საქართველოს ტექნიკური უნივერსიტეტი

თეოდორე ზარქუა

პროგრამული უზრუნველყოფის მობილურობისა და გაფართოებადობის ზოგიერთი ასპექტი

წარდგენილია დოქტორის აკადემიური ხარისხის მოსაპოვებლად

სადოქტორო პროგრამა “ინფორმატიკა”, შიფრი 0401

საქართველოს ტექნიკური უნივერსიტეტი
თბილისი, 0175, საქართველო
ივლისი, 2015 წელი

სამუშაო შესრულებულია საქართველოს ტექნიკურ უნივერსიტეტში

ინფორმატიკისა და მართვის სისტემების ფაკულტეტი

ინტერდისციპლინარული ინფორმატიკის დეპარტამენტი

ხელმძღვანელები: პროფ. ჰამლეტ მელაძე

პროფ. ოლეგ ნამიჩეიშვილი

რეცენზენტები: პროფ. ნანა გულუა

პროფ. ლევან იმნაიშვილი

დაცვა შედგება ---------- წლის ”------” ----------, --------- საათზე

საქართველოს ტექნიკური უნივერსიტეტის ----------

---------- ფაკულტეტის სადისერტაციო საბჭოს კოლეგიის

სხდომაზე, კორპუსი ----------, აუდიტორია ----------

მისამართი: 0175, თბილისი, კოსტავას 77.

დისერტაციის გაცნობა შეიძლება სტუ-ს ბიბლიოთეკაში,

ხოლო ავტორეფერატისა - ფაკულტეტის ვებგვერდზე

სადისერტაციო საბჭოს მდივანი პროფ. თინათინ კაიშაური

საქართველოს ტექნიკური უნივერსიტეტი
ინფორმატიკისა და მართვის სისტემების ფაკულტეტი

ჩვენ, ქვემორე ხელისმომწერნი, ვადასტურებთ, რომ გავეცანით თეოდორე ზარქუას მიერ შესრულებულ სადისერტაციო ნაშრომს დასახელებით: “პროგრამული უზრუნველყოფის მობილურობისა და გაფართოებადობის ზოგიერთი ასპექტი” და ვაძლევთ რეკომენდაციას საქართველოს ტექნიკური უნივერსიტეტის სადისერტაციო საბჭოში მის განხილვას აკადემიური ხარისხის მოსაპოვებლად.

_____

ხელმძღვანელი: ჰამლეტ მელაძე

_____

ხელმძღვანელი: ოლეგ ნამიჩეიშვილი

_____

რეცენზენტი: ნანა გულუა

_____

რეცენზენტი: ლევან იმნაიშვილი

_____

საქართველოს ტექნიკური უნივერსიტეტი
2015

ავტორი: თეოდორე ზარქუა

დასახელება: “პროგრამული უზრუნველყოფის მობილურობისა და გაფართოებადობის ზოგიერთი ასპექტი”

ფაკულტეტი: ინფორმატიკისა და მართვის სისტემების

ხარისხი: დოქტორი

სხდომა ჩატარდა:

ინდივიდუალური პიროვნების ან ინსტიტუტის მიერ ზემომოყვანილი დასახელების დისერტაციის გაცნობის მიზნით მოთხოვნის შემთხვევაში მისი არაკომერციული მიზნებით კოპირებისა და გავრცელების უფლება მინიჭებული აქვს საქართველოს ტექნიკურ უნივერსიტეტს.

ავტორის ხელმოწერა

ვუძღვნი ჩემი მშობლების, ათინა კეთილაძის და იასონ ზარქუას ნათელ ხსოვნას

რეზიუმე

წინამდებარე სადისერტაციო ნაშრომი “პროგრამული უზრუნველყოფის მობილურობისა და გაფართოებადობის ზოგიერთი ასპექტი” მიძღვნილია პროგრამული უზრუნველყოფის ხარისხის ისეთი მნიშვნელოვანი მახასიათებლების, როგორიცაა მობილურობა და გაფართოებადობა, გაუმჯობესებისთვის მიმართული მეთოდოლოგიური საშუალებების როგორც თეორიული, ასევე პრაქტიკული მხარეებისადმი.

ნაშრომში შესწავლილია პროგრამული უზრუნველყოფის მობილურობისა და გაფართოებადობის გაუმჯობესების ისეთი საშუალებები, რომლებიც ემყარება ფუნქციონალურ გამოსახულებათა წარმოდგენის ე.წ. პოლონურ ნოტაციას. ამასთან დაკავშირებით შემუშავებულ იქნა ფუნქციონალურ გამოსახულებათა დამუშავებისთვის განკუთვნილი პროგრამული კომპლექსი, რომელიც უზრუნველყოფს ამ გამოსახულებებთან მუშაობას ტერმინებში, რომლებიც მაქსიმალურად მიახლოებულია ბუნებრივთან პირდაპირ დაპროგრამების ე.წ. ალგორითმული ენის დონიდან. ცხადია, ეს ფაქტობრივად წარმოადგენს ალგორითმული ენის გაფართოებას, რომელიც მთელი რიგი ამოცანისთვის გადაწყვეტის პრინციპულად ახალ გზებს განაპირობებს პრობლემური პროგრამისტისთვის. წარმოდგენილი ნაშრომის ფარგლებში შესრულდა ფუნქციონალურ გამოსახულებათა დამუშავების ავტომატიზებისთვის განკუთვნილი 2 განსხვავებული რეალიზაცია.

ერთი – დაპროგრამების ენა პასკალზე და გაფორმებულია დამოუკიდებელი მოდულის სახით. აქ გამოსახულების შიდა წარმოდგენა ემყარება პოსტფიქსურ ნოტაციას და წარმოდგენილია დინამიურად ცვალებადი ბმული სიის სახით. მოდულის შემადგენლობაში შედის 2 ძირითადი პროგრამული კომპონენტი. ერთი წარმოადგენს პროცედურას, რომელიც უზრუნველყოფს სტრიქონის სახით მოცემული ინფიქსური გამოსახულების ანალიზს და თუ იგი კორექტულია, შედეგად იძლევა ამ გამოსახულების პოსტფიქსურ შესატყვისს შიდა ფორმით, ხოლო წინააღმდეგ შემთხვევაში კი იძლევა ინფორმაციას აღმოჩენილი შეცდომის შესახებ. მეორე პროგრამული კომპონენტი არის ფუნქცია, რომელიც შიდა სახით მოცემული გამოსახულებისთვის დააბრუნებს მის მნიშვნელობას პარამეტრით განსაზღვრული გამოსახულებაში შემავალი ცვლადების მნიშვნელობებისთვის. ეს რეალიზაცია ითვალისწინებს საზოგადოდ რამდენიმე ცვლადის შემცველი ისეთი გამოსახულებების დამუშავებას, რომლებშიც გამოყენებულია 4-ვე არითმეტიკული მოქმედება, 2 სახის ახარისხება (მთელ და არამთელ ხარისხად), აგრეთვე გავრცელებული ელემენტარული ფუნქციები. რეალიზაციაში გათვალისწინებულია ერთი 3 ოპერანდიანი მოქმედება. ამ რეალიზაციით შეიკრა თსუ-ს ეგმ-ების მათემატიკური უზრუნველყოფის კათედრის მიერ 1999 წელს შექმნილი გამოყენებითი პროგრამების პაკეტი, რომელიც წარმოდგენილ იქნა გამოფენაზე ინფო-99 და დაჯილდოვდა დიპლომით მე-2 ადგილისთვის.

მეორე რეალიზაცია შესრულებულია დაპროგრამების ენა C++ -ზე და ამჟამად პროგრამათა კომპლექსის სახე აქვს. აქ გათვალისწინებულია არა უმეტეს ერთი ცვლადის (x) შემცველი ფუნქციონალურ გამოსახულებათა დამუშავება, რომლებშიც შეიძლება

იყოს გამოყენებული 4-ვე არითმეტიკული მოქმედება, ახარისხება (უნივერსალური) და გავრცელებული ელემენტარული ფუნქციები. ეს რეალიზაცია ითვალისწინებს გამოსახულების შიდა წარმოადგენას ვექტორის სახით პრეფიქსულ ნოტაციაზე დაყრდნობით. ამ რეალიზაციით გათვალისწინებულია გამოსახულების ბუნებრივი ფორმიდან (სტრიქონიდან) არა მარტო შიდა ფორმაზე გადაყვანა და საპირისპირო ქმედება, არამედ მისი გაწარმოება და გამარტივება. წარმოდგენილი რეალიზაცია ქმნის წინაპირობას სპეციალური კლასის შემუშავებისთვის, რომელიც განკუთვნილი იქნება პრობლემური პროგრამისტისთვის ფუნქციონალურ გამოსახულებებთან მაქსიმალურად მარტივად მუშაობისთვის დაპროგრამების ენა C++ -დან. ამ კლასის საინტერფეისო ნაწილის აღწერა შევიდა წარმოდგენილ ნაშრომში.

მოცემული ნაშრომის ფარგლებში შემუშავდა დაპროგრამების სასტარტო სწავლების კონცეპცია, რომელიც ემყარება ტიურინგის ვირტუალური მანქანის მოდიფიკაციის გამოყენებას. ცხადია, ეს კონცეპცია არ არის დამოკიდებული სისტემურ პლატფორმაზე და ადვილად ექვემდებარება გადატანას. ამ კონცეპციაზე დამყარებული რეალიზაციები გამოიყენება სასწავლო პროცესში საქართველოს საპატრიარქოს წმიდა ანდრია პირველწოდებულის სახელობის ქართულ უნივერსიტეტში 2011 წლიდან და პანსიონ "მთიებში" 2014 წლიდან.

თემატიკის **აქტუალობა** განპირობებულია პროგრამული უზრუნველყოფის მოცულობის ზრდის სტაბილურად მაღალი ტემპებით და კომპიუტერის გამოყენების არეალის მიერ ადამიანის საქმიანობის პრაქტიკულად ყველა სფეროს დაფარვით.

ნაშრომში მიღებული ძირითადი შედეგები შემდეგია:

1. ალგებრულ გამოსახულებათა უფრჩხილებო, ე.წ. პოლონური ჩანაწერების მიმართ შეუღლებულის ცნების შემოღება და ამ ცნებასთან დაკავშირებული კანონზომიერებების დადგენა. ავტორმა მკაცრად მათემატიკურად დაამტკიცა შესაძლებლობა მთლიანად მოიხსნას პოლონური ჩანაწერების პოსტფიქსური და პრეფიქსული ფორმების ურთიერთგარდაქმნის აუცილებლობა. ეს გარემოება კი არსებითია ისეთ ამოცანებში, სადაც წარმოიშვება ფუნქციონალურ გამოსახულებათა არა მარტო რიცხვითი მნიშვნელობების მიღების, არამედ მათი ფუნქციონალური გარდაქმნის საჭიროება. ამ შედეგებზე დაყრდნობით შემუშავდა პროგრამა, რომელიც აძლევს მომხმარებელს შესაძლებლობას იმუშავოს ფუნქციონალურ გამოსახულებებთან ისევე, როგორც მონაცემთა სტანდარტულ ტიპებთან.
2. შემუშავებულია გამოსახულებათა პოლონური ჩანაწერების შეუღლებულის ცნებასთან დაკავშირებული ალგორითმები, რომლებიც უზრუნველყოფს მათ გარდაქმნას სტანდარტულ პოლონურ ჩანაწერებზე და პირიქით.
3. შემუშავებულია ინფიქსური ჩანაწერიდან პრეფიქსული ჩანაწერის მიღების პირდაპირი ალგორითმი, დამყარებული 2 სტეკის გამოყენებაზე.
4. შემუშავებულია მეთოდიკა, განკუთვნილი პრობლემური პროგრამისტის მიერ ფუნქციონალურ გამოსახულებებზე გაწარმოების გარდაქმნების განხორციელებისთვის

დაპროგრამების ე.წ. ალგორითმული ენებიდან, რომელიც ემყარება ამ მიზნით შექმნილ შიდა ენას.

5. შემუშავებულია სპეციალური კლასის საინტერფეისო ნაწილი, რომელიც უზრუნველყოფს პრობლემურ პროგრამისტს ფუნქციონალურ გამოსახულებათა დამუშავების მოსახერხებელი საშუალებებით ალგორითმული ენის დონიდან.
6. შემუშავებულია ტიურინგის ვირტუალური მანქანის მოდიფიკაცია, განკუთვნილი დაპროგრამების საწყისების სწავლებისთვის.

ნაშრომის **მეცნიერული სიახლე** განპირობებულია შემდეგით:

- ავტორის მიერ შემოღებული პოლონური ჩანაწერის შეუღლებულის ცნება და, აქედან გამომდინარე, ამ ცნებასთან დაკავშირებული მის მიერ ჩამოყალიბებული და დამტკიცებული 7 თეორემა და 8 შედეგი სრულიად ახალია;
- სიახლეს წარმოადგენს ავტორის მიერ ჩამოყალიბებული ინფიქსური ჩანაწერიდან პირდაპირ პრეფიქსული ჩანაწერის მიღების ალგორითმი, დამყარებული ერთდროულად 2 სტეკის გამოყენებაზე;
- ახალია შეუღლებულის ცნებასთან დაკავშირებული ყველა ალგორითმი;
- სიახლეს წარმოადგენს ავტორის მიერ შემოთავაზებული შიდა ენა გაწარმოების წესების აღწერისთვის, ასევე სიახლეა რეალიზაცია, რომელიც ეყრდნობა ამ ენას და გამოირჩევა როგორც გაფართოებადობით, ასევე მობილურობით;
- მთლიანობაში, სიახლეა ავტორის მიერ შემუშავებული პროგრამული საშუალებები, რომელთა მეშვეობითაც პრობლემურ პროგრამისტს ეძლევა შესაძლებლობა ალგორითმული ენის დონიდან დაამუშავოს სტრიქონის მეშვეობით ბუნებრივი სახით მოცემული ფუნქციონალური გამოსახულება. დამუშავებაში შედის ამ გამოსახულების არა მარტო მნიშვნელობების გამოთვლა (პარამეტრის გარკვეული მნიშნელობებისთვის), არამედ მისი სხვადასხვანაირი ფუნქციონალური გარდაქმნა, მათ შორის დიფენერენცირება;
- სიახლეს წარმოადგენს ავტორის მიერ შემოთავაზებული ტიურინგის ვირტუალური მანქანის მოდიფიკაცია.

ნაშრომის **პრაქტიკული ღირებულება** განპირობებულია იმ გარემოებით, რომ ყველა შემუშავებული კონცეპცია მიყვანილია კომპიუტერული რეალიზაციის დონემდე.

Abstract

The dissertation “Some aspects of software portability and extensibility” examines theoretical and practical aspects of methodological means intended to improve important parameters of software quality, such as portability and extensibility.

These methods are based on so-called “Polish notation” of representing functional expressions. In terms of this, a software package facilitating work with such expressions has been created. Actually, this can be regarded as an extension to a programming language, which provides programmers with brand new ways of handling substantial class of problems. Two different versions of the mentioned software have been designed.

As for the first version, it was developed with Pascal programming language as an independent module. Reverse Polish Notation is used for internal representation of expressions, implemented by linked lists. The module comprises two main components. The first one is a function, which analyses string containing functional expression written in infix notation and returns an internal representation of the same expression but converted into postfix form, provided that no errors are found. Otherwise, a message containing information about the error is returned. The second component, also a function, evaluates the expression using values of variables, passed as arguments to the function. Generally, the expressions can contain several variables with all four arithmetic operators, two types of exponentiation (with integer and non-integer exponent), some elementary functions and one ternary operator. This particular implementation was part of the software package developed by a department of computer software of Tbilisi State University in 1999. Exhibited within an exhibition “Info-99”, the software was awarded diploma for the 2nd place.

The second version, is designed with C++. In contrast with the first one, this implementation is limited to only single variable expressions with all four arithmetic operators, exponentiation and elementary functions. Vector data structure is used for internal representation of functional expressions. Not only does it support converting string expression into internal representation and vice versa, but also differentiation and simplification. Based on this version, a C++ class could be created, providing a better experience for programmers when dealing with certain sorts of mathematical problems. Description of an interface of this class is also covered in the following thesis.

Within the scope of this dissertation, a conception of teaching computer programming basics at introductory level has also been developed. Based on a modification of Turing Machine, yet preserving the key ideas, this methodology is platform independent, portable and easy to use. Implementations of the concept have been used in classes held at St. Andrew the First-Called Georgian University of the Patriarchate of Georgia and boarding school “IB Mtiebi” since 2014.

The topicality of the subject is provoked by stably high rates of software volume increase and the fact that the computer usage area currently covers virtually every field of human activity.

The principal results of the thesis are the following:

1. The term of a conjugate to parenthesis-free notations of algebraic expressions was introduced and the regularities associated to this term were detected. The author has given a strict mathematical proof that it is possible to withdraw the necessity of convertibility from postfix notation to prefix and vice versa. This is significant for problems which require not only computing the numeral values of functional expressions, but also their functional translation. On basis of these results, a software was developed which enables the user to operate with functional expressions as with any standard data type.
2. Set of algorithms providing functionality of converting between polish conjugate and standard polish notations has been introduced.
3. The algorithm involving two stacks for directly converting from infix to prefix notation has been designed.
4. The methodology of enabling programmers to perform differentiation of functional expressions using an internal language created for this very purpose.
5. The interface part of special class which enables the user to processing functional expressions from the level of algorithmic language was developed.
6. The conception of modification of Turing Machine intended to facilitate conveying the basics of programming to beginners has been developed.

The following paragraphs suggest the scientific originality of the thesis:

- The term of a conjugate to Polish notation, 7 stated and proven theorems and 8 results introduced by the author are brand new.
- The algorithm of directly obtaining prefix notation of an expression from the corresponding infix notation using 2 stacks is new.
- The new is also all algorithms concerning the conjugate.
- The internal language for describing the rules of differentiation and the corresponding implementation (which is portable and extensible) are completely new as well.
- Overall, the methodology enabling performing transformations of functional expressions (given as a string in a natural form), including differentiation and evaluation (with specified values of variables). Moreover, programmers are able to conduct all these operations from a programming language.
- Finally, suggested modification of Turing Machine.

The practical value of the dissertation consists in the fact that all developed concepts have been implemented as software.

შინაარსი

შესავალი....14
ლიტერატურის მიმოხილვა....17
შედეგები და მათი განსჯა....19
თავი 1. ფუნქციონალურ გამოსახულებათა დამუშავების ზოგიერთი მეთოდი....19
1.1 ფუნქციონალურ გამოსახულებათა ჩაწერის პოსტფიქსური და პრეფიქსული ნოტაციები....19
1.2 პოსტფიქსური ჩანაწერის შეუღლებულის ცნება და მისი თვისებები....33
1.3 პრეფიქსული ჩანაწერის შეუღლებულის ცნება და მისი თვისებები....41
თავი 2. ფუნქციონალურ გამოსახულებათა დამუშავება დაპროგრამების ენიდან....46
2.1 შიდა ენა სიმბოლური დიფერენცირების წესების აღწერისთვის....46
2.2 სიმბოლური დიფერენცირების რეალიზაცია....48
2.3 ფუნქციონალურ გამოსახულებათა დამუშავებისთვის განკუთვნილი კლასის ინტერფეისი....54
2.4 ფუნქციონალურ გამოსახულებათა დამუშავებისთვის განკუთვნილი მოდული (რეალიზაცია პასკალზე)....57
თავი 3. ტიურინგის ვირტუალური მანქანის მოდიფიკაცია დაპროგრამების სასტარტო სწავლებისთვის....62
3.1 დაპროგრამების სასტარტო სწავლებისთვის განკუთვნილი ტიურინგის ვირტუალური მანქანის აღწერა....62
3.2 უმარტივესი ამოცანების გადაწყვეტა ტიურინგის ვირტუალურ სასწავლო მანქანაზე....66
3.3 განშტოებების და ციკლების გაცნობა ტიურინგის ვირტუალურ სასწავლო მანქანაზე....73
3.4 ქვეპროგრამა ტიურინგის ვირტუალურ სასწავლო მანქანაზე....79
დასკვნა....84
ლიტერატურა....89
დანართი 1. მოდული gamoTvla. საწყისი ტექსტი (პასკალზე)....92
დანართი 2. პროგრამათა კომპლექსი ფუნქციონალურ გამოსახულებათა დამუშავებისთვის. საწყისი ტექსტი (C++)....105

ცხრილების ნუსხა

ცხრილი 1 პოსტფიქსური გამოსახულების A ალგორითმით გამოთვლა....23
ცხრილი 2 ინფიქსური გამოსახულებიდან პოსტფიქსურის მიღება....26
ცხრილი 3. გაწარმოების ცხრილი ტრადიციული და პრეფიქსული ნოტაციით....27
ცხრილი 4. ინფიქსური გამოსახულებიდან პირდაპირ პრეფიქსულის მიღება....30
ცხრილი 5. პოსტფიქსურიდან პირდაპირ პრეფიქსული გამოსახულების მიღება....32
ცხრილი 6. პოსტფიქსური ჩანაწერიდან მისი შეუღლებულის მიღება....35
ცხრილი 7. გაწარმოების წესები შიდა ენაზე....48
ცხრილი 8. პროგრამაში რეალიზებული გამარტივებები....53

ავტორი თავს ვალდებულად თვლის მადლობა მოუხადოს მისთვის ძვირფას და საპატივცემულო პიროვნებებს, რომელთა მხარდაჭერისა და გულისმიერების გარეშე წინამდებარე ნაშრომი ვერ შეიქმნებოდა:

ჰამლეტ მელაძეს – დისერტაციის ხელმძღვანელს, მუდმივი მზრუნველობისთვის და სწორი მიმართულებით მომართვისთვის;

ოლეგ ნამიჩეიშვილს – დისერტაციის ხელმძღვანელს, საოცარი გულისხმიერებისთვის;

კოტე ცისკარიძეს – კაცს, რომელმაც პირველად დამანახა დაპროგრამების არსი;

არჩილ ფრანგიშვილს – მუდმივი მხარდაჭერისთვის;

სერგო ვარდოსანიძეს – ყურადღებისა და მზრუნველობისთვის;

თინათინ კაიშაურს – მუდმივი ყურადღების, სულგრძელობისა და საოცარი მოთმინებისთვის;

ზურაბ წვერაიძეს – მამოტივირებელი ყურადღებისთვის;

ზურაბ ბაიაშვილს – უპრეცედენტო თანადგომისთვის;

გია სურგულაძეს – ყურადღებისა და თანადგომისთვის;

რევაზ კაკუბავას – პერმანენტული გულშემატკივრობისთვის;

ავთანდილ ცისკარიძეს – ჩემს უახლოეს მეგობარს და თანაავტორს, სისტემატიური სასიკეთო ზემოქმედებისთვის;

ნანა გულუას – ნაშრომის რეცენზენტს, მისი გაცნობისთვის გაწეული ძალისხმევის, პროფესიონალიზმისა და კოლეგიალობისთვის;

ლევან იმნაიშვილს - ნაშრომის რეცენზენტს, მისი გაცნობისთვის გაწეული ძალისხმევის, პროფესიონალიზმისა და კოლეგიალობისთვის;

ავთანდილ რუხაძეს – ჩემს მოსწავლეს და თანაავტორს, ტიურინგის სასწავლო ვირტუალური მანქანის კარგი რეალიზაციისთვის;

სანდრო ბარნაბიშვილს – ჩემს მოსწავლეს და თანაავტორს, ტიურინგის სასწავლო ვირტუალური მანქანის კარგი რეალიზაციისა და ტექსტის დამუშავებაში გაწეული დახმარებისთვის;

ანა ჯიბღაშვილს – ჩემს მეუღლეს, მუშაობისთვის პირობების შექმნისთვის;

იასონ, თათია და ნათია ზარქუებს – ჩემს შვილებს, კომფორტული სულიერი განწყობის შექმნისთვის;

ნიკა, ანანო და დავით ზარქუებს – ჩემს შვილიშვილებს, რონლებმაც ჩამომიყალიბეს სასიკეთო ემოციური ფონი;

მარიამ დორეულს და დავით ჭეჟიას – ჩემს შეძენილ შვილებს, აუცილებელი სულიერი სიმშვიდის მონიჭებისთვის;

ავტორი მადლობას უხდის ზემომოყვანილ სიაში ვერმოხვედრილ მისთვის ძვირფას ყველა პიროვნებას და გამოთქვამს სინანულს მადლიერების პერსონალურად გამოხატვის ვერშეძლებისთვის მათი სიმრავლისა (საბედნიეროდ) და, ამავე დროს, მადლიერების გამოსახატი სივრცის მოცულობის შეზღუდულობის გამო.

შესავალი

კაცობრიობის განვითარების თანამედროვე ეტაპი ხასიათდება საინფორმაციო ნაკადების ინტენსივობის მკვეთრი ზრდით და კომპიუტერული ტექნოლოგიების შეჭრით ადამიანის საქმიანობის ყველა სფეროში. ასეთ პირობებში განუხრელად იზრდება მოთხოვნები პროფილურ პროგრამულ უზრუნველყოფაზე და იზრდება მოთხოვნები მისი ხარისხის მიმართ.

თუ ჯერ კიდევ ათიოდე წლის წინ პროგრამული უზრუნველყოფის სათანადო ხარისხს უზრუნველყოფდა მისი შემუშავების სტანდარტულად მიჩნეული სქემა, რომლის მიხედვითაც პროგრამისტს არ მოეთხოვებოდა დასაპროგრამებელი მასალის საგნობრივი არის ცოდნა, დღეს უკვე მთელ რიგ შემთხვევაში მოთხოვნილი ხარისხის მიღწევა შეუძლებელი ხდება, თუ პროგრამისტი არ არის გარკვეული დასაპროგრამებელი საკითხების შინაარსში სათანადო საგნობრივი არის სპეციალისტის დონეზე.

ამის მიღწევისთვის რეალურად ადგილი აქვს ორივე შესაძლებელ პროცესებს – ერთის მხრივ, პროგრამისტები ცდილობენ გაერკვნენ მათ მიერ შესადგენი პროგრამების საგნობრივ არეში, მეორეს მხრივ კი საგნობრივი არის სპეციალისტები ცდილობენ დაეუფლონ დაპროგრამების საფუძვლებს. ამის ხელშეწყობას ემსახურება დაპროგრამების უახლესი საინსტრუმენტო საშუალებები, რომლებიც გამიზნულია პრობლემურ პროგრამისტს შეექმნას მაქსიმალურად კომფორტული პირობები პროგრამის შემუშავებისთვის.

ერთ-ერთი მძლავრი საშუალება ზემონახსენების მისაღწევად ჩადებულია დაპროგრამების ობიექტზე ორიენტირებულ პარადიგმაში, რომლის წყალობითაც დაპროგრამების ენის ფარგლებში შესაძლებელია შეიქმნას ამა თუ იმ საგნობრივი არისთვის აქტუალური კლასები (ფაქტობრივად, ენის მხარდაჭერით შემოღებული დამატებითი ტიპები), რომლებიც იძლევა შესაძლებლობას ჩამოყალიბდეს პროგრამის ტექსტი სათანადო საგნობრივი არისთვის ბუნებრივთან მაქსიმალურად მიახლოვებულ ტერმინებში.

დაპროგრამების ინსტრუმენტარის დახვეწასთან ერთად იხვეწება დაპროგრამების სტილი. ცნობილია, რომ პროგრამაში დაშვებული

შეცდომა მით ნაკლებად საზიანოა, რაც ადრეა გამოვლენილი. ამ თვალსაზრისით ძალიან მნიშვნელოვანია დაპროგრამების ისეთი სტილის გამომუშავება, რომელიც მინიმუმამდე დაიყვანს შეცდომის დაშვების შესაძლებლობას. ამავდროულად, პროგრამისტი ყოველთვის უნდა ითვალისწინებდეს, რომ დავალების არსი დროთა განმავლობაში შეიძლება იცვლებოდეს და მის მიერ შექმნილი პროგრამული უზრუნველყოფა ადვილად უნდა ექვემდებარებოდეს სათანადო მოდიფიკაციას. ანუ პროგრამის სტრუქტურამ უნდა შეუწყოს ხელი შემდგომში მასში ცვლილებების შეტანას. წინამდებარე ნაშრომში განხილულია პროგრამის **გაფართოებადობის** მიღწევის რამდენიმე შესაძლებლობა. აქედან გამოსაყოფია გაფართოებადობა შიდა ენის შემუშავების ხარჯზე, როდესაც საკუთრივ პროგრამა წარმოადგენს ამ შიდა ენის რეალიზაციას და პროგრამის ფუნქციონალის შეცვლა ფაქტობრივად მდგომარეობს მხოლოდ ამ შიდა ენაზე ჩამოყალიბებული დავალების შეცვლაში. მეორეს მხრივ, კი შემუშავებულია პროგრამული ინსტრუმენტის გაფართოების შესაძლებლობა დაპროგრამების ენაში პრობლემური პროგრამისტისთვის ახალი საშუალებების შეტანის გზით. ცხადია, ამ უკანასკნელის ბაზაზე იზრდება მოცემულ ენაზე შედგენილი ყველა სამომხმარებლო პროგრამების გაფართოებადობის მაჩვენებლებიც.

დაპროგრამების საფუძვლების ცოდნის ფართო გავრცელების აუცილებლობა დღეს უკვე არავითარ ეჭვს არ იწვევს. პროგრამირების საფუძვლების ცოდნა სასარგებლოა საგნობრივი არის სპეციალისტისთვის იმ შემთხვევაშიც, თუ მას უშუალოდ პროგრამების შექმნა არ უწევს. ასეთი ცოდნა სპეციალისტს სწორ წარმოდგენას უყალიბებს საზოგადოდ პროგრამული უზრუნველყოფის და მისი შესაძლებლობების მიმართ – ეს გარემოება კი ძნელია გადაჭარბებით შეფასდეს!

დღეს პროგრამული უზრუნველყოფის **მობილობის** ქვეშ გულისხმობენ მის თვისებას იმუშავოს სხვადასხვა აპარატულ პლატფორმაზე, ან სხვადასხვა ოპერაციული სისტემის მართვით. კომპიუტერული ტექნოლოგიების განვითარების თანამედროვე ეტაპზე მობილობა პირველ რიგში მიიღწევა პროგრამული უზრუნველყოფის შესრულებით

ისეთ საინსტრუმენტო საშუალებით (დაპროგრამების ენით), რომელიც რეალიზებულია სხვადასხვა პლატფორმაზე და სხვადასხვა ოპერაციულ სისტემაში. ამ აზრით მოცემულ ნაშრომში წარმოდგენილი ყველა პროგრამა მობილურია, რადგან შესრულებულია სხვადასხვა ოპერაციულ სისტემაში რეალიზებულ დაპროგრამების ენაზე.

ძირითადი აქცენტი ამ ნაშრომში მაინც გაკეთებულია პროგრამული უზრუნველყოფის ხარისხის განმაპირობებელ ისეთ თვისებაზე, როგორიცაა **გაფართოებადობა**. შემუშავებულ პროგრამებში გაფართოებადობა არის მიღწეული როგორც მისი შიდა აგებულების ხარჯზე, ასევე საინსტრუმენტო საშუალების (დაპროგრამების ენის) შესაძლებლობების გაფართოებით.

რაც შეეხება დისერტაციის დასკვნით ნაწილს, რომელიც ეხება დაპროგრამების სასტარტო სწავლებისთვის ინსტრუმენტის შემუშავების საკითხს, მისი ჩართვა მოცემულ ნაშრომში გარდა აქტუალობისა, გამართლებულია კონცეპციის **აბსოლუტური დამოუკიდებლობით როგორც აპარატული, ასევე სისტემური პლატფორმისგან**.

ლიტერატურის მიმოხილვა

როგორც შესავალში ითქვა, დისერტაციაში აქცენტი გაკეთდა პროგრამული უზრუნველყოფის გაფართოებადობის მიღწევის საშუალებებზე. ამ მიმართულებით დამუშავდა კომპილაციის თეორიის ის ნაწილი, რომელიც ეხება ალგებრულ გამოსახულებებთან მუშაობის ავტომატიზებას. კომპილაციის თეორიისა და პრაქტიკის საკითხები კარგად არის გადმოცემული მთელ რიგ გამოცემაში, რომლებიც უკვე პროგრამირების კლასიკად ითვლება. აქ ჩვენ გვხდება როგორც უფრჩხილებო (პოლონური) ჩანაწერების კლასიკური აღწერა, ასევე ყველა იმ ალგორითმის აღწერა, რომელიც გამოიყენა ავტორმა ამ საკითხის შესწავლის დროს. ცხადია, სწორედ ამ მასალას დაეყრდნო ავტორი უფრჩხილებო ჩანაწერებისთვის შეუღლებულის ცნების შემოღებისა და მისი თვისებების კვლევის დროს. [1-4,6,8-10].

უფრჩხილებო ჩანაწერებზე დაფუძნებული მონაცემთა სტრუქტურები დიდ როლს ასრულებს სხვადასხვა ალგორითმის აგების საკითხებში. ამავდროულად, როგორც ცნობილია, დღეს მიღწეულია აპარატული უზრუნველყოფის სიმძლავრეთა გარკვეული ზღვრული მნიშვნელობა და პირველ პლანზე გამოდის პროგრამული უზრუნველყოფის წარმადობის გაზრდა ალგორითმული საშუალებებით. ამ საკითხებს ეძღვნება შემდეგი ფუნდამენტური ნაშრომები [12,13].

პროგრამული უზრუნველყოფის ხარისხი მრავალი მაჩვენებლით განისაზღვრება და შესაბამისი საკითხები თავისთავად დიდ თეორიულ და პრაქტიკულ ინტერესს შეადგენს. ეს თემატიკა ფუნდამენტურად არის დამუშავებული [11]-ში.

პროგრამული უზრუნველყოფის ხარისხი მნიშვნელოვნად არის დამოკიდებული დაპროგრამების სტილზე. საზოგადოდ, დაპროგრამების პროცესში შერწყმულია როგორც ინდივიდუალური თვისებების გამოვლენის შესაძლებლობა, ასევე გარკვეული საყოველთაოდ აღიარებული წასების დაცვა, რათა შედეგი აღმოჩნდეს მაქსიმალურად მაღალხარისხიანი. ამ საკითხებზე ერთობ შინაარსიანად მსჯელობს ლეგენდარული დეიქსტრა თავის შესანიშნავ წიგნში [7].

პროგრამისტთა თაობებისთვის საკულტო გამოცემად გადაიქცა და ასეთად რჩება ფრედერიკ ბრუქსის განთქმული წიგნი “როგორ იქმნება პროგრამული კომპლექსები”. ამ წიგნში ავტორმა პირველად გამოიტანა სამსჯელოდ პროგრამული უზრუნველყოფის შექმნასთან დაკავშირებული წმინდა საორგანიზაციო და ღრმა ფსიქოლოგიური ქვეტექსტის მქონე საკითხები. ამ წიგნის პირველი გამოცემიდან თითქმის 50 წელი გავიდა, მიუხედავად ამისა იგი არ კარგავს აქტუალობას. ამის დასტურია მისი განახლებული გამოცემა [14].

პირველად ალგებრული გამოსახულების პრეფიქსული ჩანაწერის რევერსირების შედეგის შესახებ ავტორის მიერ ითქვა [20]-ში. მაშინ მას ჯერ კიდევ არ ქონდა გამოყენებული ტერმინი შეუღლებული.

შეუღლებულის ცნების შემოღებას მოყვა ავტორის მიერ ამ ცნების ყოველმხრივი გამოკვლევა, რაც აისახა პუბლიკაციებში [21-25].

მიღებული შედეგების გამოყენება სიმბოლური დიფერენცირებისთვის პირველად დაფიქსირდა [27]-ში.

ცალკე დიდ ლოდად არის აზიდული დონალდ კნუტის მრავალტომეული “დაპროგრამების ხელოვნება”, რომელიც პროგრამისტთა თაობებისთვის სამაგიდო წიგნად იქცა. როცა ლაპარაკია დაპროგრამების სწავლების მეთოდიკებზე, გვერდს ვერ აუვლით ამ ნაშრომში შემოთავაზებულ ვირტუალურ სასწავლო მანქანას MIX-ს [15].

საკმაოდ დიდი ხნის განმავლობაში დაპროგრამების საწყისების სწავლების დე-ფაქტო სტანდარტად მიჩნეული იყო მეთოდიკები, დამყარებული შვეიცარიელი მათემატიკოსის, ნიკლაუს ვირტის მიერ შემუშავებულ დაპროგრამების ენა პასკალზე. უნდა ითქვას, რომ ეს გამართლებული იყო არა მარტო თავად ამ ენის შესაძლებლობათა სიმდიდრით, არამედ იმ ფაქტით, რომ წლების განმავლობაში პასკალი და მასზე დამყარებული პროგრამისტული ისნტრუმენტები ერთ-ერთი ყველაზე გავრცელებულ საინსტრუმენტო საშუალებებს მიეკუთვნებოდა. დღეს ამ მხრივ სიტუაცია შეცვლილია. პასკალის განვითარება უკვე არ ხდება და ამის გამო საკმაოდ სწრაფად ამ ენის მომხმარებელთა რაოდენობა მკვეთრად შემცირდა. შედეგად, საჭირო გახდა დაპროგრამების სწავლების ახალი მეთოდოლოგიის შერჩევა [16,17].

პირველად პოსტის ვირტუალური მანქანის სასწავლო პროცესში გამოყენების იდეა გამოთქვა მოსკოვის სახელმწიფო უნივერსიტეტის თანამშრომელმა, ვლადიმერ უსპენსკიმ [18].

ეს იდეა მყისვე აიტაცა ქართველმა მათემატიკოსმა კონსტანტინე ცისკარიძემ და საკმაოდ მალე მისი ხელმძღვანელობით ახალგაზრდა სპეციალისტმა მარინა ხარაზიშვილი-დერბინიანმა შექმნა პოსტის დიალოგური მანქანის რეალიზაცია ეგმ ბესმ-6-თვის. უნდა ითქვას, რომ წლების განმავლობაში ამ რეალიზაციაზე დამყარებული მეთოდიკა წარმატებულად გამოიყენებოდა თსუ-ს გამოყენებითი მათემატიკის სამეცნიერო კვლევითი ინსტიტუტის ნორჩ მათემატიკოს-პროგრამისტთა სკოლაში [19].

ამ იდეაზე დაყრდნობით ავტორმა დაიწყო მუშაობა ეხლა უკვე ტიურინგის ვირტუალური მანქანის სასწავლო პროცესში გამოყენების შესაძლებლობაზე [4,5].

თავიდან შესრულდა სამოდელო რეალიზაციები ბესმ-6-თვის, რომლებიც არ იყო დაფიქსირებული პუბლიკაციის სახით (იყო სტუდენტთა საკურსო და სადიპლომო ნაშრომები). მაგრამ დარწმუნდა რა ამ იდეის ნაყოფიერებაში, ავტორმა უხელმძღვანელა შემუშავებული კონცეპციის რეალიზებას თანამედროვე კომპიუტერებზე [26,28].

რეალიზაციების უშუალო შემსრულებლები არიან მოყვანილი პუბლიკაციების თანაავტორები – ავთანდილ რუხაძე და სანდრო ბარნაბიშვილი. მოცემული მომენტისთვის ეს მეთოდოლოგია გამოიყენება სასწავლო პროცესში საქართველოს საპატრიარქოს წმიდა ანდრია პირველწოდებულის სახელობის ქართულ უნივერსიტეტსა და სკოლა-პანსიონ “მთიებში”.

შედეგები და მათი განსჯა

თავი 1. ფუნქციონალურ გამოსახულებათა დამუშავების ზოგიერთი მეთოდი

1.1. ფუნქციონალურ გამოსახულებათა ჩაწერის პოსტფიქსური და პრეფიქსული ნოტაციები

სისტემურ პროგრამირებაში ფართოდ გამოიყენება გამოსახულებათა ჩაწერის უფრჩხილებო ნოტაციები, რომლებსაც პოლონურ ჩანაწერებს უწოდებენ მათი ავტორის, პოლონელი მათემატიკოსის, იან ლუკასევიჩის (*Jan Łukasiewicz)* საპატივცემულოდ.

განასხვავებენ პოლონური ჩანაწერების 2 ნაირსახეობას – პოსტფიქსურს და პრეფიქსულს. პირველ შემთხვევაში ოპერაციის ნიშანი იწერება უშუალოდ შესაბამისი ოპერანდების შემდეგ, ხოლო მეორე შემთხვევაში კი უშუალოდ ოპერანდების წინ.

მაგალითად, ვთქვათ გვაქვს გამოსახულება, ჩაწერილი დაპროგრამების ენებში მიღებული სახით:

(a+b)*c-d/(e-f*g),

სადაც ლათინური ასოებით აღნიშნულია ოპერანდები. გამოსახულების ჩაწერის ასეთ ფორმას, როგორც წესი ინფიქსურს უწოდებენ (რადგან აქ 2-ადგილიანი ოპერაციების ნიშნები ოპერანდებს შორის იწერება).

ასეთ გამოსახულებას შეესაბამება შემდეგი პოსტფიქსური ჩანაწერი:

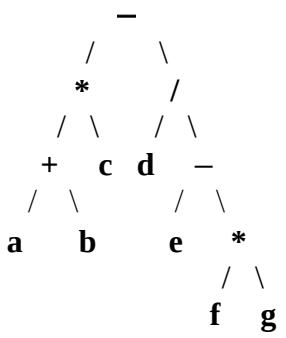
ab+c*defg*-/-

მართლაც, გამოსახულებას a+b ზემოთქმულის მიხედვით შეესაბამება ab+ , ამის შემდეგ, თუ განვიხილავთ ამ უკანასკნელს, როგორც ოპერანდს, მაშინ ab+c* იქნება (a+b)*c გამოსახულების შესატყვისი პოსტფიქსური ჩანაწერი. ანალოგიურად, ინფიქსურ ჩანაწერს e-f*g შეესაბამება პოსტფიქსური ჩანაწერი efg*- , ხოლო გამოსახულებას d/(e-f*g) კი შეესაბამება პოსტფიქსური ჩანაწერი defg*-/ . ბოლოს, მიღებული ორი პოსტფიქსური ჩანაწერის, როგორც ოპერანდების, მეშვეობით ვღებულობთ საწყისი გამოსახულების პოსტფიქსურ ექვივალენტს ab+c*defg*-/- .

აბსოლუტურად ანალოგიურად შეგვიძლია დავრწმუნდეთ, რომ საწყისი გამოსახულების პრეფიქსული ფორმაა

-*+abc/d-e*fg

თუ საწყის გამოსახულებას წარმოვიდგენთ გრაფის (ხის) სახით შემდეგნაირად (კვანძებში ოპერაციების ნიშნებია, ხოლო შესაბამისი ოპერანდები - “ფოთლებია”):



მაშინ ადვილად დავინახავთ, რომ გამოსახულების ჩაწერის ორივე ზემოგანხილული ფორმა მიიღება ამ გრაფის შემოვლით.

პოსტფიქსური გამოსახულება მიიღება, თუ გრაფის შემოვლას ვაწარმოებთ მარცხნიდან მარჯვნივ იმ პირობით, რომ გრაფის კვანძებში მოთავსებულ ოპერაციების ნიშნებს გადავიტანთ მისაღებ გამოსახულებაში მხოლოდ იმის შემდეგ, რაც მთლიანად დასრულდება მისი ყველა ფოთლის ამოღება.

გამოსახულების პრეფიქსულ ფორმას მივიღებთ იგივე გრაფის შემოვლით ზევიდან ქვემოთ მარცნიდან მარჯვნივ იმ პირობით, რომ ოპერანდების ამოღება მოხდეს იმის შემდეგ, რაც ამოღებულია შესაბამისი ოპერაციის ნიშანი.

პოსტფიქსური ფორმა არსებითად აადვილებს ალგებრულ გამოსახულებათა რიცხვითი მნიშვნელობების გამოთვლას და ფართოდ გამოიყენება პროგრამირების ენების კომპილაციის თეორიასა და პრაქტიკაში. ეს გამოწვეულია იმით, რომ პოსტფიქსური ფორმა მოსახერხებელი სახით შეიცავს გამოსახულების გამოთვლისთვის აუცილებელ სრულ ინფორმაციას. გამოსახულების ჩაწერის ამ ფორმაზე დაყრდნობით მისი მნიშვნელობა გამოითვლება მარტივი ალგორითმით, რომელსაც ქვემოთ მოვიყვანთ.

შევთანხმდეთ, რომ შემდგომში პროგრამირებაში გავრცელებული ტერმინების “არითმეტიკული გამოსახულება” და “ალგებრული გამოსახულება” სინონიმად, გამოვიყენებთ ტერმინს “ფუნქციონალური

გამოსახულება”. გარდა ამისა, პოსტფიქსური ჩანაწერის შემადგენელ ცალკეულ ოპერანდებს და ოპერაციების ნიშნებს ვუწოდოთ ჩანაწერის ელემენტები. ამრიგად, შეგვიძლია ვთქვათ, რომ პოსტფიქსური ჩანაწერი წარმოადგენს მიმდევრობას, რომლის ყოველი ელემენტი არის ან ოპერანდის აღმნიშვნელი (მოკლედ, ოპერანდი), ან ოპერაციის აღმნიშვნელი (მოკლედ, ოპერაცია). ცხადია, ზუსტად იგივე შეიძლება ითქვას პრეფიქსული ჩანაწერის შესახებ.

სანამ ჩამოვაყალიბებთ პოსტფიქსური გამოსახულების მნიშვნელობის გამოთვლის ალგორითმს, მიზანშეწონილია გავიხსენოთ სტეკის ცნება, რადგან ეს ალგორითმი იყენებს სტეკს.

სტეკის ქვეშ გულისხმობენ მონაცემთა ისეთ უმისამართო სტრუქტურას, რომელიც ფუნქიონირებს პრინციპით – ბოლო მოხვედი, პირველი მიდიხარ (LIFO – Last In First Out). ადგილს სტეკში, სადაც თავსდება ყველაზე ბოლოს მიღებული ელემენტი, სარკმელს უწოდებენ. ზემოთქმულიდან გასაგებია, რომ სტეკიდან ელემენტის აღება ხდება სწორედ მისი სარკმელიდან. სტეკის ყოველი მდგომარეობა ხასიათდება სიღრმით (სტეკში მოთავსებული ელემენტების რაოდენობით) და სარკმელის შიგთავსით. სტეკთან მუშაობისთვის აუცილებელია სულ 3 მოქმედების განხორციელება, ესენია: სტეკში მოთავსება (ჩაწერა), სტეკიდან ამოღება (წაკითხვა), სტეკის შემოწმება დაცლაზე.

ეხლა კი ჩამოვაყალიბოთ პოსტფიქსური ფორმით ჩაწერილი გამოსახულების მნიშვნელობის გამოთვლის ალგორითმი. ქვემოდ ყველგან პუნქტებიანი ალგორითმების აღწერის დროს ვიგულისხმებთ, რომ პუნქტების ნუმერაცია განსაზღვრავს ამ პუნქტების შესრულების თანმიმდევრობას, გარდა იმ შემთხვევებისა, როდესაც ესა თუ ის პუნქტი ცხადი სახით თავად მიუთითებს მის შემდეგ შესასრულებელ პუნქტს, ან მოქმედებათა დასრულებას.

1. ავიღოთ ცარიელი სტეკი;
2. განვახორციელოთ პოსტფიქსური ჩანაწერის მორიგი ელემენტის აღების (წაკითხვის) მცდელობა. თუ ეს მცდელობა წარუმატებელია, ანუ ჩანაწერი უკვე ამოიწურა, შევასრულოთ პუნქტი 5;

3. თუ წაკითხული ელემენტი წარმოადგენს ოპერანდს (ოპერანდის აღნიშვნას), მოვათავსოთ იგი სტეკში და გადავიდეთ პუქნტ 2-ზე;
4. რადგან ჩანაწერის მორიგი ელემენტი ოპერაციის ნიშანია, შევასრულოთ იგი და შედეგი მოვათავსოთ სტეკში. ამისათვის, ამოვიღოთ სტეკიდან ამ ოპერაციის შესრულებისთვის საჭირო ოპერანდების რაოდენობა (ანუ იმდენი ოპერანდი, რამდენ ადგილიანიც არის ეს ოპერაცია), შევასრულოთ ამ ოპერანდებზე (ამ ოპერანდების მნიშვნელობებზე) ხსენებული ოპერაციის შესატყვისი მოქმედება, იმის გათვალისწინებით, რომ ოპერანდების ამოღება ხდებოდა <u>შებრუნებული მიმდევრობით</u> (ვთქვათ, გამოკლების შემთხვევაში, ჯერ მაკლები, ხოლო შემდეგ კი საკლები) და მიღებული შედეგი მოვათავსოთ სტეკში, რის შემდეგაც გადავიდეთ პუნქტ 2-ზე;
5. ავიღოთ შედეგი სტეკის სარკმელიდან და დავასრულოთ ალგორითმის შესრულება.

ავღნიშნოთ ეს ალგორითმი ასოთი A. ზემომოყვანილი გამოსახულების ab+c*defg*-/- მიმართ ამ ალგორითმის მიყენების დროს **სტეკი თანმიმდევრულად მიიღებს შემდეგ მდგომარეობებს:**

წაკითხული ელემენტი	a	b	+	c	*	d	e	f	g
სტეკის მდგომარეობა	a	b a	a+b	c a+b	(a+b)*c	d (a+b)*c	e d (a+b)*c	f e d (a+b)*c	g f e d (a+b)*c

წაკითხული ელემენტი	*	–	/	–
სტეკის მდგომარეობა	f*g e d (a+b)*c	e-f*g d (a+b)*c	d/(e-f*g) (a+b)*c	(a+b)*c- d/(e-f*g)

ცხრილი 1. პოსტფიქსური გამოსახულების A ალგორითმით გამოთვლა.

როგორც ვხედავთ, არსად არ დაგვჭირდა არც ოპერაციების პრიორიტეტის და არც ფრჩხილების გათვალისწინება. ოპერაციებისგან დაგვჭირდა მხოლოდ უშუალოდ მათი შესრულების წესი, რომელიც, ცხადია, მოიცავს ამ ოპერაციების ოპერანდების რაოდენობას. მივაქციოთ ყურადღება იმას, რომ მოყვანილი გამოსახულების მნიშვნელობის მიღებისთვის საკმარისი აღმოჩნდა სტეკი მაქსიმალური სიღრმით 5. საზოგადოდ, გამოსახულების გამოთვლის პროცესში გამოყენებული სტეკის მაქსიმალური სიღრმე ამ გამოსახულების სირთულის ერთ-ერთ საზომად შეიძლება გამოდგეს.

გამოსახულებათა ჩაწერის პოსტფიქსური ფორმა ფართოდ გამოიყენებოდა არა მარტო კომპილატორების პროექტირების დროს, არამედ მიკროკალკულატორებში. მთელ რიგ მიკროკალკულატორში გათვალისწინებული იყო პოსტფიქსური ჩანაწერების მხარდაჭერა. სამეცნიერო და საინჟინრო გამთვლებისთვის შექმნილ მიკროკალკულატორებში პოსტფიქსური ჩანაწერების მხარდაჭერა პირველ რიგში იმით აიხსნებოდა, რომ შედარებით მცირე რესურსის მქონე მოწყობილობებისთვის გადამწყვეტი იყო პოსტფიქსური გამოსახულებების დამუშავების პროცესის მცირე მოთხოვნები ამ რესურსების მიმართ. კერძოდ ის, რომ პოსტფიქსური ფორმით ჩაწერილი გამოსახულების გამოთვლა არ არის დაკავშირებული შუალედური შედეგების ცალკე დამახსოვრების აუცილებლობასთან – ყველა შუალედური შედეგი სტეკში ინახება გამოსახულების დამუშავების პროცესში.

რადგან ფუნქციონალური გამოსახულების ჩაწერის პოსტფიქსური ფორმა აგრერიგად მოხერხებულია გამოსახულების მნიშვნელობის გამოთვლისთვის, ბუნებრივად ისმის კითხვა გამოსახულების ინფიქსური (ანუ ბუნებრივთან ახლო) ფორმიდან მისი პოსტფიქსური ფორმის მიღების თაობაზე. ეს საკითხი საკმაოდ ლამაზად გადაჭრა ცნობილმა ჰოლანდიელმა მათემატიკოსმა ედსგარ დეიქსტრამ (Edsgar Dijkstra), რომელიც აღიარებულია სისტემური პროგრამირების კლასიკოსად. მის მიერ შემოთავაზებულ ალგორითმში კვლავ სტეკი გამოიყენება, ოღონდ ამჯერად სტეკში ხდება ოპერაციათა ნიშნების მოთავსება.

ეს ალგორითმი გულისხმობს შესასრულებელ ოპერაციათა რანჟირებას პრიორიტეტების მიხედვით. ამ ალგორითმის თვალსაზრისით შემავალი გამოსახულება შედგება ოპერანდებისგან (ცვლადები და კონსტანტები), მოქმედებათა აღნიშვნებისგან (ფუნქციები და ოპერაციები) და ფრჩხილებისგან (გახსნილი და დახურული). ალგორითმი შემდეგნაირად ყალიბდება:

1. ავიღოთ ცარიელი სტეკი;
2. განვახორციელოთ ინფიქსური ჩანაწერის მორიგი ელემენტის წაკითხვის მცდელობა. თუ ეს მცდელობა წარუმატებელია, ანუ გამოსახულება უკვე ამოიწურა, მაშინ შეგასრულოთ პუნქტი 7;
3. თუ წაკითხული ელემენტი ოპერანდია, მივუერთოთ იგი გამოსავალ გამოსახულებას და გადავიდეთ პუნქტ 2-ზე;
4. თუ წაკითხული ელემენტი გახსნილი ფრჩხილია, ჩავწეროთ იგი სტეკში და გადავიდეთ პუნქტ 2-ზე;
5. თუ წაკითხული ელემენტი დახურული ფრჩხილია, მაშინ ამოვიღოთ სტეკიდან ყველა ელემენტი უახლოეს გახსნილ ფრჩხილამდე ჩათვლით. ყველა ამოღებული ელემენტი, გარდა გახსნილი ფრჩხილისა, ამოღების კვალობაზე მივუერთოთ გამოსავალ გამოსახულებას და გადავიდეთ პუნქტ 2-ზე;
6. თუ წაკითხული ელემენტი ოპერაციის ნიშანია, ან ფუნქციის აღნიშვნაა, მაშინ სანამ სტეკის სარკმელში იქნება მოცემული ოპერაციასთან ან ფუნქციასთან შედარებით უფრო მაღალი ან ტოლი პრიორიტეტის მქონე მოქმედების აღნიშვნა, ამოვიღოთ იგი სტეკიდან და მივუერთოთ გამოსავალ გამოსახულებას, ხოლო როგორც კი სტეკის სარკმელში არ იქნება მოცემულთან შედარებით უფრო მაღალი ან ტოლი პრიორიტეტის მქონე მოქმედების აღნიშვნა (იმ შემთხვევების ჩათვლით, როცა სტეკი ცარიელია, ან სტეკის სარკმელში გახსნილი ფრჩხილია), მოცემული მოქმედების აღნიშვნა მოვათავსოთ სტეკში და გადავიდეთ პუნქტ 2-ზე;
7. ამოვიღოთ სტეკიდან ყველა დარჩენილი ელემენტი და ამოღების კვალობაზე მოვათავსოთ გამოსავალ გამოსახულებაში. მიღებული

გამოსახულება ჩავთვალოთ ალგორითმის განხორციელების შედეგად და დავასრულოთ ალგორითმის შესრულება.

არც თუ იშვიათად, სხვადასხვა წყაროში, ამ ალგორითმს მოიხსენიებენ დასახელებით “დამხარისხებელი სადგური”. საფიქრალია, რომ ასეთი დასახელება განპირობებულია ამ ალგორითმით გათვალისწინებული მოპყრობით ოპერაციების მიმართ, რომელშიც დახარისხების ელემენტს ადგილი ნამდვილად აქვს.

ცხრილის სახით გავუკეთოთ ილუსტრაცია ამ ალგორითმის შესრულებას ჩვენს მიერ ადრე განხილული ინფიქსური გამოსახულების (a+b)*c- d/(e-f*g) მიმართ.

წაკითხული ელემენტი	(	a	+	b	)	*	c	-	d	/
სტეკის მდგომარეობა	(	(	+ (	+ (		* 	* 	- 	- 	/ -
გამოსავალი გამოსახულება		a	a	ab	ab+	ab+	ab+c	ab+c*	ab+c*d	ab+c*d

წაკითხული ელემენტი	(	e	-	f	*	g
სტეკის მდგომარეობა	(/ -	(/ -	- (/ -	- (/ -	* - (/ -	* - (/ -
გამოსავალი გამოსახულება	ab+c*d	ab+c*de	ab+c*de	ab+c*def	ab+c*def	ab+c*defg

წაკითხული ელემენტი	)	┤(გამოსახულება ამოიწურა)
სტეკის მდგომარეობა	/ -	
გამოსავალი გამოსახულება	ab+c*defg*-	ab+c*defg*-/-

ცხრილი 2. ინფიქსური გამოსახულებიდან პოსტფიქსურის მიღება.

როგორც ვხედავთ, აქაც სტეკის მაქსიმალური სიღრმეა 5.

ეხლა მივაქციოთ ყურადღება გამოსახულების ჩაწერის პრეფიქსულ ფორმას. პრეფიქსული ჩანაწერი მოსახერხებელია გამოსახულების ფუნქციონალური დამუშავებისთვის, რადგან აქ აქცენტი სწორედ მოქმედებებზეა გაკეთებული. თქმულის საილუსტრაციოდ მოვიყვანოთ რამოდენიმე ფუნქციონალური თანაფარდობის ჩაწერა ტრადიციული მათემატიკური ფორმით და შევადაროთ ისინი მათ შესატყვისებს პრეფიქსული ნოტაციის გამოყენებით.

თანაფარდობა ტრადიციული ფორმით	თანაფარდობა პოსტფიქსური ნოტაციის გამოყენებით
(const)’ = 0	‘ const = 0
(x)’ = 1	‘ x = 1
$(x^n)’ = nx^{n-1}$	‘^x n = *n^x –n 1
$(a^x)’ = a^x * \ln(a)$	‘^a x = *^a x ln a
(u+v)’ = u’+v’	‘+u v = +’u’v
(u*v)’ = u’*v+u*v’	‘*u v = +*’u v * u ’v
(sin(x))’ = cos(x)	‘ sin x = cos x
$(u/v)’ = (u’*v-u*v’)/v^2$	‘ / u v = / - * ‘u v * u ‘v ^ v 2

ცხრილი 3. გაწარმოების ცხრილი ტრადიციული და პრეფიქსული ნოტაციით.

აქ ახარისხება ავღნიშნეთ სიმბოლოთი “^”.

ბუნებრივია მოვიძიოთ შესაძლებლობა პრეფიქსული ჩანაწერის მიღებისა ინფიქსური ჩანაწერიდან. ცხადია, აქ პირველ რიგში უნდა მოისინჯოს ინფიქსურიდან პოსტფიქსური ჩანაწერის დეიქსტრასეული ალგორითმის მოდიფიცირება. ადვილად ვრწმუნდებით, რომ ეს შესაძლებელია, მაგრამ რთულდება პუნქტების 5, 6 და 7 ანალოგების ჩამოყალიბება, რადგან ამ შემთხვევაში ცალკე პრობლემაა ოპერაციის ნიშნისთვის გამოსახულებაში ადგილის მონახვა. ტექსტი “მივუერთოთ გამოსავალ გამოსახულებას”, ან “მოვათავსოთ გამოსავალ გამოსახულებაში” თავის მხრივ დაზუსტებას მოითხოვს. მართლაც, როდესაც გამოსავალი გამოსახულებაა “ab”, მაშინ ნიშანი “+” თავსდება ამ გამოსახულების თავში, ასევე უნდა მოვიქცეთ, როცა გამოსახულებაა “+abc” და მოსანახია ადგილი გამრავლების

ნიშნისთვის: “*+abc” . მაგრამ როცა მიღებული გამოსახულებაა “*+abcdefg” და ამ გამოსახულებაში მოსათავსებელია ნიშანი “*”, იგი უნდა მოთავსდეს მარჯვენა 2 ოპერანდის წინ: “*+abcde*fg”. ამის შემდეგ, ნიშანი “-” უნდა მოთავსდეს კვლავ მარჯვნიდან მეორე ოპერანდის წინ, მაგრამ ამჯერად ეს მეორე ოპერანდია “e”, რადგან “*fg” მთლიანად ოპერანდს წარმოადგენს: “*+abcd-e*fg”. ანალოგიურად “/” თავსდება “d”-ს წინ, ხოლო ბოლო ნიშანი “-” თავსდება გამოსახულების თავში: “-*+abc/d-e*fg”.

ადვილი შესამჩნევია, რომ გამოსახულებაში მოქმედების ყოველი ნიშნისთვის სწორი ადგილის მონახვისთვის საჭიროა მისი ანალიზი თავიდან (მხოლოდ მარჯვნიდან დათვალიერება არაფერს არ იძლევა, რადგან შესაძლებელია, რომ მოქმედების ნიშნები მხოლოდ თავში იყოს). ცხადია, ეს ძალზედ მოუხერხებელია.

გაცილებით გამართლებულია დეიქსტრას ალგორითმის შემდეგი მოდიფიკაცია:

1. ავიღოთ 2 ცარიელი სტეკი და მივანიჭოთ მათ ნომრები 1 და 2;
2. განვახორციელოთ ინფიქსური ჩანაწერის მორიგი ელემენტის წაკითხვის მცდელობა. თუ ეს მცდელობა წარუმატებელია, ანუ გამოსახულება უკვე ამოიწურა, მაშინ შევასრულოთ პუნქტი 8;
3. თუ წაკითხული ელემენტი ოპერანდია, მოვათავსოთ იგი №2 სტეკში და გადავიდეთ პუნქტ 2-ზე;
4. თუ წაკითხული ელემენტი გახსნილი ფრჩხილია, ჩავწეროთ იგი №1 სტეკში და გადავიდეთ პუნქტ 2-ზე;
5. თუ წაკითხული ელემენტი დახურული ფრჩხილია, მაშინ ამოვიღოთ №1 სტეკიდან ყველა ელემენტი უახლოეს გახსნილ ფრჩხილამდე ჩათვლით. ყველა ამოღებული ელემენტისთვის, გარდა გახსნილი ფრჩხილისა, ამოღების კვალობაზე შევასრულოთ პუნქტი 7 და გადავიდეთ პუნქტ 2-ზე;
6. თუ წაკითხული ელემენტი ოპერაციის ნიშანია, ან ფუნქციის აღნიშვნაა, მაშინ სანამ №1 სტეკის სარკმელში იქნება მოცემული ოპერაციის ან ფუნქციასთან შედარებით უფრო მაღალი ან ტოლი პრიორიტეტის მქონე მოქმედების აღნიშვნა, ამოვიღოთ იგი

სტეკიდან და შევასრულოთ მის მიმართ პუნქტი 7, ხოლო როგორც კი სტეკის სარკმელში არ იქნება მოცემულთან შედარებით უფრო მაღალი ან ტოლი პრიორიტეტის მქონე მოქმედების აღნიშვნა (იმ შემთხვევების ჩათვლით, როცა სტეკი ცარიელია, ან სტეკის სარკმელში გახსნილი ფრჩხილია), მოცემული მოქმედების აღნიშვნა მოვათავსოთ №1 სტეკში და გადავიდეთ პუნქტ 2-ზე;

7. მოცემული მოქმედების ნიშნის მიხედვით შევასრულოთ საჭირო ოპერანდების რაოდენობის ტოლი წაკითხვები №2 სტეკიდან, ამოღებულ ელემენტებს წინ მივუერთოთ ეს მოქმედების ნიშანი იმის გათვალისწინებით, რომ ოპერანდების ამოღება ხდებოდა პირუკუ მიმდევრობით და მიღებული გამოსახულება მოვათავსოთ №2 სტეკში, რის შემდეგ დავბრუნდეთ ალგორითმის იმ წერტილში, საიდანაც მოხდა მიმართვა პუნქტ 7-ზე;
8. ამოვიღოთ №1 სტეკიდან ყველა დარჩენილი ელემენტი და ამოღების კვალობაზე შევასრულოთ მათ მიმართ პუნქტი 7. №2 სტეკის სარკმელში მიღებული გამოსახულება ჩავთვალოთ ალგორითმის განხორციელების შედეგად და დავასრულოთ ალგორითმის შესრულება.

მოყვანილი ალგორითმის მიხედვით საწყისი ინფიქსური გამოსახულებიდან პრეფიქსული ჩანაწერის მიღება მოხდება შემდეგი ცხრილის შესაბამისად (იხ. **ცხრილი 4.**)

როგორც ვხედავთ, ინფიქსური გამოსახულებიდან პრეფიქსულის მიღება უფრო რთულია, ვიდრე იგივე ინფიქსურიდან პოსტფიქსურის მიღება, მაგრამ განსხვავება საკმაოდ მცირეა მეორე სტეკის ეფექტიანად გამოყენების წყალობით. საინტერესოა, რომ ამ შემთხვევაშიც გამოყენებული სტეკების მაქსიმალური სიღრმეა 5.

საკმაოდ კარგი გამოსავალია აგრეთვე პრეფიქსული ჩანაწერის მიღება პოსტფიქსური ჩანაწერის საფუძველზე. გარდაქმნა, რომელიც პოსტფიქსური ჩანაწერიდან შესატყვის პრეფიქსულ ჩანაწერს იძლევა ავღნიშნოთ სიმბოლოთი P. შესაბამისი ალგორითმი შემდეგნაირად გამოიყურება.

წაკითხული ელემენტი	(	a	+	b	)	*	c	-	d	/
№1 სტეკის მდგომარეობა	(	(	+ (	+ (		*	*	-	-	/ -
№2 სტეკის მდგომარეობა		a	a	b a	+ab	+ab	c +ab	*+abc	d *+abc	d *+abc

წაკითხული ელემენტი	(	e	-	f	*	g
№1 სტეკის მდგომარეობა	(/ -	(/ -	- (/ -	- (/ -	* - (/ -	* - (/ -
№2 სტეკის მდგომარეობა	d *+abc	e d *+abc	e d *+abc	f e d *+abc	f e d *+abc	g f e d *+abc

წაკითხული ელემენტი	)				┤(გამოსახულება ამოიწურა)	
№1 სტეკის მდგომარეობა	* - (/ -	- (/ -	(/ -	/ -	-	
№2 სტეკის მდგომარეობა	g f e d *+abc	*fg e d *+abc	-e*fg d *+abc	-e*fg d *+abc	/d-e*fg *+abc	-*+abc/d-e*fg

ცხრილი 4. ინფიქსური გამოსახულებიდან პირდაპირ პრეფიქსულის მიღება.

1. ავიღოთ ცარიელი სტეკი;
2. განვახორციელოთ პოსტფიქსური ჩანაწერის მორიგი ელემენტის აღების (წაკითხვის) მცდელობა. თუ ეს მცდელობა წარუმატებელია, ანუ ჩანაწერი უკვე ამოიწურა, შევასრულოთ პუნქტი 5;
3. თუ წაკითხული ელემენტი წარმოადგენს ოპერანდს (ოპერანდის აღნიშვნას), მოვათავსოთ იგი სტეკში და გადავიდეთ პუქნტ 2-ზე;
4. რადგან ჩანაწერის მორიგი ელემენტი ოპერაციის ნიშანია, ამოვიღოთ სტეკიდან ამ ოპერაციის შესრულებისთვის საჭირო ოპერანდების რაოდენობის ელემენტი (ანუ იმდენი ელემენტი, რამდენ ადგილიანიც არის ეს ოპერაცია), მივაწეროთ ამ ელემენტებს ზემოხსენებული ოპერაციის ნიშანი, იმის გათვალისწინებით, რომ ელემენტების ამოღება ხდებოდა <u>შებრუნებელი მიმდევრობით</u> და მიღებული გამოსახულება მოვათავსოთ სტეკში, რის შემდეგაც გადავიდეთ პუნქტ 2-ზე;
5. ავიღოთ შედეგი სტეკის სარკმელიდან, დავასრულოთ ალგორითმის შესრულება.

ვნახოთ, როგორ წარიმართებოდა ჩამოყალიბებული ალგორითმის მიხედვით, ჩვენს მიერ განხილული პოსტფიქსური ჩანაწერიდან

ab+c*defg*-/-

მისი შესატყვისი პრეფიქსული ჩანაწერის მიღება. ცხრილის სახით ეს პროცესი ქვემოთ არის მოყვანილი.

წაკითხული ელემენტი	a	b	+	c	*	d	e	f	g
სტეკის მდგომარეობა	a	b a	+ab	c +ab	*+ab c	d *+abc	e d *+abc	f e d *+abc	g f e d *+abc

წაკითხული ელემენტი	*	-	/	-

სტეკის მდგომარეობა	*fg e d *+abc	-e*fg d *+abc	/d-e*fg *+abc	-*+abc /d-e*fg

ცხრილი 5. პოსტფიქსურიდან პირდაპირ პრეფიქსული გამოსახულების მიღება.

როგორც ვხედავთ, პოსტფიქსური ჩანაწერიდან პრეფიქსულის მიღება საკმაოდ მარტივი ალგორითმით ხერხდება. ეს ალგორითმი სირთულით პრაქტიკულად იდენტურია ალგორითმისა, რომელიც ჩვენ ავღნიშნეთ სიმბოლოთი A და რომელიც გამოიყენება პოსტფიქსური ჩანაწერის მნიშვნელობის გამოთვლისთვის.

ადვილად შეიძლება დავრწმუნდეთ, რომ პრეფიქსული ჩანაწერის მიხედვით გამოსახულების მნიშვნელობის გამოთვლა საკმაოდ რთულია. ასევე ადვილია იმაში დარწმუნება, რომ ჩვენს მიერ ადრე მოყვანილი ფუნქციონალური გარდაქმნები პოსტფიქსური ფორმით ჩამოყალიბების შემთხვევაში საფრძნობლად მოუხერხებელი იქნება პრეფიქსული ნოტაციის მეშვეობით ჩამოყალიბებულებთან შედარებით.

ყოველივე ზემოთქმულიდან სავსებით ბუნებრივად შეიძლება გაკეთდეს შემდეგი დასკვნა – მიუხედავად იმისა რომ ორივე სახის უფრჩხილებო ჩანაწერი აიგება მსგავს პრინციპებზე დაყრდნობით და განსხვავება, ერთი შეხედვით, ერთობ უმნიშვნელოა (მოქმედების ნიშანი განთავსდეს ოპერანდების მარჯვნივ თუ მარცხნივ), ეს ჩანაწერები განსხვავებული ღირებულებისაა იმისდა მიხედვით, რისთვის არის გამიზნული მათი გამოყენება. სახელდობრ, გამოსახულებათა ჩაწერის პოსტფიქსური ნოტაცია მოსახერხებელია ამ გამოსახულების მნიშვნელობის გამოთვლისთვის, ხოლო პრეფიქსული ნოტაცია კი უფრო მოსახერხებელია გამოსახულების ფუნქციონალური გარდაქმნის შემთხვევაში. აქედან გამომდინარე გასაგებია, რომ **ფუნქციონალურ გამოსახულებათა ჩაწერის ორივე ფორმა აქტუალურია, რადგან თითოეულ მათგანს აქვს მკვეთრად გამოხატული ინდივიდუალური უპირატესობები გამოყენების განსხვავებული შემთხვევებისთვის**.

რადგან პოსტფიქსური ჩანაწერიდან პრეფიქსულის მიღება ჩვენ უკვე ვნახეთ როგორც ხდება (დავრწმუნდით, რომ საკმაოდ მარტივად),

ბუნებრივად ისმის საკითხი პრეფიქსული ფორმიდან შესაბამისი პოსტფიქსური ჩანაწერის მიღების თაობაზე, ანუ P^{-1} გარდაქმნის განხორციელებაზე. ეს საკითხი კიდევ უფრო აქტუალური ხდება იმის გათვალისწინებით, რომ არსებობს საკმაოდ მნიშვნელოვანი ამოცანები, რომელთა გადაწყვეტის პროცესში პერიოდულად იქმნება როგორც ფუნქციონალური გარდაქმნების შესრულების, ასევე გარდაქმნილი გამოსახულებების მნიშვნელობათა გამოთვლის აუცილებლობა. შესაბამისად, წარმოიშვება მოთხოვნილება ერთდროულად ვიქონიოთ აქტიურ მდგომარეობაში გამოსახულების ორივე უფრჩხილებო ფორმა. და რადგან ფუნქციონალური გარდაქმნების განხორციელება უფრო მოსახერხებელია სწორედ პრეფიქსული ნოტაციის გამოყენებით, ერთი შეხედვით აქტუალობას იძენს პრეფიქსულიდან პოსტფიქსური გამოსახულების მიღების პრობლემის გადაწყვეტა.

ამავე დროს, მიუხედავად იმისა, რომ როგორც ჩვენ უკვე ვნახეთ, პოსტფიქსურიდან პრეფიქსული ფორმა ადვილად მიიღება, საპირისპირო გარდაქმნა უშუალოდ საგრძნობლად უფრო რთული განსახორციელებელია.

მაგრამ აღმოჩნდა, რომ ამ პრობლემას მთლიანად აგვარებს პოსტფიქსური გამოსახულების შეუღლებულის ცნებაზე დამყარებული ერთი მიდგომა, რომელსაც ქვემოდ ჩამოვაყალიბებთ.

1.2. პოსტფიქსური ჩანაწერის შეუღლებულის ცნება და მისი თვისებები

პოსტფიქსური ჩანაწერის შეუღლებულის ცნება არსებითად ემყარება ჩვენს მიერ სიმბოლო A–თი აღნიშნული ალგორითმის გარკვეულ მოდიფიკაციას, რომელსაც შემდგომში ჩვენ A–ალგორითმის შეუღლებულს ვუწოდებთ.

განსაზღვრა 1. A–ალგორითმის **შეუღლებული** ვუწოდოთ ალგორითმს, რომელიც ასევე მიეყენება პოსტფიქსურ ჩანაწერებს და მიიღება A–ალგორითმიდან, თუ პუნქტ 4-ში ხაზმგასმულ ტექსტს (“შებრუნებელი მიმდევრობით”) შევცვლით ტექსტით “პირდაპირი მიმდევრობით”.

A–ალგორითმის შეუღლებული ავღნიშნოთ სიმბოლოთი A*. შევნიშნოთ, რომ განსაზღვრიდან პირდაპირ გამომდინარეობს შემდეგი:

- ალგორითმებს A და A* იდენტური სირთულე აქვთ;
- (A*)* = A, ანუ შეუღლებულობა ურთიერთშებრუნებული ყოფილა.

განსაზღვრა 2. თუ v არის პოსტფიქსური გამოსახულება, მაშინ მისი **შეუღლებული** ვუწოდოთ ისეთ w გამოსახულებას, რომ

$$A^*(w) = A(v) \qquad (1)$$

თუ ადგილი აქვს (1)-ს, ანუ თუ w არის v–ს შეუღლებული, მაშინ ჩავწერთ:

$$w = v^* \qquad (2)$$

(1)-ში A შევცვალოთ A* -ით. მაშინ, რადგან (A*)* = A, მივიღებთ, რომ A(w) = A*(v). საიდანაც პირდაპირ ვღებულობთ, რომ v = w*. რაც საბოლოო ჯამში ნიშნავს, რომ ნებისმიერი პოსტფიქსური v გამოსახულებისთვის

$$(v^*)^* = v \qquad (3)$$

ცხადია, პოსტფიქსური გამოსახულების შეუღლებული ასევე პოსტფიქსური გამოსახულებაა.

სიმბოლოთი C ავღნიშნოთ პოსტფიქსური გამოსახულების გარდაქმნა შეუღლებულზე (Conjugate - შეუღლებული). მაშინ განსაზღვრის მიხედვით ნებისმიერი v პოსტფიქსური გამოსახულებისთვის გვექნება:

$$C(v) = v^* . \qquad .$$

ცხადია, რომ ნებისმიერ პოსტფიქსურ გამოსახულებას გააჩნია შეუღლებული. გარდა ამისა, ახალი აღნიშვნის გამოყენებით (3) ჩაიწერება შემდეგნაირად:

$$C(C(v)) = v \qquad (3^*)$$

მოყვანილი განსაზღვრის მიხედვით, მაგალითად, გვექნება:

C(ab-) = ba-

C(ab-c/) = cba-/

ჩამოვაყალიბოთ C გარდაქმნის მარეალიზებელი ალგორითმი.

1. ავიღოთ ცარიელი სტეკი;
2. განვახორციელოთ პოსტფიქსური ჩანაწერის მორიგი ელემენტის აღების (წაკითხვის) მცდელობა. თუ ეს მცდელობა

წარუმატებელია, ანუ ჩანაწერი უკვე ამოიწურა, შევასრულოთ პუნქტი 5;

3. თუ წაკითხული ელემენტი წარმოადგენს ოპერანდს (ოპერანდის აღნიშვნას), მოვათავსოთ იგი სტეკში და გადავიდეთ პუქნტ 2-ზე;
4. რადგან ჩანაწერის მორიგი ელემენტი ოპერაციის ნიშანია, ამოვიღოთ სტეკიდან ამ ოპერაციის შესრულებისთვის საჭირო ოპერანდების რაოდენობის ელემენტი (ანუ იმდენი ელემენტი, რამდენ ადგილიანიც არის ეს ოპერაცია), მივაწეროთ ამ ელემენტებს ზემოხსენებული ოპერაციის ნიშანი მარჯვნიდან, იმის გათვალისწინებით, რომ ელემენტების ამოღება ხდებოდა პირდაპირი მიმდევრობით და მიღებული გამოსახულება მოვათავსოთ სტეკში, რის შემდეგაც გადავიდეთ პუნქტ 2-ზე;
5. ავიღოთ შედეგი სტეკის სარკმელიდან, დავასრულოთ ალგორითმის შესრულება.

ვნახოთ, მაგალითად როგორ მიიღება ზემოთგანხილული პოსტფიქსური ჩანაწერის შეუღლებული.

წაკითხული ელემენტი	a	b	+	c	*	d	e	f	g
სტეკის მდგომარეობა	a	b a	ba+	c ba+	cba+*	d cba+*	e d cba+*	f e d cba+*	g f e d cba+*

წაკითხული ელემენტი	*	-	/	-
სტეკის მდგომარეობა	gf* e d cba+*	gf*e- d cba+*	gf*e-d/ cba+*	gf*e-d/cba+*-

ცხრილი 6. პოსტფიქსური ჩანაწერიდან მისი შეუღლებულის მიღება.

როგორც ვხედავთ, C-ს მარეალიზებელი ალგორითმის სირთულე პრაქტიკულად იდენტურია A ალგორითმის სირთულის.

ავღნიშნოთ სიმბოლოთი R (Reverse - შებრუნებული) სასრული მიმდევრობის შებრუნების (რევერსირების) გარდაქმნა, ანუ გარდაქმნა, რომელიც წყვილ-წყვილად უნაცვლებს ადგილებს მიმდევრობის წევრებს, რომლებიც თანაბრად არიან დაშორებულნი ამ მიმდევრობის ბოლოებიდან.

ადგილი აქვს შემდეგ დებულებებს.

ლემა 1. თუ a და b სასრული მიმდევრობებია, მაშინ იმისათვის, რომ ადგილი ქონდეს ტოლობას

$b = R(a)$

აუცილებელი და საკმარისია, რომ:

არსებობდეს ისეთი ურთიერთ ცალსახა თანადობა a და b-ს წევრებს შორის, რომ:

1. $\forall a_i, a_j \in a$ & $\forall b_{i'}, b_{j'} \in b$ -თვის, თუ ამ თანადობით a_i გადადის $b_{i'}$-ში და a_j გადადის $b_{j'}$ -ში, მაშინ $i < j \Rightarrow i' > j'$.
2. წევრები, რომლებიც შეესაბამება ერთმანეთს ამ თანადობის მიხედვით, ტოლია.

დამტკიცება. აუცილებლობა ცხადია. ვაჩვენოთ საკმარისობა. ვთქვათ არსებობს დასამტკიცებელი დებულებით მოთხოვნილი ურთიერთ ცალსახა თანადობა. მაშინ, ამ თანადობის მიხედვით, a_1 შეიძლება შეესაბამებოდეს მხოლოდ b_n-ს. მეორე პუნქტის მიხედვით კი $a_1 = b_n$. ანალოგიურად, თუ კი არსებობს a_2, მაშინ ადგილი ექნება $a_2 = b_{n-1}$, და ა.შ. ამით ლემა მთლიანად დამტკიცდა.

ლემა 2. თუ b არის ნებისმიერი სასრული მიმდევრობა, მაშინ b-ს ნებისმიერი დაყოფისთვის უწყვეტ ქვემიმდევრობათა მიმდევრობად:

$b = B_1 ... B_n$

ადგილი აქვს ტოლობას:

$R(b) = R(B_n) ... R(B_1)$.

დამტკიცება. ავიღოთ b მიმდევრობის ნებისმიერი წევრი. ვთქვათ ის მოხვდა B_i ქვემიმდევრობის j ადგილზე. მაშინ b-ს ყველა წევრი, რომელიც იყო ამ წევრის მარცხნივ, R გარდაქმნის შემდეგ უნდა

მოხვდეს მის მარჯვნივ. მართლაც, თუ ეს წევრები სხვა ქვემიმდევრობაში იყო, მაშინ B_i-ს მარცხნივ მოთავსებული ყველა ქვემიმდევრომა (თავის წევრებიანად) გარდაქმნის შემდეგ აღმოჩნდება მის მარჯვნივ. თუ ეს ელემენტი იგივე ქვემიმდევრობაში იყო, მაშინ გარდაქმნის შემდეგ იგი აღმოჩნდება მის მარჯვნივ თავად B_i–ს შებრუნების შედეგად. ზუსტად ანალოგიურად დავრწმუნდებით, რომ თავდაპირველად არჩეული ელემენტის მარჯვნივ მყოფი ყველა წევრი გარდაქმნის შემდეგ მის მარცხნივ აღმოჩნდება. და რადგან მიმდევრობის ელემენტი ნებისმიერად იყო არჩეული, ამით ნაჩვენებია, რომ მიმდევრობა b–დან დასამტკიცებელი ტოლობის მარჯვენა მხარეში მოთავსებული მიმდევრობის მიმღები თანადობისთვის შესრულებულია ლემა 1-ის პირველი პირობა. და რადგან დასამტკიცებელი ტოლობის მარჯვენა მხარეში მოთავსებული მიმდევრობა შეიცავს b–მიმდევრობის ყველა წევრს და მხოლოდ მათ, აქედან ტრივიალურად გამომდინარეობს, რომ თანადობა, რომელიც იძლევა ამ მიმდევრობას აკმაყოფილებს ლემა 1-ის მეორე პირობასაც. ამით ლემა 2 მთლიანად დამტკიცებულია.

ორივე სახის უფრჩხილებო ჩანაწერს შეიძლება შევხედოთ, როგორც სასრულ მიმდევრობას, რომლის წევრები ამ ჩანაწერის ელემენტებია. მაშინ შებრუნების გარდაქმნა სავსებით ბუნებრივ აზრს იძენს ამ ჩანაწერების მიმართაც.

თეორემა 1. ნებისმიერი პოსტფიქსური v ჩანაწერისთვის ადგილი აქვს თანაფარდობას (გავიხსენოთ, რომ P(v) აღნიშნავს v–ს პრეფიქსულ შესატყვისს):

$$C(v) = R(P(v)) \qquad (4)$$

დამტკიცება. დამტკიცება ვაწარმოოთ ზოგადი ინდუქციით გამოსახულებაში **ოპერაციათა რაოდენობის მიმართ**, რომელსაც ავღნიშნავთ ასოთი n. როცა n=0, მაშინ დებულების ჭეშმარიტება ცხადია. ვთქვათ, დასამტკიცებელი თეორემა სამართლიანია ყველა არაუარყოფითი მთელი n<k–თვის, სადაც k>=1. ვაჩვენოთ მაშინ, რომ იგი იქნება სამართლიანი ასევე n=k –თვისაც. ვთქვათ, v რაიმე პოსტფიქსური გამოსახულებაა, რომლისთვისაც n=k. რადგან n>0, ამიტომ v-ში არის ერთი მაინც ოპერაციის ნიშანი. ვინაიდან ოპერაციის ერთი მაინც ნიშნის შემცველი პოსტფიქსური გამოსახულება

აუცილებლად ბოლოვდება ოპერაციის ნიშნით, v შეგვიძლია წარმოვიდგინოთ შემდეგნაირად v = w@ , სადაც სიმბოლო @ აღნიშნავს ჩანაწერის ბოლოს მოთავსებულ ოპერაციას. ვთვათ @ არის m–ადგილიანი ოპერაცია. მაშინ გვექნება $w = w_1...w_m$, სადაც თითოეული w_i წარმოადგენს პოსტფიქსურ ჩანაწერს, რომელშიც ოპერაციათა რაოდენობა მკაცრად ნაკლებია k–ზე, ანუ თითოეული w_i –თვის ინდუქციის დაშვების თანახმად დასამტკიცებელი დებულება სამართლიანია. ამიტომ შეგვიძლია დავწეროთ:

$C(v) = C(w_1...w_m@) = C(w_m)...C(w_1)@ = R(P(w_m))...R(P(w_1))@$, აქედან კი ახლადდამტკიცებული ლემა 2-ის თანახმად გვექნება:

$R(P(w_m))...R(P(w_1))@ = R(P(w_m))...R(P(w_1))R(@) = R(@P(w_1)... P(w_m)) =$

$=R(P(w_1...w_m@)) = R(P(v))$. ამით თეორემა მთლიანად დამტკიცდა.

შედეგი 1.1. ნებისმიერი პოსტფიქსური v ჩანაწერისთვის ადგილი აქვს თანაფარდობას:

$$P(v) = R(C(v)) \qquad (5)$$

დამტკიცება. (5) პირდაპირ მიიღება, თუ (4)-ის ორივე მხარეს მიუყენებთ გარდაქმნა R-ს და გავითვალისწინებთ ცხად თანაფარდობას R(R(w)) = w, სადაც w-ს ქვეშ შეგვიძლია ვიგულისხმოთ ნებისმიერი სასრული მიმდევრობა.

რადგან C(v) = v*, ამიტომ ეს შედეგი გვკარნახობს პოსტფიქსური გამოსახულებიდან პრეფიქსული გამოსახულების მიღების კიდევ ერთ გზას. სახელდობრ, *მოცემული პოსტფიქსური გამოსახულებიდან მივიღოთ შეუღლებული და შემდეგ კი ეს შეუღლებული გადავაბრუნოთ.* P-ს შებრუნებული გარდაქმნა ავღნიშნოთ P^{-1}. ცხადია, გარდაქმნა P^{-1} უნდა მიუყენოთ პრეფიქსულ ჩანაწერს, შედეგად კი მივიღებთ პოსტფიქსურ შესატყვისს.

შედეგი 1.2. ნებისმიერი პრეფიქსული x ჩანაწერისთვის ადგილი აქვს თანაფარდობას:

$$P^{-1}(x) = C(R(x)) \qquad (6)$$

დამტკიცება. ერთდროულად მიუყენოთ (4)-ის ორივე მხარეს გარდაქმნა C და ვისარგებლოდ ფორმულით (3*), მივიღებთ

$$v = C(R(P(v))) \qquad (7)$$

რადგან v – ნებისმიერი პოსტფიქსური გამოსახულებაა და ყოველ პრეფიქსულ გამოსახულებას აქვს პოსტფიქსური შესატყვისი და პირიქით, შეგვიძლია ავღნიშნოთ x = P(v). მაშინ (7)-დან პირდაპირ მივიღებთ (6)-ს.

ეხლა კი (6)-დან უშუალოდ ვასკვნით – *იმისათვის, რომ პრეფიქსული ჩანაწერიდან მივიღოთ პოსტფიქსური შესტყვისი, უნდა ავიღოთ შეუღლებული მისი გადაბრუნებით მიღებული გამოსახულებისგან.* შევნიშნოთ, რომ პრეფიქსული გამოსახულების შებრუნებით მიიღება პოსტფიქსური ჩანაწერი, თანაც, დამტკიცებულის მიხედვით, საწყისი პრეფიქსულის შესატყვისი პოსტფიქსურის მიმართ შეუღლებული.

შედეგი 1.3. თუ v - პოსტფიქსური ჩანაწერია, მაშინ:

$$R(v) = P(v^*) \qquad (8)$$

დამტკიცება. (8)-ს პირდაპირ მივიღებთ, თუ (5)-ის ორივე მხარეში v-ს ნაცვლად ჩავსვამთ v*-ს.

შედეგი 1.4. თუ x – პრეფიქსული ჩანაწერია, მაშინ:

$$R(x) = (P^{-1}(x))^* \qquad (9)$$

დამტკიცება. (9) პირდაპირ მიიღება, თუ (6)-ის ორივე მხარეს ერთდროულად მიუყენებთ გარდაქმნა C-ს.

თუ ყურადღებით დავაკვირდებით ბოლო 2 შედეგს, ადვილად დავასკვნით, რომ *პოსტფიქსური გამოსახულების დამუშავება მარჯვნიდან მარცხნივ (ანუ მისი ყოველგვარი სახეცვლილების გარეშე) ტოლძალოვანია მისი შეუღლებულის შესაბამისი პრეფიქსული გამოსახულების დამუშავებისა.*

ანალოგიურად, *პრეფიქსული ჩანაწერის დამუშავება მარჯვნიდან მარცხნივ, ტოლძალოვანია მისი პოსტფიქსური შესატყვისის შეუღლებულის დამუშავებისა.*

ფაქტობრივად, ზემოთქმულიდან გამომდინარეობს, რომ პოსტფიქსური გამოსახულების ქონა ტოლძალოვანია გვქონდეს მისი შეუღლებულის შესაბამისი პრეფიქსული გამოსახულება. ასევე, პრეფიქსული გამოსახულების ქონა ტოლძალოვანია გვქონდეს შესაბამისი პოსტფიქსური გამოსახულების შეუღლებული. მაგრამ, პოსტფიქსური გამოსახულების ქონა, თავის მხრივ, ტოლძალოვანია გქონდეს

შუღლებული, რადგან შეუღლებულისგან მოთხოვნილი შედეგი მიიღება საწყისი პოსტფიქსურიდან მის მიმართ A* ალგორითმის მიყენებით.

ყოველივე ზემოთქმული გვკარნახობს ფუნქიონალურ გამოსახულებათა დამუშავების სქემას, როდესაც ***საერთოდ არ არის საჭირო*** *პოსტფიქსური ჩანაწერის გარდაქმნა პრეფიქსულზე და პირიქით, პრეფიქსულის გარდაქმნა პოსტფიქსურზე.*

მართლაც, ვთქვათ გადასაწყვეტი ამოცანა დრო და დრო მოითხოვს საწყისი გამოსახულების ფუნქციონალურ გარდაქმნას, გარდაქმნილი გამოსახულებისთვის იმ ბლოკის გააქტიურებას, რომელიც არგუმენტების გარკვეული მნიშვნელობებისთვის მოახდენს მის დამუშავებას (ვთქვათ, გამოსახულების მნიშვნელობის გამოთვლას), შემდეგ კვლავ ფუნქციონალურ გარდაქმნას, ხსენებული ბლოკის დამუშავებას და ასე რამოდენიმეგზის. ერთი შეხედვით, შეიძლება ვიფიქროთ, რომ მსგავსი ამოცანის გადაწვეტისას აუცილებელია პრეფიქსული ჩანაწერიდან პოსტფიქსურის მიღების ალგორითმის განხორციელება პრეფიქსული ჩანაწერის ყოველი განახლების კვალობაზე და აქედან გამომდინარე, ამ ალგორითმის მაქსიმალური ოპტიმიზება. მაგრამ, ჩვენ უკვე მივაგენით სხვა გამოსავალს.

მართლაც, იმის გათვალისწინებით, რომ ალგორითმებს A და A* ახასიათებს სირთულეთა იდენტურობა, პოსტიფქსური ჩანაწერის დამუშავება არაფრით არ განსხვავდება შესაბამისი შეუღლებულის დამუშავებისგან. ასეთ შემთხვევაში, ცხადია, ფუნქციონალური გარდაქმნებისთვის უფრო მოსახერხებელია აქტიურ მდგომარეობაში გვქონდეს გამოსახულების პრეფიქსული ფორმა. **შესაბამისი პოსტფიქსური ჩანაწერის გამოყენების ეფექტს კი მივიღებთ ყოველგვარი გარდაქმნების გარეშე, თუ პრეფიქსულ ჩანაწერს დავამუშავებთ მარჯვნიდან მარცხნივ A* ალგორითმით!** ამრიგად, ეს მიდგომა, ზემონახსენები სახის ამოცანების ამოხსნისას, იძლევა შესაძლებლობას **კონცენტრაცია მთლიანად მოხდეს გამოსახულების პრეფიქსული ფორმის დამუშავებაზე**.

ბოლოს, ჩამოვაყალიბოთ კიდევ ერთი დებულება, რომელიც განსაზღვრავს პოსტფიქსური ჩანაწერის შეუღლებულის მიღების კიდევ ერთ წესს.

თეორემა 2. ნებისმიერი პოსტფიქსური v ჩანაწერისთვის ადგილი აქვს თანაფარდობას:

$$C(v) = P^{-1}(R(v)) \qquad (10)$$

დამტკიცება. (10) პირდაპირ მიიღება (6)-დან, თუ ავღნიშნავთ v=R(x) (ამის უფლება, ცხადია გვაქვს, რადგან R(x) პოსტფიქსური ჩანაწერია) და გავითვალისწინებთ, რომ მაშინ x=R(v).

ეხლა შემოვიღოთ პრეფიქსული ჩანაწერის შეუღლებულის ცნება და შევისწავლოთ მისი თვისებები. უნდა ველოდოთ სარკისებულ შედეგებს პოსტფიქსური ჩანაწერის შეუღლებულის ცნების საფუძველზე მიღებული შედეგების მიმართ.

1.3. პრეფიქსული ჩანაწერის შეუღლებულის ცნება და მისი თვისებები

განსაზღვრა 3. თუ x არის რაიმე გამოსახულების პრეფიქსული ჩანაწერი, მაშინ მისი **შეუღლებული** ვუწოდოთ ისეთ y გამოსახულებას, რომ

$$C(P^{-1}(y)) = P^{-1}(x) \qquad (11)$$

ანუ, გამოსახულების პრეფიქსული ჩანაწერის შეუღლებული ისეთი პრეფიქსული ჩანაწერია, რომლის შესატყვისი პოსტფიქსური ჩანაწერი საწყისი პრეფიქსული ჩანაწერის შესატყვისი პოსტფიქსურის შეუღლებულია.

ის ფაქტი, რომ y არის x პრეფიქსული ჩანაწერის შეუღლებული, ბუნებრივია, ავღნიშნოთ შემდეგნაირად:

$$y = x^*$$

ასევე ბუნებრივია, გარდაქმნა, რომელიც პრეფიქსული ჩანაწერიდან გვაძლევს მის შეუღლებულს ავღნიშნოთ სიმბოლოთი C.

ცხადია, რომ უშუალოდ განმარტების თანახმად ნებისმიერი პრეფიქსული x–თვის გვექნება:

$x^* = C(x)$, ასევე $(x^*)^* = x$ და $C(C(x)) = x$.

მაშინ ადრე დამტკიცებულზე და ამ განსაზღვრაზე დაყრდნობით ადვილად მიიღება შემდეგი დებულებები.

თეორემა 3. ნებისმიერი პრეფიქსული x ჩანაწერისთვის ადგილი აქვს თანაფარდობას:

$$C(x) = R(P^{-1}(x)) \qquad (12)$$

დამტკიცება. მართლაც, (11)-ში y–ის ნაცვლად ჩავწეროთ C(x), მივიღებთ:

$$C(P^{-1}(C(x))) = P^{-1}(x), \qquad (13)$$

აქედან ორივე მხარისთვის R–ის მიყენებით მივიღებთ:

$R(P^{-1}(x)) = R(C(P^{-1}(C(x))))$. ეხლა თუ გავითვალისწინებთ, რომ $P^{-1}(C(x))$ პოსტფიქსური ჩანაწერია და მის მიმართ გამოვიყენებთ (5)-ს, მივიღებთ $R(C(P^{-1}(C(x)))) = P(P^{-1}(C(x))) = C(x)$, რისი დამტკიცებაც გვინდოდა.

ეს შედეგი სარკისებულია თეორემა 1-ში ჩამოყალიბებული შედეგის მიმართ.

თეორემა 4. ნებისმიერი პრეფიქსული x ჩანაწერისთვის ადგილი აქვს თანაფარდობას:

$$C(x) = P(R(x)) \qquad (14)$$

დამტკიცება. მართლაც, (13)-ის ორივე მხარეში x შევცვალოთ C(x)-ით, მივიღებთ $C(P^{-1}(x)) = P^{-1}(C(x))$, აქედან კი, თუ ორივე მხარეს მიუყენებთ გარდაქმნა P-ს, გვექნება $C(x) = P(C(P^{-1}(x)))$, საიდანაც (9)-ის გათვალისწინებით უშუალოდ მიიღება $P(C(P^{-1}(x))) = P(R(x))$, რისი დამტკიცებაც გვინდოდა.

ცხადია, რომ ეს თეორემა 2-ის სარკისებული შედეგია.

შემდეგი 4 შედეგი სარკისებულია 1.1-1.4 შედეგების მიმართ.

შედეგი 2.1. ნებისმიერი პრეფიქსული x ჩანაწერისთვის ადგილი აქვს თანაფარდობას:

$$P^{-1}(x) = R(C(x)) \qquad (15)$$

დამტკიცება. (15) პირდაპირ მიიღება, თუ (12)-ის ორივე მხარეს მიუყენებთ გარდაქმნა R–ს.

შედეგი 2.2. ნებისმიერი პოსტფიქსური v ჩანაწერისთვის ადგილი აქვს თანაფარდობას:

$$P(v) = C(R(v)) \qquad (16)$$

დამტკიცება. (12)-ის ორივე მხარეს მიუყენოთ გარდაქმნა C, მივიღებთ $x=C(R(P^{-1}(x)))$. ეხლა (16)-ს დაუყოვნებლივ მივიღებთ, თუ ავღნიშნოთ $v=P^{-1}(x)$ და გავითვალისწინებთ, რომ მაშინ $x=P(v)$.

(15)-დან ვასკვნით, რომ *პრეფიქსული ჩანაწერიდან პოსტფიქსური შესატყვისის მიღებისთვის საჭიროა ამ პრეფიქსული ჩანაწერის შეუღლებულის შებრუნება.*

(16) კი გვკარნახობს, რომ *პოსტფიქსურო ჩანაწერის პრეფიქსული შესატყვისის მისაღებად საჭიროა პოსტფიქსური ჩანაწერის შებრუნებულის შეუღლება.*

ეს დასკვნები პრაქტიკულად ღირებული იქნებოდა რომ გვქონდეს პრეფიქსული ჩანაწერის შეუღლების პირდაპირი ალგორითმი. (12) და (6)-დან კი ვღებულობთ თანაფარდობას:

$$C(x) = R(C(R(x))), \qquad (17)$$

რაც გვაფიქრებს, რომ *პრეფიქსული ჩანაწერის შეუღლება, როგორც ჩანს,* ***არ არის*** *პოსტფიქსურის შეუღლებასთან შედარებით უფრო მოხერხებული.*

შედეგი 2.3. ნებისმიერი პრეფიქსული x ჩანაწერისთვის ადგილი აქვს თანაფარდობას:

$$R(x) = P^{-1}(x^*) \qquad (18)$$

დამტკიცება. (18) პირდაპირ მიიღება, თუ (15)-ის ორივე მხარეში x–ს შევცვლით x*-ით და გავითვალისწინებთ, რომ C(x*)=x.

შედეგი 2.4. ნებისმიერი პოსტფიქსური v ჩანაწერისთვის ადგილი აქვს თანაფარდობას:

$$R(v) = (P(v))^* \qquad (19)$$

დამტკიცება. (19)-ს დაუყოვნებლივ მივიღებთ, თუ (16)-ის ორივე მხარეს მიუყენებთ გარდაქმნა C–ს.

ბოლო ორი შედეგი გვკარნახობს პოსტფიქსური ჩანაწერების მიმართ შესაბამისი შედეგების საფუძველზე ჩამოყალიბებული დასკვნის სარკისებულ დასკვნას. კერძოდ, *პრეფიქსული გამოსახულების დამუშავება მარჯვნიდან მარცხნივ ტოლძალოვანია მისი შეუღლებულის შესაბამისი პოსტფიქსური გამოსახულების დამუშავებისა.*

ანალოგიურად, *პოსტფიქსური ჩანაწერის დამუშავება მარჯვნიდან მარცხნივ, ტოლძალოვანია მისი პრეფიქსული შესატყვისის შეუღლებულის დამუშავებისა.*

ეხლა შევნიშნოთ, რომ თანაფარდობების (5), (16) და (6), (15)-ის საფუძველზე შეგვიძლია ჩამოვაყალიბოთ და დამტკიცებულად ჩავთვალოთ შემდეგი დებულება.

თეორემა 5. თუ g არის ნებისმიერი გამოსახულების უფრჩხილებო ფორმა (ან პოსტფიქსური, ან პრეფიქსული), მაშინ ადგილი აქვს შემდეგ თანაფარდობას:

$$C(R(g)) = R(C(g)) \qquad (20)$$

სხვა სიტყვებით რომ ვთქვათ, *გარდაქმნები C და R გადაადგილებადია (კომუტაციურია).*

თუ განვაზოგადებთ აღნიშვნა P–ს პოლიმორფიზმის კონცეპციის სტილში და ჩავთვლით, რომ იგი შეიძლება მივუყენოთ როგორც პოსტფიქსურ, ასევე პრეფიქსულ ჩანაწერს, თანაც ვიგულისხმებთ, რომ პოსტფიქსურისადმი მიყენების შემთხვევაში იგი უზრუნველყოფს მისი პრეფიქსული შესატყვისის მიღებას, ხოლო პრეფიქსულისადმი მიყენების შემთხვევაში დააბრუნებს პოსტფიქსურ შესატყვისს, მაშინ ზემომოყვანილის საფუძველზე შეგვიძლია ჩამოვაყალიბოთ შემდეგი დებულებები.

თეორემა 6. თუ g არის ნებისმიერი გამოსახულების უფრჩხილებო ფორმა (ან პოსტფიქსური, ან პრეფიქსული), მაშინ მისი შეუღლებულის მისაღებად შეიძლება ან ჯერ მისი შებრუნება და შემდეგ მიღებულისთვის მეორე უფრჩხილებო შესატყვისის მიღება, ან ჯერ საწყისი ჩანაწერის მეორე უფრჩხილებო ფორმის მიღება და შემდეგ კი შებრუნება. ანუ ადგილი აქვს თანაფარდობებს:

$$C(g) = R(P(g)) \qquad (21)$$

$$C(g) = P(R(g)) \qquad (22)$$

დამტკიცება. მართლაც, როდესაც g არის პოსტფიქსური გამოსახულება, მაშინ (21) მიიღება (4)-დან, ხოლო (22) კი (10)-დან. როდესაც g არის პრეფიქსული გამოსახულება, მაშინ (21) მიიღება (12)-დან, ხოლო (22) კი (14)-დან.

თეორემა 7. ნებისმიერი უფრჩხილებო g ჩანაწერისთვის ადგილი აქვს თანაფარდობას:

$$R(P(g)) = P(R(g)) \qquad (23)$$

ანუ, გარდაქმნა *R აღმოჩნდა გადაადგილებადი განზოგადებულ P გარდაქმნასთანაც.*

დამტკიცება. დასამტკიცებელი დებულების ჭეშმარიტება პირდაპირ გამომდინარეობს (21) და (22)-დან.

ისევ დავუბრუნდეთ გამოსახულებას (a+b)*c-d/(e-f*g) და მისი წარმოდგენის ფორმას გრაფის მეშვეობით (გვერდი 20). ადრე ჩვენ უკვე ვახსენეთ, რომ ამ გამოსახულების როგორც პრეფიქსული, ასევე პოსტფიქსური ფორმები შეიძლება იყოს მიღებული შესაბამისი გრაფის გარკვეული წესით შემოვლის გზით. ადვილად დავრწმუნდებით, რომ ჩვენს მიერ განხილულ უფრჩხილებო გამოსახულებათა შეუღლებულებს შეესაბამება იგივე გრაფის შემოვლა სავსებით გარკვეული ალგორითმების მიხედვით.

კერძოდ, პოსტფიქსურის შეუღლებული gf*e-d/cba+*- მიიღება გრაფის შემოვლით მარჯვნიდან მარცხნივ იმ პირობით, რომ ოპერაციების ნიშნებს გადავიტანთ გამოსავალ გამოსახულებაში მხოლოდ იმის შემდეგ, რაც მთლიანად დასრულდება მისი ფოთლების ამოღება. პრეფისქულის შეუღლებული კი -/-*gfed*c+ba მიიღება გრაფის შემოვლით ზევიდან ქვემოთ და მარჯვნიდან მარცხნივ იმ პირობით, რომ ოპერანდების ამოღება მოხდეს იმის შემდეგ, რაც ამოღებულია შესაბამისი ოპერაციის ნიშანი. ცხადია, ყოველივე აქ მოყვანილი გვაფიქრებს, რომ ზოგიერთი სახის გრაფის ალგებრულად ჩაწერა სასარგებლო შეიძლება იყოს უფრჩხილებო ჩანაწერების ამა თუ იმ ფორმაზე დაყრდნობით. ასევე მოსალოდნელია, რომ სხვადასხვა ამოცანისთვის, საზოგადოდ, უფრჩხილებო ჩანაწერების სხვადასხვა ფორმა იქნება უფრო მოსახერხებელი.

თავი 2. ფუნქციონალურ გამოსახულებათა დამუშავება დაპროგრამების ენიდან

2.1. შიდა ენა სიმბოლური დიფერენცირების წესების აღწერისთვის

როგორც ზემით უკვე ითქვა, ფუნქციონალური გარდაქმნების წარმართვა უფრო მოსახერხებელია გამოსახულების პრეფიქსული სახიდან. ამ მხრივ არ არის გამონაკლისი არც გაწარმოება.

მოცემული ნაშრომის ფარგლებში განხორციელებულია მხოლოდ ერთი ცვლადის ფუნქციების გაწარმოება. თუმცა, ჩამოყალიბებული მიდგომა შეიძლება იყოს გადატანილი მრავალი ცვლადის ფუნქციის სიმბოლური გაწარმოებისთვის. წარმოდგენილ რეალიზაციაში ინფიქსურ ჩანაწერს მოეთხოვება ფუნქციის პარამეტრიც აუცილებლად ფრჩხილებში იყოს ჩასმული, როგორც ეს მიღებულია დაპროგრამების ენებში. გარდა ამისა, ელემენტარული ფუნქციების სიას დამატებულია ერთ პარამეტრიანი ფუნქცია დასახელებით s, (შემოკლება სიტყვისა “signed” - ნიშანი) რომელიც შემდეგნაირად განისაზღვრება:

$$s(x) = -x .$$

გაწარმოების წესების შესწავლა გვაძლევს შესაძლებლობას გავაკეთოთ შემდეგი დასკვნები ამ წესებით განსაზღვრული მოქმედებების ზოგადი დახასიათებისთვის:

1. გაწარმოების ყოველ ნაბიჯზე აუცილებელია შეგვეძლოს გამოვყოთ ის მოქმედება (ფუნქცია) რომლის მიმართაც ხდება გაწარმოება. ცხადია, ლაპარაკია ყველაზე გარე მოქმედებაზე, ანუ მოქმედებაზე, რომელიც შეიძლება შესრულდეს აღნიშნული გამოსახულების გამოთვლის ბოლოს. ამით ხდება იმ წესის დადგენა, რომელიც აქტუალურია მოცემული ნაბიჯისთვის;

2. აუცილებელია შეგვეძლოს გამოვყოთ აქტუალური მოქმედების (ფუნქციის) ყველა პარამეტრი;

3. როდესაც ცნობილია აქტუალური მოქმედება (ფუნქცია) და მისი ყველა პარამეტრი, საკუთრივ გაწარმოების წესის ჩამოყალიბება შესაძლებელია ამ პარამეტრებისა და მათი წარმოებულების მიმართ გარკვეული გამოსახულების ტერმინებში. ცხადია, იგივე წესები

შეიძლება იყოს გამოყენებული პარამეტრების გაწარმოებისთვის – გაწარმოების წესები, თავისი ბუნებით რეკურსიულია.

ყოველივე ზემოთქმული გვკარნახობს, რომ პროგრამული რეალიზაციის გაადვილების მიზნით მიზანშეწონილია შემუშავდეს შიდა ენა, რომელზედაც ჩამოყალიბდება გაწარმოების წესები. თითოეული წესი უნდა შეესაბამებოდეს ერთ მოქმედებას (ფუნქციას) და ქონდეს სათანადო პრეფიქსული ფუნქციონალური გამოსახულების ფორმის თარგის სახე, რომელშიც გამოიყენება მოქმედებათა (ფუნქციათა) აღნიშვნები, აგრეთვე ზოგიერთი კონსტანტები და გასაწარმოებელი გამოსახულების პარამეტრები და მათი წარმოებულები. აქ ძალიან არსებითია, რომ ეს თარგები იყოს შეძლებისდაგვარად თვალსაჩინო.

გასაწარმეობელი გამოსახულება		შესაბამისი გაწარმოების წესი შიდა ენაზე
პრეფიქსული სახით	შიდა ენაზე	
+uv	+ab	+AB
-uv	-ab	-AB
*uv	*ab	+*Ab*aB
/uv	/ab	/-*Ab*aB*bb
^uv	^ab	+**Ab^a-b1**A^abqa
sin u	fa	*gaA
cos u	ga	s*faA
tg u	ha	/Anga
ctg u	ia	s-Anfa
arcsin u	ja	/Ao-1na
arcos u	ka	s/Ao-1na
arctg u	ℓa	/A+1na
arcctd u	ma	s/A+1na
sqr u	na	*2*aA
sqrt u	oa	/A*2oa
exp u	pa	*paA
ℓn u	qa	/Aa

ɬg u	ta	/A*aqD
s u	sa	sA

ცხრილი 7. გაწარმოების წესები შიდა ენაზე.

იმისათვის, რომ გასაგები გახდეს ამ ენაზე ჩამოყალიბებული ზემომოყვანილი გაწარმოების წესები, საკმარისია ითქვას შემდეგი:

1. ამ ფორმულებში არითმეტიკული მოქმედებათა ნიშნები უცვლელადაა გადმოტანილი, ხოლო ახარისხება აღინიშნება სიმბოლოთი '^';
2. ფუნქციები გადანომრილია მცირე ზომის ლათინური ასოებით, დაწყებული 'f'-ით;
3. აქტუალური მოქმედების პირველი პარამეტრი აღინიშნება სიმბოლოთი 'a', ხოლო მეორე პარამეტრი კი სიმბოლოთი 'b'. მათი წარმოებულები კი აღინიშნება შესაბამისად 'A' და 'B'-თი.

დაგვრჩა ავღნიშნოთ, რომ ბოლოსწინა ფორმულაში სიმბოლოთი 'D' აღნიშნულია რიცხვი 10. გარდა ამისა, s-ით აღნიშნულია ფუნქცია, რომელიც დააბრუნებს მნიშნელობას, მიღებულს პარამეტრიდან მისი ნიშნის შეცვლით.

2.2. სიმბოლური დიფერენცირების რეალიზაცია

სიმბოლური დიფერენცირების რეალიზებისთვის დაპროგრამების ე.წ. ალგორითმული ენების საშუალებით პირველ რიგში გადასაწყვეტია შემდეგი საკითხები:

1. როგორ უნდა წარმოვადგინოთ ფუნქციონალური გამოსახულებები;
2. როგორ უნდა წარმოვადგინოთ გაწარმოების წესები.

საკუთრივ გაწარმოების წესების რეალიზება მთლიანად განპირობებულია ჩამოთლილი 2 პუნქტით.

გარკვეულობისთვის აქ ვისაუბრებთ იმ რეალიზების შესახებ, რომელიც შესრულდა C++ –ზე, თუმცა, ადვილად დავინახავთ, რომ წარმოდგენილი მიდგომა სავსებით მარტივად გადატანადია ნებისმიერ სხვა ალგორითმულ ენაზე.

თავდაპირველად შევნიშნოთ, რომ ზემოთქმულზე დაყრდნობით ფუნქციონალური გამოსახულება მიზანშეწონილია იყოს წარმოდგენილი პრეფიქსულ ნოტაციაში. და ვინადიან პრეფიქსული ჩანაწერი ფაქტობრივად წარმოადგენს მიმდევრობას, რომლის თითოეული წევრი არის ან მოქმედება (ფუნქცია) ან ოპერანდი, ბუნებრივად გამოიყურება ეს მიმდევრობა მოვათავსოთ ვექტორში, რომლის თითოეული ელემენტი ან მოქმედებაა ან ოპერანდი. აქედან გამოდინარე, ვექტორის ელემენტის ტიპად შეირჩა შემდეგი სტრუქტურა:

```
struct Telem
{char operand; // 0 ნიშნავს მოქმედებას, 1 ცვლადს, ხოლო 2 რიცხვს
 union { char nomer; // მოქმედებისთვის აღნიშნავს მის ნომერს
       double mnish; // რიცხვისთვის – მისი მნიშვნელობაა
      }
};
```

რაც შეეხება გაწარმოების წესებს, ისინი პირდაპირ ცხრილი 7-ის მიხედვით შედგენილი სტრიქონებით წარმოდგება, რომლებიც განსაზღვრავს ასეთი ვექტორის ფორმირებას. ამ ფორმულებისგან, თავის მხრივ, დგება ვექტორი, რომლის ყოველი ელემენტი შესაბამისობაშია ზუსტად ერთ მოქმედებასთან (ფუნქციასთან). ამის შემდეგ პროგრამული უზრუნველყოფის შემუშავებისთვის დაგვრჩება შემდეგი მოქმედებების რეალიზება:

1. გამოსახულების საკვანძო (რომელიც ბოლოს სრულდება) მოქმედების დადგენა. პრეფიქსულ ნოტაციაში ეს უბრალოდ ნიშნავს გამოსახულების საწყისი ელემენტის გარკვევას;
2. თუ საწყისი ელემენტი არ არის მოქმედება (ფუნქცია), მაშინ პასუხის დაბრუნება (1, თუ ცვლადია და 0, თუ კონსტანტაა). ხოლო წინააღმდეგ შემთხვევაში კი შესაბამისი ფორმულის შემცველი სტრიქონის გააქტიურება;
3. მიგნებული ფორმულის მიხედვით პასუხის აგება, რისთვისაც აუცილებელი იქნება საჭირო რაოდენობის პარამეტრების ამოღება და მათი ჩასმა პასუხში ფორმულით განსაზღვრულ ადგილზე, შესაძლოა, გაწარმოების ფუნქციით ზემოქმედების შემდეგ (საკუთარი თავის რეკურსიული გამოძახებით);

4. საბოლოო პასუხის მიღების წინ სასურველია გამოსახულების გამარტივების პროცედურის (მეთოდის) გამოძახება.

ყველაფერი მიუთითებს, რომ გამარტივების მეთოდიც არსებითად იოლდება, თუ გამოსახულების გამარტივების (შეკვეცის) წესებსაც ზემომოყვანილის მსგავს შიდა ენაზე აღვწერთ.

ამრიგად, როგორც ვხედავთ, ბუნებრივთან ახლო შიდა ენის გამოყენებამ, ერთის მხრივ, ერთიან სქემაში მოაქცია დიფერენცირების გარდაქმნის რეალიზება და ამით არსებითად გაზარდა მისი საიმედოობა და გასაგებობა. მეორეს მხრივ, შიდა ენის გამოყენება უკიდურესად ამარტივებს საჭირო შემთხვევაში სარეალიზაციო გაწარმოების წესის ჩამატებას პროგრამაში (პროგრამის გაფართოებას). ამისათვის საკმარისი ხდება ფორმულების ვექტორში ახალი ფორმულის დამატება და შესაბამისი მოქმედების (ფუნქციის) გამოცნობის გათვალისწინება მოცემული ინფიქსური გამოსახულებიდან შესატყვისი პრეფიქსული გამოსახულების მიმღებ მეთოდში.

რეალიზების გაადვილებისთვის შემოღებულ იქნა C++ ის სტანდარტული vector – ის მემკვიდრე Vector შემდეგნაირად:

template <class T>

class Vector : public vector <T>

მემკვიდრეს დაემატა კონკატენაციის ოპერაცია, რომელიც აღინიშნა “|”-თი და მეთოდი SubVec, რომლის დანიშნულებაა ქვევექტორის დაბრუნება სრული ანალოგიით იმასთან, რასაც გვაძლევს ქვესტრიქონების მიღებისთვის სტანდარტული მეთოდი Substr.

ზემოთქმულის შემდეგ გასაგებია, თუ რატომ იქნა შემოღებული ტიპი Tgam, როგორც კონტეინერული ტიპი ფუნქციონალური გამოსახულების შიდა (უფრჩხილებო) წარმოდგენის განთავსებისთვის.

აღნიშნულ ტიპთან მუშაობისთვის რეალიზებულია ფუნქციები, რომელთა აღწერას დავიწყებ მათი სათაურებით.

Tgam Togam(string x, int &res) –

ეს ფუნქცია უზრუნველყოფს სტრიქონის სახით მოცემული არა უმეტეს ერთი ცვლადის შემცველი გამოსახულების გარდაქმნას შიდა წარნიდგენაში. მეორე პარამეტრის მნიშვნელობა თუ >=0, სტრიქონი არაკორექტულია და ეს პარამეტრიც გვაძლევს პოზიციის ნომერს,

რომელშიც აღმოჩენილ იქნა შეცდომა. ასეთ შემთხვევაში, ფუნქცია დააბრუნებს ნულოვანი სიგრძის გამოსახულებას (ვექტორს). თუ ყველაფერი წესრიგშია, მეორე პარამეტრის მნიშვნელობა იქნება (-1) – ის ტოლი. ამ რეალიზაციაში იგულისხმება, რომ ცვლადი აღინიშნება ასოთი 'x'. გამოსახულებაში შესაძლებელია იყოს გამოყენებული 4-ვე არითმეტიული მოქმედება {+,-,*,/}, ახარისხება, რომელიც აღინიშნება სიმბოლოთი '^' და ცხრილი 7-ში მოყვანილი ყველა ელემენტარული ფუნქცია.

string Tostring(Tgam x) –

გამოსახულების გადაყვანა შიდა წარმოდგენიდან მისი ინფიქსური სახის შემცველ სტრიქონში. აქ არ ხდება მოცემული გამოსახულების კორექტულობის შემოწმება. იგულისხმება, რომ იგი კორექტულია. ამას უზრუნველყოფს მისი მიღება ბუნებრივი სახის შემცველი სტრიქონიდან ზემოთგანხილული ფუნქციით Togam. აქ ინტერესს წარმოადგენს 2 გარემოება. პირველი მდგომარეობს იმაში, რომ ინფიქსური გამოსახულება მიიღება პრეფიქსული გამოსახულების მარჯვნიდან მარცხნივ დამუშავებით (ფაქტობრივად, პრეფიქსულის შესატყვისი პოსტფიქსური გამოსახულების შეუღლებულიდან). მეორე კი მდგომარეობს დამხმარე ფუნქციების Br1 და Br2 გამოყენებაში აუცილებელი ფრჩხილების დასმისთვის. აქედან პირველი განსაზღვრავს პასუხს კითხვაზე – არსებობს თუ არა საჭიროება განთავსდეს ფრჩხილებში პირველი ოპერანდი, მეორე კი განსაზღვრავს პასუხს ანალოგიურ კითხვაზე მეორე ოპერანდის მიმართ. იმისათვის, რომ შეგვეძლოს დასმულ კითხვებზე პასუხის გაცემა, ამ ფუნქციების გამოძახების დროს უნდა იყოს ცნობილი არა მარტო მიმდინარე მოქმედება, არამედ შესაბამისი ოპერანდის საკვანძო (ბოლოს შესასრულებელი) მოქმედებაც. შესაბამისად, ორივე ეს ფუნქცია ოროპერანდიანია, ხოლო ძირითადი ალგორითმით გათვალისწინებული ოპერანდების განთავსებისთვის განკუთვნილი სტეკის პარალელურად პროგრამაში გათვალისწინებულია ამავე ოპერანდების საკვანძო მოქმდებების სტეკის წარმართვა.

double Gamotvla(Tgam G, double x=1) –

პირველი პარამეტრით მოცემული გამოსახულების გამოთვლა ცვლადის მნიშვნელობისთვის, რომელიც მეორე პარამეტრით არის განსაზღვრული. თუ გამოსახულება არ შეიცავს ცვლადს, ან მისი მნიშვნელობა უდრის 1-ს, ფუნქცია გამოიძახება ერთი პარამეტრით. მოყვანილ რეალიზაციაში გამოსახულების მნიშვნელობა მიიღება როგორც double ტიპის მნიშვნელობა, როგორც რიცხვითი ტიპებიდან ყველაზე ზოგადი. ოპერაციებიც სრულდება ისე, როგორც ეს მიღებულია double ტიპისთვის.

Tgam Dif(Tgam x) –

ფუნქცია უზრუნველყოფს კორექტული პრეფიქსული გამოსახულებიდან (ერთადერთი პარამეტრი) მისი წარმოებულის შესატყვისი პრეფიქსული გამოსახულების დაბრუნებას უშუალოდ ზემომოყვანილ ფორმულებზე დაყრდნობით.

Tgam Gamartiveba(Tgam G) –

ეს ფუნქცია უზრუნველყოფს პრეფიქსული გამოსახულების გამატივებას. აღნიშნული მოქმედებები აქტუალურია იმის გათვალისწინებით, რომ გაწარმოება ხდება ყველა შემთხვევაში უშუალოდ ფორმულების მიხედვით, რომელთაც ზოგადი ხასიათი აქვს.

მაგალითად, გამოსახულებიდან, რომლის ინფიქსური სახეა

(2.14*x -15)*cos(x)

ფუნქციით Dif მიიღება გამოსახულება, რომლის ინფიქსური შესატყვისია:

(0*x+2.14*1-0)*cos(x)+ (2.14*x -15)*s(sin(x)*1)

ამ გამოსახულების გამარტივებით მივიღებთ გამოსახულებას, რომლის ინფიქსური შესატყვისია:

2.14*cos(x)+ (2.14*x -15)*s(sin(x)) .

საინტერესოა, რომ გამოსახულებაში ზედმეტი ფრჩხილების მოცილება უზრუნველყოფილია მხოლოდ ფუნქციებით Togam და Tostring.

მოცემულ რეალიზაციაში გათვალისწინებულია შემდეგი სახის გამარტივებები:

გასამარტივებელი გამოსახულება	გამარტივების შედეგი
*f*C	*f*(C) – ანუ კონსტანტა გამოთვლილი მნიშვნელობით

$@C_1C_2$	$C_1@C_2$ – ანუ კონსტანტა გამოთვლილი მნიშვნელობით
ss	ანუ გამოსახულება სიგრძით 0
+a0	a
+0a	a
-a0	a
-0a	sa
*a0	0
/0a	0
*0b	0
*a1	a
*1b	b
/a1	a
^0b	0
^1b	1
^a0	1
^a1	a

ცხრილი 8. პროგრამაში რეალიზებული გამარტივებები

მოყვანილ ცხრილში:

f – პროგრამაში გათვალისწინებული ნებისმიერი ელემენტარული ფუნქციაა,

@ - პროგრამაში გათვალისწინებული ნებისმიერი 2 ადგილიანი მოქმედებაა,

a – გამოსახულების პირველი ოპერანდი,

b – გამოსახულების მეორე ოპერანდი,

C,C_1,C_2 – ნებისმიერი რიცხვითი კონსტანტები.

ჩამოთვლილი ფუნქციების მუშაობას აადვილებს შემდეგი დამხმარე ფუნქციები, რომლებითაც შეიძლება პრობლემურმა პროგრამისტმა პირდაპირაც ისარგებლოს.

Telem SetTelem(char x=2, char y=0, double z=0) –

ფუნქცია უზრუნველყოფს Telem ტიპის მნიშვნელობის ჩამოყალიბებას მისი პარამეტრების საფუძველზე (ოპერატორობის ნიშანი, მოქმედების ნომერი, მნიშვნელობა).

Tgam SetTgam(Telem x) –

ფუნქცია დააბრუნებს ერთელემენტიან პრეფიქსულ გამოსახულებას, რომლის ერთადერთი ელემენტი განსაზღვრულია პარამეტრით.

void Printelem(Telem x) –

პარამეტრის მნიშვნელობის ამობეჭდვა.

void PrintGam(Tgam x) –

პარამეტრის მნიშვნელობის ამობეჭდვა. ამჯერად პარამეტრი წარმოადგენს პრეფიქსულ გამოსახულებას, ამიტომ მისი მნიშვნელობის ამობეჭდვა მიიღწევა ცალკეული ელემენტების ამობეჭდვით მათ შორის თითო ჰარის მოთავსებით.

Tgam a(Tgam x) –

პარამეტრით განსაზღვრული პრეფიქსული გამოსახულებიდან პირველი ოპერანდის გამოყოფა.

Tgam b(Tgam x) –

პარამეტრით განსაზღვრული პრეფიქსული გამოსახულებიდან მეორე ოპერანდის გამოყოფა.

int Prior(int x) –

პარამეტრით განსაზღვრული ნომრის მქონე ოპერაციის პრიორიტეტის დაბრუნება.

int IsCorrectMnish(string x) –

სტრიქონის სახით მოცემული ჩანაწერის კორექტულობის განსაზღვრა მისი რიცხვად აღქმის ჭრილში C++ –ის სინტაქსის მიხედვით. ფუნქცია დააბრუნებს (-1)-ს, თუ პარამეტრი რიცხვის კორექტულ ჩანაწერს წარმოადგენს, ხოლო წინააღმდეგ შემთხვევაში კი დააბრუნებს იმ პოზიციის ნომერს, რომლიდანაც შეუძლებელი გახდა პარამეტრის აღქმა რიცხვად.

2.3. ფუნქციონალურ გამოსახულებათა დამუშავებისთვის განკუთვნილი კლასის ინტერფეისი

ყოველივე ზემომოყვანილი ქმნის წინაპირობას ფუნქციონალურ გამოსახულებებთან მუშაობისთვის განკუთვნილი სპეციალური კლასის შემუშავებისთვის.

აქ მოვიყვანოთ ასეთი კლასის მხოლოდ ინტერფეისის აღწერა.

შევთანხმდეთ, რომ ასეთ კლასს დავარქვათ Cgam და დავიწყოთ იმით, რომ ფუნქციონალური გამოსახულების შიდა წარმოდგენა ზემომოყვანილის იდენტური იქნება. ეხლავე ვთქვათ, რომ ასეთი კლასის ინფორმაციული ველების შემადგენლობის საკითხი მრავალნაირად შეიძლება გადაწყდეს. მე აქ მოვიყვან ისეთ გადაწყვეტას, რომელიც დაემყარება მინიმალურად აუცილებელი საინფორმაციო შემადგენლობის ქონას.

ამრიგად, ამ კლასს ექნება ერთი ინფორმაციული ველი, რომელიც იქნება საკუთრივი (პრივატული) და რომლის ტიპია Tgam.

დისკუსია შესაძლებელია საკითხზე – ღირს თუ არა მოყვანილ კლასში გათვალისწინებულ იქნას გამოსახულების პოსტფიქსური სახე. საქმე ისაა, რომ ზემოაღწერილი გზით ყოველგვარი სახეცვლილების გარეშე შესაძლებელია პოსტფიქსური გამოსახულების წარმოდგენაც. ასეთ შემთხვევაში აუცილებელი იქნებოდა მეთოდი, რომელიც მიწოდებული გამოსახულების მიხედვით დააბრუნებდა პასუხს კითხვაზე არის თუ არა იგი პრეფიქსული (თუ პრეფიქსული არ არის, მაშასადამე პოსტფიქსურია). ამ ფუნქციას გამოიყენებდა კლასში გადატვირთული ყველა ოპერაცია და მეთოდების აბსოლუტური უმრავლესობა, რათა განესაზღვრა გამოსახულება დაემუშავებინა როგორც პოსტფიქსური, თუ პრეფიქსული. ცხადია, ასეთ შემთხვევაში მომხმარებლისთვის სულ ერთი იქნებოდა გამოსახულების შიდა წარმოდგენა პოსტფიქსურია თუ პრეფიქსული. ცხადია, უკვე წარმოდგენილის საფუძველზე არავითარი პრობლემა არ იქნებოდა მოყვანილი გამოსახულებისთვის გაგვეთვალისწინებინა ყველა შესაძლო გარდაქმნები, რომლებიც დაფარავდა არა მარტო ინფიქსურიდან სტანდარტული პოლონური ნოტაციის ორივე სახეზე და პირიქით გარდაქმნებს, არამედ მოიცავდა გარდაქმნებს ჩვენს მიერ შემოღებული პოლონური ჩანაწერების შეუღლებულების მონაწილეობით... არ გამოვრიცხავ, რომ დროთა განმავლობაში გამოჩნდეს ასეთი გადაწყვეტის მიზანშეწონილობა,

მაგრამ მოცემულ ნაშრომში აქცენტს ვაკეთებ რეალიზაციის მაქსიმალურ გარკვეულობაზე და ორიენტაციაზე შესაძლებლობათა სრული პაკეტით შეიარაღდეს ალგორითმული ენიდან მომუშავე პრობლემური პროგრამისტი. ამისათვის კი სავსებით საკმარისია (როგორც ჩვენ უკვე დავრწმუნდით) ვიქონიოთ გამოსახულების მხოლოდ ერთი შიდა სახე, რომელიც ემყარება მის პრეფიქსულ წარმოდგენას. ამიტომ, ამ აღწერაში ვიგულისხმებ, რომ კლასს აქვს ერთადერთი ინფორმაციული ველი – დასამუშავებელი გამოსახულების შიდა წარმოდგენა, დამყარებული პრეფიქსულ ნოტაციაზე.

კლასში გადატვირთული იქნება ოთხივე არითმეტიკული ოპერაცია და ახარისხება (ეს უკანასკნელი, ისევ ნიშანი '^'-ის გამოყენებით). ამის საშუალებას გვაძლევს C++ –ის ოპერაციათა გადატვირთვის შესაძლებლობა. სიტყვამ მოიტანა **და აქ გამოჩნდა რითია გამართლებული ვექტორების კონკატენაციის ოპერაციად არა + , არამედ | ოპერაციის გადატვირთვა.**

ყველა ელემენტარული ფუნქცია, რომელიც შედის ზემოგანხილული პროგრამული კომპლექსის შემადგენლობაში, როგორც ეს დაშვებულია გამოსახულების ინფიქსური ფორმისთვის, შეიძლება ბუნებრივი სახით გადატვირთული იყოს ამ კლასის მეთოდებად. ცხადია, თითოეული მათგანი დააბრუნებს შედეგად გარდაქმნილი გამოსახულების შიდა წარმოდგენას, ანუ ამავე კლასის ობიექტს. გამოსახულების წარმოდგენის პრეფიქსული ნოტაცია უკიდურესად აიოლებს როგორც ოპერაციების, ასევე ელემენტარული ფუნქციების გადატვირთვას...

საქმეს სერიოზულად გააიოლებს ტოლობის ოპერაციის გადატვირთვა. მოსაფიქრებელია, რა შინაარსი მიეცეს შედარების დანარჩენ ოპერაციებს.

ცხადია, კლასში გასათვალისწინებელია პუნქტ 2.2.-ში მოყვანილი ყველა ფუნქციის შესატყვისი მეთოდი. აქ, როგორც ჩანს აუცილებელი იქნება ერთი გამონაკლისის დაშვება. ფუნქცია Togam-ის ანალოგი ჯობია გაფორმდეს როგორც მეგობრული ფუნქცია, რადგან ღირებულია მისი მეორე პარამეტრით დაბრუნებული ინფორმაცია, თუმცა აშკარად სასარგებლო იქნება კლასის ერთი კონსტრუქტორი სწორედ ამ ფუნქციის საფუძველზე აიგოს.

წმინდა პრაქტიკული მოსაზრებებით გამოსახულების გამარტივება პირდაპირ უნდა ჩაიდოს დიფერენცირების მეთოდში – საბოლოო შედეგის დაბრუნებამდე. ამავე დროს, სასარგებლოა გამარტივების მეთოდის გამოყენება ცალკეც შეგვეძლოს.

საინტერესოდ გამიყურება არითმეტიკული ოპერაციების გადატვირთვა იმ შემთხვევებისთვის, როდესაც ოპერაციაში მონაწილეობს არამარტო მოცემული კლასის ობიექტი, არამედ რიცხვითი ტიპის კონსტანტა.

2.4. ფუნქციონალურ გამოსახულებათა დამუშავებისთვის განკუთვნილი მოდული (რეალიზაცია პასკალზე)

ეს რეალიზაცია განკუთვნილია საზოგადოდ მრავალი ცვლადის ფუნქციების დამუშავებისთვის და შესრულებულია მოდულის სახით, რომელსაც დაერქვა gamotvla და რომელიც შედგება 2 პროგრამული კომპონენტისგან.

ამ კომპონენტებიდან პირველი, scanfunq, წარმოადგენს პროცედურას, რომელიც უზრუნველყოფს სტრიქონის სახით მიწოდებული ინფიქსური გამოსახულების გარდაქმნას პოსტფიქსურ ნოტაციაზე დამყარებულ შიდა სახეზე.

პროცედურის სათაურს შემდეგი სახე აქვს:

procedure scanfunq(x:string; var y : mimt; var mistake, npoz : byte; var masind : masindp);

აქ პირველი პარამეტრი წარმოადგენს საწყის გამოსახულებას, მოცემულს ინფიქსური სახით სტრიქონის მეშვეობით.

ყველა დანარჩენი პარამეტრი კი გამომუშავდება პროცედურის მუშაობის შედეგად. აქედან მეორე პარამატრი წარმოადგენს საწყისი გამოსახულების პოსტფიქსურ შესატყვისს, მოცემულს ბმული სიის სახით.

```
gamelem = Record case moqm:Boolean of
        True:(monom:integer;
             adgil:1..3;
            );
        False:(Case cvladi:Boolean of
              True:(cnom:integer);
```

```
                False:(mnish:Real);
              );
        end;
mimt = ^gamosah;
gamosah = Record
        inf:gamelem;
        mim:mimt
       end;
```

როგორც ვხედავთ, mimt წარნოადგენს მითითებას 2 ველიან კომბინირებულ ტიპზე (gamosah). ამ უკანასკნელის პირველი ველი (inf) იძლევა გამოსახულების ერთ ელემენტს. ეს ელემენტი შეიძლება იყოს მოქმედების აღნიშვნელი და მაშინ საჭიროა ვიცოდეთ მისი ნომერი და შესრულებისთვის აუცილებელი ოპერანდების რაოდენობა (შესაბამისად, ველები monom და adgil), ხოლო თუ ელემენტი ოპერანდს აღნიშნავს, მაშინ თუ იგი ცვლადია, უნდა ვიცოდეთ მისი ნომერი (cnom), ხოლო თუ იგი რიცხვია, უნდა ვიცოდეთ მისი მნიშვნელობა (mnish). მეორე ველი (mim) კი წარმოადგენს მიმთითებელს იგივე ტიპის ობიექტზე (მიეკუთვნება ტიპს mimt). ამ გზით ხდება გამოსახულების პოსტფიქსური შესატყვისის შემცველი ბმული სიის აგება.

ცხადია, ამ პარამეტრით მიღებული მნიშვნელობა საჭიროებს შემდგომ დამუშავებას, თუ საწყის გამოსახულებაში შეცდომა არ იყო. ამის შესახებ კი ინფორმაციას იძლევა მესამე და მეოთხე პარამეტრი. თუ მესამე პარამეტრი (mistake) არ უდრის 0-ს, ეს ნიშნავს, რომ დაფიქსირდა შეცდომა, რომლის ადგილმდებარეობას (პოზიციის ნომერს) იძლევა მეოთხე პარამეტრი (npoz), ხოლო თუ მესამე პარამეტრმა დააბრუნა 0, მაშინ გამოსახულება შეცდომებს არ შეიცავს, მისი შიდა სახე მეორე პარამეტრშია, ხოლო ბოლო პარამეტრი (masind) კი იძლევა ინფორმაციას ამ გამოსახულებაში გამოყენებული ცვლადების შესახებ. იგი წარმოადგენს მასივს:

masindp = array[1..np] of boolean;

ელემენტი ინდექსით 1 პასუხიბს კითხვას – იყო თუ არა გამოსახულებაში ცვლადი დასახელებით x, ელემეტი ინდექსით 2 პასუხობს ანალოგიურ კითხვას y ცვლადის შესახებ და ა.შ.

ეს პროცედურა თავის მუშაობაში იყენებს რამდენიმე შიდა ქვეპროგრამას.

function shecdoma:boolean;

ეს ფუნქცია უზრუნველყოფს სათანადო სასრული გამომცნობით განსაზღვრული შიდა გადასვლების ლოგიკის რეალიზებას და აბრუნებს პასუხს კითხვაზე გასაანალიზებელ სტრიქონში შეცდომა იყო თუ არა.

procedure relem(var mm: tipelem; var lakm:Boolean);

ეს პროცედურა უზრუნველყოფს მორიგი სინტაქსური ელემენტის მიღებას და გამოცნობას (რომელია მორიგი ელემენტი?). გამოცნობის შედეგი ბრუნდება პირველი პარამეტრის მეშვეობით, რომელიც მიეკუთვნება ტიპს tipelem:

```
tipelem = Record case romelem:selem of
        -1:();
         0:(mmnish:real);
         1:(cvnom:integer);
         2:(monom:integer);
         3:();
         4:()
        end;
```

აქ სელექტორის მნიშნელობები შეესაბამება (-1) – შეცდომაა, 0 – რიცხვია (და გვჭირდება მისი მნიშვნელობა), 1 – ცვლადია (და გვჭირდება მისი ნომერი), 2 – მოქმედებაა (და გვჭირდება მისი ნომერი), 3 – გახსნილი ფრჩხილია, 4 – დახურული ფრჩხილია.

პროცედურის მეორე პარამეტრი პასუხობს კითხვას – სწორია თუ არა, რომ მორიგი ელემენტი არ არის. ანუ, თუ მისი მნიშვნელობა არის true, ეს ნიშნავს, რომ სტრიქონი უკვე ბოლომდე დამუშავდა.

ეს პროცედურა იყენებს 2 შიდა უპარამეტრო პროცედურას.

procedure neprob;

აქ უზრუნველყოფილია გასაანალიზებელი სტრიქონის მიმდინარე ადგილიდან უახლოეს ისეთ სიმბოლომდე მისვლა, რომელიც არ წარმოადგენს ჰარს. ამით ხდება იმ ლოგიკის განხორციელება, რომლის

მიხედვითაც ზედიზედ განთავსებული რამდენიმე ჰარიდან მხოლოდ ერთი ითვლება ნიშნად სიმბოლოდ – ყველა დანარჩენი იგნორირდება.

procedure prob;

ეს პროცედურა კი უზრუნველყოფს გასაანალიზებელი სტრიქონის იმ უბნის ამოღებას, რომელიც იწყება მიმდინარე ადგილას და თავდება უახლოესი გამყოფით. აქ განხორციელებულია ლოგიკა, რომლის მიხედვითაც მორიგი ელემენტის გამოცნობის დროს გამყოფების ნუსხა შეიძლება იცვლებოდეს (ვთქვათ, თუ დავიწყეთ რიცხვის წაკითხვა, მაშინ ციფრი უკვე არ არის გამყოფი).

procedure wmstack(x:integer);

ეს პროცედურა უზრუნველყოფს მორიგი ელემენტის ჩაწერას მოქმედებების სტეკში. რეალურად სტეკში ხდება მოქმედებების ნომრების ჩაწერა.

procedure rmstack(var x:integer);

ეს პროცედურა კი უზრუნველყოფს მოქმედებათა სტეკიდან მორიგი ელემენტის ამოღებას. აქაც ხდება მოქმედების ნომრის ამოღება.

ბოლოს ჩამოვთვალოთ ელემენტარული ფუნქციები, რომელთა გამოყენება ნებადართულია ამ მოდულისთვის მიწოდებულ ინფიქსურ გამოსახულებაში:

sin, cos, tg, arcsin,

arccos , arctg, abs,

sqr, sqrt, exp, ln, lg .

ვფიქრობ, ამ სიას რამე დამაზუსტებელი კომენტარი არ ჭირდება. რაც შეეხება ერთადერთ სამოპერანდიან მოქმედებას, იგი მოიცემა სიმბოლოებით ‘?’ და ‘:’. მის სინტაქსს შემდეგი სახე აქვს:

$$g_1?g_2:g_3$$

აქ თითოეული g_i წარმოადგენს რიცხვითი მნიშვნელობის მქონე გამოსახულებას. მთლიანობაში გამოსახულების მნიშვნელობაა g_2–ის ტოლი, თუ $g_1>=0$, წინააღმდეგ შემთხვევაში, გამოსახულების მნიშვნელობას იძლევა g_3.

მეორე პროგრამული კომპონენტი წარმოადგენს ფუნქციას, რომელიც უზრუნველყოფს პირველი პარამეტრით მოცემული გამოსახულების

შიდა სახიდან მნიშვნელობის მიღებას მეორე პარამეტრით განსაზღვრული ცვლადების მნიშვნელობებისთვის.

ამ ფუნქციის სათაურს შემდეგი სახე აქვს:

function gam(x:mimt; y:maspar):real;

ფუნქცია ახორციელებს პოსტფიქსური ჩანაწერის მნიშვნელობის გამოთვლის კლასიკურ ალგორითმს. მუშაობის პროცესში იგი იყენებს რამდენიმე შიდა ქვეპროგრამას.

procedure wstack(x:real);

ეს პროცედურა უზრუნველყოფს ზემონახსენები ალგორითმით გათვალისწინებულ მნიშვნელობათა სტეკში პარამეტრად მიწოდებული მნიშვნელობის მოთავსებას;

procedure rstack(var x:real);

აქ კი, პირიქით, მნიშვნელობათა სტეკიდან ხდება 1 ელემენტის ამოღება. ამოღებული მნიშვნელობა თავსდება მეორე პარამეტრში, რომელიც წარმოადგენს პარამატრ-ცვლადს.

function arcsin(x:real):real;

ფუნქცია განკუთვნილია arcsin(x) მნიშვნელობის გამოთვლისთვის, სადაც პარამეტრის მნიშცნელობა იგულისხმება მოცემულად რადიანებში.

თავი 3. ტიურინგის ვირტუალური მანქანის მოდიფიკაცია დაპროგრამების სასტარტო სწავლებისთვის

3.1. დაპროგრამების სასტარტო სწავლებისთვის განკუთვნილი ტიურინგის ვიტუალური მანქანის მოდიფიკაციის აღწერა

საზოგადოების განვითარების თანამედროვე ეტაპზე პროგრამისტის პროფესია გახდა ერთ-ერთი ყველაზე მოთხოვნილი და მნიშვნელოვანი. უკვე თამამად შეიძლება ითქვას, რომ არ არსებობს ადამიანის საქმიანობის არცერთი სფერო, რომელშიც არ გამოიყენება კომპიუტერი. ამ პირობებში ტექნოლოგიური თვალსაზრისით ყველაზე მოწინავე სახელმწიფოებიც კი ცდილობენ საგანგებო ზომები გაატარონ იმისათვის რომ მოახერხონ მაღალკვალიფიციური კადრების საჭირო რაოდენობით მომზადება, რათა არ აღმოჩნდენ პროგრამისტულ პოზიციებზე სპეციალისტების გარედან მოწვევის აუცილებლობის წინაშე.

ცხადია, ეს საკითხი აქტუალურია საქართველოსთვისაც. საკითხის აქტუალობა მნიშვნელოვნად იზრდება იმის გათვალისწინებით, რომ საქართველოს მოსახლეობას ოდითგანვე ახასიათებს მაღალი ინტელექტუალური პოტენციალი და, საზოგადოდ, ლტოლვა ინტელექტუალური საქმიანობისკენ. დასტურად შეიძლება გამოდგეს უმდიდრესი ხალხური შემოქმედება (ხალხური პროზა და პოეზია, უპრეცედენტო ჟღერადობის, სირთულის და მრავალფეროვნების ხალხური სიმღერები, უნიკალური ცეკვები, ხალხური რეწვა, ...). ინტელექტუალური საქმიანობა ქართველი კაცისთვის ყოველთვის მაღალპრესტიჟულად ითვლებოდა. აქედან გამომდინარე, ლაპარაკია იმაზე რომ მაქსიმალურად გამოყენებულ იქნას მოსახლეობის მაღალი ინტელექტუალური პოტენციალი.

ზემოთქმულიდან გასაგებია, თუ რაოდენ მნიშვნელოვანია მაღალკვალიფიციური პროგრამისტული კადრების მომზადებისთვის გამიზნული სასწავლო პროცესის ორგანიზება. დღეს ამ საქმეს ძირითადად უმაღლესი სასწავლებლები ემსახურება. მათი მდგომარეობა მნიშვნელოვნად გართულებულია იმ გარემოებით, რომ ჩვენს სკოლებში (რამოდენიმე გამონაკლისის გარდა) დაპროგრამების საფუძვლები არ

შეისწავლება. ამის გამო უმაღლეს სასწავლებლებს უწევთ დაპროგრამების სასტარტო სწავლების ორგანიზებაც.

დღეისათვის არ არსებობს დე ფაქტო სტანდარტად მიჩნეული დაპროგრამების სასტარტო სწავლების მეთოდოლოგია. ასეთად ვერ გადაიქცა ვერც დონალდ კნუტის მიერ შემოთავაზებული ვირტუალურ MIX და MMIX კომპიუტერებზე დამყარებული მეთოდოლოგია და ვერც ნიკლაუს ვირტის მიერ შემოთავაზებული პასკალის სწავლებაზე დამყარებული მეთოდოლოგია. უფრო სწორედ, ეს უკანასკნელი უკვე არ არის დე ფაქტო სტანდარტი, რადგან შეწყდა დაპროგრამების ენა პასკალის ტექნოლოგიური მხარდაჭერა (სამართლიანობა მოითხოვს შევნიშნოთ რომ 90-იან წლებში და 2000-იან წლების დასაწყისში სწორედ ეს მეთოდოლოგია ითვლებოდა ყველაზე უფრო გამართლებულად).

ასეთ პირობებში საკმაოდ ინტენსიურად ხდება სხვადასხვა მიდგომების ჩამოყალიბება. მაგრამ ჩვენი მდგომარეობა ერთიორად გართულებულია იმით, რომ არცერთი სხვა სახელმწიფოს უმაღლესი სკოლა არ არის მსგავს მდგომარეობაში. ეს ნიშნავს, რომ გარედან თუნდაც აღიარებული მეთოდიკის გადმოღება ძირშივე მცდარია, რადგან ის გათვალისწინებულია სულ სხვა სასტარტო პირობებზე.

საკითხის სიმწვავე იზრდება იმით, რომ არც დრო არის მოუმზადებელი ექსპერიმენტების ჩასატარებლად. მოუმზადებელი ექსპერიმენტი შეიძლება აღმოჩნდეს გონიერი ახალგაზრდისთვის საკუთარ თავში რწმენის ჩაკვლის გამომწვევი, რაც ჩვენს პირობებში ყოვლად დაუშვებელია.

წარმოდგენილ ნაშრომში შემოთავაზებულია მეთოდიკა, რომელიც ეყრდნობა ტიურინგის ვირტუალური მანქანის მოდიფიკაციის რეალიზებას.

ტიურინგის ვირტუალური მანქანის ავტორი, როგორც ცნობილია, არის გამოჩენილი ინგლისელი მათემატიკოსი ალან ტიურინგი, რომელიც ითვლება თანამედროვე პროგრამირების პირველ თეორეტიკოსად, ინფორმატიკისა და ხელოვნური ინტელექტის თეორიის ერთერთ ფუძემდებლად. საყოველთაოდ ცნობილია მისი უდიდესი ღვაწლი მეორე

მსოფლიო ომში გერმანელების საიდუმლო მიმოწერის გაშიფრვის საქმეში.

ტიურინგის ვირტუალურ მანქანას აქვს ორი ნაწილი:

1. ორივე მხრიდან შემოუსაზღვრელი წრფივი ლენტა, რომელიც დაყოფილია უჯრედებად;
2. პროგრამის მიერ მართვადი ჩაწერა–წაკითხვის თავაკი.

ყოველ უჯრედში თითო სიმბოლო შეიძლება განთავსდეს. მანქანის ყოფაქცევა განისაზღვრება მიმდინარე სიმბოლოთი (რომელზეც დგას თავაკი) და ე.წ. მდგომარეობათა ცხრილის მიმდინარე მდგომარეობით. ტიურინგის მანქანის სრული აღწერა გადის მოცემული ნაშრომის ფარგლებს გარეთ. აქ მხოლოდ იმას შევნიშნავთ, რომ ტიურინგის მანქანა გამოიყენება მათემატიკის ისეთ რთულ და განყენებულ დარგში, როგორიცაა ალგორითმების თეორია...

სწავლებისთვის განკუთვნილი ტიურინგის ვირტუალური მანქანის მოდიფიკაცია შემუშავდა შემდეგ მოსაზრებებზე დაყრდნობით:

1. შენარჩუნებული ყოფილიყო ტიურინგის მანქანის ფუძემდებელი ნაწილები - წრფივი ლენტა, მიმდინარე პოზიციის განმსაზღვრელი თავაკი, და მდგომარეობათა ცხრილის ფაქტობრივი ექვივალენტი.
2. შენარჩუნებულ ყოფილიყო ალგორითმის გამომსახველობით საშუალებათა მინიმალურობა მათი უნივერსალობის პირობებში, თუკი ლენტი უსასრულოა.
3. მდგომარეობათა ცხრილის ექვივალენტის როლი ეტვირთა პროგრამას, დაწერილს ტიურინგის მანქანის შემუშავებული მოდიფიკაციის ბრძანებათა ტერმინებში მინიმალური დამატებითი საშუალებების ჩართვის პირობებში.
4. რეალიზაცია ყოფილიყო ადვილი გამოყენებისთვის სასწავლო პროცესის სხვადასხვა კომპონენტის ინტერესებიდან გამომდინარე.
5. რეალიზაცია ყოფილიყო მაქსიმალურად თვალსაჩინო.

ყოველივე ზემოთქმულის საფუძველზე შემუშავდა ტიურინგის მანქანის მოდიფიკაცია.

ტიურინგის ვირტუალური მანქანის რეალიზაციის იდეა წარმოიშვა თსუ-ს გამოყენებითი მათემატიკის სამეცნიერო კვლევით ინსტიტუტთან

არსებული ნორჩ მათემატიკოს-პროგრამისტთა სკოლის გამოცდილებაზე დაყრდნობით. ამ სკოლაში (ხელმძღვანელი ავთანდილ ცისკარიძე) წლების განმავლობაში მოსწავლეები დაპროგრამების საფუძვლებს ეუფლებოდნენ პოსტის ვირტუალური მანქანის რეალიზაციის მეშვეობით. სასწავლო პროცესში პოსტის ვირტუალური მანქანის გამოყენების იდეა ეკუთვნის მოსკოვის სახელმწიფო უნივერსიტეტის თანამსრომელს, ვლადიმერ უსპენსკის. ეს იდეა მყისვე აიტაცა ქართველმა მათემატიკოსმა კონსტანტინე ცისკარიძემ და საკმაოდ მალე მისი ხელმძღვანელობით ახალგაზრდა სპეციალისტმა მარინა ხარაზიშვილი-დერბინიანმა შექმნა პოსტის დიალოგური მანქანის რეალიზაცია ეგმ ბესმ-6-თვის. უნდა ითქვას, რომ წლების განმავლობაში ამ რეალიზაციაზე დამყარებული მეთოდიკა წარმატებულად გამოიყენებოდა თსუ-ს გამოყენებითი მათემატიკის სამეცნიერო კვლევითი ინსტიტუტის ნორჩ მათემატიკოს-პროგრამისტთა სკოლაში.

მოცემულ ნაშრომში წარმოდგენილი ტიურინგის ვირტუალური მანქანის მოდიფიკაცია შედგება 2048 უჯრედიანი წრფივი ლენტისგან, ნიშნულისგან (თავაკისგან) რომელიც მიუთითებს ამ ლენტის მიმდინარე პოზიციას, და იმართება სულ 6 ბრძანებით.

ეს ბრძანებებია:

1. **L** - თავაკის გადაადგილება ერთი პოზიციით მარცხნივ;
2. **R** - თავაკის გადაადგილება ერთი პოზიციით მარჯვნივ;
3. **S** - პროგრამის შესრულების დამთავრება (ტიურინგის ვირტუალური მანქანის გაჩერება);
4. **Wს** - ლენტის მიმდინარე პოზიციაში **ს** სიმბოლოს ჩაწერა;
5. **Gჭ** - უპირობო გადასვლა **ჭ** ჭდეზე, ანუ **ჭ** ნიშნულის მქონე ბრძანებაზე;
6. **Ისჭ** - გადასვლა **ჭ** ჭდეზე, იმ შემთხვევაში, თუ ლენტის მიმდინარე პოზიციაში მოთავსებულია სიმბოლო **ს**.

თითოეული ბრძანება იწერება ცალკე სტრიქონში. ასევე ცალკე სტრიქონში იწერება **ჭდე**, რომელიც **გარეგნულად წარმოადგენს არა**

უმეტეს 5 ნიშნა ათობით მთელ რიცხვს. ჯდე იწერება სტრიქონში, რომელიც მოთავსებულია **უშუალოდ მოსანიშნი ოპერატორის წინ.**

ტიურინგის ვირტუალური მანქანის ზემოაღწერილ ბრძანებათა ენას სავსებით შეიძლება ამ მანქანის ასემბლერი ვუწოდოთ. ზემოთქმულის საილუსტრაციოდ განვიხილოთ რამდენიმე კონკრეტული ამოცანა. ყველგან ქვემოთ ვიგულისხმებთ, რომ ლენტის საწყისი და დასკვნითი მდგომარეობის განსაზღვრისთვის სავსებით საკმარისია გამოცხადებული სიგრძე 2048 სიმბოლო. სხვა სიტყვებით, რომ ვთქვათ, ვიგულისხმოთ, რომ არც საწყისი, არც შუალედური და არც დასკვნითი მონაცემები არ ითხოვს 2048-ზე უფრო დიდი სიგრძის ლენტას.

შევნიშნოთ, რომ ტიურინგის ვირტუალურ მანქანაზე ამოსახსნელი ყოველი ამოცანისთვის უნდა განისაზღვროს ლენტის და თავაკის სასტარტო მდგომარეობა, აგრეთვე ამავე ლენტის და თავაკის დასკვნითი მდგომარეობა.

3.2. უმარტივესი ამოცანების გადაწყვეტა ტიურინგის ვირტუალურ სასწავლო მანქანაზე

ამოცანა 1. სიმბოლოთა მიმდევრობის მარჯვენა კიდეზე გადასვლა, თუ თავაკი ამ მიმდევრობაზეა.

ლენტის სასტარტო მდგომარეობა. ლენტზე მოთავსებულია სიმბოლოთა ნებისმიერი მიმდევრობა, რომელიც არ შეიცავს ჰარებს, ამ მიმდევრობისგან მარცხნივ და მარჯვნივ ყველა პოზიცია შევსებულია ჰარებით (ჩვენ გთავაზობთ სიტყვა ”ჰარი” გამოვიყენოთ იმ სიმბოლოს აღნიშვნისთვის, რასაც ინგლისურად ეწოდება ”space”, ხოლო რუსულად კი ”пробел”; ”ჰარი” სვანური წარმოშობის სიტყვაა და აღნიშნავს თავისუფალ სივრცეს 2 მოხნულ ყანას შორის).

თავაკის სასტარტო მდგომარეობა. თავაკი განთავსებულია მიმდევრობის მიერ დაკავებული სივრცის შიგნით (ანუ ამ მიმდევრობის ნებისმიერ სიმბოლოზე).

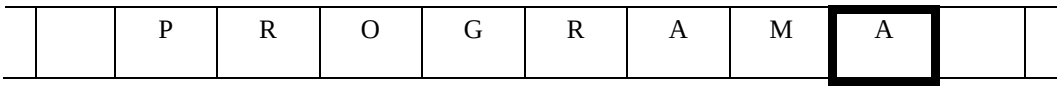
ლენტის დასკვნითი მდგომარეობა. იგივეა, რაც სასტარტო.

თავაკის დასკვნითი მდგომარეობა. თავაკი უნდა განთავსდეს სიმბოლოთა საწყისი მიმდევრობის მარჯვენა კიდეზე.

მაგალითი 1.

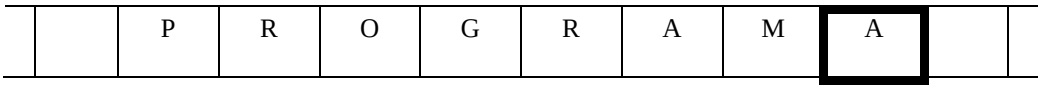
საწყისი მდგომარეობა:

დასკვნითი მდგომარეობა:

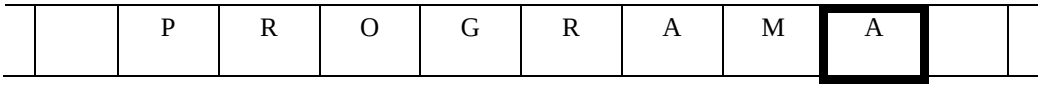


მაგალითი 2.

საწყისი მდგომარეობა:

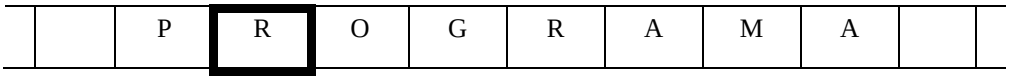


დასკვნითი მგომარეობა:

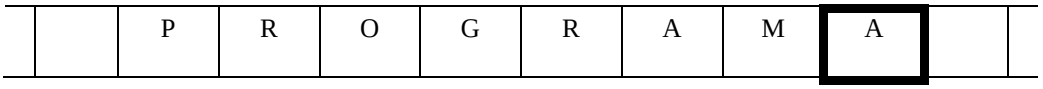


მაგალითი 3.

საწყისი მდგომარეობა:

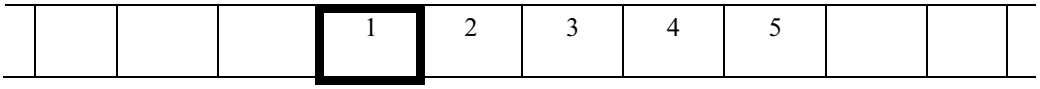


დასკვნითი მგომარეობა:



მაგალითი 4.

საწყისი მდგომარეობა:



დასკვნითი მგომარეობა:

მივაქციოთ ყურადღება, რომ მაგალითები, რომლებიც ახლავს პირობას, ამ პირობის შემადგენელი ნაწილია და აადვილებს პირობის გაგებას.

ალბათ უკვე უჩემოდაც გასაგებია, რომ მიმდინარე პოზიცია გამოყოფილია სქელი ჩარჩოთი.

ჯერ ფორმალურად აღვწეროთ ალგორითმი.

1. თავაკი გადავიდეს ერთი პოზიციით მარჯვნივ;
2. თუ მიმდინარე პოზიციაში არ არის ჰარი, განხორციელდეს გადასვლა 1-ზე;
3. თავაკი გადავიდეს ერთი პოზიციით მარცხნივ;
4. პროგრამა დასრულდეს.

როგორც ვხედავთ, პუნქტ 3-ში მოსვლა ნიშნავს, რომ მარჯვნივ მოძრაობით თავაკი პირველად მიადგა პოზიციას, რომელშიც წერია სიმბოლო ჰარი. ეს კი ნიშნავს, რომ მიმდევრობის მარჯვენა კიდე ამ პირველი ჰარის მეზობლად მარცხნივ არის განთავსებული. ცხადია, ამ აღწერის ბოლომდე გაგებისთვის გასათვალისწინებელია, რომ ჩამოყალიბებული პუნქტები სრულდება თანმიმდევრობით, გარდა შემთხვევებისა, როდესაც თავად პუნქტის შესრულება განსაზღვრავს მომდევნო პუნქტს. ვთქვათ, პირველი პუნქტის მერე აუცილებლად სრულდება მე-2 პუნქტი. მე-3-ს მერე კი აუცილებლად სრულდება პუნქტი 4. მაგრამ პუნქტ 2-ს შემდეგ პუნქტი 3 შესრულდება მხოლოდ იმ შემთხვევაში, თუ მიმდინარე სიმბოლო არის ჰარი, წინააღმდეგ შემთხვევაში, შესრულდება პუნქტი 1.

პროგრამა შეიძლება ასე ჩაიწეროს:

```
10
R
I 20
G10
20
L
S
```

როგორც ვხედავთ, პროგრამა ვერ აღმოჩნდა აღწერილ ალგორითმთან სრულ შესაბამისობაში ბიჯების მიხედვით. მიზეზი მდგომარეობს იმაში, რომ ჩვენ ვერ ჩამოვაყალიბეთ პირობა - ”მიმდინარე სიმბოლო არ არი ჰარი”. ამის ნაცვლად, ჩვენ გამოვიყენეთ პირობა ”მიმდინარე სიმბოლო არის ჰარი” და გავითვალისწინეთ ასეთ შემთხვევაში გადასვლა

პროგრამის დასკვნით ნაწილზე, წინააღმდეგ შემთხვევაში კი პროგრამა გადადის მომდევნო ბრძანებაზე, რომელიც უზრუნველყოფს 10-ით მონიშნულ ბრძანებაზე დაბრუნებას და მარჯვნივ მოძრაობის გაგრძელებას.

მივაქციოთ ყურადღება პროგრამის მესამე სტრიქონს. ამ სტრიქონში სიმბოლო I-სა და რიცხვ (ჭდე) 20-ს შორის განთავსებულია სიმბოლო ჰარი. ეს სტრიქონი უზრუნველყოფს პროგრამის შესრულებაში ე.წ. განშტოების განხორციელებას. მართლაც, თუ მიმდინარე სიმბოლო არის ჰარი, პროგრამის შესრულება გრძელდება ბრძანებიდან, რომლის ჭდეა 20 (ანუ პროგრამის მე-5 სტრიქონიდან), ხოლო წინააღმდეგ შემთხვევაში სრულდება უშუალოდ მომდევნო ბრძანება (ანუ ბრძანება, რომელიც მოთავსებულია მე-4 სტრიქონში).

აქვე ვხედავთ ბრძანებების ჯგუფის ჯერადი შესრულების (ციკლის) შესაძლებლობას, რაც განპირობებულია ამოცანის პირობით. თავაკმა უნდა იმოძრაოს მარჯვნივ, სანამ არ მიადგება პირველ ჰარს. რის შემდეგაც უნდა დაბრუნდეს ერთი პოზიციით მარცხნივ და შეწყვიტოს შესრულება. ამას უზრუნველყოფს პროგრამის პირველი 5 სტრიქონი. პროგრამა მიადგება ამ სტრიქონებიდან მე-5-ს (ჭდე 20) მაშინ და მხოლოდ მაშინ, თუ თავაკი მივიდა მიმდევრობის მარჯვნივ მოთავსებულ პირველ ჰარზე.

მოყვანილ პროგრამას შეესაბამება ალგორითმის ასეთი აღწერა:

1. თავაკი გადავიდეს ერთი პოზიციით მარჯვნივ;
2. თუ მიმდინარე პოზიციაში არის ჰარი, განხორციელდეს გადასვლა 4-ზე;
3. განხორციელდეს გადასვლა 1-ზე;
4. თავაკი გადავიდეს ერთი პოზიციით მარცხნივ;
5. პროგრამა დასრულდეს.

ეხლა კი მოდით, ოდნავ გავართულოთ მოყვანილი ამოცანა. დავუშვათ, რომ თავდაპირველად თავაკი შეიძლება იყოს არა მარტო საკუთრივ მოცემულ მომდევრობაზე, არამედ მის მარცხნივაც.

ამოცანა 2. სიმბოლოთა მიმდევრობის მარჯვენა კიდეზე გადასვლა, თუ თავაკი თავდაპირველად მიმდევრობაზეა ან მის მარცხნივ.

ლენტის სასტარტო მდგომარეობა. ლენტზე მოთავსებულია სიმბოლოთა ნებისმიერი მიმდევრობა, რომელიც არ შეიცავს ჰარებს, ამ მიმდევრობისგან მარცხნივ და მარჯვნივ ყველა პოზიცია შევსებულია ჰარებით

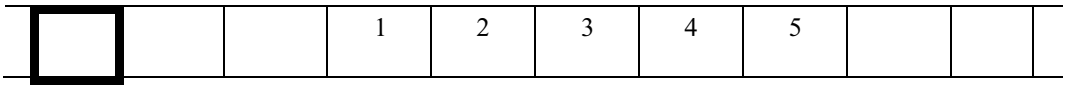
თავაკის სასტარტო მდგომარეობა. თავაკი განთავსებულია მიმდევრობის მიერ დაკავებული სივრცის შიგნით (ანუ ამ მიმდევრობის ნებისმიერ სიმბოლოზე) ან ამ სივრცის მარცხნივ.

ლენტის დასკვნითი მდგომარეობა. ლენტის დასკვნითი მდგომარეობა იგივეა, რაც სასტარტო.

თავაკის დასკვნითი მდგომარეობა. თავაკი უნდა განთავსდეს სიმბოლოთა საწყისი მიმდევრობის მარჯვენა კიდეზე.

მაგალითი 1:

საწყისი მდგომარეობა

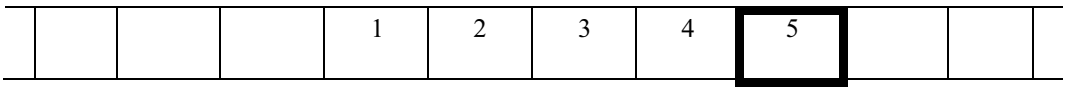


დასკვნითი მდგომარეობა

მაგალითი 2:

საწყისი მდგომარეობა

დასკვნითი მდგომარეობა



ცხადია, აქ თავდაპირველად მისაღწევი იქნება მდგომარეობა, როდესაც თავაკი მიმდევრობაზეა და მხოლოდ ამის შემდეგ გაგრძელდება შესრულება ჩვენთვის უკვე ცნობილი გზით.

მოდით, შევეცადოთ პირდაპირ შევადგინოთ სათანადო პროგრამა:

```
5
R
```

I 5

10

R

I 20

G10

20

L

S

თითქოს ყველაფერი უკიდურესად მარტივად არის. პირველი 3 სტრიქონი უზრუნველყოფს თავაკის საწყისი მდგომარეობის მოყვანას ამოცანა 1-ის საწყის მდგომარეობაში. შემდეგ კი მოდის ამოცანა 1-ის ამოხსნა ყოველგვარი ცვლილებების გარეშე. მაგრამ, აქ ჩვენ გაგვეპარა შეცდომა. ადვილად დავრწმუნდებით, რომ **პროგრამა არასწორად იმუშავებს, თუ ვირტუალური მანქანის სასტარტო მდგომარეობა აკმაყოფილებს დასკვნით პირობას**. მართლაც, მიმდევრობის მარჯვენა კიდეზე მდებარე თავაკის გადანაცვლება მარჯვნივ გამოიწვევს მის განთავსებას ჰარის შემცველი უჯრის გასწვრივ, რის შემდეგაც პროგრამა დაბრუნდება მე-5 ჭდით მონიშნულ ბრძანებაზე და მოხდება ის, რასაც პროგრამისტები ეძახიან **ჩაციკლვას**. ჩვენ შემთხვევაში, ლენტი უსასრულო არ არის, ამიტომ მოხდება თავაკის მისვლა ლენტის მარჯვენა საზღვარზე და, რადგან იმის მარჯვნივ არაფერი არ არის, პროგრამის შესრულება შეწყდება ისე, რომ მიზანი არ იქნება მიღწეული.

ეს მაგალითი მიგვანიშნებს, თუ **რამდენად ყურადღებით უნდა იყოს პროგრამისტი, რათა არ დატოვოს პროგრამის არასწორად შესრულების შესაძლებლობა.**

ასეთი შეცდომა არ წარმოიშვებოდა, თავიდანვე რომ მომხდარიყო მიმდინარე სიმბოლოს შემოწმება, მაგალითად ასე:

3

I 5

G10

5

R

G3
10
R
I 20
G10
20
L
S

თუმცა, დარწმუნებული ვარ, ბევრს უფრო მოეწონება ამოხსნის გზა, რომელიც ემყარება დასაწყისში მარცხნივ 1 გადაადგილების შესრულებას:

L
5
R
I 5
10
R
I 20
G10
20
L
S

ეხლავე დავაფიქსიროთ, რომ უმარტივესი ამოცანებიც კი შეიძლება გადაიჭრას სხვადასხვა პროგრამებით (ანუ სხვადასხვა გზით). სინამდვილეში, მთელ რიგ შემთხვევაში, ვერც ვიტყვით რომელი ამოხსნაა უკეთესი. საზოგადოდ, აქ ვლინდება პროგრამისტის, როგორც ინდივიდის, აზროვნების თავისებურება. ერთგვარად, რომ შევაჯამოთ ამ საკითხთან დაკავშირებული პრობლემატიკა, უნდა ვთქვათ, რომ ყოველი პროგრამისტი გამოიმუშავებს დაპროგრამების საკუთარ სტილს. მისი დაწერილი პროგრამის ის ვარიანტი ჩაითვლება უკეთესად, რომელიც უფრო უკეთ შეესაბამება ამ პიროვნების აზროვნების კონსტიტუციას, რომელიც მისთვის უფრო გასაგები და გამჭვირვალეა. და რაც უფრო გამჭვირვალეა ამოხსნა მისი ავტორისთვის, მით ნაკლებია შანსი ამ

ამოხსნაში მან რამე შეცდომა დაუშვას, ხოლო თუ მაინც გაეპარა შეცდომა, მას გაუადვილდება ამ შეცდომის ლოკალიზება (აღმოჩენა) და გასწორება.

და მოდით ახლავე შევთანხმდეთ, რომ დაპროგრამების მასწავლებელმა არამც და არამც არ უნდა მოახვიოს მოსწავლეს დაპროგრამების რომელიმე სტილი. მან უნდა დაანახოს ამოხსნათა სპექტრის სიმდიდრე, ჩამოაყალიბოს ამოხსნათა შედარების მთავარი კრიტერიუმები, ხოლო საბოლოო არჩევანი კი დაუტოვოს თავად მოსწავლეს. ისევე, როგორც ქართულის კარგი მასწავლებელი არ ასწავლის მოსწავლეს მეტყველებას და წერას გამზადებული შტამპებით და მუსიკის კარგი მასწავლებელი არ დათრგუნავს თავისი მოსწავლის ინდივიდუალობას.

3.3. განშტოებების და ციკლების გაცნობა ტიურინგის ვირტუალურ სასწავლო მანქანაზე

როგორც ვხედავთ, ჩვენ ჯერ-ჯერობით არ გამოგვიყენებია ჩაწერის ბრძანება (W). ამ ბრძანების გამოყენების საილუსტრაციოდ ჩამოვაყალიბოთ კიდევ ერთი, ცოტა უფრო რთული ამოცანა.

ამოცანა 3. ორობითი კომბინაციის ინვერტირება.

ლენტის სასტარტო მდგომარეობა. ლენტზე მოთავსებულია სიმბოლოთა არაცარიელი მიმდევრობა, რომელიც შედგება მხოლოდ ’0’ და ’1’-გან, ამ მიმდევრობისგან მარცხნივ და მარჯვნივ ყველა პოზიცია შევსებულია ჰარებით.

თავაკის სასტარტო მდგომარეობა. თავაკი განთავსებულია მიმდევრობის მიერ დაკავებული სივრცის შიგნით (ანუ ამ მიმდევრობის ნებისმიერ სიმბოლოზე).

ლენტის დასკვნითი მდგომარეობა. ლენტის დასკვნით მდგომარეობაში მოცემული მიმდევრობის ყველა სიმბოლო უნდა იყოს შეცვლილი: ’0’ უნდა შეიცვალოს ’1’-ით და პირიქით.

თავაკის დასკვნითი მდგომარეობა. თავაკი უნდა განთავსდეს სიმბოლოთა დასკვნითი მიმდევრობის მარჯვენა კიდეზე.

მაგალითი 1.

საწყისი მდგომარეობა:

		0	0	1	**1**	1	0	1	0		

დასკვნითი მდგომარეობა:

		1	1	0	0	0	1	0	**1**		

მაგალითი 2.

საწყისი მდგომარეობა:

		0	0	1	1	1	0	1	**0**		

დასკვნითი მდგომარეობა:

		1	1	0	0	0	1	0	**1**		

საფიქრალია, რომ აქ აზრი აქვს თავიდან თავაკი განვათავსოთ მიმდევრობის მარცხენა კიდეზე, საიდანაც ვამოძრაოთ მარჯვენა კიდემდე და გზად კი შევცვალოთ ყველა შემხვედრი სიმბოლო.
პროგრამა შეიძლება შემდეგნაირად გამოიყურებოდეს:

```
10
L
I 20
G10
20
R
I 40
I030
W0
G20
30
W1
G20
40
L
S
```

აქ პირველი 4 სტრიქონი უზრუნველყოფს იმ პოზიციაზე მისვლას, რომელიც განთავსებულია მოცემული მიმდევრობის კიდურა მარცხენა პოზიციის მეზობლად მარცხენა მხრიდან. რის შემდეგაც იწყება მოძრაობა მარჯვნივ. როგორც კი ამ მოძრაობაში შეგვხდება სიმბოლო ჰარი, განხორციელდება ერთი პოზიციით მარცხნივ გადასვლა და მოხდება პროგრამის მუშაობის შეწყვეტა, მანამდე კი გზად შემხვერდი ყველა ’0’ იცვლება ’1’-ით, ხოლო ყველა ’1’ კი ’0’-ით. მიაქციეთ ყურადღება, რომ მარჯვნივ მოძრაობის დროს, ჰარისგან განსხვავებული ყოველი სიმბოლოსთვის ბრძანებებიდან W0 და W1 შესრულდება ზუსტად ერთი.

ჩვენ არსად არ ჩამოგვიყალიბებია მოთხოვნები ჭდეების მიმართ, გარდა იმისა, რომ ისინი გარეგნულად წარმოადგენს არა უმეტეს 5-ნიშნა არაუარყოფით ათობით მთელ რიცხვებს და თითოეული მათგანი იკავებს ცალკე სტრიქონს. ამ დროს, ჭდე გამოიყენება პროგრამის ამა თუ იმ ბრძანების მონიშვნისთვის, ანუ ადგილმდებარეობის დასახელებისთვის. ამის გამო, სასურველია, ჭდეები განთავსებული იყოს პროგრამაში მათი რიცხვითი მნიშნელობის ზრდით, რათა დამატებით ურთიერთგანთავსების ინფორმაციასაც იძლეოდეს... თუმცა, ისიც ვთქვათ, რომ დაპროგრამების განვითარების ერთ-ერთი უმნიშვნელოვანესი ეტაპი უკავშირდება ჭდეებთან ბრძოლას. შესაბამისად, დაპროგრამების თანამედროვე ენებში მოთხოვნილება ჭდეებზე მინიმუმამდეა დაყვანილი... ცოტა წინ გავუსწრებთ მოვლენებს და ვიტყვით, რომ ბრძოლა ჭდეების წინააღმდეგ ჩვენც გვექნება ჩვენი სასწავლო ვირტუალური მანქანის განვითარების კვალობაზე.

ეხლა კი ჩამოვაყალიბოთ ახალი ამოცანა. ეს ამოცანა სირთულით აღემატება ადრე განხილულებს.

ამოცანა 4. არაუარყოფითი სამობითი რიცხვის ერთით გაზრდა.

ლენტის სასტარტო მდგომარეობა. ლენტზე მოთავსებულია სიმბოლოთა არაცარიელი მიმდევრობა, რომელიც წარმოადგენს არაუარყოფით რიცხვს, ჩაწერილს სამობით სისტემაში; ამ მიმდევრობისგან მარცხნივ და მარჯვნივ ყველა პოზიცია შევსებულია ჰარებით.

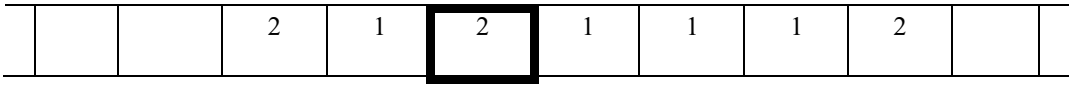
თავაკის სასტარტო მდგომარეობა. თავაკი განთავსებულია მიმდევრობის მიერ დაკავებული სივრცის შიგნით (ანუ ამ მიმდევრობის ნებისმიერ სიმბოლოზე) ან მის მარცხნივ.

ლენტის დასკვნითი მდგომარეობა. ლენტის დასკვნით მდგომარეობაში მოცემული მიმდევრობა უნდა შეესაბამებოდეს თავიდან მოცემული მიმდევრობის შესაბამის რიცხვთან შედარებით ერთით გაზრდილ რიცხვს, ჩაწერილს სამობით სისტემაში.

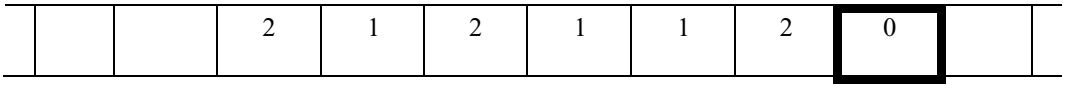
თავაკის დასკვნითი მდგომარეობა. თავაკი უნდა განთავსდეს სიმბოლოთა დასკვნითი მიმდევრობის მარჯვენა კიდეზე.

მაგალითი 1.

საწყისი მდგომარეობა:

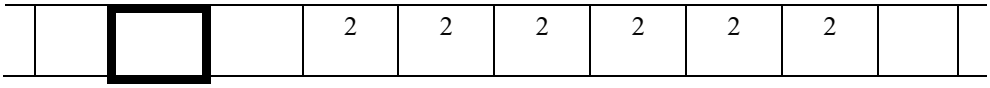


დასკვნითი მდგომარეობა:

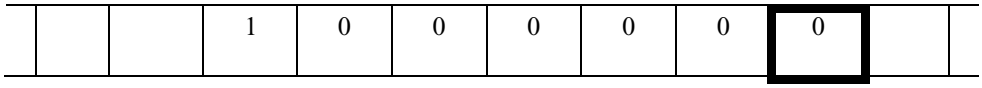


მაგალითი 2.

საწყისი მდგომარეობა:

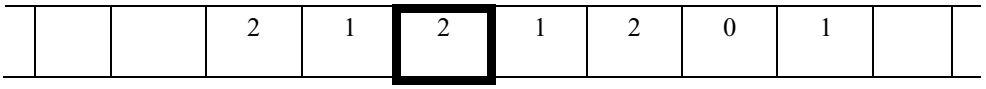


დასკვნითი მდგომარეობა:

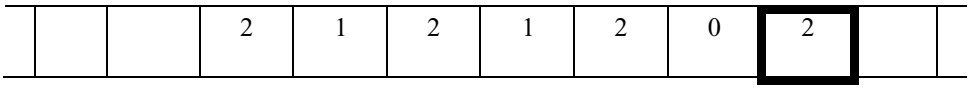


მაგალითი 3.

საწყისი მდგომარეობა:

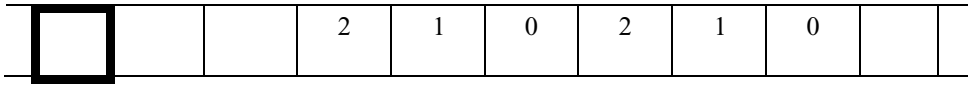


დასკვნითი მდგომარეობა:

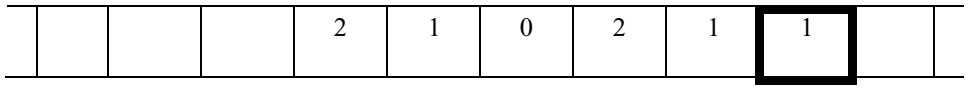


მაგალითი 4.

საწყისი მდგომარეობა:



დასკვნითი მდგომარეობა:



მოდით, ჯერ ჩამოვაყალიბოთ ამოხსნის ალგორითმი:

1. თავაკი გადავიდეს ერთი პოზიციით მარცხნივ;
2. თავაკი გადავიდეს ერთი პოზიციით მარჯვნივ;
3. თუ მიმდინარე პოზიციაში არის ჰარი, განხორციელდეს გადასვლა 2-ზე;
4. თავაკი გადავიდეს ერთი პოზიციით მარჯვნივ;
5. თუ მიმდინარე პოზიციაში არის ჰარი, განხორციელდეს გადასვლა 7-ზე;
6. განხორციელდეს გადასვლა 4-ზე;
7. თავაკი გადავიდეს ერთი პოზიციით მარცხნივ;
8. თუ მიმდინარე პოზიციაში არის 1, განხორციელდეს გადასვლა 16-ზე;
9. თუ მიმდინარე პოზიციაში არის 2, განხორციელდეს გადასვლა 18-ზე;
10. მიმდინარე პოზიციაში ჩაიწეროს 1;
11. თავაკი გადავიდეს ერთი პოზიციით მარჯვნივ;
12. თუ მიმდინარე პოზიციაში არის ჰარი, განხორციელდეს გადასვლა 14-ზე;
13. განხორციელდეს გადასვლა 11-ზე;
14. თავაკი გადავიდეს ერთი პოზიციით მარცხნივ;
15. პროგრამის შესრულება შეწყდეს;
16. მიმდინარე პოზიციაში ჩაიწეროს 2;
17. განხორციელდეს გადასვლა 11-ზე;
18. მიმდინარე პოზიციაში ჩაიწეროს 0;
19. განხორციელდეს გადასვლა 7-ზე;

უკვე შეგვიძლია დავინახოთ, რომ პუნქტები 1-3 უზრუნველყოფს თავაკის მოხვედრას მოცემული მიმდევრობის სივრცეში (თუ თავაკი თავიდანვე ამ სივრცეში იყო, საწყის პოზიციაზე დარჩება, ხოლო თუ იყო მარცხნივ, გაჩერდება მიმდევრობის კიდურა მარცხენა პოზიციაზე). პუნქტები 4-6 უზრუნველყოფს თავაკის განთავსებას მიმდევრობის მიერ დაკავებული სივრცის გარეთ, ამ სივრცის მარჯვენა კიდის მარჯვნიდან მეზობელ პოზიციაში. პუნქტები 7-10 და 16-19 უზრუნველყოფს რიცხვის ერთით გაზრდას. თანაც პუნქტ 10-ზე მივდივართ მაშინაც, როცა დასამუშავებელი სიმბოლო არის '0' და მაშინაც, როცა ეს სიმბოლო არის ჰარი. ხოლო, თუ კიდურა პოზიციაში არის 2, მაშინ მუშაობს ციკლი პუნქტებისგან 9,18,19,7. და ბოლოს, პუნქტები 11-15 უზრუნველყოფს დასკვნითი მოქმედებების შესრულებას, ანუ თავაკის გადაყვანას მიმდევრობის კიდურა მარჯვენა პოზიციაში და პროგრამის შესრულების შეჩერებას.

ვირტუალურ მანქანას ახლადგაცნობილი კაცი, ეჭვგარეშეა, გაიკვირვებს ალგორითმის პირველი 2 პუნქტით. მართლაც, რა აზრი აქვს თავაკის ჯერ მარცხნივ გაწევას, ხოლო იქვე, მომდევნო სვლით, უკან დაბრუნებას. მაგრამ ჩვენ ხომ უკვე ვიცით, რომ პუნქტები 2,3 ქმნიას ციკლს, რომელიც უზრუმველყოფს თავაკის გადაადგილებას მარჯვნივ, სანამ იგი არ მიადგება ჰარისგან განსხვავებულ პირველ სიმბოლოს. ამ დროს პუნქტი 1 ამ ციკლში არ შედის, მაგრამ ის ასრულებს მოქმედებას, რომლის შემდეგ ციკლი შეიძლება განხორციელდეს დაუბრკოლებლივ. მუსიკის თეორიას გაცნობილები უთუოდ შეამჩნევენ აქ ანალოგიას ე.წ. გარე ტაქტთან. და გარე ტაქტის როლში პუნქტი 1 გვევლინება...

სათანადო პროგრამის დაწერა უკვე არ წარმოადგენს რაიმე სირთულეს.

```
L
10
R
I 10
20
R
I 30
```

G20
30
L
I160
I270
W1
40
R
I 50
G40
50
L
S
60
W2
G40
70
W0
G30

3.4. ქვეპროგრამა ტიურინგის ვირტუალურ სასწავლო მანქანაზე

დაპროგრამებისთვის ქვეპროგრამის ცნების მნიშვნელობის გადაჭარბებით შეფასება ერთობ ძნელია. ფაქტია, რომ ამ ცნებას ემყარება საზოგადოდ დაპროგრამების ტექნოლოგიებისა და პარადიგმების განვითარება.

აქამდე ჩამოყალიბებული საშუალებებით ქვეპროგრამის და, მით უფრო რეკურსიის, ცნების გაცნობა ტიურინგის სასწავლო ვირტუალურ მანქანაზე შეუძლებელი იყო.

მრავალი მცდელობის შემდეგ ტიურინგის სასწავლო მანქანის ენის სათანადო გაფართოებისთვის მოინახა ლამაზი გადაწყვეტა, რომლის ავტორია **სანდრო ბარნაბიშვილი**.

ამ გადაწყვეტით მოხდა გადასვლის ოპერაციების (G და I) გაფართოება მათზე დამატებით დაბრუნების მოთხოვნის დაკისრებით, თუ კი ბრძანება დასრულდებოდა სიმბოლოთი R. ასეთ პირობებში გარეგნულად არშეცვლილ ბრძანებას S დაეკისრა ყველაზე ბოლოს ჩამოყალიბებული დაბრუნების მოთხოვნის განხორციელება, თუკი მისი შესრულების მომენტისთვის ასეთი ერთი მაინც დაუკმაყოფილებელი მოთხოვნა იარსებებდა. წინააღმდეგ შემთხვევაში ეს ბრძანება ჩვეულებრივ ამთავრებს დავალების შესრულებას.

განვიხილოთ ამ ბრძანების საილუსტრაციო 2 მაგალითი.

ჯერ მოვიყვანოთ ბოლოს შესრულებული ამოცანის ახალი ამოხსნა.

ამოსხნის იდეა ასეთია – დავყოთ ამოცანა ნაწილებად, რომელთაც დაეკისრება, შესაბამისად:

1. მიმდინარე მდგომარეობიდან ლენტზე მოთავსებულ სიმბოლოთა მიმევრობის მარჯვენა კიდეზე მისვლა (ცხადია, აქ უნდა გვქონდეს გარანტია, რომ თავიდან თავაკი ან მიმდევრობაზეა, ან მის მარცხნივ);
2. სამობითი რიცხვის ერთით გაზრდა იმ პირობით, რომ თავაკი სტარტზე ამ რიცხვის მარჯვენა კიდეზე დგას.

თუ ეს ნაწილები უკვე გვაქვს, მაშინ ახალი ბრძანებების მეშვეობით ჯერ შევასრულოთ პირველი ნაწილი, შემდეგ მეორე და შემდეგ კი ისევ პირველი.

ასეთი მიდგომა ითხოვს დაპროგრამების სხვა სტილის დაუფლებას. ვნახოთ, როგორ გამოიყურება ამოცანის ახალი ამოხსნა.

```
G80
10
L
15
R
I 15
20
R
I 30
G20
```

30
L
S
40
I160
I270
W1
50
S
60
W2
G50
70
W0
L
G40
80
G10R
G40R
G10R
S

აქ ჭდიდან 10 იწყება ჩვენს მიერ ზემონახსენები პირველი ნაწილი, ხოლო 40-დან კი ზემონახსენები მეორე ნაწილი. რეალურად ერთიც და მეორეც წარმოადგენს ქვეპროგრამებს. და მიუხედავად იმისა, რომ პროგრამის სიგრძე არ შემცირებულა (პირიქით, ცოტა გაიზარდა), დარწმუნებული ვარ, ბევრს ეს ვარიანტი უფრო მოეწონება, რადგან პროგრამის ლოგიკა აშკარად გამარტივდა.

მეორე ამოცანა კი გვაძლევს რეკურსიის გამოყენების ერთობ ეფექტიანი და არც თუ მარტივი შემთხვევის ილუსტრაციას.

ამოცანა 5. ფრჩხილებიანი მიმდევრობის კორექტულობის შემოწმება. ფრჩხილებიანი მიმდევრობის კორექტულობა გაგებულია ჩვეულებრივი აზრით. კონკრეტულად, მრგვალი ფრჩხილებისგან შედგენილ

მიმდევრობას ვუწოდებთ კორექტულს (კფმ), თუ იგი აკმაყოფილებს პირობას:

<კფმ>::= | (<კფმ>)

ამ განმარტებით ცარიელი მიმდევრობაც ითვლება კორექტულ მიმდევრობად, მაგრამ ჩვენი ამოცანის პირობით მოცემული მიმდევრობა ცარიელი არ შეიძლება იყოს.

ლენტის სასტარტო მდგომარეობა. ლენტზე განთავსებულია მხოლოდ მრგვალი ფრჩხილებისგან შედგენილი არაცარიელი მიმდევრობა. ლენტის ყველა დარჩენილი პოზიცია შევსებულია ჰარებით.

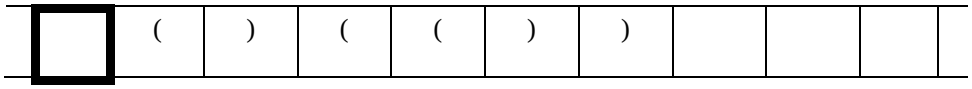
თავაკის სასტარტო მდგომარეობა. თავაკი თავიდან დგას ფრჩხილებისგან შედგენილი მიმდევრობის მარცხენა კიდის მარხენა მეზობელ პოზიციაზე.

ლენტის დასკვნითი მდგომარეობა. დასკვნითი მდგომარეობა იდენტურია სასტარტოსი.

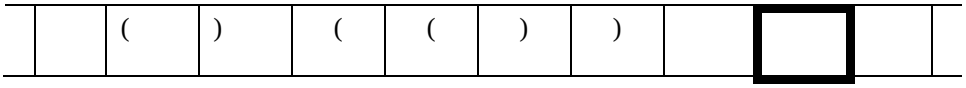
თავაკის დასკვნითი მდგომარეობა. თუ მიმდევრობა კორექტულია, თავაკი უნდა განთავსდეს მიმდევრობის მარჯვენა კიდესგან 2 პოზიციით მარჯვნივ, თუ არა და ნებისმიერ სხვა მდგომარეობაში.

მაგალითი 1.

საწყისი მდგომარეობა:



დასკვნითი მდგომარეობა:



პროგრამა, რომელიც წყვეტს ამ ამოცანას შემდეგნაირად გამოიყურება:

```
10
R
20
I(10R
I(20
R
S
```

ვფიქრობ უდავია, რომ პროგრამა არც ისე ტრივიალურია, თუმცა ზომით საკმაოდ პატარაა.

მოცემული მომენტისთვის არსებობს ტიურინგის სასწავლო მანქანის 2 რეალიზაცია. ორივე შესრულებულია დაპროგრამების ენაზე C# ჩემი ხელმძღვანელობით. პირველის ავტორია **ავთანდილ რუხაძე**. მან რეალიზაცია შეასრულა 2011 წელს სამაგისტრო ნაშრომის ფარგლებში წმიდა ანდრია პირველწოდებულის სახელობის ქართულ უნივერსიტეტში. მეორე რეალიზაცია ეკუთვნის ამავე უნივერსიტეტის ბაკალავრიატის სტუდენტს, **სანდრო ბარნაბიშვილს** და შესრულებულია 2013 წელს. რუხაძის რეალიზაცია ითვალისწინებს სისტემასთან მუშაობას ვებ-აპლიკაციის რეჟიმში, ხოლო ბარნაბიშვილის რეალიზაცია გამოირჩევა უფრო დახვეწილი დიზაინით და ზემოაღწერილი გაფართოებული საშუალებების ჩამატებით, მაგრამ მუშაობა ამ რეალიზაციასთან შესაძლებელია მხოლოდ იმ ასლთან, რომელიც მოცემულ კომპიუტერზეა დაინსტალირებული. მოცემულ მომენტში შემოთავაზებული მეთოდიკა გამოიყენება სასწავლო პროცესში საქართველოს საპატრიარქოს წმიდა ანდრია პირველწოდებულის სახელობის ქართულ უნივერსიტეტში და სკოლა-პანსიონ "მთიებში".

დასკვნა.

პირველი თავი მიძღვნილია ალგებრულ გამოსახულებათა ჩაწერის სხვა და სხვა ნოტაციის შესწავლისა და ამ ნოტაციებს შორის ურთიერთკავშირების დადგენისადმი. ამ თავში ავტორმა თავი მოუყარა ალგებრულ გამოსახულებათა ე.წ. პოლონურ ნოტაციასთან დაკავშირებულ და სისტემურ პროგრამირებაში ნაყოფიერად გამოყენებულ ფაქტებს. რის შემდეგაც ავტორი გვთავაზობს სიახლეს – ინფიქსური ნოტაციიდან გამოსახულების პრეფიქსულ ნოტაციაში გადაყვანის პირდაპირ ალგორითმს, რომელიც ერთდროიულად 2 სტეკის გამოყენების იდეას ემყარება. ამის შემდეგ ავტორს შემოაქვს შეუღლებულის ცნება ალგებრული გამოსახულების პოსტფიქსური ჩანაწერის მიმართ. შეუღლებულის ცნების გამოკვლევით დადგინდა მისი მთელი რიგი თვისებები. ამ თვისებებზე დაყრდნობით ავტორმა მიიღო პრაქტიკული მნიშვნელობის დასკვნა – **პრეფიქსული ჩანაწერის თანხლება პოსრტფიქსურით საჭირო არ არის**, რადგან გამოსახულების მნიშვნელობის მიღება შესაძლებელია პირდაპირ მისი პრეფიქსული სახიდან. ამისათვის კი საკმარისია პრეფიქსული ჩანაწერის დამუშავება მარჯვნიდან მარცხნივ პოსტფიქსური ჩანაწერისთვის განსაზღვრული სტანდარტული ალგორითმის შეუღლებული ალგორითმით. იმის გათვალისწინებით, რომ ზემოხსენებული სტანდარტულის შეუღლებული ალგორითმის სირთულე იდენტურია საკუთრივ სტანდარტულის სირთულისა, შედეგად მთლიანად მოიხსნა **პრეფიქსულიდან პოსტფიქსურში გამოსახულების გადაყვანის აუცილებლობა**. ამავე თავში შემოღებულ იქნა შეუღლებულის ცნება გამოსახულების ეხლა უკვე პრეფიქსული ჩანაწერის მიმართ და, როგორც მოსალოდნელი იყო, მიღებულ იქნა პოსტფიქსური ჩანაწერის შეუღლებულისთვის დადგენილი ყველა თანაფარდობის სარკისებული ანალოგი. მეტიც, განხილულ იქნა უფრჩხილებო ჩანაწერის გამნზოგადოებული ცნება და მისთვისაც დამტკიცებულ იქნა სათანადო დებულებები, რომლებსაც პრაქტიკული მნიშვნელობა აქვს.

შემდგომი კვლევისთვის საინტერესოა ძარღვის დამუშავება, რომელიც მდგომარეობს იმაში, რომ ამ თავში განხილული **ყველა სახის**

უფრჩხილებო ჩანაწერი წარმოადგენს გრაფის (ხის) სახით მოცემული ინფორმაციის ალგებრულ წარმოდგენას.

მეორე თავი მიეძღვნა სიმბოლურ დიფერენცირებას და პირველ თავში მიღებული შედეგების პრაქტიკულ გამოყენებას. სიმბოლური დიფერენცირებისთვის შემოღებულ იქნა შიდა ენა, რომელზედაც აღიწერა გაწარმოების ცნობილი ფორმულები. ავტორმა დაასაბუთა ამ ენის შემოღების მიზანშეწონილობა და იქვე მოგვცა მისი რეალიზაცია C++-ზე. ამ რეალიზაციამ, ფაქტობრივად უზრუნველყო პრეფიქსული სახით მოცემული **ერთი ცვლადის ფუნქციონალური გამოსახულებების ანალიტიკური გაწარმოება**. ფუნქციონალური გამოსახულებების შიდა წარმოდგენისთვის, რაც დაეფუძნა პრეფიქსულ ნოტაციას, შემოთავაზებულ იქნა სტანდარტული ვექტორის ტიპის კონტეინერის განზოგადოება. გაწარმოების ფუნქცია შევიდა ამავე თავში შემუშავებულ პროგრამათა კომპლექსში, რომელმაც მოიცვა – ინფიქსური სახით (სტრიქონით) მოცემული ფუნქციონალური გამოსახულების გადაყვანა პრეფიქსულ სახეზე, პრეფიქსული სახიდან გამოსახულების გადაყვანა ინფიქსურ სახეზე (სტრიქონში), პრეფიქსული სახით მოცემული გამოსახულების გამოთვლა უცნობის გარკვეული მნიშვნელობისთვის, პრეფიქსული გამოსახულების გამარტივება. ამ კომპლექსმა უკვე **შეაიარაღა პრობლემური პროგრამისტი ფუნქციონალური გამოსახულებების დამუშავების მრავალმხრივი ინსტრუმენტით ალგორითმული ენის დონიდან**, რამაც **შეიძლება განაპირობოს გამოთვლითი მათემატიკის, ფიზიკის და სხვა დარგების წიაღში წარმოქმნილი ამოცანებისთვის გადაწყვეტის პრინციპიალურად ახალი შესაძლებლობები**. აქვე ავტორმა მოიყვანა დაპროგრამების ენის დონიდან ფუნქციონალურ გამოსახულებათა მრავალმხრივი დამუშავებისთვის განკუთვნილი კლასის საინტერფეისო ნაწილის აღწერა. ამავე თავში მოყვანილია ავტორის მიერ პასკალზე შესრულებული ალგებრულ გამოსახულებათა დამუშავებისთვის განკუთვნილი მოდულის აღწერა. აქ შესაძლებელია დამუშავდეს რამდენიმე ცვლადის შემცველი გამოსახულება, მაგრამ ამ რეალიზაციაში გათვალისწინებულია მხოლოდ გამოსახულების გადაყვანა სტრიქონის სახით მოცემული ინფიქსური ფორმიდან

პოსტფიქსურ წარმოდგენაზე დამყარებულ შიდა ფორმაზე და შიდა ფორმით მოცემული გამოსახულების მნიშვნელობის გამოთვლა ამ გამოსახულებაში შემავალი ცვლადების გარკვეული მნიშვნელობებისთვის.

მეორე თავში მოყვანილი მასალის **შემდგომი განვითარება** შესაძლებელია ასეთი მიმართულებებით:

1. **დაიხვეწოს და, საჭიროების შემთხვევაში, შეივსოს ავტორის მიერ შემოთავაზებული კლასის პროექტი და განხორციელდეს მისი რეალიზაცია;**
2. **შემუშავდეს რამდენიმე ცვლადის შემცველი ფუნქციონალური გამოსახულებების დამუშავებისთვის განკუთვნილი პროგრამული რეალიზაცია, რომელიც მოიცავს კერძო წარმოებულების მიღებას;**
3. **გამოკვლეულ იქნას გამოსახულებათა შემოკლება ამ მიზნისთვთვის განკუთვნილი უფრო მძლავრი აპარატის შემუშავებისთვის;**
4. **ავტორის მიერ შემოთავაზებული მიდგომა გამოყენებულ იქნას გაწარმოების შესწავლისთვის განკუთვნილი სასწავლო პროგრამის შემუშავებისთვის;**
5. **მოისინჯოს მსგავსი რეალიზაციების შესრულება დაპროგრამების სხვა ენებზე. შემუშავდეს ამ რეალიზაციების სხვა ენებზე გადატანისთვის განკუთვნილი კონვერტორები;**
6. **მოისინჯოს ავტორისეული მიდგომა სიმბოლური ინტეგრების ამოცანის გადასაწყვეტად;**
7. **განისაზღვროს ამოცანათა კლასი, რომლისთვისაც შემოთავაზებული ინსტრუმენტარი განსაკუთრებით ეფექტიანია.**

მესამე თავი მიეძღვნა ავტორის მიერ შემუშავებულ ტიურინგის ვირტუალური მანქანის მოდიფიკაციას, განკუთვნილს სასწავლო პროცესში გამოყენებისთვის. ავტორმა ჩამოაყალიბა მოთხოვნები თანამედროვე სასწავლო პროცესში დაპროგრამების სასტარტო შესწავლისთვის განკუთვნილი საშუალებების მიმართ. ამის შემდეგ მას მოყავს სასწავლო პროცესში გამოყენებისთვის გამიზნული ტიურინგის ვირტუალური მანქანის მისეული მოდიფიკაციის აღწერა. რის შემდეგაც

ამ თავში განიხილება რამდენიმე კონკრეტული მაგალითი, რომლებიც სწორ წარმოდგენას ქმნის შემოთავაზებული სასწავლო საშუალების ფარგლებში პოტენციურად დაფარული დავალებების სირთულის დიაპაზონის შესახებ. ამ თავის ბოლოს ავტორი გვაცნობს შემოთავაზებული სასწავლო მანქანის შემავალი ენის გაფართოებას, რომლის მეშვეობითაც ხდება ქვეპროგრამისა და რეკურსიის ცნების გაცნობა. აქვე ხდება სათანადო საილუსტრაციო მაგალითების განხილვა. რეალურად, შემოთავაზებული მეთოდიკა **იძლევა შესაძლებლობას მოსწავლე შეუდგეს უშუალოდ კომპიუტერთან დავალებების შესრულებას სწავლების მეორე საათიდან**. აღწერილი მეთოდიკა უკვე დანერგილია საქართველოს საპატრიარქოს წმიდა ანდრია პირველწოდებულის სახელობის ქართულ უნივერსიტეტში (2011 წლიდან) და პანსიონ "მთიებში" (2014 წლიდან). ავტორი გულისხმობს, რომ შემოთავაზებული საშუალება უმაღლეს სასწავლებლების სასწავლო პროცესში მიზანშეწონილია გამოყენებულ იქნას მხოლოდ დაპროგრამების სწავლების სასტარტო ეტაპზე და სულ 3-4 გაკვეთილის ხანგრძლივობით. ავტორის აზრით, ტიურინგის სასწავლო ვირტუალური მანქანის ათვისება კარგი წინაპირობაა დაპროგრამების საფუძვლების შემდომი წარმატებული დაუფლების პროცესის გაგრძელებისთვის, ვინაიდან მოსწავლე გარდა აუცილებელი უნარების შეძენისა, იქმნის კარგ ფსიქოლოგიურ და ემოციურ ფონს დაპროგრამების რეალური ალგორითმული ენის შესწავლისთვის. ამ მიმართულებით **სამომავლოდ შეიძლება გაკეთდეს** შემდეგი:

1. **შემუშავდეს შიდა ენის გაფართოება, რომელიც მოგვცემს შესაძლებლობას შემოღებულ იქნას პრეპროცესინგის ელემენტები. ეს კი, თავის მხრივ, ხელს შეუწყობს მოდულური დაპროგრამების გაცნობას;**
2. **რეალიზაცია გატანილ იქნას საიტზე და ხელმისაწვდომი გახდეს გარე მომხმარებლისთვის;**
3. **რეალიზაციაში გათვალისწინებულ იქნას მომხმარებლისთვის ამოსახსნელი ამოცანების შეთავაზება, მის მიერ შესრულებული ამოხსნების სისწორის ავტომატური შემოწმება, სხვადასხვა ამოხსნების შედარება (მათ შორის ვირტუალური მანქანისთვის**

დამახასიათებელი მაჩვენებლებით, როგორიცაა სიმოკლე, ამოხსნის დროს შესრულებული ბრძანებების რაოდენობა,...);

4. გამოკვლეულ იქნას ამოხსნათა სისწორის შემოწმების ფორმალიზების შესაძლებლობები;
5. გამოკვლეულ იქნას ტესტების გენერირების ავტომატიზების შესაძლებლობა.

ლიტერატურა

1. Ахо А., Ульман Дж. Теория синтакстического анализа, перевода и компиляции. Синтаксический анализ: Том 1 – Москва, Книга по требованию, 2012, 613 с.
2. Aho Alfred V., Seti Ravi, Ullman Jevvrey D. Compilers. Principles, Techniques, and Tools. Addison Wessley Publishing Company, 2001, 768 p.
3. Льюис Ф., Розенкранц Д., Стирнз Р. Теоретические основы пректирования компиляторов. Москва, «Мир», 1979,654 с.
4. Пратт Т.,Зелковиц М. Языки программирования. Санкт-Петербург, Питер, 2002, 677 с.
5. Andrew Hodges. Alan Turing: The Enigma. Vintage, 2012, 592 p.
6. Непейвода Н.Н., Скопин. И.Н. Основания программирования. Москва, 2002, 913 с.
7. Э.Дейкстра. Дисциплина программирования. Москва, «Мир», 1978, 277 с.
8. Маккиман У.,Хорнинг Дж.,Уортман Д. Генератор компиляторов. Москва, «Статистика», 1980, 527 с.
9. Альфред Ахо, Джеффри Ульман, Джон Хопкрофт. Построение и анализ вычислительных алгоритмов. Москва, Книга по требованию, 2012, 524 с.
10. Джон Хопкрофт, Раджив Мотвани, Джеффри Уильямс. Введение в теорию автоматов, языков и вычислений. Москва, Вильямс, 2015, 525 с.
11. Barry W. Bouehm, John R. Brown, Hans Kaspar, Myron Lipow, Gordon J. MacLeod, Michael J. Merritt. Charasteristics of software quality. Amsterdam, North-Holland Publisher Company, 1978, 202 p.
12. Седжвик Роберт. Алгоритмы на C++. Москва, Вильямс, 2014, 1056 с.
13. Томас Кормен, Чарльз Лейзерсон, Рональд Ривест, Клиффорд Штайн. Алгоритмы. Построение и анализ. Москва, Вильямс, 2013, 1328 с.
14. Фредерик Брукс, Хилл Чапел. Мифический человеко-месяц, или как создаются программные комплексы. Москва, Символ-плюс, 2010, 304 с.
15. Дональд Кнут. Искусство программирования. Том1. Основные алгоритмы. Санкт-Петербург, Вильямс, 2015, 720 с.
16. Йенсен К., Вирт Н. Паскаль. Руководство для пользователя и описание языка. Москва, Финансы и статистика, 1989, 256 с.
17. ზარქუა თ. Pascal. დაპროგრამების საფუძვლები. თბილისი, 1998, 96 გ.

18. Успенский В. А. Машина Поста. Москва, Наука, 1979, 96 с.
19. Цискаридзе, Дербинян. Реализация машины Поста на ЭВМ БЭСМ-6 и ее использование при обучении основам программирования. Материалы школы юных математиков-программистов ИПМ им.академика И.Н.Векуа ТГУ. Тбилиси, издательство ТГУ, 1984, 56 с.
20. Т.Заркуа. К свойству перевернутой польской записи. Internet-Edication-Sciense-2008. New Informational and Computer Tecnologies in Education and Sciense. H. Intelligence Information Systems. Vinnitsia, 2008, p.543-544.
21. Т.Заркуа. Некоторые способы обработки функциональных выражений. Winter Programming School. Материалы зимней школы по программированию, Харьков, ХНУРЭ, 2009, p 205-211.
22. თ.ზარქუა. ერთი მიდგომა ფუნქციონალურ გამოსახულებათა დამუშავების ავტომატიზების საკითხისადმი (ქართულ და ინგლისურ ენებზე). საქართველოს საპატრიარქოს წმიდა ანდრია პირველწოდებულის სახელობის ქართული უნივერსიტეტის ბიულეტენი №2, საქართველოს საპატრიარქოს წმიდა ანდრია პირველწოდებულის სახელობის ქართული უნივერსიტეტი, 2009, გვ. 50-61.
23. Т.Заркуа. Автоматизация обработки функциональных выражений на уровне алгоритмических языков. Internet-Edication-Sciense-2010. New Informational and Computer Tecnologies in Education and Sciense. Vinnitsia, 2010, p 201-206.
24. თ.ზარქუა. სამომხმარებლო პროგრამებიდან ფუნქციონალური გამოსახულებების დამუშავების ავტომატიზების საკითხისადმი (მოხსენების თეზისები ქართულ და რუსულ ენებზე). ნიკო მუსხელიშვილის გამოთვლითი მათემატიკის ინსტიტუტი, საქართველოს საპატრიარქოს წმიდა ანდრია პირველწოდებულის სახელობის ქართული უნივერსიტეტი. საერთაშორისო კონფერენცია "ინფორმაციული და გამოთვლითი ტექნოლოგიები", მიძღვნილი პროფესორების ელენე დეკანოსიძის და მურმან წულაძის ხსოვნისადმი. თეზისების კრებული. თბილისი, 2010 წელი, გვ. 52-53.
25. Т.Заркуа. К вопросу о реализации преобразования функциональных выражений на алгоритмических языках. აკადემიკოს ი.ფრანგიშვილის დაბადების 80 წლისთავისადმი მიძღვნილი საერთაშორისო სამეცნიერო კონფერენცია "საინფორმაციო და კომპიუტერული ტექნოლოგიები, მოდელირება,

მართვა". მოხსენებათა თეზისები. საქართველო, თბილისი, 1-4 ნოემბერი, 2010 წელი. საგამომცემლო სახლი "ტექნიკური უნივერსიტეტი", 2010, გვ.135.

26. თ.ზარქუა, ა.რუხაძე. ტიურინგის მანქანის მოდიფიკაციის შემუშავება სასწავლო პროცესისათვის. საქართველოს ტექნიკური უნივერსიტეტი საერთაშორისო სამეცნიერო კონფერენცია "მართვის ავტომატიზებული სისტემები და თანამედროვე საინფორმაციო ტექნოლოგიები". მოხსენებათა თეზისები, საგამომცემლო სახლი "ტექნიკური უნივერსიტეტი", თბილისი, 2011, გვ. 196-197.
27. Т.Заркуа. К вопросу об использовании понятия сопряженного по отношению к бесскобочным выражениям. Internet-Education-Sciense-2012. New Informational and Computer Tecnologies in Education and Sciense. Vinnitsia, 2012, p.156-157.
28. Т.Заркуа, С.Барнабишвили. К вопросу стартового обучения программированию. Internet-Education-Sciense-2014. New Informational and Computer Tecnologies in Education and Sciense. Vinitsia, 2014, p.269-271.

.

დანართი 1. მოდული Gamotvla. საწყისი ტექსტი (პასკალზე).

```
unit gamotvla;
interface
const np=3;
type selem = -1..4;
  shesvla = ( operandi, psevdoo, moqmedeb, dahurva, bolo );
  gadasvla = (o, m, f, t );
  maspar = array[1..np] of real;
  masindp = array[1..np] of boolean;
  tipelem = record case romelem:selem of
             -1:();
              0:(mmnish:real);
              1:(cvnom:integer);
              2:(monom:integer);
              3:();
              4:()
            end;
  tipstack = ^moqmed;
  moqmed = record
            monom:integer;
            mit:tipstack
           end;
  gamelem = record case moqm:boolean of
              true:(monom:integer;
                    adgil:1..3;
                   );
              false:(case cvladi:boolean of
                       true:(cnom:integer);
                       false:(mnish:real);
                     );
             end;
  mimt = ^gamosah;
  gamosah = record
```

```
            inf:gamelem;
            mim:mimt
           end;
 tstack = ^ts;
 ts = record
       inf:real;
       next:tstack
      end;
procedure scanfunq(x:string; var y:mimt; var mistake, npoz:byte; var
masind:masindp);
function gam(x:mimt; y:maspar):real;


implementation

procedure scanfunq(x:string; var y:mimt; var mistake, npoz:byte; var
masind:masindp);
 label 100;
 const
   adg:array[0..21] of byte = (1,1,2,2,2,2,2,2,1,1,1,1,1,1,1,1,1,1,1,3,1);
   prior:array[-2..21] of byte = (0,0,7,6,2,2,3,3,4,4,5,5,5,5,5,5,5,5,5,5,5,1,6);
 var
   mm : tipelem;
   pr,lakm,first : boolean;
   moqmst : tipstack;
   r : mimt;
   n,ns,ks,mnom,nompd : integer;
   mdg : gadasvla;
 function shecdoma:boolean;
   const maga:array[o..m,shesvla] of gadasvla = ((m,o,f,f,f),(f,f,o,m,t));
   var
     x:shesvla;
   begin
     if mm.romelem = -1 then begin shecdoma := true; mistake :=1 end
```

```
      else
       begin if lakm then x := bolo
         else
          if mm.romelem in [0,1] then x := operandi
           else
           if mm.romelem = 3 then x:=psevdoo
            else
             if mm.romelem = 4 then x := dahurva
              else
               if (adg[mm.monom] = 1) and (mm.monom <> 21) then x := psevdoo
                 else
                  x := moqmedeb;
                 mdg := maga[mdg,x];
                 if mdg = f then begin
                                  shecdoma := true;
                                  mistake := 3+ord(x)
                                 end
                  else
                   shecdoma := false
       end
  end;

 procedure relem(var mm:tipelem; var lakm:boolean);
  const
   sr = 12;
   sitmas:array[-2..sr] of string[10] =
('x','y','z','sin','cos','tg','arcsin','arccos','arctg','abs','sqr','sqrt','exp','ln','lg');
  var
   qss:char;
   qs:string[10];
   i,code:integer;
   musha:real;
  procedure neprob;
   begin
```

```
    while (ns <= n ) and (x[ns] = ' ') do ns := ns+1
  end;
procedure prob;
  var
    turn:0..2;
    gamkop:set of char;
  begin
    turn := 0;
    ks := ns;
    if x[ks] in ['0'..'9'] then
      begin
        gamkop := [chr(0)..chr(255)] - ['0'..'9','.','e','E'];
        ks := ns+1;
        turn := 1
      end
      else
        gamkop := [' ','+','-','*','/','(',')','^']+[';','#','?','0'..'9'];
    while (ks <= n) and not (x[ks] in gamkop) do
      begin
        if turn = 1 then
          if x[ks] in ['e','E'] then
            begin
              gamkop := gamkop + ['.','e','E']-['-','+'];
              turn := 2
            end
            else
          else
            if turn = 2 then gamkop := gamkop + ['-','+'];
            ks := ks+1
      end;
    ks := ks-1
  end;
begin {start relem}
  neprob;
```

```
lakm := ns=n+1;
if not lakm then
  begin
    qss := x[ns];
    ks := ns;
    if qss = '(' then mm.romelem := 3
      else
        if qss = ')' then mm.romelem := 4
          else
            if qss in ['-','+','*','/','^','#','?',':',','] then
              begin
                mm.romelem := 2;
                case qss of
                  '-': if (y = nil) or first then mm.monom := 1
                       else mm.monom := 3;
                  '+': if (y = nil) or first then mm.monom := 0
                       else mm.monom := 2;
                  '*': mm.monom := 4;
                  '/': mm.monom := 5;
                  '^': mm.monom := 7;
                  '#': mm.monom := 6;
                  '?': mm.monom := 20;
                  ':': mm.monom := 21;
                  ',': mm.monom := 0
                end
              end
            else
              begin
                prob;
                qs := copy(x,ns,ks-ns+1);
                i := -3;
                repeat
                  i :=i+1
                until (qs = sitmas[i]) or (i = sr);
```

```
                if qs = sitmas[i] then
                  begin
                    mm.romelem := 2;
                    case i of
                      1..12 : mm.monom := i+7;
                      -2..0 : begin
                                mm.romelem := 1;
                                mm.cvnom := i+3
                              end
                    end
                  end
                 else
                  begin
                    val(qs, musha, code);
                    if code = 0 then
                      begin
                        mm.romelem := 0;
                        mm.mmnish := musha
                      end
                     else
                       mm.romelem := -1
                  end
              end;
          ns := ks+1;
          first := qss in ['(','?',':',',']
    end { if not lakm}
end;

procedure wmstack(x:Integer);
  var
    r:tipstack;
  begin
    new(r);
    r^.monom := x;
```

```
    r^.mit := moqmst;
    moqmst := r
  end;

procedure rmstack(var x:integer);
 var
  r:tipstack;
 begin
   x := moqmst^.monom;
   r := moqmst;
   moqmst := moqmst^.mit;
   dispose(r)
 end;
begin {start scanfunq}
  mdg := o;
  for ns := 1 to np do masind[ns] := false;
  moqmst := nil;
  n := length(x);
  y := nil;
  first := true;
  pr := true;
  mistake := 0;
  ns := 1;
  relem(mm,lakm);
  if shecdoma then goto 100
    else
      while not lakm do
        begin
          with mm do
            if romelem < 2 then {it is operand}
              begin
                if pr then {that is first}
                  begin
                    new(y);
```

```
      r := y;
      pr := false;
    end
   else
    begin
      new(r^.mim);
      r := r^.mim
    end;
 with r^.inf do
  begin
    moqm := false;
    cvladi := romelem = 1;
    if cvladi then begin
                    cnom := cvnom;
                    masind[cvnom] := true
                   end
      else mnish := mmnish
  end
 end
else
 if romelem = 2 then {es moqmedebaa}
  begin
    while (moqmst <> nil) and (prior[moqmst^.monom] > prior[monom]) do
      begin
        rmstack(mnom);
        new(r^.mim);
        r := r^.mim;
        with r^.inf do
          begin
            moqm := true;
            monom := mnom;
            adgil := adg[mnom]
          end;
      end;
```

```
          wmstack(monom)
        end;
      with mm do
       if romelem = 3 then wmstack(-1)
        else
         if (romelem = 2) and (monom = 20) then wmstack(-2)
          else
           if (romelem = 4) or (monom = 21) then
            begin
             if romelem = 4 then nompd := -1 else nompd := -2;
             while (moqmst <> nil) and (not (moqmst^.monom+2 in [0,1])) do
              begin
               rmstack(mnom);
               new(r^.mim);
               r := r^.mim;
               with r^.inf do
                begin
                 moqm := true;
                 monom := mnom;
                 adgil := adg[mnom]
                end
              end;
             if (moqmst = nil) or (moqmst^.monom <> nompd) then
              begin
               mistake := 2;
               goto 100
              end
             else rmstack(mnom)
            end;
           relem(mm,lakm);
       if shecdoma then goto 100
  end; {while not lakm}
while moqmst <> nil do
  begin
```

```
          rmstack(mnom);
          if (mnom = -1) or (mnom = -2) then
            begin
              if mnom = -1 then mistake := 8 else mistake := 9;
              goto 100
            end;
          new(r^.mim);
          r := r^.mim;
          with r^.inf do
            begin
              moqm := True;
              monom := mnom;
              adgil := adg[mnom]
            end
        end;
      r^.mim := nil;
      100:npoz := ns-1
end; {scanfunq}

function gam(x:mimt; y:maspar):real;
  var
    r:mimt;
    a,b,c:real;
    stack:tstack;
    procedure wstack(var x:real);
      var r:tstack;
      begin
        new(r);
        r^.inf := x;
        r^.next := stack;
        stack := r
      end; //wstack
    procedure rstack(var x:real);
      var r:tstack;
```

```
    begin
      x := stack^.inf;
      r := stack;
      stack := stack^.next;
      dispose(r)
    end; //rstack
  function har(x:real; y:integer):real;
    var
      b:boolean;
      i:integer;
      p:real;
    begin
      b := y<0;
      y := abs(y);
      p := 1;
      for i := 1 to y do p := x*p;
      if b then har := 1/p else har := p;
    end;//har

function arcsin(x:real):real;
  begin
    if x = 1.0 then arcsin := pi*0.5
      else
        if x = -1.0 then arcsin := -pi*0.5
          else arcsin := arctan(x/sqrt(1-sqr(x)))
  end;//arcsin

begin
  r := x;
  stack := nil;
  while r <> nil do
    begin
      with r^.inf do
        if moqm then
```

```
 if monom in [0,21] then
   else
     begin
       rstack(b);
       if adgil >=2 then rstack(a);
       if adgil = 3 then rstack(c);
       case monom of
         21,0  : a := b;
            1  : a := -b;
            2  : a := a+b;
            3  : a := a-b;
            4  : a := a*b;
            5  : a := a/b;
            6  : a := har(a,round(b));
            7  : a := exp(b*ln(a));
            8  : a := sin(b);
            9  : a:= cos(b);
            10 : a := sin(b)/cos(b);
            11 : a := arcsin(b);
            12 : if b>0 then a := arcsin(1-sqr(b))
                   else a := pi-arcsin(1-sqr(b));
            13 : a := arctan(b);
            14 : a := abs(b);
            15 : a := sqr(b);
            16 : a := sqrt(b);
            17 : a := exp(b);
            18 : a := ln(b);
            19 : a := ln(b)/ln(10);
            20 : if c>=0 then else a := b;
       end;
       wstack(a);
     end
else
 if cvladi then wstack(y[cnom])
```

```
            else wstack(mnish);
          r := r^.mim
        end;
      rstack(a);
      gam := a;
    end;//gam

end.
```

დანართი 2. პროგრამათა კომპლექსი ფუნქციონალურ გამოსახულებათა დამუშავებისთვის. საწყისი ტექსტი (C++).

```
#include <cstdlib>
#include <iostream>
#include <string>
#include <set>
#include <vector>
#include <map>
#include <sstream>
#include <cmath>

using namespace std;
template <class T>
 class Vector :public vector <T>
     {  public:
            Vector operator | (Vector y)
            {int i,N=y.size(); Vector x(*this);
             for(i=0;i<N;i++)
              x|=y[i];
              return x;
            }
            Vector operator | (T y)
            {Vector x(*this);
             x.push_back(y);
             return x;
            }
            Vector operator |= (Vector y)
            {*this=(*this)|y;
             return *this;
            }
            Vector operator |= (T y)
            {return *this=(*this)|y;
```

```
            }
            Vector SubVec(int n,int m=10000)
            {Vector x;
             for(int i=0; n+i<this->size() && i<m;i++)
             x|=(*this)[n+i];
             return x;
            }
            int Find(T x)
            {for(int i=0;i<this->size();i++)
             if((*this)[i]==x)return i;
             return -1;
            }
            int Card()
            {Vector x(*this);set <T> R(x.begin(),x.end());
             return R.size();
            }
      };
string Formulebi[]={"+AB","-AB","+*Ab*aB","/-*Ab*aBnb",
      "+**Ab^a-b1**B^abqa",
      "*gaA","s*faA","/Anga","s-Anfa","/Ao-1na","s/Ao-1na",
      "/A+1na","s/A+1na","*2*aA","/A*2oa","*paA","/Aa","/A*aqD","sA"};
string Funqciebi[]={"sin","cos","tg","ctg","arcsin","arcos",
      "arctg","arcctd","sqr","sqrt","exp","ln","lg","s"};
string mq="+-*/^";
struct Telem
{char operand;
 union {char nomer;
        double mnish;
       };
};
typedef Vector<Telem> Tgam;
Telem SetTelem(char x=2, char y=0, double z=0)
{ Telem a;
  a.operand=x;
```

```
 if(x==2)a.mnish=z;
 else if(!x)a.nomer=y;
 return a;
}
Tgam SetTgam(Telem x)
{
 Tgam ans;
 ans|=x;
 return ans;
}
void Printelem(Telem x)
{
 if(x.operand)
    if(x.operand==1)cout<<"x";
    else cout<<x.mnish;
 else if(x.nomer<6)cout<<mq[x.nomer-1];
      else cout<<Funqciebi[x.nomer-6];
 cout<<' ';
}
void PrintGam(Tgam x)
{
 int i,N=x.size();
 for(i=0;i<N;i++)
 Printelem(x[i]);
 cout<<endl;
}
Tgam a(Tgam x)
{int i,k;
 for(i=k=1;k;i++)
 if(x[i].operand)k--;
 else k+=x[i].nomer<6;
 return x.SubVec(1,i-1);
}
Tgam b(Tgam x)
```

```
{
 int i;
 i=a(x).size();
 return a(x.SubVec(i));
}
double Gamotvla(Tgam G, double x=1)
{int i,N=G.size(),n=0,m;
 double st[30],a,b; Telem g;
 for(i=N-1;i>=0;i--)
 {g=G[i];
  if(g.operand)
     {if(g.operand==1)st[n++]=x;
      else st[n++]=g.mnish;
     }
  else {a=st[--n];
        m=g.nomer;
        if(m>5)
          switch(m)
             {case 6:a=sin(a);break;
              case 7:a=cos(a);break;
              case 8:a=tan(a);break;
              case 9:a=1/tan(a);break;
              case 10:a=asin(a);break;
              case 11:a=acos(a);break;
              case 12:a=atan(a);break;
              case 13:a=3.14159265358979323846*.5-atan(a);break;
              case 14:a*=a;break;
              case 15:a=sqrt(a);break;
              case 16:a=exp(a);break;
              case 17:a=log(a);break;
              case 18:a=log10(a);break;
              case 19:a=-a;break;
              default:;
             }
```

```
        else {b=st[--n];
              switch(m)
                {case 1:a+=b;break;
                 case 2:a-=b;break;
                 case 3:a*=b;break;
                 case 4:a/=b;break;
                 case 5:a=pow(a,b);break;
                 default:;
                }
             }
       st[n++]=a;
      }
 }
 return st[0];
}
Tgam Gamartiveba(Tgam G)
{int i,N=G.size(),n=0,m;
 Tgam st[30],a,b; Telem g;
 for(i=N-1;i>=0;i--)
  {
   g=G[i];
   if(g.operand)st[n++]=SetTgam(g);
   else
    { m=g.nomer;
      a=st[--n];
      if(m>5)
       {
        if(a[0].operand==2)
          {
           a[0].mnish=Gamotvla(SetTgam(SetTelem(0,m))|a);
           st[n++]=a;
          }
        else if(!a[0].operand && a[0].nomer==19 && m==19)
           {
```

```
      a=a.SubVec(1);
      st[n++]=a;
     }
 else st[n++]=SetTgam(g)|a;
}
else
{
 b=st[--n];
 int k=n;
 if(a[0].operand==2 && b[0].operand==2)
    st[n++]=SetTgam(SetTelem(2,0,Gamotvla(SetTgam(g)|a|b)));
 else if(m==1)
        {if(a[0].operand==2 && a[0].mnish==0)
        st[n++]=b;
        else if(b[0].operand==2 && b[0].mnish==0)
        st[n++]=a;
        }
 else if(m==2)
        {if(a[0].operand==2 && a[0].mnish==0)
        st[n++]=SetTgam(SetTelem(0,19))|b;
        else if(b[0].operand==2 && b[0].mnish==0)
        st[n++]=a;
        }
 else if(m==3)
        {if(a[0].operand==2 && a[0].mnish==0)
            st[n++]=SetTgam(SetTelem(2,0,0));
        else if(a[0].operand==2 && a[0].mnish==1)
            st[n++]=b;
        else if(b[0].operand==2 && b[0].mnish==0)
            st[n++]=SetTgam(SetTelem(2,0,0));
        else if(b[0].operand==2 && b[0].mnish==1)
            st[n++]=a;
        }
 else if(m==4)
```

```
                {if(a[0].operand==2 && a[0].mnish==0)
                    st[n++]=SetTgam(SetTelem(2,0,0));
                else if(b[0].operand==2 && b[0].mnish==1)
                    st[n++]=a;
                }
        else if(m==5)
                {if(a[0].operand==2 && a[0].mnish==0)
                    st[n++]=SetTgam(SetTelem(2,0,0));
                else if(a[0].operand==2 && a[0].mnish==1)
                    st[n++]=a;
                else if(b[0].operand==2 && b[0].mnish==0)
                    st[n++]=SetTgam(SetTelem(2,0,1));
                else if(b[0].operand==2 && b[0].mnish==1)
                    st[n++]=a;
                }
        if(k==n)
            st[n++]=SetTgam(g)|a|b;
       }
     }
  }
 return st[0];
}

Tgam Dif(Tgam x)
{
 if(x[0].operand)
   {
    if(x[0].operand==1)
      {
       x[0].operand=2;
       x[0].mnish=1;
      }
    else x[0].mnish=0;
    return x;
```

```
  }
 string f(Formulebi[x[0].nomer-1]);
 Tgam ans; Telem g;
 int i,N,k;
 string mq="+-*/^";
 map<char,int>MQ;
 for(i=0;i<mq.size();i++)MQ[mq[i]]=i+1;
 for(i=0,N=f.size();i<N;i++)
  if(f[i]=='a')ans|=a(x); else
  if(f[i]=='b')ans|=b(x); else
  if(f[i]=='A')ans|=Dif(a(x)); else
  if(f[i]=='B')ans|=Dif(b(x)); else
  {
   if(MQ.count(f[i]))
      {
       g.operand=0; g.nomer=MQ[f[i]];
      }
   else if(f[i]>='f' && f[i]<='s')
       {
        g.operand=0; g.nomer=f[i]-'f'+6;
       }
   else {
         g.operand=2;
         if(f[i]=='D')g.mnish=10;
         else g.mnish=f[i]-'0';
        }
   ans|=g;
  }
  return Gamartiveba(ans);
}
int Br1(int x,int y)
{
 if(y>=6 || y==0)return 0;
 if(x>=6 || x<=2)return 0;
```

```
 if(y<=2)return 1;
 if(x==5)return 1;
 return 0;
}
int Br2(int x,int y)
{
 if(x<=1)return 0;
 if(y==0 || y>=5)return 0;
 if(y<3)return 1;
 return x>3;
}
int Prior(int x)
{
 if(x<1)return 0;
 if(x<3)return 1;
 if(x<=5)return 2+(x==5);
 return 4;
}
int IsCorrectMnish(string x)
{int mg[10][7]={{1,2,3,4,-1,-1,-1},
                {-1,2,3,4,-1,-1,-1},
                {-1,-1,-1,5,6,-1,10},
                {-1,3,3,5,6,-1,10},
                {-1,5,5,-1,-1,-1,-1},
                {-1,5,5,-1,6,-1,10},
                {7,8,9,-1,-1,-1,-1},
                {-1,8,9,-1,-1,-1,-1},
                {-1,-1,-1,-1,-1,-1,10},
                {-1,8,8,-1,-1,-1,10}
               },
 i,N=x.size(),k,g=0;
 for(i=0;i<=N && g>-1;i++,g=mg[g][k])
 if(i==N)k=6; else
 if(x[i]=='+' || x[i]=='-')k=0; else
```

```
 if(x[i]>='0' && x[i]<='9')k=1+(x[i]!='0'); else
 if(x[i]=='.')k=3; else
 if(x[i]=='e' || x[i]=='E')k=4;
 else k=5;
 if(g<0)return i-1;
 return -1;
}
string Tostring(double x)
{
 stringstream SS;
 SS<<x;
 return SS.str();
}
string Tostring(Tgam x)
{
 int i,N=x.size(),n=0;
 string ST[30]; int st[30];
 string mq="+-*/^";
 for(i=N-1;i>=0;i--)
  if(x[i].operand)
    {
     if(x[i].operand==1)ST[n]="x";
     else ST[n]=Tostring(x[i].mnish);
     st[n++]=0;
    }
  else
    {string A,B;
     int a,b,c=x[i].nomer;
     A=ST[--n];a=st[n];
     if(c>=6)
       {
        st[n]=c;
        ST[n++]=Funqciebi[c-6]+"("+A+")";
       }
```

```
    else
      {
       B=ST[--n]; b=st[n];
       if(Br1(c,a))A="("+A+")";
       if(Br2(c,b))B="("+B+")";
       ST[n]=A+mq[c-1]+B; st[n++]=c;
      }
   }
 return ST[0];
}
double Todouble(string x)
{stringstream SS(x);
 double ans;
 SS>>ans;
 return ans;
}
Tgam Togam(string x, int &res)
{
 string mq="+-*/^()",X=x;
 map<char,int>MQ;
 int i,N=x.size(),n=0,L=-1,c,k,bn=0,m=0,g=0,p=0,pr,sk;
 for(i=0;i<mq.size();i++)MQ[mq[i]]=i+1;
 Tgam ST[30],A,B; int st[30];
 map<string,int>FD;
 for(i=0;i<14;i++)
 FD[Funqciebi[i]]=i+6;
 vector<int>P;
 for(i=N;i>=0;i--)
 if(MQ.count(x[i])){
  x=x.substr(0,i)+" "+x[i]+" "+x.substr(i+1);
  P.insert(P.begin(),i+1);
 }
stringstream SS(x);
 int mg[4][9]={{0,-1,1,0,-1,3,1,-1,-1},
```

```
            {2,2,-1,-1,1,-1,-1,-1,4},
            {-1,-1,1,0,-1,3,1,-1,-1},
            {-1,-1,-1,0,-1,-1,-1,-1,-1}
           };
while(g>=0 && SS>>x)
 {
  if(x.size()==1 && MQ.count(x[0]))
   {p=P[++L];
    k=MQ[x[0]];
    if(k<6)
      {pr=Prior(k);
       sk=k/3;
       if(g==0){if(k==1); else st[m++]=19; goto L10;}
       while(m && Prior(st[m-1])>=pr)
         {c=st[--m];
          B=ST[--n];
          if(c>5)ST[n++]=SetTgam(SetTelem(0,c))|B;
          else {A=ST[--n];
                ST[n++]=SetTgam(SetTelem(0,c))|A|B;
               }
         }
       st[m++]=k;
      }
    else if(k==6){st[m++]=-1; bn++; sk=3;}
    else {if(bn--<1){A.resize(0); res=p; return A;}
          sk=4;
          while(st[--m]!=-1)
                {
                 B=ST[--n]; c=st[m];
                 if(c>5)ST[n++]=SetTgam(SetTelem(0,c))|B;
                 else {A=ST[--n];
                       ST[n++]=SetTgam(SetTelem(0,c))|A|B;
                      }
                }
```

```
        }
    }
    else
    {p++;
     if(x=="x"){ST[n++]=SetTgam(SetTelem(1)); sk=2;}
     else if(FD.count(x))
         {sk=5;
          k=FD[x];
          pr=4;
          while(n && m && Prior(st[m-1])>=pr)
           {c=st[--m];
            B=ST[--n];
            if(c>5)ST[n++]=SetTgam(SetTelem(0,c))|B;
            else {A=ST[--n];
                 ST[n++]=SetTgam(SetTelem(0,c))|A|B;
                 }
           }
         st[m++]=k;
         }
     else {sk=6;
          k=IsCorrectMnish(x);
          if(k>=0){A.resize(0); res=p+k; return A;}
          double d;
          d=Todouble(x);
          ST[n++]=SetTgam(SetTelem(2,0,d));
         }
    }

L10:  g=mg[g][sk];
 }
 if(g<0){A.resize(0); res=p; return A;}
 if(bn){A.resize(0); res=X.size(); return A;}
 while(m--)
         {
```

```
                B=ST[--n]; c=st[m];
                if(c>5)ST[n++]=SetTgam(SetTelem(0,c))|B;
                else {A=ST[--n];
                     ST[n++]=SetTgam(SetTelem(0,c))|A|B;
                    }

         }
 sk=8;
 if(g>-1)g=mg[g][sk];
 if(g==4){res=-1;
         return ST[0];
        }
 A.resize(0); res=X.size(); return A;
}
int main(int argc, char *argv[])
{int pos;

Tgam G;

//   G=Togam("(2.5-x)*3/(cos(3/x+2E3)+tg(14.3*x)*x/12.3242)-17.77",pos);

//   G=Togam("((2.5-x)*7/sin(x))+tg(((0.1234)*x-(256))-26.134+4*23.123)",pos);
//    G=Togam("((ln((x)+1e2+3)))/tg(x^2.3)",pos);
//     G=Togam("(3-2)*x+x*(2/2)+(3-3)*x+x*(5-5)",pos);
//     G=Togam("4*arctg(1)",pos);
//       G=Togam("x^1",pos);
        G=Togam("((2.14*x-(15))*(cos(x)))",pos);
 if(pos>-1)cout<<"Error in pos "<<pos<<endl;
 else {cout<<Tostring(G)<<endl;
      PrintGam(G);
      cout<<Gamotvla(G)<<endl;
      G=Gamartiveba(G);
      cout<<Tostring(G)<<endl;
      PrintGam(G);
```

```
      G=Gamartiveba(G);
      cout<<Tostring(G)<<endl;
      PrintGam(G);
      G=Dif(G);
      cout<<Tostring(G)<<endl;
      PrintGam(G);
      G=Gamartiveba(G);
      cout<<Tostring(G)<<endl;
      PrintGam(G);

     }
    system("PAUSE");
   return EXIT_SUCCESS;
}
```