.

ნებისმიერი ტიპის ბულისეული ცვლადისთვის **false** მნიშვნელობას შეესაბამება **0**, ხოლო **true**-ს - ნებისმიერი არანულოვანი მნიშვნე-

ლობა. სხვადასხვა ტიპის ბულისეული ცვლადები მათზე ოპერაციების შესრულების დროს სხვადასხვანაირად იქცევა.

3.1.11.3. ლოგიკური ოპერაციები

ლოგიკური ოპერაციების მეშვეობით შეგვიძლია ჩამოვაყალიბოთ ნებისმიერი სირთულის მქონე ლოგიკური გამოსახულებები. ლოგიკურ ოპერაციებს ხშირად იყენებენ მარტივი ლოგიკური გამოსახულებებისთვის.

დაპროგრამების ენა **Pascal**-ში არსებობს შემდეგი ლოგიკური ოპერაციები:

• **კონიუნქცია** (ლოგიკური გამრავლება, გადაკვეთა) - **and**. გამოსახულება **a and b** ჭეშმარიტ, ანუ **true** მნიშვნელობას იძლევა მხოლოდ იმ შემთხვევაში, თუ **a** და **b** ცვლადებს აქვს **true** მნიშვნელობა. ყველა დანარჩენ შემთხვევაში მას აქვს **false** (ყალბი) მნიშვნელობა (**ლისტინგი 20**).

```
true and true = true;
true and false = false;
false and true = false;
false and false = false.
```

ლისტინგი 20.

• **დიზიუნქცია** (ლოგიკური შეკრება, გაერთიანება) – **or,**. გამოსახულება **a or b** ყალბ ანუ **false** მნიშვნელობას იძლევა მხოლოდ იმ შემთხვევაში, თუ **a** და **b** ცვლადებს აქვს **false** მნიშვნელობა. ყველა დანარჩენ შემთხვევაში მას აქვს **true** (ჭეშმარიტი) მნიშვნელობ (**ლისტინგი 21**).

```
true and true = true;
true and false = true;
false and true = true;
false and false = false.
```

ლისტინგი 21

```
not true = false;
not false = true.
```

ლისტინგი 22

• **უარყოფა** (ინვერსია)) **– not**. გამოსახულება **not a** გამოსახულებას აქვს **a** არგუმენტის მნიშვნელობის შებრუნებული მნიშვნელო- (**ლისტინგი 22**)

• **2-ის მოდულით შეკრება** (mod2-ით შეკრება, გამომრიცხავი ან) – **xor,**. გამოსახულება **a xor b** ჭეშმარიტ ანუ **true** მნიშვნელობას იძლევა მხოლოდ იმ შემთხვევაში, თუ **a** და **b** ოპერანდებს აქვს სხვადასხვა მნიშვნელობა, ანუ, ფაქტობრივად იგივე გამოდის, როდესაც მხოლოდ ოპერანდს აქვს **true** მნიშვნელობა (**ლისტინგი 23**).

```
true and true = false;
true and false = true;
false and true = true;
false and false = false.
```

ლისტინგი 23.

ლოგიკური ოპერატორების შესრულების მიმდევრობა ასეთია: **not, and, or**.

პასკალის ენაში თავდაპირველად სრულდება ლოგიკური (and, or, xor, not), ხოლო შემდეგ მიმართებების, ანუ შედარებების (>, >=, <, <=, <>, =) ოპერატორები, ამიტომ რთულ ლოგიკურ ოპერაციებში არ უნდა დაგვავიწყდეს ფრჩხილების განთავსება.

რთული ბულისეული გასმოსახულებები შეგიძლია გამოთვლების დასასრულებამდე არ დავამუშაოთ იმ შემთხვევაში თუ გამოთვლების გაგრძელება არ ცვლის შედეგს. ბულისეული გამოსახულების დამუშავება. განსაკუთრებულ შემთხვევაში თუ საჭიროა ბულისეული გამოსახულების უფრო ადრე დამუშავება, მაშინ ამისათვის საჭიროა კომპილაციის **{B+}** დირექტივა ჩავრთოთ.

პასკალში არსებობს მულისეული შემდეგი **სტანდარტული ფუნქციები:**

• **odd(x) = true**, თუ x არგუმენტი არის კენტი (x მთელი ტიპის ცვლადია);

• **eoln(x) = true**, თუ ტექსტურ ფაილებში შეგვხვდა სტიქონის დასასრული (ბოლო);

• **eof(x) = true**, თუ შეგვხდა x ფაილის ბოლო.

ყველა დანარჩენ შემთხვევაში ეს ფუნქციები იღებს false მნიშვნელობას.

3.1.12. ბიტური არითმეტიკა და ოპერაციები ბიტებზე

Pascal–ში მთელი ტიპის **(**byte, shortint, word, integer, longint და მათი დიაპაზონები**)** ცვლადების შემთხვევაში დასაშვებია შესრულდეს **ბიტური ოპერაციები**.

3.1.12.1. ლოგიკური ოპერაციები ბიტებზე

ორი მთელი ოპერანდის ბიტებზე შეიძლება შესრულდეს ზემოთ განხილული ლოგიკური **not, and, or, xor** ოპერაციები. **ბიტური ოპერაციები** და **ლოგიკურ ოპერაციები** ერთმანეთისაგან იმით განსხვავდება, რომ ბიტური (თანრიგობრივი) ოპერაციები სრულდება ოპერანდების ცალკეულ ბიტებზე და არა ათობითი (ჩვეულებრივი) წარმოდგენის დროს მათ მნიშვნელობებზე.

მაგალითად, ერთ ბაიტად ორობითი წარმოდგენის დროს რიცხვი **5**-ს აქვს სახე 00000101. მასზე **not** ოპერაციების შესრულების დროს ინვერტირდება ამ რიცხვის ბიტები და მიიღება 11111010, ანუ რიცხვი 250. ბიტურ **or** ოპერაციას თუ შევასრულებთ რიცხვებზე **5** (00000101) და **3** (00000011), მაშინ მივიღებთ რიცხვს **6** (0000110).

3.1.12.2. ლოგიკური ძვრის ოპერაციები

პასკალში მთელი ტიპის მონაცემებზე შესასრულებლად კიდევ ორი ოპერაცია არის განსაზღვრული, რომელთა პრიორიტეტის დონე and, *, /, div და mod ოპერაციების პრიორიტეტის დონის ტოლია. ეს ოპერაციებია shl და shr. მათი საშუალებით ხდება დასახული რაოდენობის პოზიციებით ბიტების მიმდევრობით შესაბამისად მარცხნივ და მარჯვნივ ძვრა, ამ დროს იკარგება ბიტები, რომლებიც ტოვებს თანრიგთა ბადეს.

ბიტების მარცხნივ დაძვრის shl ოპერაციის შესრულების შემდეგ თანრიგთა ბადის მარჯვნივ განთავისუფლებული თანრიგები შეივსება ნულებით.

ბიტების მარცხნივ დაძვრის shr ოპერაციის შესრულების შემდეგ თანრიგთა ბადის მარცხნივ განთავისუფლებული თანრიგები:

ა) ნულებით შეივსება დადებითი რიცხვის ძვრის შემთხვევაში;

ბ) ერთიანებით შეივსება უარყოფითი რიცხვის ძვრის შემთხვევაში.

shl ოპერაციის დახმარებით შეგვიძლია 2-ის ხარისხების ტოლი რიცხვების გამრავლების ოპერაცია შევასრულოთ. ერთნაირ შედეგებს გვაძლევს შემდეგი გამოსახულებები:

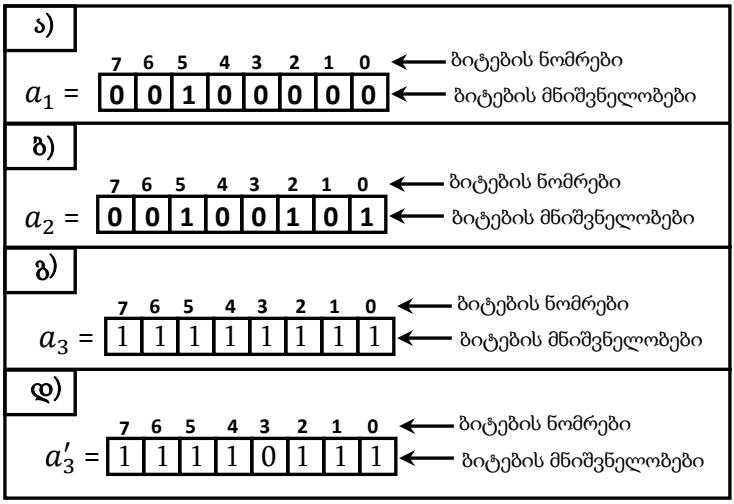
- (a shl 1) = a * 2,
- (a shl 2) = a * 4,
- (a shl3) = a * 8.

3.1.12.3. ბიტური ოპერაციების პრაქტიკული მნიშვნელობა

and ოპერაცია პრაქტიკულად ყოველთვის გამოიყენება 1-ის მნიშვნელობის მქონე ბიტების არსებობის შესამოწმებლად ან ზოგიერთი ბიტების გასანულებლად.

მსგავსი შემოწმება საჭიროა, თუ რიცხვი წარმოადგენს ორი შესაძლო მნიშვნელობის მქონე ნიშნების ნაკრებს (ალმების ნაკრებს). მაგალითად, მეხსიერების მრავალი სისტემური უჯრედი შეიცავს ცნობებს კომპიუტერის კონფიგურაციის ან მისი მდგომარეობის შესახებ. ამ დროს კონკრეტული ნომრის მქონე ბიტის დაყენება მდგომარეობა 1-ში რაიმე რეჟიმის ჩართვად, ხოლო მდგომარეობა 0-ში - ამ რეჟიმის ამორთვად განიმარტება.

განვიხილოთ byte ტიპის a_1 ცვლადი (ნახ.7,ა.). იგი არის რვა ალმის (ბიტის) შემცველი ბაიტი (8 თანრიგიანი რიცხვი). მისი ერთი ბიტი, რომლის ნომერია 5, დაყენებულია მდგომარეობა 1-ში. დანარჩენი ბიტების მდგომარეობები განულებულია. ნომერ 5-ის მქონე ბიტის მდგომარეობა 1 არის $2^5 = 32$ (00100000) რიცხვი. ამიტომ თუ a_1 ცვლადის მეხუთე ბიტში დგას ერთიანი, მაშინ სრულდება პირობა (a_1 and 32) = 32. ეს შეგვიძლია შევამოწმოთ განშტოების if ოპერატორში.



ნახ. 7. *a_i ცვლადების მნიშვნელობები*

ერთდროულად მდგომარეობა **1**-ში დაყენებული რამდენიმე ბიტის შესამოწმებლად შესაბამისი რიცხვი შეგვიძლია გამოვითვალოთ როგორც რიცხვ **2**-ის ხარისხების ჯამი, რომელთა ხარისხის მაჩვენებლებად გამოყენებულია მდგომარეობა **1**-ში დაყენებული ბიტების ნომრები.

ზემოთ აღნიშნულის თვალნათლივ საჩვენებლად, განვიხილოთ a_2 ცვლადი (**ნახ. 7,ბ).** მასში მდგომარეობა **1** ერთდროულად დაყენებულია ნომრებია **5**, **2** და **0** ნომრების მქონე ბიტები. ერთდოულად მათი მდგომარეობის შესამოწმებლად უნდა გამოვითვალოთ რიცხვი:

$$2^5 + 2^2 + 2^0 = 32 + 4 + 2 = 37.$$

a_2 ცვლადში თუ ერთიანები დაყენებული იქნება **5**, **2** და **0** ნომრებიან ბიტებში მაშინ შესრულებული იქნება პირობა **(a and 37) = 37**.

განვიხილოთ byte-ს ტიპის a_3 ცვლადში (ნახ. **7**, **გ**) რომელიმე, დავუშვათ ნომერი **3**-ის მქონე, ბიტის გაუქმების საჭიროება. მის განულებამდე გამოვითვალოთ ის მაქსიმალური რიცხვი, რომელიც შეიძლება ჩაიწეროს ამ ცვლადში. ამისათვის მისი ყველა ბიტი დაყენებული უნდა იყოს მდგომარეობა **1**-ში, როგორც ეს **ნახ. 7,გ**-ზეა ნაჩვენები. ცხადია, რომ საძებნი მაქსიმალური რიცხვი ასე გამოითვლება:

$$2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = 255.$$

ახლა გავანულოთ ნომერ **3**-ის მქონე ბიტი. მივიღებთ a_3' ცვლადს (ნახ. **8,დ**). განულებული მესამე ბიტის მნიშვნელობაა $2^3 = 8$. ამიტომ a_3' ცვლადის მნიშვნელობა იქნება **255 – 8 = 247**. მიღებულ a_3' რიცხვს თუ ლოგიკურად გავამრავლებთ a_3 ცვლადზე, მაშინ ამ უკანასკნელში, დამოუკიდებლად მასში არსებული ბიტების მნიშვნელობებისა, აუცილებლად განულდება ნომერ **3**-ის მქონე ბიტი. მაშასადამე, გვექნება $a_3 := a_3$ **and (255-8).**

ანალოგურად შეგვიძლია გავანულოთ რამდენიმე ბიტი.

■ or ოპერაცია გამოიყენება ორობითი მთელი რიცხვის ცალკეული ბიტების მდგომარეობა 1-ში დასაყენებლად. მაგალითად, ორობით **a** ცვლადში ნომერ 4-ის მქონე ბიტის მდომარეობა 1-ში ისე დასაყენებლად, რომ დანარჩენმა ბიტებმა არ შეიცვალოს საკუთარი მდგომარეობები, საჭიროა შევასრულოთ ოპერაცის **a:= a or 16**, სადაც **16** არის **2**-ის მეოთხე ხარისხი (**16** = 2^4). ანალოგურად შეგვიძლია მდგომარეობა **1**-ში დავაყენოთ რამდენიმე ბიტი.

■ xor ოპერაცია გამოიყენება ბიტის მნიშვნელობის შესაცვლელად საწინააღმდეგო მნიშვნელობით: **1**-ის შესაცვლელად 0-ით და 0-ის შესაცვლელად **1**-ით. მაგალითად, **a** ცვლადში ნომერ 3-ის მქონე ბიტის მდგომარეობა რომ შევცვალოთ საწინააღმდეგო მნიშვნელობით, საჭიროა შევასრულოთ ოპერაცია **a:= a or 8,** სადაც **8** არის **2**-ის მესამე ხარისხი **8** = 2^3).

3.1.13. ოპერაციების რეალიზების სამყარო

3.1.13.1. ოპერაციების შესრულების თანამიმდევრობა

რთულ გამოსახულებაში ოპერაციების თანამიმდევრობა განისაზღვრება ოპერაციათა პრიორიტეტის შესაბამისად. ერთნაირი პრიორიტეტის მქონე ოპერაციები მარცხნიდან მარჯვნივ სრულდება. მრგვალი ფრჩხილების გამოყენების მეშვეობით შეგვიძლია მათი შე-

სრულების თანამიმდევრობა შევცვალოთ. სხვა ოპერაციების შესრულებამდე უნდა განვსაზღვროთ ფუნქციების მნიშვნელობები. ოპერაციების პრიორიტეტული დონეები (პრიორიტეტის კლების მიხედვით) ასეთია:

1. ერთადგილიანი (უნარული) **+, -, not** ოპერაციები;

2. მულტიპლიკაციური ***, /, div, mod, and** ოპერაციები;

3. ადიტიური **+, -, or, xor** ოპერაციები;

4. მიმართებების (შედარებაბის): <, <=, >, >=, =, <> ოპერაციები

3.1.13.2. პირობითი ოპერატორები

პროგრამის რალიზების პროცესში საჭიროა ბრძანებათა სხვადასხვა ნაკრები შესრულდეს იქნეს მის შესრულებამდე მომხდარ მოვლენებზე დამოკიდებულებით. დაპროგრამების ენებში ეს მიიღწევა სპეციალური კონსტრუქციების - **პირობითი ოპერატორების** დახმარებით.

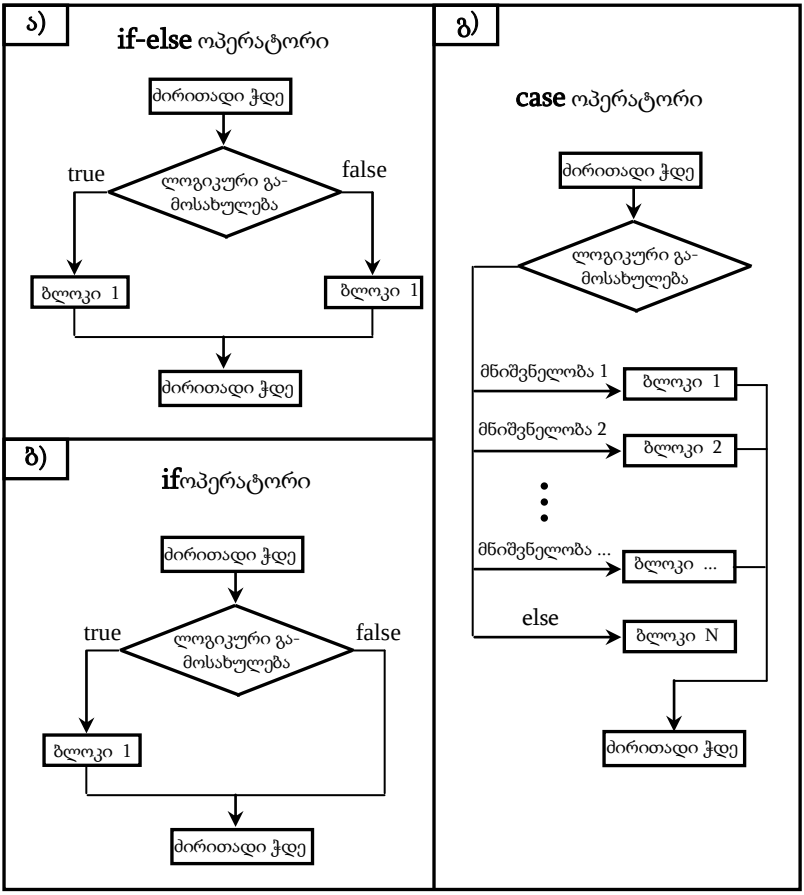
დაპროგრამების ენებში პირობით ოპერატორად ყველაზე ხშირად გამოიყენება if-else ოპერატორი ან მისი შემოკლებილი if ვარიანტი. არსებობს აგრეთვე ამორჩევის case ოპერატორი, რომელსაც უფრო სპეციფკურად იყენებენ.

▉ if-else ოპერატორი (ნახ. 8,ა).

პროგრამის ძირითადი შტოს შესრულება როდესაც მიაღწევს პირობით **if-else** ოპერატორამდე, მაშინ მის თავში არსებული ლოგიკური გამოსახულების მნიშვნელობაზე დამოკიდებულებით კოდის სხვადასხვა ბლოკი შესრულდება. ლოგიკურმა გამოსახულებამ თუ მიიღო **true** (ჭეშმარიტი) მნიშვნელობა, მაშინ შესრულდება ერთ-ერთ ბლოკი (პასკალში იგი იწყება სიტყვით **then**), ხოლო თუ მიიღო **false** (მცდარი) მნიშვნელობა - მაშინ შესრულდება მეორე ბლოკი (პასკალში იგი იწყება სიტყვით **else**). ჩალაგებული ბლოკებიდან ერთ-ერთი ბლოკის შესრულების შემდეგ პროგრამა უბრუნდება ძირითად შტოს. მეორე ბლოკი არ შესრულდება.

დავშვათ, რომ გვინდა შევადგინოთ პროგრამა, რომელმაც უნდა გაარკვიოს მომხმარებელმა ლუწი თუ კენტი რიცხვი შეიტანა და ამის

შესახებ ეკრანზე უნდა გამოიტანოს სათანადო შეტყობინება. ასეთ პროგრამას პასკალის ენაზე ექნება **ლისტინგ 24**-ზე ნაჩვენები სახე.



***ნახ. 8**. პირობითი **if-else** (ა), **if** (ბ) და **case** ოპერატორები*

```
var n: integer;
begin
    write (‘ შეიტანე მთელი რიცხვი: ‘);
     readln (n);

     if n mod 2 = 0 thn
        write (‘ რიცხვი  ლუწია. ’)
      else
         write (‘ რიცხვი  კენტია. ’)

 readln
 end
```

ლისტინგი 24.

if ოპერატორი (ნახ. 8,ბ)

არსებობს პირობითი ოპერატორების არასრული ფორმები. ასეთ შემთხვევაში **if**-ში ჩალაგებული ბლოკი მხოლოდ მაშინ შესრულდება, როდესაც პროგრამის თავში არსებული ლოგიკური გამოსახულება მიიღებს **true** მნიშვნელობას. **false** მნიშვნელობის დროს პროგრამის შესრულება მაშინვე ძირითად ბლოკს გადაეცემა. ცხადია, რომ ამ შემთხვევაში არ არსებობს **else** შტო.

```
var n: integer;
begin
    write (‘ შეიტანე მთელი რიცხვი: ‘);
     readln (n);

     if n < 0 thn
        n := abs(n);
      write (n);

 readln
 end
```

ლისტინგი 25.

ლისტინგ 25-ზე მოყვანილ მაგალითში ცვლადის მნიშვნელობა თუ 0-ზე ნაკლებია, მაშინ მისი მნიშვნელობა შეიცვლება (ხდება რიცხვის მოდულის პოვნა). ცვლადის მნიშვნელობა თუ თავიდანვე 0-ზე მეტია, მაშინ **if** ოპერატორთან არსებული კოდის ბლოკი საერთოდ არ სრულდება, რადგან არ არის დაცული პირობა (n < 0).

პირობად შეიძლება გამოვიყენოთ ნებისმიერი გამოსახულება, რომლის შედეგი იქნება ჭეშმარიტი, ე.ი. **true**, ან მცდარი, ე. ი. **false**.

უშუალოდ **then**-ის შემდეგ შეიძლება იდგეს მხოლოდ ერთი ოპერატორი. რამდენიმე ოპერატორის საჭიროებისამებრ ისინი უნდა ჩავსვათ ზემოთ აღნიშნულ ე. წ. **ოპერატორულ ფრჩხილებში beginend**. **ლისტინგ 26-**ზე მოყვანილია პროგრამა, რომელიც ცვლის ცვლადების მნიშვნელობათა ადგილებს, თუ ეს მნიშვნელობები განსხვავდება ერთმანეთისაგან. მასში **if** ბლოკი შეიცავს ოთხ გამოსახულებას, ამიტომ ისინი ჩასმულია ოპერატორულ **begin-end** ფრჩხილებში.

```
war
   a,b,c: integer
begin
    write(‘ a = ’);
     readlin(a);
     write(‘ b = ’);

      if a <> b then  begin
         c : = a;
         a : = b;
         b : = c;
         writeln(‘ a = ’, a,;’ b= ’ , b);
      end
      else
          writeln(‘ შეტანილია ერთნაირი რიცხვები ’);

readln;
end.
```

ლისტინგი 26.

შესაძლებელია ერთი **if** (ან **if-else**) ოპერატორი მეორე ოპერატორში იყოს ჩადებული. ამ დროს ფრთხილად უნდა მოვიქცეთ, რადგან ძნელი გასარკვევია თუ რომელ (გარე თუ შიგა) **if**-ს მიეკუთვნება **else** შტო. რეკომენდებულია, რომ **if**-ის ჩადებული კონსტრუქციები გამოვიყენოთ **else** შტოში. გარდა ამისა, პასკალის ენაში დაცულია შემდეგი წესი: თითოეულ **then**-ს შეესაბამება სხვა **then**-თან შესაბამის დამყარების დროს აუმოქმედებელი უახლოესი **else. if** ოპერატორების ჩალაგებულობის სიღრმე შეიძლება რაგინდ დიდი იყოს, ოღონდ ასეთ კოდში გარკვევა ძალიან რთულია.

case ოპერატორი (**ამორჩევის ოპერატორი**) (ნახ. **8,გ**)

ზემოთ განხილული **if** და **if-else** ოპერატორებისა პასკალში გათვალისწინებულია **გადამრთველად** წოდებული ოპერატორი **case**. იგი შეგვიძლია განვიხილოთ დიდი რაოდენობის (და არა **if-else** ოპერატორის შემთხვევაში არსებული ორი) პასუხის მქონე კითხვად. ოღონდ **if** ოპერატორისაგან განსხვავებით **case** ოპერატორს აქვს მთელი რიგი შეზღუდვა. მისი ფორმატი **ლისტინგ 27**-ზეა მოყვანილი

```
case სელექტორი of
                    მნიშვნელობა 1: ოპერატორი 1;
                    მნიშვნელობა 2: ოპერატორი 2;
                    მნიშვნელობა 3: ოპერატორი 3;
                    ...
                    else ოპერატორი N
end
```

ლისტინგი 27.

case ოპერატორის სათაურში ლოგიკური გამოსახულების ნაცვლად ფიგურირებს **სელექტორად** წოდებული ცვლადი. მანამდე პროგრამაში მას მიენიჭება რაიმე მნიშვნელობა. ეს შეიძლება იყოს მხოლოდ **ჩამოთვლადი ცვლადი. case** ოპერატორის შესრულების პროცესში ცვლადი-სელექტორის მნიშვნელობა უდარდება მასში აღწერილ სხვადასხვა ალტერნატივებს (ჭდე-მნიშვნელობებს). თანხვდენის პოვნისთანავე სრულდება მოცემული ჭდის კოდის ბლოკი და ხდება პროგ-

რამის ძირითად შტოზე გადასვლა. მნიშვნელობა-ჭდეები არის კონსტანტები, რომლებიც შეიძლება მნიშვნელობად მიიღოს სელექტორმა. მათი ტიპები და სელექტორის ტიპი მინიჭების მიხედვით შეთავსებადები უნდა იყოს. თანხვდენის არარსებობისას სრულდება **else** ბლოკი. ამ ბლოკის არარსებობის დროს (იგა არაა სავალდებულო) **case** ოპერატორში არავითარი ბლოკი არ სრულდება, როგორც ეს **ლისტინგ.28**-ზე მოყვანილ პროგრამაშია ნაჩვენები.

```
var n: integar;
begin
    write (‘ შეიტანე სკოლის კლასი ’)’;
     readln (n);

     case n of
        1..4: writeln (‘ უმცროსი  კლასები. ’);
         5..9: writeln (‘ საშუალო  კლასები. ’);
         10..11: writeln (‘ უფროსი კლასები. ’);
         12: writeln (‘ გამოსაშვები კლასი. ’);
          else writeln (‘ error ’)
      end

readln
end
```

ლისტინგი 28.

ამორჩევის ოპერატორის გამოყენებისას არსებობს შემდეგი ორი შეზღუდვა:

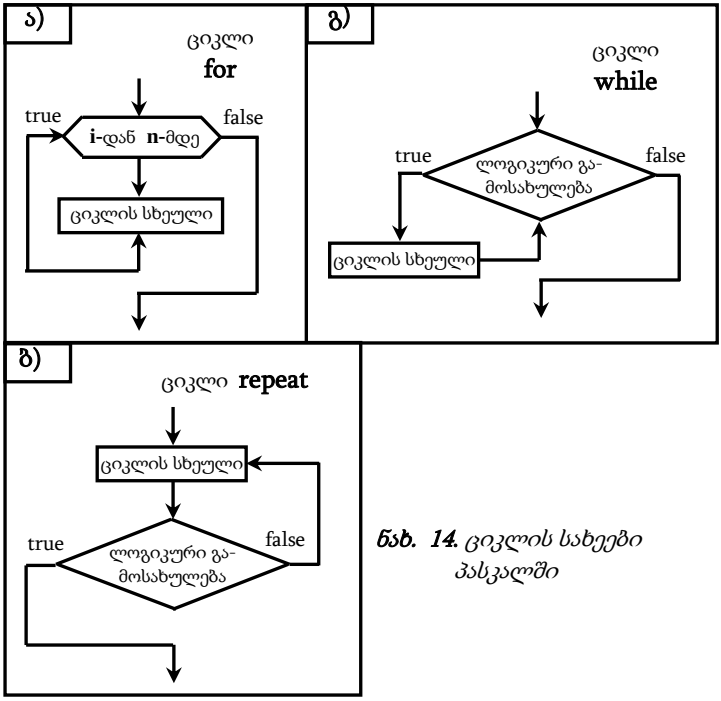
1. სელექტორს რაიმე რიგითული ტიპის უნდა იყოს;

2. ნებისმიერი ალტერნატივა უნდა იყოს კონსტანტა, დიაპაზონი, დიაპაზონთა სია. იგი არ უნდა იყოს ცვლადი მონაცემი ან გამოსახულება.

3.1.13.3. პასკალში გამოყენებული ციკლები

ამოცანების გადაწყვეტის დროს შესაძლებელია საჭირო გახდეს ერთი და იგივე მოქმედება რამდენჯერმე შესრულდეს. დაპროგრამებაში

კოდის ბლოკები, რომლებიც არაერთხელ უნდა იყოს გასამეორებელი, **ციკლებად** წოდებულ სპეციალურ კონსტრუქციებად გარდაიქმნება. ციკლში გამოიყოფა სათაური და სხეული. **სათაური** განსაზ-ღვრავს სანამდე ან რამდენჯერ გამეორდება ციკლის სხეული. **სხეული** შეიცავს გამოსახულებებს, რომლებიც მაშინ უნდა შესრულდეს, როდესაც ციკლის სათაურში გამოსხულება იძლევა ჭეშმარიტ (**True**, არანულოვან) მნიშვნელობას. სხეული როდესაც შეასრულებს უკ-ანასკნელ ინსტრუქციას, შესრულების ნაკადი ხელახლა უბრუნდება ციკლის სათაურს. ხელახლა მოწმდება ციკლის შესრულების პირობა. შედეგზე დამოკიდებულებით ციკლის სხეული მეორდება, ან ციკლის დასრულების შემდეგ ხდება ახალ გამოსახულებაზე გადასვლა.

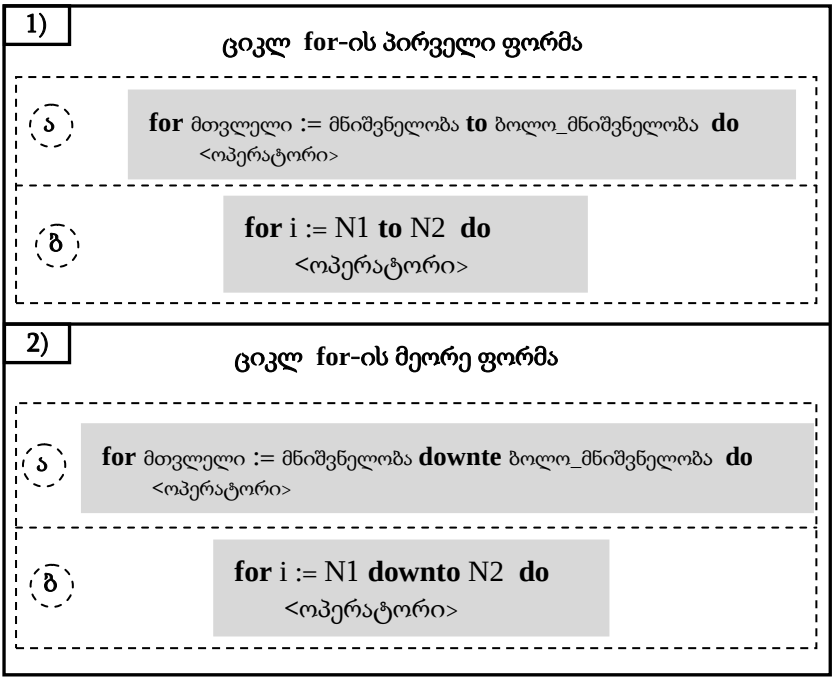


ნახ. 14. *ციკლის სახეები პასკალში*

დაპროგრამები ენა პასკალში არსებობს სამი სახის ციკლური კონსტრუქცია: **for** (ნახ.14,ა), **vhile** (ნახ.14,ბ) და **repeat** (ნახ.14,გ). განვიხილოთ ისინი.

ციკლი for

for ციკლს ხშირად **მთვლელიან** ან **პარამეტრიან** ციკლსაც უწოდებენ. იგი მაშინ გამოიყენება, როდესაც გამეორებათა რაოდენობა არ არის დამოკიდებული ციკლის სხეულზე. ეს იმას ნიშნავს, რომ ეს უკანასკნელი წინასწარ შეიძლება განისაზღვროს (თუმცა იგი არ გამოითვლება).



ნახ. 10. for-ციკლის ორი ფორმა

არსებობს **for** ციკლის ორი ფორმა (**ნახ. 10**). ***მთვლელი*** *შეიძლება იყოს ზემოთ განხილული ნებისმიერი (მთელი, ბულის, დიაპაზონური ან ჩამოთვლადი) ტიპის ცვლადი. საწყის და ბოლო მნიშვნელობად შეიძლება გამოყენებული იქნეს არა მარტო რიცხვები, არამედ*

გამოსახულებებიც. მათ შორის თუ დგას დამხმარე სიტყვა ***to****, მაშინ ციკლის თითოეულ ბიჯზე პარამეტრის მნიშვნელობა გაიზრდება ერთი ერთეულით, ხოლო თუ მოთავსებულია დამხმარე სიტყვა* **downto - მაშინ იგი შემცირდება ერთი ერთეულით.**

for ციკლის იტერაციის რაოდენობა ცნობილი ხდება მის შესრულებამდე, მაგრამ არა მთელი პროგრამის შესრულებიოს დაწყების წინ. მაგალითად, **ლისტინგ 29**-ზე მოყვანილ მაგალითში ციკლის შესრულების რაოდენობას განსაზღვრავს მომხმარებელი. მნიშვნელობა მიენიჭება ცვლადს, ხოლო შემდეგ გამოიყენება ციკლის სათაურში, მაგრამ როდესაც დაიწყება მისი გამოყენება, ციკლმა უკვე ზუსტად იცის თუ რამდენჯერ უნდა შესრულდეს იგი.

```
var
  i, n: integer;

begin
    write (‘ ნიშნების რაოდენობა ’);
     readln (n);

     for  i  := 1 to n do
        write ( ‘ (*) ‘);

readli
end.
```

ლისტინგი 29

ციკლი while არის წინაპირობიანი ციკლი. ციკლის სათაურში მოყვანილია ლოგიკური გამოსახულება. პროგრამის სხეული შესრულდება, თუ ჭეშმარიტია (**true**) ლოგიკური გამოსახულება, და არ შესრულდება საწინააღმსეგო შემთხვევაში, ე. ი. როდესაც ლოგიკური გა-მოსახულება მცდარია (**false**).

ციკლი როდესაც შესრულდება, პროგრამის შესრულება ხელახლა იწყება თავიდან. სხეულის შესრულების პირობა ხელახლა მოწმდება (განისაზღვრება ლოგიკური გამოსახულების მნიშვნელობა). ციკლის სხეული იმდენჯერ შესრულდება, სანამ ლოგიკური გამოსახულება გვაძლევს **true** მნიშვნელობას. ამიტომ ძალიან მნიშვნელოვანია, რომ ციკლის სხეულმა იმგვარად გაითვალისწინოს ციკლის სათაურში

არსებული ცვლადის ცვლილება, რომ აუცილებლად დადგება ისეთი სიტუაცია, რომ ლოგიკური გამოსახულებაა მიიღებს **false** მნიშვნელობას. საწინააღმდეგო შემთხვევაში უსასრულოდ განმეორდება პროგრამის შესრულება, ე. ი. პროგრამა **ჩაიციკლება**. ჩაციკვლა დაპროგრამებაში ერთ-ერთი ყველახე არასასურველი შეცდომაა (**ლისტინგი 30**).

```
var
  i, n: integer;

begin
    write (‘ ნიშნების რაოდენობა ’);
     readln (n);

     i := 1
     while i <= n do begin
         write (‘ (*) ‘);
          i := i + 1
      end

 readli
end.
```

ლისტინგი 30

ციკლი repeat

ზემოთ განხილული **while** ციკლი შეიძლება არასდროს არ შესრულდეს, თუ სათაურში არსებულმა ლოგიკურმა გამოსახულებამ მაშინვე მოგვცა **false** მნიშვნელობა. ასეთი სიტუაცია ყოველთვის ვერ იქნება მისაღები. არსებობს შემთხვევა, როდესაც ლოგიკური გამოსახულების ჭეშმარიტობისაგან დამოუკიდებლად აუცილებელია ციკლის სხეული ერთხელ მაინც შესრულდეს. ასეთ შემთხვევაში გამოიყენება **პოსტპირობიანი** ციკლი **repeat**.

repeat ციკლში ლოგიკური გამოსხულება დგას ციკლის სხეულის შემდეგ. ამასთანავე, **while** ციკლისაგან განსხვავებით აქ ყველაფერი შებრუნებითაა: **true** შემთხვევაში ხდება ციკლიდან გამოსვლა, ხოლო **false** შემთხვევაში - მისი შესრულება.

ლისტინგ 31-ზე მოყვანილი პროგრამა ერთ ვარსკვლავს მაშინაც კი დაბეჭდავს, როდესაც **n = 0**.

```
var
  i, n: integer;

begin
    write (‘ ნიშნების რაოდენობა ’);
     readln (n);

     i := 1
     repeat
        vrite (‘ (*) ‘);
        i := i +1
     until i > n;

readln
end.
```

ლისტინგი 31

3.1.13.4. goto, break, continue და პროგრამის შეწყვეტის ოპერატორები

ოპერატორი goto

პასკალი მიეკუთვნება სტრუტურულებულ ენათა ჯგუფს. ამის მიუხედავად მისთვის დამხასიათებელია დაპროგრამების ენების განვითარების ადრეული ეტაპებისათვის დამახასიათებელი თავისებურებები. ამ დროს ჯერ კიდევ არ იყო ჩამოყალიბებული ლოგიკურად დაკავშირებული ბლოკებისაგან ფორმირებული სისტემის სახით პროგრამის ჩამოყალიბების იდეა. ამიტომ პროგრამის წრფივი განვითარებდან გადახრის საჭიროების წარმოშობისას დამპროგრამებლები იყენებდნენ უპირობო გადასვლის **goto** ოპერატორს.

მოგვიანებით დამპროგრამებლებმა უარი თქვეს **goto** ოპერატორის რეგულარულ გამოყენებაზე, მაგრამ ზოგჯერ სასარგებლო ხდება მისი გამოყენება.

აუცილებელია ვიცოდეთ, რომ ყოველთვისაა შესაძლებელი თავი ავარიდთ **goto** ოპერატორს. მათი გამოყენება ართულებს პროგრამის წაკითხვასა და გაგებას.

goto ოპერატორით გადასვლა ხდება სპეციალური ჭდით მონიშნულ ოპერატორზე, რომელიც თავად ოპერატორისაგან ორწერტილითაა გამოყოფილი. ჭდედ შეგვიძლია გამოვიყენოთ ოთხზე მეტი ციფრისაგან შედგენილი ნებისმიერი მთელი უნიშნო რიცხვი ან ნებისმიერი სახელი. ჭდე რომ გამოვიყენოთ, საჭიროა იგი აუცილებლად გამოვაცხადოთ პროგრამის აღწერითი ნაწილში არსებულ ჭდეების განყოფილობაში. ეს განყოფილება იწყება სიტყვით label, რომლის შემდეგაც აუცილებლად ორწერტილი უნდა იყოს დასმული.

მონიშნულ ოპერატორზე გადასასვლელად გამოიყენება გადასვლის **ლისტინგ 32**-ზე ნაჩვენები სახის ოპერატორი.

```
label goback
var num: real;

begin
   goback:
   write (‘ შეიტანე რიცხვი: ‘);
   readln (num);

   if num < 0 then
     goto goback;

  num := sqrt (num);

  write (‘ კვადრატული ფესვი:  ‘, num: 5:2);

readln
end.
```

ლისტინგი 32

ოპერატორები break და continue

ზოგჯერ ხდება, რომ ციკლის შესრულების მიზანი ამ ციკლიდან გამოსვლის მომენტის დადგომამდე მიიღწევა. ამაგალითად იმის დასადგენად, რომ მოცემული **n** რიცხვი მარტივია თუ არა, ციკლი ზოგადად (**n div 2-1**)-ჯერ უნდა შესრულდეს; ხოლო ის, რომ რიცხვი მარტივი არ არის, შეგვიძლია ციკლის საწყის ნაბიჯებზეც დავად-

გინოთ. ბიჯების რაოდენობის შესამცირებლად შეგვიძლია გამოვიყენოთ **goto** ოპერატორები ან ჩამოვაყალიბოთ ციკლის შეწყვეტის რთული პირობები. მაგრამ ციკლის შესრულების პროცესის შეწყვეტისათვის არსებობს სპეციალური **break და continue** ოპერატორები. მათგან **break** ოპერატორი წყვეტს ყველა შესაძლო, ხოლო **continue** ოპერატორი - მხოლოდ მიმდინარე იტერაციას.

break და continue ოპერატორები სრულდება ნებისმიერ ცნობილ (**repeat, while, for**) ციკლში და გამოიყენება მხოლოდ შინაგანი იტერრაციისათვის. მაგალითად, ორმაგი ციკლიდან იძულებითი გამოსვლისათვის **break** ოპერატორი უნდა განვათავსოთ როგორც შინაგან, ისე გარეგან ციკლში. **break და continue** ოპერატორები თავისი არსით ცნობილი წერტილიანი **goto** ოპერატორის სახესხვაობაა.

ლისტინგ 33-ში მოყვანილ ალგორითმში მომხმარებლისაგან ხუთჯერ მხოლოდ მაშინ მოითხოვება რიცხვის შეტანა, თუ იგი არ შეიტანს ნულს.

```
var
   num: real;
   i: integer;

begin
   for i := 1 to 5 do begin
      vrite (‘ შეიტანე რიცხვი: ‘);
      readln (num);
      if num = 0 then
         break;
      writeln (num)
    end

readln
end
```

ლისტინგი 33

ლისტინგ 34-ში მოყვანილ მაგალითში მოითხოვება ხუთი რიცხვი და მათ ჩამონათვალში არსებული მხოლოდ დადებითი რიცხვები ჯამდება.

```
var
   num, sum: real;
   i: integer;

begin
   sum := 0;

   for i := 1 to 5 do begin
      write (‘ შეიტანე რიცხვი: ‘);
      readln(num);
      if num < 0 then
         continue;
      sum := sum + num
   end;

   vrite (sum: 10:2);

readln
end.
```

ლისტინგი 34

პროგრამის იძულებითი შეწყვეტა

პროგრამა ჩვეულებრივ მხოლოდ ბოლო (ე. ი. წერტილიან **end**) ოპერატორთან მიღეწევის შემდეგ მთავრდება. როდესაც სადღაც პროგრამის შიგნით ხდება საჭირო მისი შეწყვეტა, მაშინ შეგვიძლია ვისარგებლოთ **halt** პროცედურით. იგი შეგვიძლია გამოვიძახოთ მრგვალ ფრჩხილებში მოთავსებული **0**-დან **255**-მდე მთელი არაუარყოფითი რიცხვის სახის პარტამეტრის მოცემით. ეს მნიშვნელობა ოპერაციულ სისტემაში შეცდომის (**ERRORLEVEL**) კოდის სახით ბრუნდება და მოცემული პროგრამის საკომანდო ფაილიდან ამუშავების შემთხვევისას შეიძლება **DOS**-ში გაანალიზდეს. **halt** პროცედურაში პარამეტრის არარსებობა შეესაბამება პარამეტრის 0-თან ტოლობას (პროგრამის ნორმალური დამთავრება).

მეორე პროცედურა, რომლის დახმარებითაც შეგვიძლია შევწყვიტოთ პროგრამის შესრულება, არის პროგრამის შემსრულებელ ნაწილში უპარამეტრო **exit** პროცედურის განთავსება. უფრო ხშირად ეს

პროცედურა გამომძახებელი პროგრამის შეუწყვეტლად ქვეპროგრამიდან გამოსვლის დროს გამოიყენება.

ფსევდოშემთხვევითი რიცხვების გენერატორი

არსებობს შემთხვევები, როდესაც მოითხოვება, რომ პროგრამის მუშაობის შედეგი გარკვეულ ფარგლებში იყოს შემთხვევითი. ასეთი შესაძლებლობის რეალიზებისათვის დაპროგრამების ბევრ ენაში არსებობს ჩაშენებული ფუნქციები, რომელთა კოდი წარმოქმნის რიცხვებს, რომლებსაც **შემთხვევით რიცხვებად** მიიჩნევენ. სინამდვილეში შეუძლებელია ხელოვნურად შემთხვევითობის რეალიზება. ჩვეულებრივ აიღება გარკვეული კოეფიციენტი და მისი საშუალბით გამოითვლება თითოეული მომდევნო „შემთხვევითი“ რიცხვი. ამდენად ამგვარად მიღებულ რიცხვებს **ფსევდოშემთხვევითი რიცხვები უნდა ეწოდოს**.

დაპროგრამება ენა პასკალში დასახულ დიაპაზონებში ფსევდოშემთხვევითი რიცხვების გენერირებისათვის გამოიყენება ფუნქცია **random** (ინგ. random - შემთხვევითი). მისი გამოყენების წინ ხდება შემთხვევითი რიცხვების გადამწოდის ინიციალიზება, რასაც **randomize** (**რანდომიზების**) ანუ **გაშემთხვევითობის პროცედურა** ეწოდება. რანდომიზების გარეშე პროგრამა ერთსა და იმავე შედეგს ჩამოაყალიბებს. რანდომიზების დროს დაისახება რიცხვების საწყისი მიმდევრობა, რომლისგანაც გამოითვლება ყველა დანარჩენი მიმდევრობები. პროგრამის ყოველი ამუშავების დროს მიიღება რიცხვების სხვადასხვა მიმდევრობა, ე. ი. **random** ფუნქციის მუშაობის შედეგები იქნება სხვადასხვანაირი.

random ფუნქცია შემთხვევით რიცხვებს წარმოქმნის **0**-იდან (**0**-ის ჩათვლით) ფრჩხილში მითითებულ რიცხვამდე (ამ რიცხვის ჩაუთვლელად) დიაპაზონში. ასე რომ, **random (10)** გამოსახულება ნიშნავს, რომ **[0, 10)** დიაპაზონიდან მიიღება ნებისმიერი რიცხვი. რაიმე ისეთი სხვა დიაპაზონიდან რიცხვების მიღებას, რომელიც არ იწყება 0-დან მაშინ მათემატიკურ ხრიკს მიმართავენ. მაგალითად, შემთხვევითი რიცხვი რომ **-100**-დან **100**-მდე დიაპაზონიდან მივიღოთ, საკმარისია ჩავწეროთ ასეთი გამოსახულება:

random (200) – 100.

ამის შედეგად რიცხვები ჯერ მიიღება **[0, 199)** დიაპაზონიდან, ხოლო შემდეგ მათ გამოაკლდება **100**. ამორჩეული შემთხვევითი რიცხვი თუ **100**-ზე ნაკლები აღმოჩნდება, მაშინ შედეგი იქნება უარყოფითი.

```
var n, i, x: integer;

bedin
    randomize;

   n := random(7) + 5;

  for i := 1 to n do begin
     x := random(100) – 50;
     write (x:5)
  end;

readlin
end.
```

ლისტინგი 34

ლისტინგ 35-ზე მოცემულ პროგრამაში **randomize** პროცედურის დახმარებით **n** ცვლადს შემთხვევითი მნიშვნელობა მიენიჭება **[5, 12)** დიაპაზონიდან. **n** ცვლადის მნიშვნელობა გამოიყენება **for** ციკლის იტერაციის რაოდენობის განსაზღვრისათვის. **for** ციკლში შემთხვევითი რიცხვები **[0, 50)** დიაპაზონიდან იქნება გენერირებული და ეკრანზე გამოტანილი.

3.2. დაპროგრამების ენა Q a s i k (Q ბეისიკი)

3.2.1 ისტორიული ექსკურსი

სახელწოდება „Basic“ არის წინადადების „**B**eginner's **A**ll-purpose **I**nstruction **C**ode” აბრევიატურა“, რომლის შინაარსი („სიმბოლური ბრძანებების მრავალმიზნობრივი ენა დამწყებებისათვის“ ზუსტად გადმოსცემს დაპროგრამების მოცემული ენის არსს. ბეისიკისათვის ძირითადი თავისებურება არა მისი სიმარტივეა, არამედ ის, რომ მან შესაძლებელი გახადა ამოცანები კომპიუტერთან დიალოგის რეჟიმში გადაგვეწყვიტა. აღნიშნულიდან გამომდინარე, შეგვიძლია სხვაგვარადაც შევაფასოთ ცნობილი ინფორმატიკოსის, ტიურინგის პრემიის ლაურეატისა (**1972** წ.) და „კომპიუტერული პიონერის“ საპატიო წოდების (**1980** წ.) მფლობელის, სტრუქტურული დაპროგრამების ერთ-ერთი შემქმნელის, **ედსგერ დეიკსტრას** ცნობილი გამონათქვამი იმის შესახებ, რომ „სტუდენტებს, რომლებმაც ადრე შეისწავლეს **ბეისიკი**, პრაქტიკულად შეუძლებელია ასწავლოთ კარგი დაპროგრამება. რამდენადაც პოტენციურმა დამპროგრამებლებმა ამით შეუქცევადი გონებრივი დეგრადაცია განიცადეს“. მისი ასეთი შეფასება იმან განაპირობა, რომ იგი იყო **არასტრუქტურირებული ენა**, მაგრამ იგი არ შეძლება ბეისიკის სასიკვდილო განაჩენად მივიჩნიოთ, მით უფრო, რომ მასზე არანაკლები დარტყმა დეიკსტრისაგან მიიღო დაპროგრამების არა ნაკლებ მნიშვნელოვანმა მეორე ენა **კობოლმაც,** რომელზედაც იგი წერდა, რომ „**კობოლზე** დაპროგრამება ასახიჩრებს ტვინს, ამიტომ მისი სწავლება დანაშაულად უნდა იქნეს მიჩნეული“.

ზემოთ აღნიშნულის საპირისპიროდ ბეისიკის ღირსებად უდავო ღირსებად უნდა ჩაითვალოს უნდა ჩაითვალოს ის ფაქტი, რომ **1960**-იანი წლების ბოლო პერიოდში ბეისიკი იყო მაღალი დონის **ერთადერთ საერთო დანიშნულების ენა**. ამ ფაქტმა ხელი შეუწყო იმას, რომ მომდევნო წლებში დამუშავებული იქნა მისი მრავალი მოდიფიკაცია. ავტორიტეტის დამკვიდრების სფეროში კი შემობრუნების მომენტად

გადაიქცა **Visual Basic**-ს გამოჩენა. მანამდე მისვლამდე ბეისიკმა გაიარა საკმაოდ საინტერესო გზა, რომლის ცოდნა უდავოდ გაზრდის მკითხველის თვალსაწიერს.

1950-იანი წლების შუა პერიოდში დაპროგრამებაში მანქანური ენების გამოყენების როლმა შემცირება დაიწყო. გამოჩნდა უფრო მაღალი დონის ენები - **Fortran, Algol, Cobol** და ა. შ.

ზემოთ ჩამოთვლილი ენებიდან პირველ და ყველაზე გავრცელებული იყო **1954** წელს **IBM** ფირმის დამპროგრამებელთა ჯგუფის მიერ შექმნილი ენა **Fortran-**ის პირველი ვერსია (სახელი მიღებულია **FOR**mula **TRAN**s-დან, რაც „ფორმულების მთარგმნელს“ ნიშნავს). იგი არ იყო საერთო მოხმარების და ორიენტირებული იყო მათემატიკური ხასიათის სამეცნიერო-ტექნიკური გაანგარიშებებისათვის.

დაპროგრამების ენა **Algol**-ის (**ALGO**rithmic **L**anguage, რაც „ალგორითმულ ენას“ ნიშნავს) პირველი ვერსია გამოჩნდა **1958** წელს. როგორც მისი სახელწოდებიდან ჩანს იგი **ალგორითმების ჩასაწერად** იყო გამიზნული. ზუსტი ლოგიკური სტრუქტურის წყალობით იგი სამეცნიერო და ტექნიკურ ლიტერატურაში ალგორითმების ჩაწერის სტანდარტულ საშუალებად გადაიქცა.

ზემოთ განხილულ პერიოდში მაღალი დონის ენებისათვის საგნობრივი ორიენტაცია იყო დამახასიათებელი. გარდა ამისა, მსგავსი ენები ძალიან რთული იყო არაინჟინრული მიმართულების სპეციალისტებისათვის.

მასობრივი გამოყენებისათვის განკუთვნილი დაპროგრამების ენის შექმნის იდეას უკავშირებენ **ინგლისელი მისიონერის ცნობილ ისტორიას**.

XIX საუკუნეში ერთ ინგლისელ მისიონერს, რომელიც ცდილობდა გაეადვილებინა ინგლისის მიერ დაპყრობილ ქვეყანაში მცხოვრებ მკვიდრ მოსახლეობასთან კონტაქტი, თავში მოუვიდა გენიალური იდეა: ინგლისური ენიდან გამოეყო ყველაზე მარტივი მისი ნაწილი, რომელსაც პრაქტიკულად არ სჭირდებოდა გრამატიკა და რომელშიც შევიდოდა **300**-ზე არაუმეტესი რაოდენობის ყველაზე გავრცელებული სიტყვა. ინგლისური ენის ამ შეკვეცილ ქვესახეს ეწოდა **Basic English**. როგორც პრაქტიკამ გვიჩვენა, იგი მართლაც ძალიან ადვილად შესათვისებელი აღმოჩნდა და მან პოპულარობა მოიპოვა არა მხოლოდ კოლონიური ქვეყნების მკვიდრი მოსახლეობისათვის, არამედ

ინგლისში მცხოვრები ემიგრანტებისათვისაც, რომლებისთვისაც ინგლისური არ იყო მშობლიური ენა.

ასი წლის შემდეგ ანალოგური გზა აირჩია **დარტმუნდტის** (აშშ) კოლეჯის მათემატიკური ფაკულტეტის თანამშრომლებმა. ისინი ცდილობდნენ დაპროგრამების „გამარტივებული“ ენის მეშვეობით **გაემარტივებინათ ურთიერთკავშირი კომპიუტერსა და არაპროფესიონალ მომხმარებელს შორის**.

1964 წელს **თომას კურცმა** *Thomas Eugene Kurtz, 1928, 91 წლის*)და **ჯონ კამენიმ** (*John George Kemeny, 1926 - 1992*) შექმნა **ინგლისური ენის მარტივი სიტყვებისაგან შემდგარი** დაპროგრამების სპეციალიზებული ენა. ამ ენას ეწოდა **Basic**.

ისინი ამ ენას სტუდენტებისათვის დაპროგრამების ენის სწავლებისათვის იყენებდნენ. ენა იმდენად მარტივი და გასაგები აღმოჩნდა, რომ მისი გამოყენება სხვა სასწავლო დაწესებულებებშიც დაიწყეს.

Basic-ზე სერიოზული გავლენა მოახდინა **Fortran II**-მ და **Algol 60**-მა. გარდა ამისა, შემქმნელებმა - **Basic**-ში მოახდინეს დროითი განაწილების რეჟიმის მქონე სისტემასთან მუშაობის, აგრეთვე ტექსტის დამუშავებისათვის და მატრიცული არითმეტიკის გამოსაყენებლად საჭირო მექანიზმების რეალიზაცია. თავდაპირველად, **Basic**-ის ენის კომპილატორი შეიქმნა მეინფრეიმ **GE-265**-თვის.

1970-იან წლებში გამოჩნდა უფრო კომპაქტური პერსონალური კომპიუტერები. ეს გარემოება წარმატებით გამოიყენეს ფირმა **Microsoft**-ის დამმფუძნებლებმა **ბილ გეიტსმა** და **პოლ ალენმა.** მათ პირველი **MITS Altair** კომპიუტერისათვის შექმნეს **Basic**-ის ახალი ვერსია, რომელსაც შეეძლო ემუშავა **4** კბ მოცულობის ოპერატიული მეხსიერების შემთხვევაში. შემდგომში ეს ვერსია გადაიქცა დაპროგრამების მსოფლიოში ერთ-ერთ ყველაზე პოპულარულ ენად. ალბათ, ამიტომ ბილ გეიტსი თავის ტიტულებს დღესაც უმატება წოდებას „**ბეისიკ-დამპროგრამებელი**“.

საწყისი **Basic**-ის ყველაზე მეტად გაფურჩქვნისა და განვითარების პერიოდად შეიძლება ჩაითვალოს **1970**-იანი წლების ბოლო პერიოდი და **1980**-იანი წლების დასწყისი. მისი ვერსიები გაჩნდა სხვა პლატფორმებზეც და მისი სხვადასხვა ვერსიები მილიონობით იყიდებოდა კომპიუტრულ ბაზარზე. ერთ-ერთი მისი ყველაზე პოპულა-

რული ვერსია იყო **Applesoft Basic,** რომელიც **Apple II**-თვის სტანდარტულ ენად გადაიქცა.

ოპერაციული სისტემა **CP/M**-თვის შეიქმნა **Basic-80** ვერსია, რომელმაც დიდი ხნის განმავლობაში განსაზღვრა ენის განვითარება.

იგი სტანდარტულ ენას წარმოადგენდა მოგვიანებით გამოსულ **IBM PC** კომპიუტერებისთვისაც, ოღონდ **GW-Basic** ვერსიის სახით.

1980-იან წლებში **Basic**-ის გამოყენება დაიწყეს დაპროგრამებადი კალკულატორების რთულ მოდელებშიც. იგი იყო გამოყენებული ჩემი ახალგაზრდობს დროს ჩვენთან მეტად პოპულარულ კალკულატორ „**Электроника МК-85**“-შიც.

1985-90 წლებში ***Microsoft***-მა დაამუშავა ***QuickBasic*** (ინგ. ***Quick –*** „სწრაფი“) რომელმაც დიდი წარმატება მოიხვეჭა. მასში რეალიზებულია ტრადიციული ბეისიკ-ტექნოლოგიებისა და რთული პროგრამების სისტემათა შექმნის კლასიკური მეთოდების შერეული გამოყენების სქემა. ***1990*** წლიდან გამოჩნდა ***QuickBasic***-ის **შეკვეცილი ვარიანტი,** რომელმაც მიიღო ***QBasic***-ის სახელწოდება ***MS-DOS*** სისტემაში. უნდა შევნიშნოთ, რომ ***Qbasic*** **და** ***QuickBasic*** **ერთი და იგივე სისტემა არ არის.**

1991 წელს პოპულარული გახდა სურვილი იმისა, რომ ნებისმიერ დამწყებ დამპროგრამებელს შეძლებოდა ***Windows***-ისათვის თავად შეექმნა საკუთარი პროგრამა. სწორედ ამ სურვილის რეალიზების მიზნით გამოჩნდა ახალი ინსტრუმენტული საშუალებების პირველი ვერსია ***Microsoft Visual Basic (VB).*** იმ მომენტში თვით ამ ვერსიის ავტორი ***Microsoft***-იც კი საკმაოდ მოკრძალებულად აფასებდა ამ სისტემის შესაძლებლობებს და მას მიიჩნევდა მხოლოდ დამწყები და არაპროფესიონალი დამპროგრამებლებისათვის ორიენტირებულ სისტემად. იმ პერიოდში თვით გამოცდილი დამპროგრამებლისათვის პრობლემა იყო ახალ ***Windows*** გარემოში დაპროგრამება. ამიტომ კომპანიამ მიზნად დაისახა ამ პრობლემის გადასაწყვეტად დაემუშავებინა აღნიშნულ გარემოში მუშაობის მარტივი და მოსახერხებელი ინსტრუმენტი. ამიტომ ***VB***-ს ***1,0*** ვერსია დამუშავების მომავალი გარემოს მაკეტს უფრო ჰგავდა, ვიდრე მუშა ინსტრუმენტს. მიუხედავად ამისა უკვე მაშინაც ***VB***-ს **პრინციპული სიახლე** იყო ***Windows***-ის გარემოში ხდომილობით მართვადი და ვიზუალური დაპროგრამების იდეებს რეალიზება, რომლებიც რადიკალურად განსხვავდებოდა პროგრამების დამუშავების კლასიკური სქემებისაგან. საყოველთაო აღიარებით ***VB***

სათავე დაუდო ინსტრუმენტების ახალ თაობას, რომელსაც დღეს უწოდებენ **პროგრამათა სწრაფად დამუშავების საშუალებებს** (***Rapid Application Development, RAD***). დღეს ეს ჩვეული იდეოლოგიაა, მაშინ კი იგი სრულიად ახალ იდეოლგიად ეჩვენებოდათ, რაც ძველი ყაიდის დამპროგრამებლებს სერიოზულ (მათ შორის, წმინდა ფსიქოლოგიური ხასიათის) პრობლემებს უქმნიდა.

ზემოთ აღნიშნულის მიუხედავად სწრაფად იზრდებოდა ***VB***-მომხმარებელთა რაოდენობა, რამაც მნიშვნელოვანწილად შეუწყო ხელი მისი წინაპარ ***QuickBasic***-ის ვეებერთელა პოპულარობამ. ამის წყალობით ***VB*** ძალიან სწრაფად „ვაჟკაცდებოდა“ და ***1917*** წელს უკვე ***VB 15.0*** ვერსიამაც იხილა დღის სინათლე; ამასთანავე, როგორც ანალიზი გვიჩვენებს, მზის ჩასვლა ჯერ კიდევ შორსაა.

1993 წელს ***Microsoft***-მა განაცხადა თავისი სურვილი ***VB***-ეს საფუძველზე დაემუშავებინა დაპროგრამების უნივერსალური სისტემა გამოყენებითი პროგრამების პროგრამებისათვის, რომელსაც ეწოდა ***Visual Basic for Applications***, ანუ ***VBA***. განვლილ პერიოდში დამუშავეთბული იქნა მისი მრავალი ვერსია და ბოლოს ***2010*** წელს დღის სინალე იხილა ***VBA 7.0*** ვერსიამ. დღეისთვის იგი გამოიყენება ექვსივე საოფისე (***Word, Excel, PowerPoint, Access, Outlook***) პროგრამაში.

საოფისე პროგრამებში რეალიზებული ***VBA***-ს მექანიზმების ათვისებით თქვენს წინაშე გაიხსნება ათობითა და ასობით პროგრამებთან მუშაობის დროს მიღებული ცოდნისა და ჩვევების ფართოდ გამოყენების შესაძლებლობები. უმარტივესი მიკრობრძანებების შედგენის დაწყების შემდეგ სურვილის შემთხვევაში ერთი ინსტრუმენტარიუმის ფარგლებში ჩარჩოებში შეძლებთ, დაამუშაოთ ნებისმიერი სირთულის პროგრამული სისტემები. დღეს არსებული დამპროგრამებელთა აბსოლუტური უმრავლესობა ერთ-ერთ ინსტრუმენტად იყენებს ***VB***-ს ან ***VBA***-ს.

ჩვენ ყურადღებას მხოლოდ ***QuickBasic***-ის შეკეცილ ***Qbasic*** ვარიანტზე შევაჩერებთ. მისი შესწავლით თქვენ დაეუფლებით შემდგომი დამოუკიდებელი მუშაობისათვის საჭირო თეორიულ ბაზისს.

Qbasic გარემო ხელმისაწვდომია ***IBM*** შეთავსებადი ნებისმიერი მოდელისათვის - იგი დისკზე იკავებს სულ რაღაც ***32*** კილობაიტის ტოლ მოცულობას. იმავე დროს, აქვს რა საკმაოდ დიდი შესაძლებლობები, ენა ადვილად ასათვისებელია და საშუალებას გვაძლევს შევადგინოთ გამოყენებითი ხასიათის სერიოზული რეალური ამო-

ცანების გადასაწყვეტ პროგრამათა კომპლექსები. იგი შეგვიძლია განვიხილოთ **ბეისიკის ენის პროფესიონალურად ორიენტირებული ვერსიებისთან დამაკავშირებელ ხიდად.**

მოცემულ თავში მოყვანილია ენის კონსტრუქციების ის მინიმალური (ბაზისური) ნაკრები, რომლის ცოდნა აუცილებელია დაპროგრამების სწავლების საწყის ეტაპზე. მკითხველი რომ ჩაწვდეს დაპროგრამების პროცესის არსს, საჭიროა ამ ეტაპზე იგი არ უნდა გადავტვირთოთ ზედმეტი ინფორმაციით.

3.2.2 ენის ალფაბეტა

ენაში სიმბოლოებად გამოიყენება ლათინური ასოების მთავრული ასოები ***A***-დან ***Z***-მდე, არაბული **0, 1, ..., 9** ციფრები.

გარდა ზემოთ აღნიშნულისა, ბეისიკის ენაში გამოიყენება სპეციალური სიმბოლოები, რომ რომლებიც ***4.1*** ცხრილშია მოყვანილი.

ცხრილი 1.1. ენა ბეისიკის სპეციალური სიმბოლოები

+	მიმატება	<>	არატოლია
-	გამოკლება	.	წერტილი
*	გამრავლება	,	მძიმე
/	გაყოფა	;	წერტილ-მძიმე
^	ახარისხება	:	ორი წერტილი
\	მთელრიცხვული გაყოფა (*QBasic*)	!	ნამდვილი სიდიდის ნიშანი
MOD	მოდულით გაყოფა (*QBasic*)	#	ორმაგი სიზუსტის ნამდვილი სიდიდის ნიშანი
=	ტოლია	%	მთელი სიდიდის ნიშანი
>	მეტია	&	გრძელი მთელი სიდიდის ნიშანი (*QBasic*)
<	ნაკლებია	$	ტექსტური სიდიდის ნიშანი
>=	ნაკლებია ან ტოლია	( )	მრგვალი ფრჩხილები
<=	მეტია ან ტოლია	"	ბრჭყალები

4.1 ცხრილში ჩამოთვლილი სიმბოლოების გარდა ბეისიკის ენაში გამოიყენება ზოგიერთი ინგლისური სიტყვა, როგორიცაა, მაგალითად, ***LET*** (*ნების დართვა*)**, GO TO** (*გადასვლა -ზე*) და სხვები. მათ საჭიროების კვალობაზე განვიხილავთ. ბეისიკის ენაში გამოიყენება აგრეთვე სხვა სალაპარაკო ენების ალფაბეტას ასოები, ოღონდ მხოლოდ ტექსტურ კონსტანტებში.

3.2 3 მონა - ცემები

ბეისიკში გამოყენებული მონაცემები იყოფა კონსტანტებად (მუდმივებად) და ცვლადებად. ამასთანავე ცვლადებს ყოფენ მარტივ ცვლადებად და მასივებად.

4.2 ცხრილში მოყვანილია საკვანძო ცნობები მონაცემების შესახებ.

ცხრ. 4.2. *საკვანძო ცნობები მონაცემების შესახებ (დასაწყისი)*

საკვანძო ცნობები	მაგალითები
კონსტანტები	
ბეისიკში გამოიყენება რიცხვითი და ტექსტური კონსტანტები. **რიცხვითი კონსტანტა** პროგრამაში ჩაიწერება კონკრეტული რიცხვის სახით. ასეთი კონსტანტები იყოფა მუდმივ და მთელ კონსტანტებად.	
ნამდვილი კონსტანტა, რომელიც შედგება ნიშნიანი ან უნიშნო ციფრების მიმდევრობისა და წერტილისაგან შეიცავს წერტილს, ან მთავრდება სიმბოლოთი „!“. წერტილი ერთმანეთისაგან განაცელკევებს მთელ და წილადურ ნაწილს. კონსტანტაში არსებობს **7**-ზე არაუმეტესი რაოდენობის ციფრი.	87 -57.27 5296!
ასეთი ციფრები კომპიუტერში ჩვეულებრივ წარმოიდგინება გარკვეული თუმცა ძალიან მცირე) ცდომილობით.	4.67E5 (ე.ი. $4.67 \cdot 10^7$ = = 46 700 000).
ბეისიკში ნამდვილი რიცხვები შეიძლება წარმოვადგინოთ **ექსპონენციალური** სახითაც - მცურავი წერტილიანი რიცხვის სახით.	
მაშასადამე **კონსტანტას ნამდვილობის ნიშნებია:**	
1) ნიშანი ! ; **2) E** ასო ჩაწერის ექსპონენციალურ ნაწილში; 3) რიცხვის ნებისმიერი ჩანაწერი, რომელშიც ტიპი არ არის მითითებული.	765! -1,584E07
QBasic-ში რიცხვით კონსტანტებს, რომლებში ციფრების რაოდენობა **7**-ს არ აღემატება უწოდებენ **ერთმაგი სიზუსტის ნამდვილ რიცხვს**. მასში არსებობს 17 ციფრანი ნამდვილი კონსტანტები. მათ **ორმაგი სიზუტის ნამდვილი კონსტანტები** ეწოდება. ასეთი კონსტ-	

დასასრული იხ, მეორე გვერდზე

3.2.4. ჩაშენებული ფუნქციები

კომპიუტერზე მრავალი გამოყენებითი და ამოცანის გადაწყვეტის დროს ხშირად წამოიჭრება სხვასხვადასხვა ელემენტალური მათემატიკური *ფუნქციის (სინუსის, კოსინუსის, ლოგარითმის და ა. შ. გამოთვლის აუცილებლობა, რაც მოითხოვს ერთსა და იმავე პროგრამის მრავალჯერადად შე-*

სრულებას. ამის თავიდან ასაცილებლად ყველაზე გავრცელებული ფუნქცი-

ცხრ. 4.2. *საკვანძო ცნობები მონაცემების შესახებ (დასასრული)*

საკვანძო ცნობები	მაგალითები
ანტის ნიშნებია; **1)** რიცხვის ბოლოში ნიშანი #; **2) D** ასო ჩაწერის ექსპონენციალურ ნაწილში. *QBasic*-ში ტიპის მითითების გარეშე წერტილიანი რიცხვი ითვლება ერთმაგი სიზუსტის მთელ რიცხვად. **მთელი კონსტანტა** წარმოადგენს ათობითი ციფრების ნაკრებს, რომლის წინ მითითებულია ან არაა მითითებული ნიშანი, ხოლო ბოლოში მოთავსებულია სიმბოლო % . იგი მოთავსებულ უნდა იყოს დიაპაზონში -32 768% -დან 32 767-მდე. *QBasic*-ში გამოიყენება **გრძელი მთელი კონსტანტა**, რომელიც მოთავსებულია დიაპაზონში -2 147 483 647- -დან 2 147 483 648-მდე და მთავრდება სიმბოლოთი &. მთელი კონსტანტები კომპიუტერში წარმოიდგინება ზუსტად. **ტექსტური (სტრიქონული) კონსტანტა** არის ენის ალფაბეტას სიბოლოების ბრჭყალებში ჩასმული მიმდევრობა. ტექსტური კონსტანტის სიგრძე არ უნდა აჭარბებდეს 255 სიმბოლოს. *QBasic*-ში სიმბოლოების რაოდენობა 32 567-მდეა გაზრდილი	573.43# 23 418 367.1428# -1.346276574D05 128% -3592% 8725 674 287& „WELL” „ХОРОШО” “Y = AX + BZ + C“
ც ვ ლ ა დ ე ბ ი	
ცვლადი არის სიდიდე, რომელიც პროგრამის შესრულების დროს შეიძლება შეიცვალოს. მას აღვნიშნავთ სახელით, რომელსაც **იდენტიფიკატორს** ვუწოდებთ. იდენტიფიკატორი (ცვლადის სახელი) შედგება ლათინური ასოებისა და ათობითი ციფრებისაგან; იგი ბოლოვდება მისი ტიპის განმსაზღვრელი სუფიქსით (ბოლოსართით). იდენტიფიკატორის შემადგენელი სიმბოლოების მაქსიმალური რაოდენობა აღინიშნება ***L*** -ით. *QBasic*-თვის ***L***=40. ნებისმიერი ტიპის ცვლადს შეუძლია მიიღოს მხოლოდ ის მნიშვნელობა, რომელიც დასაშვებია შესაბამისი ტიპის კონსტანტისათვის. *QBasic*-ში, გარდა ამისა, გამოიყენება **ორმაგი სიზუსტის ნამდვილი ცვლადები** (მისი სუფიქსია #) სუფიქსების არსებობისას ცვლადი ითვლება ნამდვილ ცვლადად. *QBasic*-ში კიდევ ერთი ტიპის ცვლადი გამოიყენება. ესაა - გრძელი მთელი (მისი სუფიქსია &).	*AD#* DAX245# *LXBB145AAA&*

ების გამოთვლის პროგრამები ჩაწერილია კომპიუტერის მეხსიერებაში, პროგრამათა ბიბლიოთეკებში, ხოლო ეს ფუნქციები შეტანილია დაპროგრამების ენათა შემადგენლობაში. ამასთანავე, მაქსიმალურადაა გამარტივებული მათთან მიმართვა. ასეთ ფუნქციებს **ჩაშენებული მათემატიკური ფუნქციები** ეწოდება. დაპროგრამებათა ენებში, მათ შორის, ბეისიკში, ჩაშენებული მათემატიკური ფუნქციების გარდა გამოიყენება **ჩაშენებული ტექსტური ფუნქციებიც. *4.3*** ცხრილში მოყვანილია ბეისიკის პრაქტიკულად ყველა ვერსიაში ჩართული ხშირად გამოყენებადი მათემატიკური ფუნქციები. თითოეულის გამოსათვლელად საკმარისია პროგრამაში მივუთითოთ მათი სახელი და ***a*** არგუმენტის მნიშვნელობა (***a*** წარმოადგენს გარკვეულ არითმეტიკულ გამოსახულებას).

ცხრ. 4.2. *ბეისიკის ენაში ჩაშენებული მათემატიკური ფუნქციები*

მათემატიკური გამოსახულება	sin *a*	cos *a*	e^a	*ln a*	| *a* |	arctg*a*	$\sqrt{a}$
ჩანაწერი ბეისიკში	SIN *a*	COS *a*	*EXP a* ($a \le 78$)	*LOGa* ($a > 0$)	ABS(a)	ATN(a)	ATN(a)

4.2 ცხრილის გამოყენებით შეგვიძლია ჩავწეროთ რთული ჩაშენებული ფუნქციები. მაგალითად e^{x+5}→ EXP (X+5); |ln*x*|→ ABS(LOG(X).

3.2.5. გამოსახულებები

გამოსახულება წარმოადგენს ჩანაწერს რომელიც მიგვითითებს რა ოპერაციები უნდა ჩავატაროთ მონაცემებზე იმისათვის, რომ მივიღოთ საჭირო მნიშვნელობები. არსებობს ორი სახის - არითმეტიკული და ტექსტური გამოსახულებები.

არითმეტიკული გამოსახულება ეწოდება რიცხვებისაგან, ცვლადების სახელებისაგან, მასივების ელემენტებისაგან, ჩაშენებული ფუნქციებისაგან, არითმეტიკულ ოპერაციათა ნიშნებისაგან და მრგვალი ფრჩხილებისაგან შედგენილ სიმბოლურ ჩანაწერს, რომელსაც აზრი აქვს მათემატიკის თვალსაზრისით.

რამდენადმე მეტი დრო სჭირდება ტექსტური გამოსახულების რაობის გაგებას. ბეისიკს შეუძლია მუშაობა არა მარტო რიცხვებთან, არამედ ტექსტებთანაც. ესენი მონაცემთა თვისობრივად განსხვავებული სახეებია, ოღონდ ორივესთან კომპიუტერის მუშაობის პრინციპები საკმაოდ ჰგავს ერთმანეთს. ტექსტური სიდიდეებიც იყოფა კონსტანტებად და ცვლადებად. ტექსტური კონსტანტას შესახებ საუბარი გვქონდა ***4.2*** ცხრილში. **ტექსტური კონსტანის** სიგრძე **255** სიმბოლოს არ აღემატება. მის მსგავსად **ტექსტური ცვლადიც** აღინიშნება სახელით, ოღონდ მას ბოლოში ეხმარება სუფიქსი $. მაგალითად,

ტექსტური ცვლადებია *D$, K7&*. ჩვეულებრივ ტექსტური ცვლადის სიგრძეც არ აღემატება **255** სიმბოლოს, ოღონდ *QBasic*-ში იგი ***3256***-ის ტოლია.

ტექსტურ მონაცემებზე ტარდება გარკვეული, რიცხვითი მონაცემებისაგან განსხვებული, არამათემატიკური ოპერაციები. ესენია შეერთების, მინიჭებისა და შედარების ოპერაციები. ჩვენი სახელმძღვანელოს ფარგლებს სცილდება ტექსური ინფორმაციის დეტალური განხილვა; ამიტომ შემოვიფარგლებით იმ ფაქტის აღნიშვნით, რომ რიცხვითი ინფორმაციის ანალოგურად ტექსტურ ინფორმაციაზე მუშაობის პროცესშიც წარმოიქმნება გარკვეული სახის გამოსახულებები, რომლებსაც ეწოდება არა არითმეტიკული, არამედ **ტექსტური გამოსახულებები.** დასასრულს შევნიშნავთ, რომ კერძო შემთხვევაში **ტექსტურ გამოსახულებას** წარმოადგენს ტექსტური ცვლადი. მას ელემენტალური ტექსტური გამოსახულებაც კი ვუწოდოთ.

დავუბრუნდეთ არითმეტიკულ გამოსახულებას. მათი გარდაქმნისათვის გამოიყენება არითმეტიკული ოპერაციები. ამ ოპერაციებიდან შეკრების, გამოკლების, გამრავლების, გაყოფის, ახარისხებისა და ამოფესვის ოპერაციები საშუალო სკოლის კურსიდან არის ცნობილი, რასაც ვერ ვიტყვით **მოდულით გაყოფასა** და **მთლიანად გაყოფაზე,** რომლებიც, როგორც ***4.1*** არის აღნიშნული, *QBasic*-ში გამოიყენება. ორიოდე სიტყვით შევეხოთ ამ ოპერაციებს.

დავუშვათ, რომ გვაქვს გასაყოფი $a = 78$ და $b = 33$. 78-ის 33-ზე გაყოფისას მივიღებთ არასრულ განაყოფს $q = 2$ და დაგვრჩება ნაშთი $r = 12$, ამიტომ შეგვიძლია ჩავწეროთ:

$$\frac{a}{b} = b\,q + r,$$

ანუ, თუ შევიტანთ სიდიდეთა მნიშვნელობებს, მივიღებთ: $\frac{78}{33} = 33 \cdot 2 + 12.$

მოდულით გაყოფის დროს ვეძებთ ***r*** ნაშთს, ხოლო მთელრიცხვული გაყოფის დროს - არასრულ ***q*** განაყოფს, ე. ი.

a MOD b = r;	*78 MOD 33 = 12;*
a \ b = q;	*78 \ 33 = 2.*

არითმეტიკული გამოსახულებების დაწერისას უნდა დავიცვათ შემდეგი წესები და შეზღუდვები:

- გამოსახულების ყველა სიმბოლო ერთ სტრიქონში უნდა ჩაიწეროს. **აკრძალულია** მრავალსართულიანი გამოსახულებები, აგრეთვე ზედა და ქვედა ინდექსები;
- ერთმანეთის გვერდით არ უნდა განთავსდეს ერთმანეთის გვერდით. არ შეიძლება გამრავლების ნიშნის გამოტოვება;
- არითმეტიკული გამოსახულების ტიპს განსაზღვრავს შედეგის ტიპი;

ამასთანავე:

- მთელ რიცხვზე გაყოფის ოპერაცია გვაძლევს ნამდვილ შედეგს;

- გამოსახულება ერთდროულად თუ შეიცავს მთელ და ნამდვილ სიდიდეებს, იგი ითვლება ნამდვილ სიდიდედ;

- გამოსახულება თუ შეიცავს ორმაგი სიზუსტის სიდიდეებს, იგი ორმაგი სიზუსტის გამოსახულებად ითვლება.

• არითმეტიკული ოპერაციები უფროსობის შემდეგი ნიშნის მიხედვით უნდა შერულდეს:

1) ფრჩხილებში მოთავსებული ოპერაციები;

2) ჩაშენებული ფუნქციების გამოთვლა;

3) ახარისხება;

4) გამრავლებისა და გაყოფის ოპერაციები;

5) მთელრიცხვული გაყოფის ოპერაცია;

6) მოდულის მიხედვით გაყოფა;

7) გამრავლებისა და გაყოფის ოპერაცია.

• თანაბარი უფროსობის ოპერაციები უნდა შესრულდეს ერთმანეთის მიყოლებით მარცხნიდან მარჯვნივ. გამონაკლისია: $A \wedge B \wedge C = A \wedge (B \wedge C)$.

მოვიყვანთ არითმეტიკული გამოსახულების ჩაწერის ორ მაგალითს:

1) $\frac{\sqrt{3x^5 + 4,5}}{Y-7} A + D \quad \rightarrow \quad$ SQR (3 * X% ^ 5 + 4,5) / (Y# - 7) * D& ;

2) $\frac{\sqrt[5]{4x - \sin x}}{7,8 + |Y|^3} \quad \rightarrow$ (4 * X – SIN(x) ^ (1 / 5) / (7,8 + ABS(Y) ^ 3)

3.2.6. ოპერატორისა და პროგრამის ცნებები

ბეისიკის ენაზე დაწერილი პროგრამა ეწოდება ბეისიკის ენაზე ჩაწერილ ამოცანის გადაწყვეტის ალგორითმს. იგი კომპიუტერს მიუთითებს ამოცანის გადასაწყვეტად რომელი ოპერაციები რა თანამიმდევრობით უნდა შესრულოს.

სალაპარაკო, მაგალითად, ქართულ ენაზე შედგენილი ტექსტის მსგავსად **დაპროგრამების ენაზე** დაწერილი ტექსტიც შედგება წინადადებებისგან. დაპროგრამების **ბეისიკის ენაზე** ტერმინ „წინადადებების“ ნაცვლად გამოიყენება ტერმინი „**ოპერატორები**“.

თითოეული ოპერატორი მკაცრად განსაზღვრული სახით ჩაიწერება. იგი, როგორც წესი, ამა თუ იმ სახით შეიცავს **სახელს, მონაცემებსა** და კომპიუტერს მიუთითებს გარკვეული ოპერაცია რომელ მონაცემებზე უნდა შეასრულოს. განვიხილოთ, მაგალითად, ოპერატორი: INPUT A, B, C.

მასში **INPUT** (*მონაცემების შეტანა*) არის სახელი, ხოლო A, B, C არგუმენტების სახელები (იდენტიფიკატორები).

არსებობს მხოლოდ სახელის შემცველი ოპერატორები, რომლებსაც **უარგუმენტო ოპერატორებს** უწოდებენ. ასეთი ოპერატორებია, მაგალითად, STOP (სდექ) და RESTORE (გადასვლის ოპერატორი).

ბეისიკის ენაზე დაწერილი წარმოადგენს სტრიქონების თანამიმდევრობას. თითოეული სტრიქონის წინ ისმება მისი ნომერი. სტრიქონები ინომრება რიგის მიხედვით, ჩვეულებრივ, ***10***-ის ტოლი ბიჯით, ე. ი. ***10, 20, 30, 40, 50*** და ა. შ. QBasic-ში დანომვრა არაა სავალდებულო; საჭიროა დავნომროთ მხოლოდ ის სტრიქონი, რომელზეც გადასვლა **GOTO** ოპერატორითაა გათვალისწინებული.

QBasic-ში პროგრამის სტრიქონის სიგრძე არ უნდა აღემატებოდეს ***255*** სიმბოლოს (დისპლეის ეკრანის სიგრძეა ***80*** პოზიცია).

3.2.7. ბეისიკის ენის ოპერატორები

მოცემულ განყოფილებაში განვიხილავთ დაპროგრამების მხოლოდ პირველდაწყებითი გაცნობისათვის საჭირო I NPUT, PRINT, LET, REM, STOP ოპერატორებს.

მონაცემების შეტანის *I NPUT* ოპერატორი გამოიყენება პროგრაამის შესრულების პროცესში კომპიუტერის კლავიატურიდან საწყისი სიდიდეების შესატანად და ამ სიდიდეებისათვის მეხსიერებაში გამოყოფილ უჯრედებში მათ განსათავსებლად.

ოპერატორი საშუალებას გვაძლევს ერთი და იგივე ამოცანა საწყისი სიდიდეთა მნიშვნელობებისათვის პროგრამის შეუცვლელად გადავწყვიტოთ. მაგალითად, INPUT A, B$, C% ოპერატორს კლავიატურიდან შეყავს *A, B$, C%* ცვლადების მნიშვნელობები, ე. ი. მათ ანიჭებს კონკრეტულ მნიშვნელობებს.

ოპერატორის ზოგადი სახეა:

INPUT <კარნახი> ; $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ (შესატან სიდიდეთა სია),

სადაც INPUT არის ოპერატორის სახელი, <კარნახი> - ტექსტური კონსტანტა, რომელიც გამოიყენება იმის გამოსარკვევად, თუ რომელი სიდიდეები რა თანამიმდევრობით უნდა შევიტანოთ (შეიძლება მისი გამოტოვება); $\mathbf{x}_i$ ($i = 1, \ldots, n$) ცვლადის ან მასივის ელემენტის სახელი, ანუ იდენტიფიკატორი).

მაგალითი: *ოპერატორს*

20 INPUT „შეიტანე X %", D!, G$, L%" Y%, D!, G$, L$

კომპიუტერის ეკრანზე გამოყავს შეტყობინება შეიტანე X%, D!, G$, L$ და დაელოდება ამ ოთხი სიდიდის მნიშვნელობებს.

ოპერატორ *INPUT* -ის შესრულებისას კომპიუტერი მუშაობს მომხმარებელთან დიალოგის რეჟიმში:

1. ოპერატორი წყვეტს პროგრამის შესრულებას, ეკრანზე გამოაქვს საკარნახო ტექსტი და კითხვის „?“ ნიშანი, რითაც თხოვს მომხმარებელს შეიტანოს მონაცემები;

2. მომხმარებელი ვალდებულია კლავიატურიდან შეიტანოს თითოეული $\mathbf{x}_i$ ცვლადის v_i მძიმეებით განცალკევებული მნიშვნელობები, რომლის შედეგადაც ეკრანზე გამოჩნდება სია:

$$? \, v_1, v_2, \ldots, v_n \, .$$

შენიშვნა. რომ მთელი მთელი v_i კონსტანტა „%“ ნიშნის გარეშე ჩაიწერება, ხოლო ტექსტური კონსტანტა შეიძლება შემზღუდველი აპოსტროფების გარეშეც ჩავწეროთ.

მაგალითი:

შეიტანე X%, Y, G$, L%” 7, 9.4, ლაშა, “ია”

ე. ი. თუ v_i არის რიცხვითი ან ტექსტული კონსტანტა, მაშინ მისი ტიპი უნდა დაემთხვეს $\mathbf{x}_i$-ის სიდიდის ტიპს.

3. მონაცემების ნაკრების შეტანის შემდეგ მომხმარებელმა თითი უნდა დააჭიროს კლავიატურაზე არსებულ *Enter* (შეტანა) კლავიშზე, რის შედეგადაც მონაცემები ეკრანიდან შევა კომპიუტერში და თითოეულ $\mathbf{x}_i$ სიდიდეს მიენიჭება მისი v_i მნიშვნელობა; იმავდროულად მოწმდება ამ სიდიდეთა ტიპების შესაბამისობა და შეტანილ მნიშვნელობათა რაოდენობა. ამ დროს:

- ტიპების შეუსაბამობის აღმოჩენის დროს გაიცემა შეტყობინება შეცდომის შესახებ და საჭირო გახდება მონაცემების მთელი $v_1, v_2, \ldots, v_n$ სია ხელახლა ავკრიფოთ;

- არასაკმარისი რაოდენობის მნიშვნელობების შეტანის შემთხვევაში ეკრანზე ხელახლა გაჩნდება კითხვის „?“ ნიშანი და კომპიუტერი დაელოდება შეტანის გაგრძელებას:

- შეტანილი ზედმეტი მნიშვნელობები იგნორირდება.

QBasic-ში თითოეული მსგავსი შემთხვევის შესახებ გაიცემა შეტყობინება შეცდომის დაშვების შესახებ და მოითხოვება განმეორდეს ყველა მონაცემის შეტანა

მინიჭების LET ოპერატორი გამოსახულებას ანიჭებს გარკვეულ მნიშვნელობას. მაგალითად, განვიხილოთ ოპერატორის ჩანაწერი:

```
40 LET L = 7* Y + 9 * COS (X) ^ 5
50 LET A$ = “მიხო ძია”
```

მასში პირველი ოპერატორი L-ის მნიშვნელობას გამოითვლის ფორმულით $L = 7Y + 9\cos x$, ხოლო მეორე ოპერატორი A$ ცვლადს ანიჭებს ტექსტური ცვლადის მნიშვნელობას.

ოპერატორის ზოგადი სახეა:

LET **X = A**,

სადაც LET არის არის ოპერატორის სახელი (ქართულად „ ვთქვათ“), რომლის გამოტოვება შეიძლება, **X** - ცვლადის ან მასივის ელემენტის სახელი, ხოლო **A** - არითმეტიკული ან ტექსტური გამოსახულება.

აღნიშნული ოპერატორი გამოითვლის **A** გამოსახულების მნიშვნელობას და მას ანიჭებს **X** ცვლადს, ე. ი. მოცემულ შემთხვევაში „=“ სიმბოლო გამოხატავს არა **X** და **A** სიდიდეთა ტოლობას, არამედ მინიჭების ოპერაციის შესრულებას.

ოპერატორის ფუნქციონირების თავისებურებებია:

1. თუ **X** არის მთელი, ხოლო **A** ნამდვილი სიდიდეა, მაშინ ოპერაციის შედეგი უახლოეს მთელ სიდიდემდე დამრგვალდება. მაგალითად, თუ გვაქვს:

50 X = 6,8

60 Q% = X,

მაშინ Q% ცვლდს მიენიჭება 7;

2. თუ **X** არის ერთმაგი სიზუსტის ნამდვილი რიცხვი, ხოლო **A** - ორმაგი სიზუსტის ნამდვილი რიცხვი, მაშინ -ს მნიშვნელობა მრგვალდება;

3. დასაშვებია **X=X+A** სახის ოპერატორი. ამ შემთხვევაში „=“ ნიშნიდან მარჯვნივ მდგარი **X** სიმბოლო განიხილება როგორც **X** ცვლადის ადრინდელი (წინა) მნიშვნელობა, ხოლო „=“ ნიშნიდან მარხცნივ მდგარი **X** სიმბოლო - **X** ცვლადის მომდევნო მნიშვნელობად.

გამოტანის *PTINT* ოპერატორი გამოიყენება პროგრამის შესრულების პროცესში მონიტორის ეკრანზე სიდიდეთ მნიშვნელობების გამოსატანად.

მაგალითი:

10 X=7 : D$ = "მტრედი”

20 PRINT X; D$; “მორჩა”

ოპერატორი PRINT ამ მაგალითში ეკრანზე გამოიტანს სიდიდეებს X, D$, “მორჩა” .

ოპერატორ PRINT-ის (ითარგმნება, როგორც „დაბეჭდე“) ზოგადი სახეა:

PRINT ; $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ ან

PRINT ; $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ (გამოსატან სიდიდეა სია),

სადაც PRINT არის ოპერატორის სახელი, რომელიც ითარგმნება როგორც „დაბეჭდე“), $\mathbf{x}_i$, ($i = 1, \ldots., n$) - შესატან სიდიდეთა სიის ელემენტი. ეს უკანასკნელი შეიძლება იყოს კონსტანტა, ცვლადი, მასივის ელემენტი, გამოსახულება ან TAB ფუნქცია (მას ქვემოთ განვიხილავთ).

PRINT ოპერატორის ჩაწერის მაგალითია:

60 PRINT “ამოცანა 1” ; D; C-A/4; TAB(40) ; 30.

მუშაობისას ოპერატორი მიმდევრობით ათვალიერებს; $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ სიის თითოეულ წევრს და ყოველი $\mathbf{x}_i$-სათვის ეკრანზე გამოაქვს მისი მნიშვნელობა. ამ დროს:

- $\mathbf{x}_i$-ის რიცხვითი სიდიდის მნიშვნელობა $|\mathbf{x}_i| < 10^{-7}$-ის ან $|\mathbf{x}_i| > 10^{-7}$- -ის დროს გამოიტანება მცურავი წერტილიანი რიცხვის სახით;

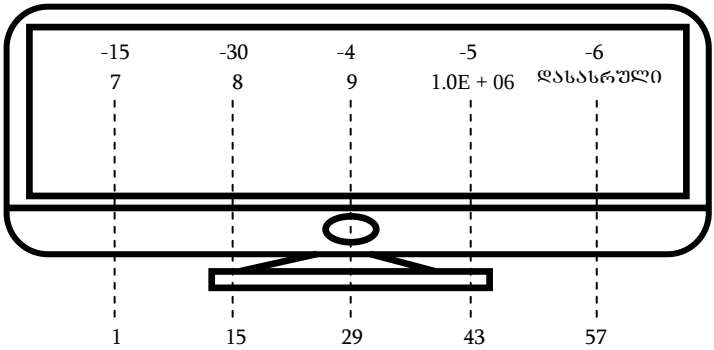
- რიცხვის წინ არსებული + ნიშანი იცვლება პრობელით:

- მთელი სიდიდეები იბეჭდება % ნიშნის, ხოლო ტექსტური სიდიდეები - ბრჭყალების გარეშე;

- სიდიდეთა მნიშვნელობები გამოიტანება სტრიქონობრივად - მოცემული სტრიქონის შევსების შემდეგ ავტომატურად ხდება მომდევნო სტრიქონზე გადასვლა.

სტრიქონში გამოსატანი მნიშვნელობები ლაგდება სიის ელემენტების შემდეგი დამყოფების მეშვეობით:

ა) წერტილ-მძიმით (;). ამ შემთხვევაში რიცხვითი მნიშვნელობის ბოლოში გაითვალისწინება ერთი პრობელი; ტექსტურ სიდიდეთა მნიშვნელობები ყოველგვარი დამატების გარეშე გამოიტანება;



ნახ. 4.1. *მონიტორის ეკრანზე მონაცემების გამოტანის მაგალითი*

ბ) მძიმით (,). ამ შემთხვევაში ეკრანი იყოფა ხუთ სვეტად, რომელთაგანაც თითოეული შეიცავს 14 პოზიციას. თითოეული სიდიდის მნიშვნელობა იბეჭდება მომდევნო სვეტის დასაწყისიდან.

მოვიყვანოთ QBasic-ში ოპერატორ PRINT-ის გამოყენების მაგალითები.

მაგალითი: განვიხილოთ ოპერატორები:

```
40  D=-15
50  L=D
```

60 F=1000000

70 PRINT D, D+L, -4, -5, -6, 7, 8, 9. "ᲓᲐᲡᲐᲡᲠᲣᲚᲘ"

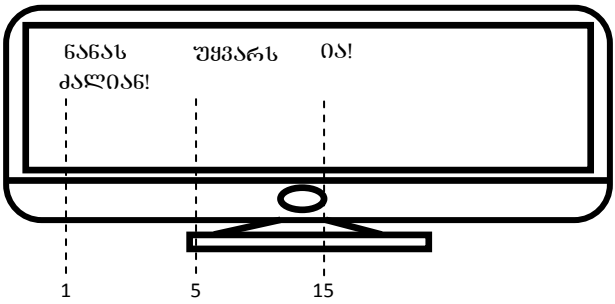
4.1 ნახაზზე მოყვანილია მონიტორის ეკრანი, რომელზედაც ზემოთ მოყვანილი ოპერატორების შედეგად გამოყვანილია ორი სტრიქონი. ამ ნახაზზე 1, 15, 29, 43, 57 რიცხვები აღნიშნავს მონიტორის ეკრანის პოზიციების ნომრებს (ისინი და მასთან არსებული პუნქტირული ხაზები ეკრანზე არ გამოიტანება)

PRINT ოპერატორის ჩაწერისა და ფუნქციონირების თავისებურებაბი:

1. PRINT ოპერატორი შეგვიძლია ჩავწეროთ სიდიდეების სიის გარეშე. ამ შემთხვევაში იგი ეკრანზე გამოიტანს ცარიელ (პრობელების) სტრიქონს.

2. PRINT-ის ერთ ოპერატორში შეგვიძლია გამოვყენოთ სხვადასხვა (",“ და „;“) მაცალკევებელი;

3. პროგრამა თუ რამდენიმე PRINT ოპერატორს შეიცავს, მაშინ ისინი გამოსატან სიდიდეთა ერთიანი სიის მქონე ერთ ოპერატორივით იქცევა; ამ დროს გვაქვს ერთადერთი გამონაკლისი: PRINT ოპერატორის ბოლოში მაცალკევებელის ((",“-ის ან „;“-ის) არ არსებობისას პროგრამაში მის შემდეგ არსებულ PRINT ოპერატორს სიდიდე ახალი სტრიქონის დასაწყისიდან გამოჰყავს.



ნახ. 4.2. *რამდენიმე PRINT ოპერატორის ფუნქციონირების მაგალითი*

მაგალითი: პროგრამის შემდეგი ფრაგმენტის:

60 PRINT „ᲜᲐᲜᲐᲡ“;

70 PRINT „ᲣᲧᲕᲐᲠᲡ“,

80 PRINT „ᲘᲐ!“

90 PRINT „ᲫᲐᲚᲘᲐᲜ!”

4.2 ნახაზზე ნაჩვენებია პროგრამის ზემოთ მოყვანილ ფრაგმენტის მიერ მონიტორის ეკრანზე გამოტანილი ინფორმაცია. აქ **1, 5** და **15** არის ეკრანის სტრიქონის პოზიციათა ნომრები.

PRINT და INPUT ოპერატორების მეშვეობით მ

ომხმარებელსა და კომპი-უტერს (პროგრამას) შორის **დიალოგის ორგანიზებაცაა** შესაძლებელი. ასეთი დიალოგის ფრაგმენტი მოყვანილია ქვემოთ.

```
10 PRINT „მე ვარ IBM კომპიუტერი! რა ქვიათ თქვენ?”
20 INPUT „ბიგი”
30 PRINT „გამარჯობა, ბიგი!“
40 PRINT „როგორი ამინდია?”
50 INPUT „წვიმიანი”
60 PRINT „ცუდი ამინდი ჟოფილა!”
```

ასეთი დიალოგი შეიძლება დიდხანს გაგრძელდეს.

–TAB ფუნქცია *PTINT* ოპერატორში გამოიყენება სიდიდეთა მნიშვნელობების ეკრანის გარკვეულ პოზიციებზე გამოსატანად ან მათ დასაბეჭდად პრინტერში არსებული ფურცლის სათანადო პოზიციებზე. მისი ზოგადი სახეა:

TAB (*n*),

სადაც *n* არის რიცხვი ან არითმეტიკული გამოსახულება.

PRINT TAB (*n*) ; *X*

ოპერატორს *X*-ის მნიშვნელობა გამოყავს მონიტორის სტრიქონის *n*-ურ პოზიციაზე. მაგალითად,

PRINT TAB (57); G

ოპერატორს G-ს მნიშვნელობა გამოყავს მონიტორის ეკრანის 57-ე პოზიციაზე.

REM ოპერატორი გამოიყენება პროგრამის ტექსტში ისეთი განმარტებების ჩასართავად, რომლებიც აუცილებელია პროგრამის გასაგებად. მაგალითად: 20 REM ინტეგრალის გამოთვლა. ლოგუა.

ოპერატორის ზოგადი სახეა:

REM <ტექსტი>,

სადაც REM არის (ინგ. *remark* -დამ, რაც ნიშნავს შენიშვნას, მითითებას) ოპერატორის სახელი, ხოლო <ტექსტი> - ბეისიკის ენის ნებისმიერ სიმბოლოთა მიმდევრობა.

STOP ოპერატორი წყვეტს პროგრამის შესრულებას და გასცემს შეტყობინობას:

გაჩერება სტრიქონში *n*,

სადაც ***n*** არის იმ სტრიქონის ნომერი, რომელშიც დგას STOP ოპერატორი. ოპერატორი შეგიძლია პროგრამის ნებისმიერ სტრიქონში მოვათავსოთ. ასეთი ოპერტატორების რაოდენობა არ იზღუდება. იგი სასარგებლოა პროგრამის გამართვისათვის. მისი შესრულების შემდეგ შეგვიძლია ზოგიერთი ცვლადის მნიშვნელობები შევცვალოთ, PRINT... ბრძანებით შეგვიძლია ეკრანზე

გამოვიტანოთ ცვლადების მნიშვნელობები (მაგრამ პროგრამის რედაქტირება არ შეუძლებელია), ხოლო შემდეგ ბრძანება CONT-ით იმ წერტილიდან გავაგრძელოთ პროგრამის მუშაობა, სადაც მოხდა მისი გაჩერება.

QBasic-ში პროგრამის შეწყვეტის გამომწვევი STOP ოპერატორი ტექსტში გამოირჩევა კაშკაშა ფერით, არავითარი შეტყობინება არ გაიცემა.შეწტვეტის წერტილიდან პროგრამის მუშაობის გასაგეძელებლად სრულდება ბრძანება:

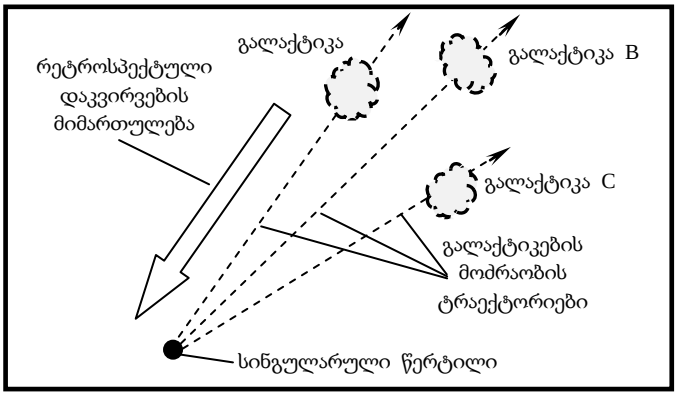
/Run/Continue (/ᲐᲛᲣᲨᲐᲕᲔᲑᲐ/ᲒᲐᲒᲠᲫᲔᲚᲓᲔᲡ).

END ოპერატორი წყვეტს პროგრამის შესრულებას და პროგრამის ბოლო ოპერატორია.

დ ა ნ ა რ თ ი

სინგულარული წერტილის წარმოშობის პროცესის რეტროსპექტული სურათი

ნახაზზე ნაჩვენებია კოსმოსში გარკვეული მიმართულებით მოძრავი **A, B** და **C** გალაქტიკები. მათ მოძრაობაზე დაკვირვება ისეთ შთაბეჭდილებას ტოვებს, რომ ისინი საერთო წერტილიდან გარკვეული ძალის ზემოქმედებით უნდა იყოს გატყორცნილი. ლოგიკის თანახმად, აღნიშნული წერტილი გალაქტიკების მოძრაობის ტრაექტორიების რეტროსპექტულად გაგრძელებების გადაკვეთაზე უნდა მდებარეობდეს. **დიდი აფეთქების** თეორიაში ითვლება, რომ უსასრულო დიდი სიმკვრივისა და ტემპერატურის მქონე ამ წერტილში, რომელსაც **სინგულარული წერტილი** ეწოდება, შეყურსული იყო მთელი მატერია. დროის გარკვეულ მომენტში, რომელიც **ნულოვანი მომენტად** ითვლება (აქედან დაიწყო დროის ათვლა), იგი აფეთქდა და მასში არსებული მთელი შიგთვსი კოსმოსში გაიტყორცნა; დაიწყო მატერიის გაფართოების პროცესი, რომლის შედეგად ჩამოყალიბდა ვარსკვლავები, პლანეტები, გალაქტიკები და სხვა ობიექტები. მათმა ერთობლიობამ წარმოქმნა დღევანდელი **სამყარო**. ამ უკანასკნელის გაფართოება-სახეცვლილების პროცესი დღემდე გრძელდება: მისი გალაქტიკები პერმანენტულად ფართოვდება და შორდება როგორც სინგულარულ წერტილს, ისევე ერთმანეთს.



ლ ი ტ ე რ ა ტ უ რ ა

1. ა. დუნდუა. კომპიუტერული სისტემებისა და ტექნლოგიების თეორიული საფუძვლები - თბ, „ტექნიკური უნივერსიტეტი“, **2014** - 258 გვ.

2. ა. დუნდუა. კომპიუტერული სისტემების ტექნიკური საშუალებები (Hsrdware) - თბ, საგამომცემლო სახლი „ტექნიკური უნივერსიტეტი“, **2019.** - 144 გვ.

3. ა. დუნდუა. კომპიუტერული სისტემების პროგრამული საშულებები (Hsrdware) - თბ, საგამომცემლო სახლი „ტექნიკური უნივერსიტეტი“, **2019**. - 164 გვ.

4. ა. დუნდუა. ტრანსპორტზე მიკროპროცესორული ტექნიკის გამოყენების საფუძვლები - თბ, საგამომცემლო სახლი „ტექნიკური უნივერსიტეტი“, **2017.** - 344 გვ.

5. ა. დუნდუა, **ვ. საპოჟნიკოვი, ვლ. საპოჟნიკოვი** დისკრეტულ მოწყობილობათა თეორიის საკითხები. თბ, საქართველოს პოლიტექნიკური ინსტიტუტი, **1990**. -120 გვ

6. თ. მაჭარაძე, ზ. წვერაიძე. ინფორმატიკის საფუძვლები. - თბ, „ტექნიკური უნივერსიტეტი“, **2009**. – 342 გვ.

8. ი. ჰარარი. საპიენსი - კაცობრიობის მოკლე ისტორია. - თბ, სულაკაურის გამომცემლობა, **2020. –** 552 გვ.

9. В. В. Сапожников, **Вл. В. Сапожников**, **Д. В. Ефанов**. Основы теории надежности и технической диагностики. Идательство „Лань“, **2019**. – 588 с.

10. Информфтика / под ред. В.В. Трофимова. – М, Издательство Юр-айт; ИД Юрайт, **2013**. -917 с.

11. А.В. Могилев, Н.И. Пак, У.К.Хенер. Информатика. – М, Издательский центр „Академия“, **2012.** -848 с.

12. В. Ф. Ляхович, В.А. Молодцов, Е.И. Рыжкова. Основы информатики. **–** М, КНОРУС, **2016**. – 348 с.

13. Ф. Ф.Хлебников. Информационные технологии. – М, КНОРУС, **2014**. – 472 с.

14. С. М.Окулов. Информатика: развитие интеллекта школьников. М, БИНОМ. Лаборатория знаний, **2015**. – 212 с.

15. С. М.Окулов. Основы программирования. – М, БИНОМ. Лаборатория знаний, **2015**. – 336 с.

16. Н.В. Макарова. Основы программированияю – М, КНОРУС, **2017**. -432 с.

17. А.А. Хлебников. Информационные технологии. М, КНОРУС, **2014.** – 472 с.

18. Е. В. Андреева. Программирование – это так просто просто, программирование – это так сложно. – М, МЦНОМО, **2015**. – 184 с.

19. И. В. Черепков. – Основы программирования. – М, Издательство Юрайт, **2016**. - 214 с.

20. В. С. Еовичков, Н.И. Парфилова, А.Н. Пылкин. Паскаль. – М, Высш. Шк. 1990. – 223 с.

21. Р. С. Гутер, Ю.Л. Полунов. От абака до компьютера. – М, Издательство „Знание“**, 1981**. – 207 с.

22. ა. შეროზია. ასახვა. - ქართული საბჭოთა ენციკლოპედია, ტომი 1-ლი, გვ. 627-628. **1975**.

23. უ. ბაინემი. მეცნიერება. მოკლე ისტორია- - თბ, სულაკაურის გამომცემლობა, **2022**. -370 გვ.

24. ა. გეგეჭკორი. აზროვნების სათავეებთან. თბ, „ნაკადული“, **1982.** – 295 გვ.

25. მსოფლიოს ისტორია. I , - თბ, პალიტრა L, **2024. -**324 გვ.

26. უცხო სიტყვათა ლექსიკონი. შემდგენელი მიხეილ ჭაბაშვილი. თბ, განათლება, **1989. –** 598 გვ.

27. **ქართული საბჭოთა ენციკლოპედია**, ტ 8. - თბ, - 720 გვ.

28. ე. გომბრიხი. მსოფლიოს მოკლე ისტორია - თბ, სულაკაურის გამომცემლობა, **2022**. -417 გვ.

29. რ. ნათაძე. აზროვნება - ქართული საბჭოთა ენციკლოპედია, ტომი 1-ლი, გვ. 220. **1975**.

30. რ. ნათაძე. აზროვნების გენეზისი - ქართული საბჭოთა ენციკლოპედია, ტომი 1, გვ. 220. **1975**.

31. რ. ნათაძე, ინტელექტი - ქართული საბჭოთა ენცილოპედია, ტ 5, გვ. 172. **1980**.

32. მ. სალუქვაძე. მართვა (ტექ.) - ქართული საბჭოთა ენციკლოპედია, ტ 6, გვ. 434. **1983**.

33. ა. ჩიქობავა. ენა. - ქართული საბჭოთა ენციკლოპედია, ტ 4, გვ. 141 – 143. **1979**.

34. ე. ხარაძე. ასტრონომიის საფუძვლები // ორ ტომად - თბ, თბილისის უნივერსიტეტის გამომცემლობა, – ტ 1, 412 გვ; ტომი მე-2. 448 გვ. **1991**.

35. Знакомьтесь: компьютер / [П. Брирли, М. Ротену, Р. Заксу и др.]; Перевод с англ. К. Г. Батаева; Под ред. В. М. Курочкина. – М, Мир, **1989**. – 238 с.

36. Н. Луман. Социальные системы. Санкт Петербург, „Наука“, 2007. -680 с.

37. Al-Khwarizmi. https://en.wikipedia.org/wiki/Al-Khwarizmi

38."Let us Calculate!" Leibniz, Llull, and the Computational Imagination https://publicdomainreview.org/essay/let-us-calculate-leibniz-llull-and-the-computational-imagination/

39. George Bool. https://www.lindahall.org/about/news/scientist- of-the-day/george-boole/

40. Виталий Мацарский. Джордж Буль — предтеча искусственного интеллекта. https://elementy.ru/nauchno-populyarnaya biblioteka/436174/ Dzhordzh_Bul_predtecha_iskusstvennogointellekta

41. **Леонид Черняк Шесть веков истории логических машин.** https://www.osp.ru/os/2005/03/185427

42. Валерий Шилов. История логических машин. https://refdb.ru/look/2401603-pall.html

43. Двоичная система. http://esyr.org/uneex_disc/mounted/FreeCode/436/work/ss/bin_ss.html

44. Ramon Llull. https://en.wikipedia.org/wiki/Ramon_Llull

45. Лейбниц: идея символической логики https://studfile.net/preview/3875463/page:8/

46. Big Bang Big. https://tn. Wikipedia.org/wiki/Bing_Bang

47. Big Bang Timeline. https://www.physicsoftheuniverse.com/topics_bigbang_timeline.

48. Natural units. https://en.wikipedia.org/wiki/Natural_units

49. Planck units. https://en.wikipedia.org/wiki/Planck_units

50. Human history. https://en.wikipedia.org/wiki/Human_history

51. Intelligense. https://en/wikipedia.org/wiki/Intelligence

52. The Planck era: Imagining our infant universe. https:// www**.** astronomy.com/science/the-planck-era-imagining-our-infant-universe/

53. Cosmic History. https://science.nasa.gov/universe/overview/

54. Есть ли у растений интелект? https:// www.bbc .com/ukrainian/ukraine_in_russian/2015/12/151203_ ru_ s_ plants_ underestima-ted _intelligent

55. https://ka.wikipedia.org/wiki/Ვოლოცენი

56. Neolithic Revolution. https://en.wikipedia.org/wiki/Neolithic _ Revolution

57. The history of the universe: Big Bang to now in 10 easy steps https://www.space.com/13320-big-bang-universe-10-steps-explainer.html

58. Почему именно Homo sapiens пережил другие виды? И как на это повлял яяык, https://meduza.io/feature/2021/07/25/ pochemu-imenno-homo-sapiens-perezhil-drugie-vidy-i-kak-na-eto-povliyal-yazyk#:~:text= Социальные%20связи%2C%20сотрудничество%20и%20столкновения, процессе%20решающую%20роль%2C%20—%20язык.

59. Quark. https://en.wikipedia.org/wiki/Quark

60. Will Humans Ever Go Extinct? https://www.scientificamerican.com/article/will-humans-ever-go-extinct/

61. https://biomolecula.ru/articles/kris-stringer-ostalis-odni-edinstvennyi-vid-liudei-na-zemle-retsenziia

62. Industrial Revolution. https://en.wikipedia.org/wiki/Industrial_Revolution

63. Holocene calendar. https://en.wikipedia.org/wiki/Holocene_calendar

64. Holocene calendar https://simple.wikipedia.org/wiki/Holocene_calendar

65. neolithic revolution**.** https://en.wikipedia.org/wiki/Neolithic _Revolution

66. Professor John McCarthy. What is AI? / Basic Questions http://jmc .stanford.edu/artificial-intelligence/what-is-ai/index.html

67. Professor John McCarthy. What is AI? / Branches of AI http://jmc.stanford.edu/artificial-intelligence/what-is-ai/branches-of-ai.html

68. Professor John McCarthy. What is AI? /Applications of AI http://jmc.stanford.edu/artificial-intelligence/what-is-ai/applications-of-ai.htm

69. Professor John McCarthy. What is AI? / More questions http://jmc.stanford.edu/artificial-intelligence/what-is-ai/more-questions.html

70. Artificial intelligence. https:// en.wikipedia.org/wiki/Artifical_Intelligence

71. День рождения искусственного интеллекта. https://dzen.ru/a/ZHnbQNvzSzu37E70

72. What Is Artificial Intelligence? Definition, Uses, and Types. https://www.coursera.org/articles/what-is-artificial-intelligence

73. Первый философ Искусственного Интеллекта https://habr.com/ru/articles/694934/

74. Искусственный интеллект. https://ru.wikipedia.org/wiki/Искусственный_интеллект

75. Искусственный интеллект https://secrets.tinkoff.ru/glossarij/artificial-intelligence/

76. https://secrets.tinkoff.ru/glossarij/artificial-intelligence/

77. https:// www.resonancedaily. com/mobile/index.php?id_ rub=11&id_ artc=212646

78. https://gtu.ge/Archive/22817/

79. https://www.facebook.com/watch/?v=318532184629728

80. https://www.interpressnews.ge/ka/article/761767-rektori-davit-gurgenize-sakartvelos-teknikuri-universitetis-mier-giorgi-nikolazis-unikaluri-gamogonebis-agadgenam-shesazlebelia-sakartveloshi-inpormatikisa-da gamo- tvliti-teknikis-istoria-shecvalos/.

81. ნ. ბერძენიშვილი. საქართველოს ისტორიის საკითხები. თბ, „მეცნიერება“, 1990**.** – 661.

82. Р. С. Гиляревский. Информатика как наука об информации**.** https://www.mathnet.ru/links/26fcdddeb4c22b7c1aee8d3066865368/ssi51.pdf

83. Р. С. Гиляревский. Основы информатики. Курс лекций. https://ma.hse.ru/data/2013/03/05/1292987258/БИ_курс%20лекций.pdf

84. master - https://www.merriam-webster.com/dictionary/master

85. мастер - https://ru.wikipedia.org/wiki/Мастер

86. И. Х. Дворецкий. Латино-русский словарь. – М, издательство „Русский язык“, 1976. – 1096 с. https://ashishkin.ru/wp-content/uploads/ 2019/03/Latinskii---Dvoretskii--.pdf

რედაქტორი ბ. ცხადაძე

გადაეცა წარმოებას 27.01.2025. ხელმოწერილია დასაბეჭდად 06.03.2025. ქაღალდის ზომა 60X84 1/16. პირობითი ნაბეჭდი თაბახი 14,5. №3694.

საგამომცემლო სახლი „ტექნიკური უნივერსიტეტი“,
თბილისი, კოსტავას 77

Verba volant,
scripta manent

ალექსანდრე დუნდუა
საქართველოს ტექნიკური
უნივერსიტეტის პროფესორი

ISBN 978-9941-512-94-0