ინგა აბულაძე

დაპროგრამების ენა
Turbo Pascal

„ტექნიკური უნივერსიტეტი“

საქართველოს ტექნიკური უნივერსიტეტი

ინგა აბულაძე

დაპროგრამების ენა
Turbo Pascal

დამტკიცებულია სტუ-ს
სარედაქციო-საგამომცემლო
საბჭოს მიერ

თ ბ ი ლ ი ს ი
2 0 0 7

Uაკ 681.3

დამხმარე სახელმძღვანელოში „დაპროგრამების ენა ***Turbo Pascal***“ განხილულია დაპროგრამების ენა პასკალის ძირითადი ელემენტები და სტანდარტული ფუნ-ქციები, პროგრამის ჩაწერის სტრუქტურა, ალგორითმის სახეები, არჩევის ოპერატორი; აგრეთვე განხილულია პასკალში სტრუქტურულ მონაცემთა ტიპები: ერთი და ორ განზომილებიანი მასივების, ქვეპროგრამების, ჩანაწე-რების, ფაილებისა და ტექსტური ფაილების გამოყენების ძირითადი პრინციპები.

გადმოცემულია თეორიული მასალის შესაბამისი ალგორითმები და პროგრამები ტურბო-პასკალზე. ისინი დალაგებულია თემების მიხედვით, რაც მასალის უკეთ ათვისების საშუალებას იძლევა.

განკუთვნილია ინფორმატიკის ფაკულტეტის სტუდენ-ტებისა და მაგისტრანტებისათვის. აგრეთვე, დაპროგრამე-ბის შემსწავლელთა ფართო წრისათვის.

რეცენზენტი პროფესორი **ზურაბ ბაიაშვილი**

ა ვ ტ ო რ ი ს ა გ ა ნ

დაპროგრამების ენა ***Pascal*** შექმნილია შვეიცარიელი (ციურიხის ტექნოლოგიური ინსტიტუტის) პროფესორის ნიკლაუს ვირტის მიერ 1969-71 წწ. სახელწოდება ***Pascal*** დაარქვა ცნობილი ფრანგი მათემატიკოსისა და ფილოსოფოსის ბლეზ პასკალის პატივსაცემად.

შემდეგ ამერიკული კორპორაციის „***Borland International***“ (აშშ) პრეზიდენტმა ფრენკ ბორლანდმა დაამუშავა პასკალის ახალი ვერსია სახელწოდებით – ***Turbo Pascal***. იგი წარმოადგენს მსოფლიოში ერთ-ერთ თანამედროვე მაღალი დონის დაპროგრამების ენას.

1. დ ა პ რ ო გ რ ა მ ე ბ ი ს ე ნ ა პ ა ს კ ა ლ ი

1.1. ე ნ ი ს ძ ი რ ი თ ა დ ი ე ლ ე მ ე ნ ტ ე ბ ი

ამოცანის ამოხსნის ალგორითმის დასაწერად რომელიმე ენაზე აუცილებელია ვიცოდეთ ბრძანებების ჩაწერის ელემენტარული წესები.

პასკალზე პროგრამა ფორმირდება ასოების, ციფრების და სპეციალური სიმბოლოების გამოყენებით მიღებული სინტაქსის საფუძველზე.

პასკალში გამოიყენება შემდეგი სიმბოლოები:

ა) ლათინური ანბანის 26 (მთავრული და ნუსხური) ასო;

ბ) ციფრები: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9;

გ) სპეციალური სიმბოლოები:

+ პლუსი	< ნაკლებობის ნიშანი
- მინუსი	> მეტობის ნიშანი
* ვარსკვლავი	() მრგვალი ფრჩხილები
/ დახრილი ხაზი	[] კვადრატული ფრჩხილები
= ტოლობის ნიშანი	{} ფიგურული ფრჩხილები
. წერტილი	ჰარი (ცარიელი პოზიცია)
; წერტილ-მძიმე	
: ორი წერტილი	
’ აპოსტროფი	

დ) კომბინირებული სიმბოლოები: <> – არ არის ტოლი, < = – ნაკლებია ან ტოლი, > = – მეტია ან ტოლი, : = – მინიჭების ნიშანი, .. – მნიშვნელობათა დიაპაზონი.

ე) დარეზერვებული სიტყვები:

AND და
ARRAY მასივი
BEGIN დასაწყისი
CASE არჩევა
CONST კონსტანტა
DIV გაყოფის შედეგად მიღებული მთელი რიცხვი
DO შესრულება
DOWNTO შემცირება
ELSE წინააღმდეგ შემთხვევაში
END დასასრული
FILE ფაილი
FOR თვის
FUNCTION ფუნქცია
GO TO გადასვლა
IF თუ
IN ში
LABEL ჭდე
MOD მოდული
NIL მიმთითებლის არარსებობა
NOT უარყოფა
OF დან
OR ან
PACKED ჩალაგებული
PROCEDURE პროცედურა
PROGRAM პროგრამა
RECORD ჩანაწერი
REPEAT განმეორება
SET სიმრავლე
THEN მაშინ
TO გაზრდა
TYPE ტიპი
VAR ცვლადი
WHILE სანამ
WITH დან

1.2. პასკალის სტანდარტული ფუნქციები

პასკალში გამოიყენება 20-მდე სხვადასხვა ფუნქცია, ქვემოთ მოყვანილია ზოგიერთი მათგანი:

ფუნქცია	ჩანაწერი პასკალზე	არგუმენტის ტიპი	შედეგის ტიპი
x^2	sqr(x)	მთელი ან ნამდვილი	მთელი ან ნამდვილი
$\sqrt{x}$	sqrt(x)	მთელი ან ნამდვილი	მთელი ან ნამდვილი
x^n	Exp(n*Ln(x))	მთელი ან ნამდვილი	ნამდვილი
$\sqrt[n]{x}$	Exp(1/n*Ln(x))	მთელი ან ნამდვილი	ნამდვილი
e^x	Exp(x)	მთელი ან ნამდვილი	ნამდვილი
ln x, lg x	Ln(x)	მთელი ან ნამდვილი	ნამდვილი
sin x	sin(x)	მთელი ან ნამდვილი	ნამდვილი
cos x	cos(x)	მთელი ან ნამდვილი	ნამდვილი
Arctg(x)	arctg(x)	მთელი ან ნამდვილი	ნამდვილი
$\lvert x \rvert$	abs(x)	მთელი ან ნამდვილი	მთელი ან ნამდვილი

2. ალგორითმის სახეები

დაპროგრამების ენა ***Pascal***-ში არსებობს ალგორითმის წარმოდგენის სამი სახის სტრუქტურა. ესენია: წრფივი სტრუქტურის, განშტოებადი სტრუქტრისა და ციკლური სტრუქტურის ალგორითმები. ყველა რთული ამოცანის გადაწყვეტა შესაძლებელია ზემოთ ჩამოთვლილი სტრუქტურების ალგორითმების საშუალებით.

2.1. წრფივი სტრუქტურის ალგორითმები

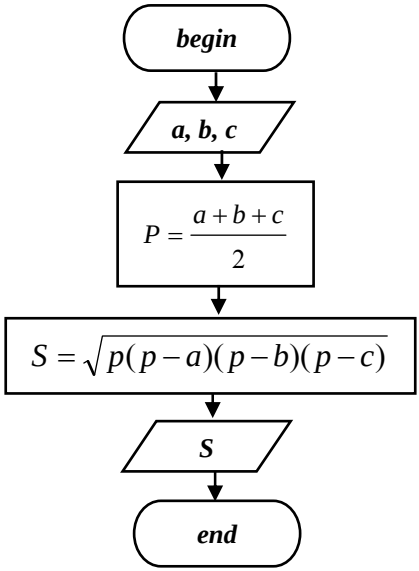
წრფივი სტრუქტურის ალგორითმი ეწოდება ალგორითმს, სადაც ბლოკები სრულდება ერთმანეთის მიმდევრობით სქემაში მოცემული რიგის მიხედვით.

მაგალითი 2.1. დაწერეთ პროგრამა, რომელიც გამოთვლის სამკუთხედის ფართობს ჰერონის ფორმულით.

ჰერონის ფორმულის თანახმად:

$$S = \sqrt{p(p-a)(p-b)(p-c)},$$

სადაც ***p*** ნახევარპერიმეტრია და გამოითვლება შემდეგი ფორმულით: $p = (a+b+c)/2$. შესაბამის ალგორითმს და პროგრამას გააჩნია სახე:




```
Program Geron;
{გამოვთვალოთ სამკუთხედის ფართობი
  ჰერონის ფორმულით}
Var
   a,b,c: integer;
   P , S: Real;
begin
   Write ('Enter a=');
   Readln (a);
   Write ('Enter b=');
   Readln (b);
   Write ('Enter c=');
   Readln (b);
   P: = (a+b+c)/2;
   S: = Sqrt (p* (p-a)* (p-b) *(p-c));
  Writeln ('S  =' ,S);
 Readln
End.
```

2.2. განშტოებადი სტრუქტურის ალგორითმები

პრაქტიკაში იშვიათად გვხვდება ისეთი ამოცანები, რომელთა გადაწყვეტა შესაძლებელია წრფივი სტრუქტურის საშუალებით. უმრავლეს შემთხვევაში საჭიროა პირობის მიხედვით ხან ერთი, ხან კი მეორე და ა.შ. ფორმულის მიხედვით გამოთვლა. ზემოხსენებულიდან გამომდინარეობს, რომ „რაღაც“ ლოგიკურ პირობაზე დამოკიდებულებით გამოთვლითი პროცესი მიმდინარეობს სხვადასხვა შტოს მიხედვით.

ალგორითმის სტრუქტურას, რომლის მიხედვით გამოთვლა მიმდინარეობს ხან ერთი, ხან კი მეორე ფორმულის მიხედვით ***ალგორითმის განშტოებადი სტრუქტურა*** ეწოდება.

ასეთი სახის სტრუქტურის ალგორითმების შემთხვევაში შტოების რიცხვი არ არის აუცილებელი ორის ტოლი იყოს.

პირობით ოპერატორს აქვს შემდეგი ზოგადი სახე:

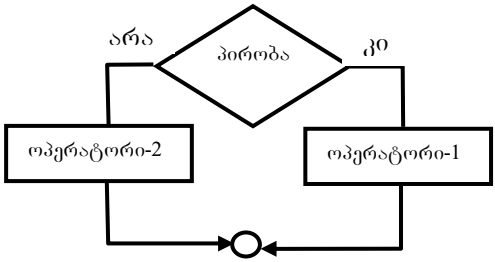
if **<პირობა>** **then** **<ოპერატორი-1>**
else **<ოპერატორი-2>**

ან

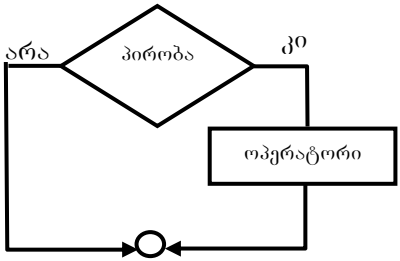
if **<პირობა>** ***then*** **<ოპერატორი>**

სადაც ***if, then*** და ***else*** დარეზერვებული სიტყვებია. თუ პირობა ჭეშმარიტია, მაშინ შეასრულე <**ოპერატორი–1**>, თუ არა (წინააღმდეგ შემთხვევაში) შესრულდება <**ოპერატორი–2**>.

ამ ოპერატორის ალგორითმს აქვს ქვემოთ ჩამოთვლილთაგან ერთ-ერთი სახე (ა, ბ):



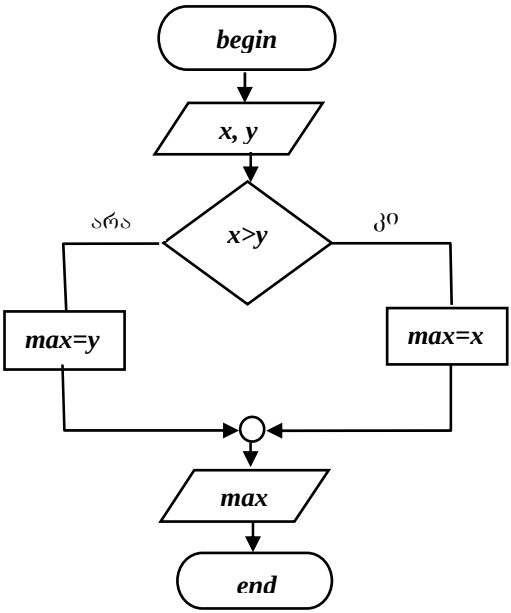
ა)



ბ)

მაგალითი 2.2. დაწერეთ პროგრამა, რომელიც გამოთვლის ორ რიცხვს შორის მაქსიმალურ მნიშვნელობას.

მაგალითის შესაბამის ალგორითმს და პროგრამას გააჩნია შემდეგი სახე:




```
Program maximum;
{ გამოვთვალოთ ორ რიცხვს შორის მაქსი-
  მალური მნიშვნელობა }
Var
   x, y, max: Real;
begin
   Readln (x, y);
   if  x>y then max: = x  else  max := y;
   Writeln (max);
   Readln
End.
```

2.3. ციკლური სტრუქტურის ალგორითმები

ზოგიერთი ამოცანის გადაწყვეტისათვის აუცილებელია ერთი და იგივე მათემატიკური ფორმულის მრავალჯერადი გამოყენება. ასეთი მრავალჯერადი განმეორებითი პროცესის ალგორითმებს ***ციკლური სტრუქტურის ალგორითმები*** ეწოდება.

ციკლური სტრუქტურის განსახორციელებლად არსებობს რამდენიმე ციკლის ოპერატორი. ქვემოთ განვიხილავთ თითოეულ მათგანს.

2.3.1. ციკლის ოპერატორი ციკლის განმეორების წინასწარ ცნობილი რიცხვით

ციკლის ოპერატორს გააჩნია შემდეგი ზოგადი სახე:

for i: = m_1 to m_2 do <ოპერატორი>

სადაც ***for, to*** და ***do*** დარეზერვებული სიტყვებია;

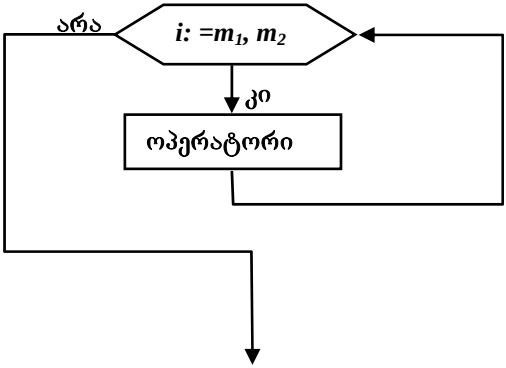
i - ციკლის პარამეტრი; ***m_1*** – ციკლის პარამეტრის საწყისი მნიშვნელობა; ***m_2*** – ციკლის პარამეტრის საბოლოო მნიშვნელობა; **<ოპერატორი>** – პასკალის ნებისმიერი ოპერატორი.

ასეთი სტრუქტურის შემთხვევაში ციკლის განსახორციელებლად აუცილებელია:

1) ***$m_1 \le m_2$***

2) ***i: = i+1***

ამ ოპერატორის ალგორითმს აქვს შემდეგი სახე:

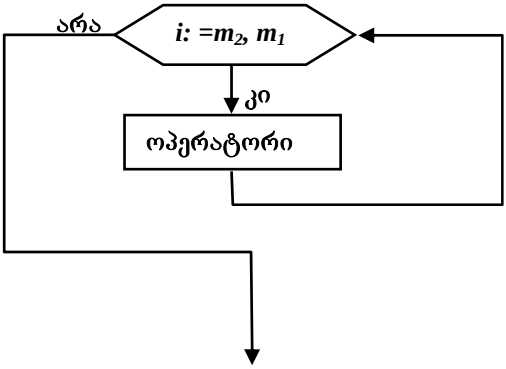


არსებობს ***for*** ციკლის ოპერატორის მეორე სახის ჩანაწერი:

for i = m2 downto m1 do <ოპერატორი>

1) $m_2 \leq m_1$
2) ***i:= i-1.***

ამ ოპერატორის ალგორითმს აქვს შემდეგი სახე:



2.3.2. *ციკლის ოპერატორი წინა პირობით*

ხშირად გვხვდება ისეთი ციკლური გამოთვლითი პროცესი, როდესაც ამოცანის გადასაწყვეტად მოცემულია „რაღაც“ პირობა. ასეთი ციკლური სტრუქტურის განსახორციელებლად ***Pascal***-ში არსებობს ორი სახის ოპერატორი: ციკლის ოპერატორი წინა პირობით და ციკლის ოპერატორი ბოლო პირობით.

ციკლის ოპერატორს წინა პირობით გააჩნია შემდეგი ზოგადი სახე:

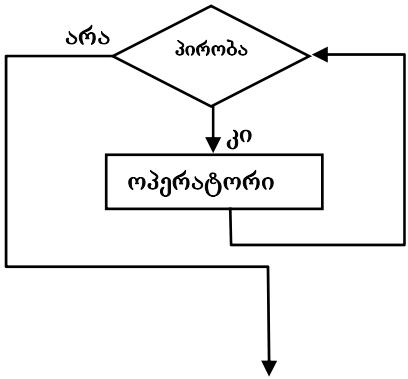
While **<პირობა>** ***do*** **<ოპერატორი>**

სადაც ***While*** და ***do*** დარეზერვებული სიტყვებია
(სანამ პირობა ჭეშმარიტია, შეასრულე);

<პირობა> - ლოგიკური გამოსახულება;

<ოპერატორი> – პასკალის ნებისმიერი ოპერატორი.

ამ ოპერატორის ალგორითმს აქვს შემდეგი სახე:



ასეთი სახის ციკლის ოპერატორის შემთხვევაში ციკლი ბრუნავს მანამ, სანამ პირობა ჭეშმარიტია (ლოგიკური გამოსახულება იღებს ***true*** მნიშვნელობას); მცდარი პირობის შემთხვევაში (ლოგიკური გამოსახულება იღებს ***false*** მნიშვნელობას) კი – ციკლიდან გამოვდივართ და სრულდება ***While***-ს შემდეგ მოთავსებული ოპერატორი.

2.3.3. *ციკლის ოპერატორი ბოლო პირობით*

ციკლის ოპერატორს ბოლო პირობით გააჩნია შემდეგი ზოგადი სახე:

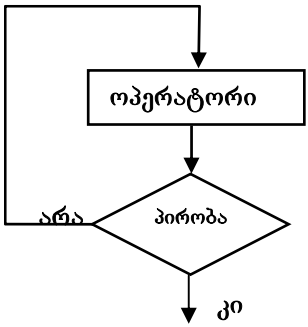
Repeat **<ოპერატორი>** ***Until*** **<პირობა>**

სადაც ***Repeat*** და ***Until*** დარეზერვებული სიტყვებია (გაიმეორე, სანამ პირობა მცდარია);

<ოპერატორი> – პასკალის ნებისმიერი ოპერატორები;

<პირობა> – ლოგიკური გამოსახულება.

ამ ოპერატორის ალგორითმს აქვს შემდეგი სახე:



ასეთი სახის ციკლის ოპერატორის შემთხვევაში ციკლი ბრუნავს მანამ, სანამ პირობა მცდარია (ლოგიკური გამოსახულება იღებს ***false*** მნიშვნელობას); ჭეშმარიტი პირობის შემთხვევაში (ლოგიკური გამოსახულება იღებს ***true*** მნიშვნელობას) კი – ციკლიდან გამოვდივართ და სრულდება ***until***-ს შემდეგ მოთავსებული ოპერატორი.

3. პ რ ო გ რ ა მ ი ს ს ტ რ უ ქ ტ უ რ ა

დაპროგრამების ენა ***Pascal***-ში პროგრამა შედგება 3 ნაწილისაგან: პროგრამის სათაური, აღწერითი ნაწილი და ოპერატორების განყოფილება. პროგრამის სათაური იწყება დარეზერვებული სიტყვით ***Program***, შემდეგ მოდის პროგრამის სახელწოდება, რომელიც აუცილებლად უნდა იწყებოდეს ლათინური ანბანის ასოთი; შემდეგ კი შესაძლებელია არაბული ციფრებისა და „–“ სიმბოლოს გამოყენება.

აღწერითი ნაწილი მკაცრად განსაზღვრულია და აქვს შემდეგი განყოფილებები: ***Label, Const, Type, Var, Procedure*** ან ***Function***.

ოპერატორების განყოფილებაში ***begin*** და ***end*** დარეზერვებულ სიტყვებს შორის მოთავსებულია ოპერატორები, რომელთაც ეწოდებათ პროგრამის ტანი.

Pascal-ზე პროგრამის დაწერის სტრუქტურას გააჩნია შემდეგი სახე:

Program **<პროგრამის სათაური>**;
Label
< ჭდეების აღწერა>;

```
Const
        <კონსტანტების აღწერა>;
Type
        <ტიპების აღწერა>;
Var
        <ცვლადების აღწერა>;

Procedure
        <პროცედურის აღწერა >;
 Function
        <ფუნქციის აღწერა>;
Begin
        <ოპერატორები>;
end.
```

3.1. ჩამოთვლითი და შემოსაზღვრული ტიპები

დაპროგრამების ენა ***Pascal***-ში პროგრამისტს საშუალება აქვს შექმნას ახალი ტიპები. მათი ფორმირებისათვის გამოიყენება დარეზერვებული სიტყვა ***TYPE***. ჩვენ განვიხილავთ ჩამოთვლით და შემოსაზღვრულ (ინტერვალურ) ტიპებს.

ჩამოთვლით ტიპს გააჩნია შემდეგი ზოგადი სახე:

```
Type
      სახელი = (მნიშვ.1, მნიშვ.2, ... ,მნიშვ.n
      );
```

ამასთან მნიშვნელობები შეიძლება იყოს მხოლოდ სახელები.

მაგალითად:

```
Type
    Day=( Monday, Tuesday, Wednesday, Thursday,
          Friday,  Saturday,  Sunday);
    Color = ( Red, Blue, Green, Rose );
Var
  WeekDay:Day;
  C:Color;
```

პროგრამის ტანში **WeekDay** ცვლადმა შეიძლება მიიღოს **Day** ტიპში მითითებული შვიდი მნიშვნელო-ბიდან მხოლოდ ერთი.

ერთ პროგრამაში შეუთავსებელია შემდეგი ორი ტიპი:

```
TYPE
    T1= ( One, Two, Three );
    T2= (Three, Four, Five);
```

რადგან დაუშვებელია ორ სხვადასხვა ტიპში ერთი და იგივე მნიშვნელობის არსებობა.

WeekDay ცვლადი აღწერილია როგორც **Day** ცვლადი, ხოლო **C** ცვლადი — როგორც **Color** ცვლადი. ამიტომ შესაძლებელია ჩავწეროთ შემდეგი მინიჭების ოპერატორები:

```
WeekDay:= Monday;
C:= Rose;
```

მინიჭების ოპერატორში მარცხნივ და მარჯვნივ მდგომი ცვლადები ერთნაირი ტიპის უნდა იყოს.

ჩამოთვლით ტიპებს შორის არსებობს თანა-ფარდობა — წინა მნიშვნელობა ნაკლებია მომდე-ვნო მნიშვნელობაზე, მაგალითად

Monday < Tuesday ან Green < Rose.

ჩამოთვლით ტიპებთან სამუშაოდ გამოიყენება შემდეგი სტანდარტული ფუნქციები:

ORD (x) — ფუნქცია განსაზღვრავს ***x***–ის რიგით ნომერს (ჩამოთვლითი ტიპის რიგის ათვლა წარმო-ებს 0-დან).
მაგალითად: ***ORD(Blue)* = 1** და ***ORD(C)* = 3.**

PRED(X) — ფუნქცია გვაძლევს არგუმენტის წინა მნიშვნელობას.

PRED(Rose) = Green,
PRED(WeekDay) = Sunday.

SUCC(x) – ფუნქცია გვაძლევს მომდევნო მნიშვნელობას.

SUCC(Rose) = Red,
SUCC(Tuesday) = Wednesday.

Pascal-ში შეუძლებელია შეტანისა და გამო-ტანის ოპერატორებით გარე მოწყობილობებზე ჩამო-თვლითი ტიპების შეტანა-გამოტანა.

შემოსაზღვრული ტიპს აქვს შემდეგი ზოგადი სახე:

```
TYPE
სახელი = სიდიდე1 .. სიდიდე 2;
```

მაგალითად:

```
TYPE
    Day=( Monday, Tuesday, Wednesday,
          Thursday, Friday,  Saturday,
          Sunday);
Var
   i, j, k:10..100;
   Symbol:'A' .. 'Z';
   WorkDay=Monday..Friday;
```

i, j, k ტიპის ცვლადებს შეუძლიათ მიიღონ 10-დან 100-მდე მთელი ტიპის მნიშვნელობები, ***Symbol*** ტიპის ცვლადს — ლათინური ანბანის

ასოების მნიშვნელობები, ***WorkDay*** ტიპის ცვლადებს კი — ***Monday***-დან ***Friday***-მდე მნიშვნელობები, რომლებიც განსაზღვრულია ***Day*** ტიპში.

შემოსაზღვრულ ტიპში თავდაპირველად მიეთი-თება ქვედა საზღვარი შემდეგ კი ზედა. შეცდომა იქნება თუ შემოსაზღვრული ტიპის ცვლადს მივა-ნიჭებთ იმ მნიშვნელობას, რომელიც მითითებულ დიაპაზონში არ შედის.

ტიპს, რომლისგანაც აირჩევა ინტერვალი, ეწოდება ბაზური ტიპი. შემოსაზღვრული ტიპის ცვლადზე შესაძლებელია ჩატარდეს ყველა ის ოპერაცია, რომლის ჩატარებაც შესაძლებელია მის შესაბამის ბაზურ ტიპზე.

4. არჩევის ოპერატორი

CASE ოპერატორის გამოყენებით შესაძლებელია რამდენიმე ალტერნატივიდან ერთ-ერთის ამორჩევა. არჩევის ოპერატორს გააჩნია შემდეგი ზოგა-დი სახე:

```
CASE <გამოსახულება> OF
   <კონსტანტა-1>: <ოპერატორი-1>;
   <კონსტანტა-2>: <ოპერატორი-2>;
                  .
                  .
                  .
  <კონსტანტა-n>   : <ოპერატორი-n>
END;
```

ან

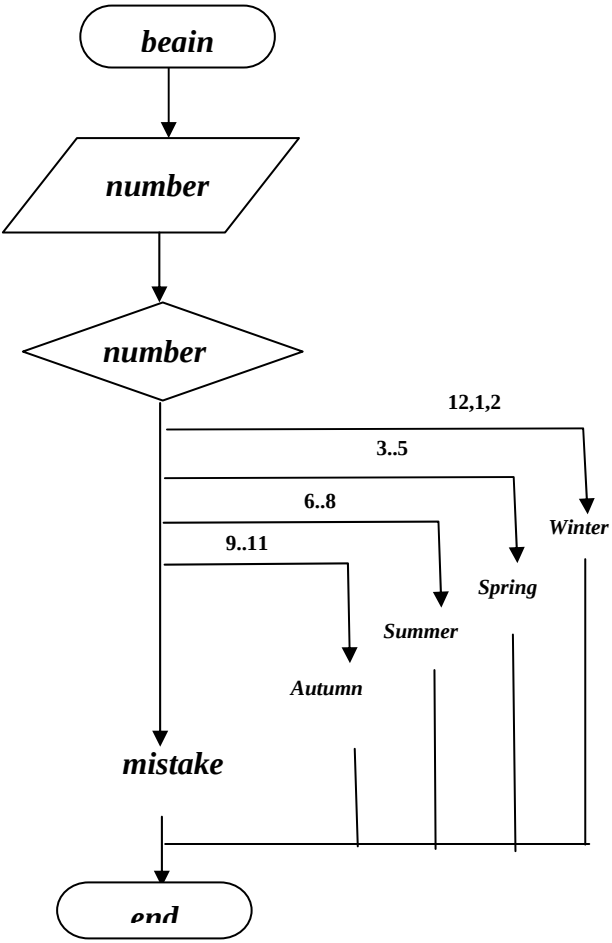
CASE **<გამოსახულება>** ***OF***
<კონსტანტა-1>: <ოპერატორი-1>;
<კონსტანტა-2>: <ოპერატორი-2>;
.
.
.
<კონსტანტა-*n*> : <ოპერატორი-*n*>;
ELSE
<ოპერატორი>
END;

გამოსახულებას, რომელიც ***CASE*** (შემთხვევა, ვარიანტი) და ***OF*** (დან) დარეზერვებულ სიტყვებს შორისაა მოთავსებული გადამრთველი (სელექტორი) ეწოდება.

კონსტანტა უნდა იყოს ისეთივე ტიპის, როგორი ტიპისაცაა გამოსახულება. გამოსახულების მნი-შვნელობა შეიძლება იყოს ნებისმიერი ტიპის, გარდა ნამდვილი ტიპისა. ერთი და იგივე კონსტანტა არ შეიძლება ჰქონდეს ორ სხვადასხვა ალტერნატივას.

არჩევის ოპერატორის შესრულების წესი განისაზღვრება გამოსახულების (სელექტორის) მნიშვნელობის მიხედვით, რომელიც უნდა დაემთხვეს კონსტანტის მნიშვნელობას რომ შესრულდეს შესაბამისი ალტერნატივა (ოპერატორი). თუ სელე-ქტორის მნიშვნელობა არ დაემთხვა ალტერნატივას (ოპერატორის არც ერთ ჭდეს), მაშინ სრულდება ოპერატორი, რომელიც მოსდევს ***ELSE*** დარე-ზერვებულ სიტყვას. თუ ***ELSE*** სიტყვა მითითებული არ არის, მაშინ მართვა გადაეცემა ოპერატორს, რომელიც მოსდევს ***CASE*** დარეზერვებულ სიტყვას.

მაგალითი 4.1. დაწერეთ პროგრამა, რომელიც მონიტორზე გამოიტანს წელიწადის დროს შეტანილი თვის რიგითი ნომრის მიხედვით.



```
Program   Season_1;
{ დავბეჭდოთ წელიწადის დრო თვის შესაბამისი
რიგითი ნომრის მიხედვით }
Var
   number : integer;
   s : string;
 begin
    Writeln ('Enter number = =>');
    Readln  (number);
    Case      number of
        12, 1, 2 :   S : =  'Winter ';
          3, 4, 5 :   S : = ' Spring';
          6, 7, 8 :   S : = ' Summer';
       9, 10, 11: S : = ' Autumn'
     end;  {case}
     Writeln ('S= ', S);
    Readln
 end.
```

ამ ამოცანის გადასაწყვეტად პროგრამაში ***Case*** ოპერატორის გამოყენებისას შეიძლება მივუთითოთ თვეების დიაპაზონი, რის შემდეგ პროგრამა მიიღებს შემდეგ სახეს:

```
Program  Season_2;
{ დავბეჭდოთ წელიწადის დრო თვის შესაბამისი
რიგითი ნომრის მიხედვით }
Var
   number : integer;
begin
    Writeln ('Enter number = =>');
    Readln  (number);
```

```
    Case     number  of
        12, 1, 2 :   WriteLn('Winter ');
           3.. 5 :   WriteLn(' Spring');
           6.. 8 :   WriteLn(' Summer)';
          9.. 10:  WriteLn(' Autumn')
      end; { Case }
Readln
end.
```

5. სტრუქტურულ მონაცემთა ტიპები

დაპროგრამების ენა ***Pascal***-ში არსებობს შემდე-გი სახის ტიპები: მასივი, სიმრავლე, ჩანაწერი და ფაილი. სტრუქტურული ტიპის ცვლადი ანუ მუდმივა ყოველთვის შეიცავს რამდენიმე კომპონენტს (ელე-მენტს).

5.1. ერთგანზომილებიანი მასივი

ერთგვაროვანი ელემენტების (რიცხვების, სიმბოლოების, სტრიქონების და ა.შ.) ერთობლიობას ეწოდება ***მასივი.*** მასივის ელემენტებს აქვთ საერთო სახელწოდება (იდენტიფიკატორი) და განსხვავდებიან ინდექსით. არსებობს მთელი, ნამდვილი, სიმბოლურ და სხვა სახის მასივები. მასივის ელემენტი შეიძლება იყოს ნებისმიერი ტიპის მონაცემი. მასივის ელემენტების რიცხვი განისაზღვრება მისი აღწერის დროს და პროგრამის შესრულების პროცესში არ იცვლება.

მასივის ელემენტებთან მიმართვა ხორციელდება ინდექსის საშუალებით. ინდექსის ტიპი შეიძლება იყოს ნებისმიერი სკალარული ტიპის (ყველაზე მეტად გამოიყენება მთელი, შემოსა-ზღვრული და ჩამოთვლითი) მონაცემი, გარდა ნამდვილისა. ერთგანზომილებიან მასივს აქვს ერთი ინდექსი, ორგანზომილებიან მასივს — ორი და ა.შ. ორგანზომილებიან მასივს ეწოდება მატრიცა.

ერთგანზომილებიანი მასივის მაგალითებია: $a_1, a_2, \ldots, a_{25}$ ან ***A*(25),** მასივის წევრები ***Pascal***-ში აღინიშნება ასე: ***A*[1], *A*[2], … ,*A*[25]**. ზოგადი ***i***-ური ელემენ-ტი აღინიშნება ***A*[i]**-ით.

ორგანზომილებიანი მასივის მაგალითია:

$$B_{ij} = \begin{vmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{vmatrix}$$

i-ური სტრიქონისა და ***j***-ური სვეტის გადაკვეთაზე მყოფი ელემენტი შემდეგნაირად აღინიშნება ***B*[i, j].**

ინდექსი წარმოადგენს მასივის ელემენტის მა-ჩვენებელს, იგი მოთავსებულია სახელის შემდეგ კვადრატულ ფრჩხილებში.

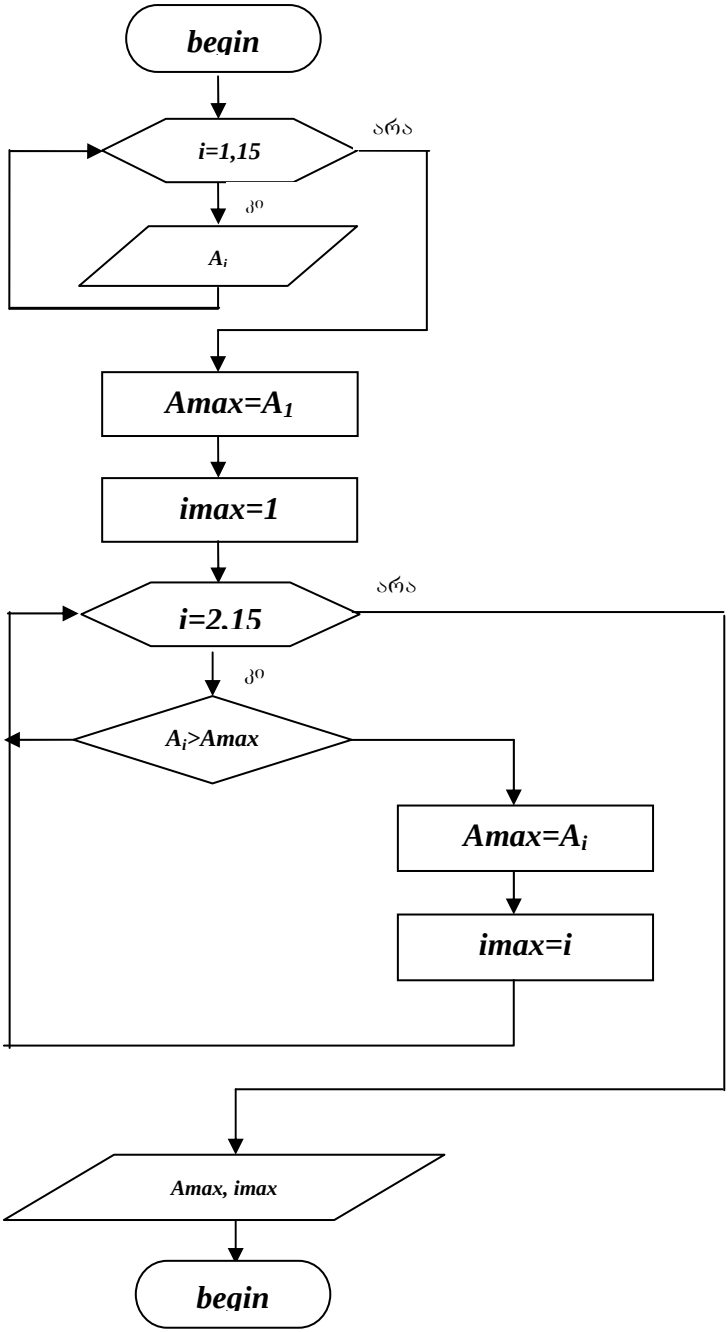
თუ პროგრამაში გამოყენებული გვაქვს მასივი, ისევე როგორც ყველა ცვლადი, აღწერით ნაწილში ისიც უნდა აღიწეროს. ერთგანზომლებიანი მასივის აღწერა პროგრამაში შესაძლებელია შემდეგნაირად: მაგალითად:

```
VAR
   A: ARRAY[1..25] OF REAL;
```

მასივის გამოცხადების დროს გამოიყენება დარეზერვებული სიტყვები ***ARRAY*** და ***OF*** (გან).

ARRAY-ის შემდეგ კვადრატულ ფრჩხილებში მოთავსებულია დიაპაზონი, რომლის საშუალებითაც კომპილატორი განსაზღვრავს მასივის ელემენტების რაოდენობას, ხოლო ***OF*** დარეზერვებული სიტყვის შემდეგ მითი-თებულია მასივში შემავალი ელემენტების ტიპი. პროგრამაში მასივის ელემენტების ინდექსი არ უნდა გასცდეს მის დიაპაზონს.

მაგალითი 5.1. გამოთვალეთ **A**(15) მასივის უდიდესი ელემენტი და მისი რიგითი ნომერი.


beqin
i=1,15
არა
კი
Ai
Amax=A1
imax=1
i=2.15
არა
კი
Ai>Amax
Amax=Ai
imax=i
Amax, imax
beqin

```
PROGRAM  MAXSIMUM;
VAR
    A: ARRAY [1..15] OF REAL;
   AMAX: REAL;
   i, iMAX : INTEGER;
BEGIN
    FOR i:=1 TO 15 DO
       READ (A[i]);    (* მასივის ელემენტების
შეტანა *)
    AMAX:=A[1];
    iMAX:=1;
    FOR i:=2 TO 15 DO
       IF A[i]>AMAX   THEN
                          BEGIN
                             AMAX:=A[i];
                            iMAX:=i
                         END;
    WRITELN ('MAX=',  MAX);
    WRITELN  ('iMAX=',  iMAX);
    READLN
END.
```

5. 2. რ თ უ ლ ი ს ტ რ უ ქ ტ უ რ ი ს ც ი კ ლ ე ბ ი

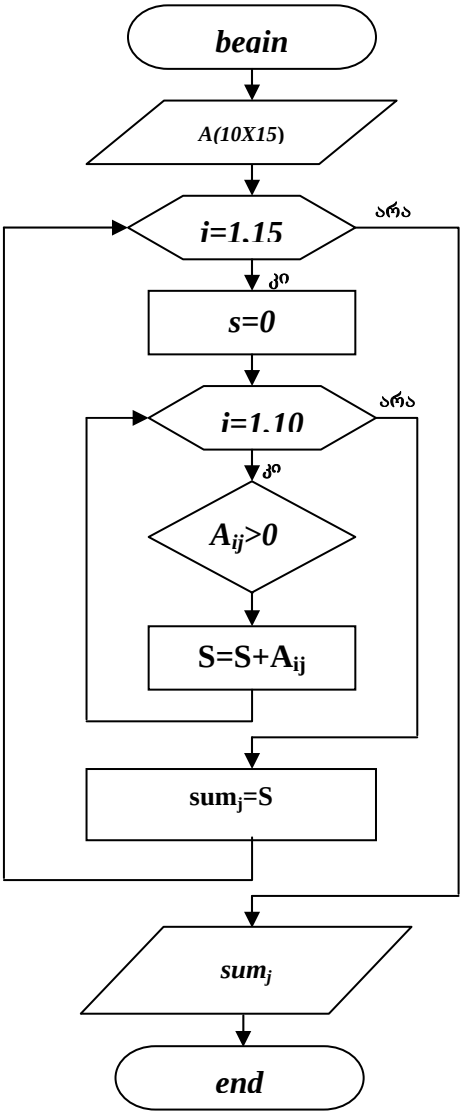
თუ ციკლი (გარე ციკლი) შეიცავს ერთ ან რამდენიმე ციკლს (შიდა ციკლი) ამ შემთხვევაში გვაქვს ***რთული სტრუქტურის ციკლი*** ანუ ციკლში ციკლი. გარე და შიდა ციკლების პარამეტრები სხვადასხვაა და მათი ცვლილება არ ხდება ერთდროულად. ამ დროს ციკლი მუშაობს შემდეგნაირად: გარე ციკლის პარამეტრის

თითოეული მნიშვნელობისათვის შიდა ციკლის პარამეტრი იღებს ყველა შესაძლო მნიშვნელობას.

მაგალითი 5.2. გამოვთვალოთ **A(10X15)** მატრიცის თითოეული სვეტის დადებითი ელემენტების ჯამი.

რთული (ჩაწყობილი) ციკლების შემთხვევაში დიდი მნიშვნელობა ენიჭება შიდა და გარე ციკლების სწორად განსაზღვრას. ჩვენი მაგალითის შემთხვევაში გარე ციკლი განსაზღვრავს სვეტის ინდექსის ცვლილებას, ხოლო შიდა ციკლი სტრიქონის ინდექსის ცვლილებას. თითოეული სვეტის დადებითი ელემენტების გამოთვლისათვის საჭიროა ციკლის ორგანიზება, რომელიც გადა-არჩევს ამ სვეტის ყველა ელემენტს. შიდა ციკლში პარამეტრი იქნება ***i*** სტრიქონის ნომერი. ამ ციკლის წინ ჯამს უნდა მივანიჭოთ საწყისი მნიშვნელობა ***S*** **= 0**. ამ მოქმედების განმეორება საჭიროა გარე ციკლში, სადაც იცვლება ***j*** სვეტის ინდექსი და შესაბამისად მივიღებთ 15 ჯამს.

ამ მაგალითის შესაბამის ალგორითმს და პრო-გრამას აქვთ შემდეგი სახე:


beqin
A(10X15)
i=1.15
არა
კი
s=0
i=1.10
არა
კი
$A_{ij}>0$
$S=S+A_{ij}$
$sum_j=S$
sum_j
end

```
Program Summa;
(* მატრიცის თითოეული სვეტის დადებითი
ელემენტების ჯამის გამოთვლა *)
var
   A:array[1..10,1..15] of Real;
   sum:array[1..15] of Real;
   s:real;
   i, j:Integer;
begin {Summa}
Write('Enter a value of array A(10X15) of
         reals');
   for i:=1 to 10 do
     for j:=1 to15 do
       read(A[I, j]);
   for j:=1 to 15 do
     begin
       s:=0; (* თითოელი სვეტის  დადები-
თი ელემენტების ჯამი *)
       for i:=1 to 10 do
       if A[i, j]>0 then s:=s+A[i,j];
        sum[j]:=s
     end;
   for j:=1 to 15 do
     write(sum[j]:2:5);
  ReadLn
end. {Summa }
```

6. ქვეპროგრამების გამოყენება

ქვეპროგრამა წარმოადგენს პროგრამის დამოუკიდებელ ნაწილს (მოდულს). თითოეული ქვეპროგრამა აღიწერება ერთხელ და მისი გამოძახება შესაძლებელია მრავალჯერ. დაპროგრამების ენა ***Pascal***-ში არსებობს ორი ტიპის ქვეპროგრამა: ფუნქცია და პროცედურა, რომლებიც აღიწერებიან პროგრამის აღწერით ნაწილში.

ქვეპროგრამების გამოყენება რეკომენდებულია პროგრამის მოცულობის შესამცირებლად, მისი გამოყენებით პროგრამა უფრო თვალსაჩინოდ გამოიყურება და ადვილი გასაგებია. ამასთან, პროგრამაში ქვეპროგრამის გამოყენება ამცირებს შე-ცდომების დაშვების ალბათობას და აადვილებს პროგრამის გამართვის პროცესს.

ქვეპროგრამა, ისევე როგორც პროგრამა, შედგება სათაურის, აღწერითი ნაწილისა (ჭდეები, კონსტანტები, ტიპები, ცვლადები; აგრეთვე დამატებითი ფუნქციის ან პროცედურისაგან, რომელიც მოცემული ქვეპროგრამისათვის წარმოადგენს ლოკა-ლურს) და ქვეპროგრამის ტანისაგან. ქვეპროგრამის სათაურში ***FUNCTION*** ან ***PROCEDURE*** დარეზერვებული სიტყვის შემდეგ მიეთითება ქვეპროგრამის სახელი, ხოლო მრგვალ ფრჩხილებში – ფორმალური პარა-მეტრების სია მათი ტიპების მითითებით. პროცე-დურისაგან განსხვავებით ფუნქციის სათაურში მი-ეთითება შედეგის ტიპი.

ქვეპროგრამაში მოქმედებები მიმდინარეობს ფორმალური პარამეტრების გამოყენებით. ქვეპროგრამასთან მიმართვას ახორციელებს ფაქტიური პა-რამეტრები, რომლებიც უნდა დაემთხვეს ფორმა-ლურს რიცხვით, ტიპით და პოზიციით (მათი ადგილ-მდებარეობით).

6.1. ქ ვ ე პ რ ო გ რ ა მ ა პ რ ო ც ე დ უ რ ა

პროგრამაში პროცედურა აღიწერება აღწერით ნაწილში. ნებისმიერი პროცედურა პროგრამის ანა-ლოგიურად შედგება სათაურის, აღწერითი ნაწილი-სა და პროცედურის ტანისაგან.

პროცედურის სათაურს აქვს შემდეგი ზოგადი სახე:

PROCEDURE **<სახელი> (<პარამეტრების სია>);**

სადაც ***PROCEDURE*** დარეზერვებული სიტყვაა; **სახელი** – პროცედურის სახელი; **პარამეტრების სია** – საწყისი მონაცემებისა და პროცედურის შედეგების სია მათი ტიპების მითითებით.

პარამეტრების სიაში ჩამოთვლილ პარამეტრებს ეწოდებათ ***ფორმალური პარამეტრები***.

აგრეთვე, შესაძლებელია პროცედურის აღწერა პარამეტრების გარეშე:

PROCEDURE **<სახელი>;**

აღწერით ნაწილში თავდაპირველად აღიწერება ჭდე, შემდეგ კონსტანტა, ტიპი, ცვლადი და პროცედურა ან ფუნქცია.

პროცედურის ტანი აღიწერება ***begin*** და ***end*** დარეზერვებულ სიტყვებს შორის და მთავრდება წერტილ-მძიმით.

მაგალითი 6.1. გააფორმეთ $Y=X^n$ გამოსახულების ალგორითმი პროცედურის სახით.

```
PROCEDURE STEP1 (N: INTEGER; X: REAL; VAR
Y:REAL);
 VAR
    i: INTEGER;
BEGIN
    Y:=1;
    FOR  i:=1  TO N DO
    Y: =Y*X ;
END;
```

STEP1 პროცედურის სათაურში ჩამოთვლილია ***N*** და ***X*** საწყისი ცვლადები და ***Y*** პარამეტრი – პროცე-დურის შესრულების შედეგი. აგრეთვე, მითითე-ბულია ყველა ფორმალური პარამეტრის ტიპი.

პროცედურის აღწერით ნაწილში განსაზღვრუ-ლია ***i*** პარამეტრის ტიპი, რომლის გამოყენებას აზრი აქვს მხოლოდ ქვეპროგრამის შიგნით და ეწოდება ***ლოკალური პარამეტრი*** (ძირითად პროგრამაში ლო-კალური პარამეტრი კარგავს თავის მნიშვნელობას).

მაგალითი 6.2. გააფორმეთ $Y=X^n$ გამოსახულების ალგორითმი პროცედურის სახით პარამეტრების გარეშე.

```
PROCEDURE STEP2;
 VAR
    i: INTEGER;
BEGIN
    Y:=1;
    FOR  i:=1  TO N DO
    Y: =Y*X ;
END;
```

ამ შემთხვევაში პროცედურა ***STEP2*** არ შეიცავს ფორმალური პარამეტრების სიას. პროცედურაში აღიწერება მხოლოდ ***i*** ლოკალური პარამეტრიც; ხო-ლო ***X, Y*** და ***N*** პარამეტრები აღიწერება ძირითად პრო-გრამაში და მათ ეწოდებათ ***გლობალური პარამეტ-რები.*** გლობალური პარამეტრების გამოყენებას მნი-შვნელობა აქვს როგორც ძირითად პროგრამაში, ასე-ვე ქვეპროგრამაში.

მაგალითი 6.3. დაწერეთ პროგრამა, რომელიც გამოთვლის $z=\dfrac{x^{k_1}x^{k_2}}{s_1+s_2}$ ფუნქციის მნიშვნელობას.

სადაც ***S_1*** *და* ***K_1*** *–* ***$A(A_1, A_2, A_3, \ldots, A_{50})$*** მასივის და-დებითი ელემენტების ჯამი და რაოდენობაა, ხოლო ***S_2*** *და* ***K_2*** ***$B(B_1, B_2, B_3, \ldots, B_{30})$*** მასივის დადებითი ელემენტების ჯამი და რაოდენობა.

მასივში დადებითი ელემენტების ჯამისა და რაოდენობის გამოსათვლელად გამოვიყენოთ პროცე-დურა.

პროცედურაში ***SUMMA*** სახელით გამოვთვალოთ ***S*** – ***MAS*** მასივის დადებითი

ელემენტების ჯამი და ***K*** – ამ მასივის დადებითი ელემენტების რაოდენობა.

პროგრამას აქვს შემდეგი სახე:

```
PROGRAM   SUM;
CONST     RAZM = 100;
TYPE      INDEX  = 1..RAZM;
          VECTOR = ARRAY[INDEX] OF REAL;
VAR   I, N, M, K1, K2: INTEGER;
        Z, S1, S2, X : REAL;
              A, B : VECTOR;
PROCEDURE SUMMA (MAS : VECTOR; NN : INDEX;
                                  VAR S : REAL; K
: INTEGER);
VAR
   J: INTEGER;
BEGIN
  S : = 0;
 K : = 0;
  FOR   J : = 1 TO NN DO
  IF  MAS [J]>0   THEN
                   BEGIN
                     S : = S + MAS [J];
                     K : = K + 1
                   END;
END;
BEGIN
  READ (N);
  FOR  I : = 1  TO  N  DO READ (A [I]);
  SUMMA (A, N, S1, K1);
  READ (M);
 FOR  I : = 1   TO M  DO  READ  (B[I]);
  SUMMA (B, M, S2, K2);
 READLN (X);
 Z : =EXP (K1*Ln(x))*EXP (K2*Ln(x))/( S1+ S2);
```

WRITE (' Z = ', Z)
END.

თუ პროცედურაში და ძირითად პროგრამაში გამოიყენება ერთი და იგივე პარამეტრები (პროცედურა უშუალოდ დაკავშირებულია ძირითად პროგრამასთან გლობალური პარამეტრებით), მაშინ პროცედურა შეიძლება იყოს უპარამეტრო.

6.2. ქ ვ ე პ რ ო გ რ ა მ ა ფ უ ნ ქ ც ი ა

ფუნქცია – ესაა ქვეპროგრამა, რომლის შესრულების შედეგს წარმოადგენს სკალარული სიდიდე. საბოლოოდ, ფუნქცია პროცედურისაგან პრინციპიალურად განსხვავდება იმით, რომ: I ფუნქციის შესრულების შედეგად მიიღება ერთი მნიშვნელობა, ხოლო პროცედურის – ერთი ან რამდენიმე; II ფუნქციის შესრულების შედეგი გადაეცემა ძირითად პროგრამას, როგორც ამ ფუნქციის მნიშვნელობა, ხოლო პროცედურის შესრულების შედეგები – როგორც მისი პარამეტრების მნიშვნელობები.

ფუნქციის აღწერა პროცედურის აღწერის ანალოგიურია და შედგება სათაურის, აღწერითი ნაწილისა და ფუნქციის ტანისაგან

ფუნქციის სათაურს აქვს შემდეგი ზოგადი სახე:

FUNCTION **<სახელი> (<პარამეტრების სია>): <ტიპი>;**

სადაც ***FUNCTION*** – დარეზერვებული სიტყვაა; **სახელი** – ფუნქციის სახელი;

პარამეტრების სია – ფორმალური პარამეტრების (საწყისი მონაცემების) ჩამონათვალი მათი ტიპის მითითებით; **ტიპი** – ფუნქციის შესრულების შედეგად მიღებული შედეგის ტიპი.

შესაძლებელია აგრეთვე ფუნქციის აღწერა პარამეტრების გარეშე:

FUNCTION <სახელი> : <ტიპი>;

მაგალითი 6.4. გააფორმეთ $Y=X^n$ გამოსახულების ალგორითმი ფუნქციის სახით.

```
FUNCTION STEP3(N:INTEGER; X:REAL):REAL;
VAR
   i:INTEGER;
   Y:REAL;
BEGIN
   Y:=1;
   FOR i:=1 TO N DO
      Y:=Y*X;
      STEP3:=Y
END;
```

STEP3 ფუნქციის სათაურში ჩამოთვლილია ***N*** და ***X*** პარამეტრები, რომლებიც წარმოადგენს საწყის მონაცემებს. ფუნქციის შესრულების შედეგი (***y*** ლოკალური პარამეტრის მნიშვნელობა) მიენიჭება ***STEP3*** ფუნქციის სახელს. ***Y*** ლოკალური ცვლადის გამოყენება არ არის აუცილებელი, მაგრამ მისი საშუალებით ფუნქცია უფრო თვალსაჩინოდ გამოიყურება.

6.3. ქვეპროგრამასთან მიმართვა

პროცედურის (ფუნქციის) აღწერა მოიცემა პროგრამის აღწერით ნაწილში, რომელიც არავითარ მოქმედებას არ ასრულებს. იმისათვის, რომ შევასრულოთ პროცედურა (ან ფუნქცია) საჭიროა პროგრამაში მოვათავსოთ ამ უკანასკნელთან მი-მართვის სპეციალური ოპერატორი. პროცედურასთან (ან ფუნქციასთან) მიმართვის განხორციელება შესაძლებელია პროცედურის გამოძახების სპეცი-ალური ოპერატორის საშუალებით, რომელსაც გააჩნია შემდეგი ზოგადი სახე:

< სახელი > (<არგუმენტების სია>);

სადაც **სახელი** – პროცედურის (ფუნქციის) სახელია, რომლის საშუალებითაც ხორციელდება მიმართვა (გამოძახება); **არგუმენტების სია** – კონ-კრეტული სახელებისა და მნიშვნელობების სია, რომლითაც მიმართვისას (გამოძახებისას) შეიცვლე-ბა ფორმალური პარამეტრები. პროცედურის (ან ფუნქციის) გამოძახებისას მის სათაურში მითითე-ბული ფორმალური პარამეტრები შეიცვლება შესა-ბამისი არგუმენტებით: მარცხნიდან პირველ პარა-მეტერს შეესაბამება შესაბამისად პირველი არგუმე-ნტი, მეორეს – მეორე არგუმენტი და ა.შ.

არგუმენტებს, რომლებიც ჩამოთვლილია პრო-ცედურის (ფუნქციის) გამოძახების სპეციალურ ოპერატორში ეწოდებათ ***ფაქტიური პარამეტრები***.

7. ჩ ა ნ ა წ ე რ ე ბ ი

ამოცანა 7.1. დაწერეთ პროგრამა, რომელიც დაბეჭდავს სტუდენტების გვარებს, მათ სტიპენდიებს და შესაბამისად გამოთვლის სესიის შედეგების საშუალო არითმეტიკულებს. დავუშვათ არსებობს სამი სახის სტიპენდია: I სტუდენტს, რომელსაც ყველა საგანში აქვს შეფასება 5, დაენიშნოს ფრიადოსნის სტიპენდია; II სტუდენტს, რომელსაც აქვს შეფასებები 5, 4 და ერთი 3, მაგრამ აქტიურად მონაწილეობს საზოგადოებრივ საქმიანობაში, და-ენიშნოს ჩვეულებრივი სტიპენდია; III სტუდენტს, რომელსაც აქვს თუნდაც ერთი შეფასება 2, სტიპენდია არ დაენიშნოს და მისი შეფასების საშუალო არითმეტიკული იყოს 0-ის ტოლი. სესიის შედეგი წარმოდგენილი უნდა იყოს მატრიცის სახით.

პროგრამა უნდა დაიწეროს 4 გამოცდის შეფასების საფუძველზე და იმის მიხედვით, აქტიურად მონაწილეობს თუ არა სტუდენტი საზოგადოებრივ საქმიანობაში. შესაბამის უჯრაში ჩაწერეთ 1, წინააღმდეგ შემთხვევაში – 0. ფრიადოსნის სტი-პენდია აღნიშნეთ 2–ით, ჩვეულებრივი სტიპენდია 1–ით და ის სტუდენტი, რომელიც სტიპენდიას არ მიიღებს მის შესაბამის უჯრაში ჩაწერეთ 0.

ამოცანის ამოხსნის შესაბამის პროგრამას აქვს შემდეგი სახე:

```
Program Session;
Label 5;
Const
    n=15;
    m=4;
    nm=35;
```

```
Type
    LastName = array[1..n] of char;
    rez = array[1..4] of Integer ;
    sa = Real;
    activ = Integer;
    stip = Real;
    STUDENT = RECORD
                          LN: LastName;
                            g: rez;
                            s: sa;
                          act: activ;
                        priz: stip
                END;
var
  st: array[1..nm] of STUDENT;
 i , j: Integer;
 sum: Real;
begin { Session }
   for i:=1 to nm do
     begin
        for j:=1 to n do
          Read(st[i].LN[j]);
            for j:=1 to m do
              Read(st[i].g[j]);
                Read(st[i].Act)
     end;
for i:=1 to nm do
         begin
           sum:=0;
           for j:=1 to m do
             if st[i].g[j]<3 then GOTO 5
                             else sum:=sum+st[i].g[j];
           st[i].s:=sum/m;
           if (sum=20) and (st[i].act=1) then st[i].priz:=2
```

```
            else if (sum>=15) and (sum<20) and
(st[i].act=1)

then st[i].priz:=1;
          5: st[i].s:= 0;
             st[i].priz:= 0
        end;
      for i:=1 to nm do
        begin
           for j:= 1 to n do
             write(st[i].LN[j]);
             write(st[i].s:2:0,'  ',st[i].priz)
        end
      end. { Session }
```

8. ფ ა ი ლ ე ბ ი

ფაილი დაპროგრამების ენა ***Pascal***-ზე წარმოადგენს მონაცემების სტრუქტურულ ტიპს, რომელიც შედგება ერთნაირი ტიპის და ერთნაირი სიგრძის კომპონენტებისაგან. ყველაზე ხშირად ფაილის კომპონენტებს წარმოადგენს ჩანაწერი. ფაილის ტიპის აღწერა იწყება დარეზერვებული სიტყვით ***FILE OF***, შემდეგ კი მიეთითება კომპონენტების ტიპი; ტექსტურ ფაილი აღიწერება დარეზერვებული სიტყვით ***TEXT***, ხოლო ფაილი ტიპის გარეშე – დარეზერვებული სიტყვით ***FILE***.

ნებისმიერ ფაილს აქვს 3 დამახასიათებელი თვისება: I მას აქვს სახელი, II ფაილი შეიცავს ერთი ტიპის კომპონენტებს, მისი კომპონენტები შეიძლება იყოს ნებისმიერი ტიპის, გარდა ფაილური ტიპის; არ შეიძლება შევქმნათ „ფაილების ფაილი”; III შექმნილი ფაილის სიგრძე ანუ მასში კომპონენტების რიცხვი

დამოკიდებულია მხოლოდ კომპიუტერის შიდა მეხსიერებაზე.

ფაილური ტიპი შეიძლება აღვწეროთ ქვემოთ მოყვანილი 3 სახიდან ერთ-ერთით:

1) Type

<ფაილის სახელი> = *FILE OF* <კომპონენტების ტიპი>;

var

<იდენტიფიკატორი> : <ფაილის სახელი>;

2) <იდენტიფიკატორი> : *TEXT*;

3) <იდენტიფიკატორი> : *FILE*;

განვიხილოთ ფაილის ჩაწერის პირველი სახე: მაგალითად:

```
Type
  karta = FILE  OF   RECORD
                       d: Byte;
                       v: Real;
                     END;
     var
       kartfile: karta;
```

ფაილის ჩაწერის პირველი სახე შეიძლება ასეც გამოიყურებოდეს:

```
Type
   < კომპონენტების ტიპი > = RECORD

              < კომპონენტი 1 > :  < ტიპი >;

              < კომპონენტი 2 > :  < ტიპი >;
                                    .
                                    .
                                    .
```

```
            < კომპონენტი n > :  < ტიპი >;
                                  END;
var
   F : FILE  OF  < კომპონენტების ტიპი >;
   r : < კომპონენტების ტიპი >;
მაგალითად:
Type
     student= RECORD
         NUM: Integer;
          N,LN: string[25];
         Oklad: Real
                   END;
var
   st: FILE OF student;
  s: student;
```

9. ტ ე ქ ს ტ უ რ ი ფ ა ი ლ ე ბ ი

ამოცანა 9.1. შექმენით **C** დისკის მთავარ საქაღალდეში **TF** ტექსტური ფაილი. ფაილი შეიცავს საქმიანი წერილის სტრიქონებს. ამასთან, ყოველი სტრიქონი შედგება მაქსიმუმ 60 სიმბოლოსაგან. როდესაც სტრიქონის მნიშვნელობა ტოლი იქნება სიტყვის „**Title**“, მაშინ შეწყვიტეთ სტრიქონის ჩაწერა ფაილში. დისკზე პროგრამა შეინახეთ **SOZDT** პროცედურის სახელით.

```
Program TextFiles;
Type
     d=string[60];
var
  FileName:string[10];
  TF: text;
```

```
  st: d;
  ch: char;
Procedure SOZDT;
begin {SOZDT}
   Write(' შემოიტანეთ შესაქმნელი ტექსტური
          ფაილის სახელი ==> ');
   ReadLn(FileName);
   assign(TF,FileName);
   rewrite(TF);
while TRUE do
    begin
      ReadLn(st);
      if st=' Title' then
                    begin
                      close(TF);
                      exit
                    end;
     WriteLn(TF,st) { ფაილში ჩაწერა }
    end
end; {SOZDT}
. . .
```

ამოცანა 9.2. დაბეჭდეთ 7.1 ამოცანაში შექმნილი ფაილის ყველა კომპონენტი. დისკზე პროგრამა შეინახეთ ***OTKT*** პროცედურის სახელით.

```
Procedure OTKT;
begin{OTKT}
{$I-}
  repeat
Write('შემოიტანეთ დასაბეჭდი ტექსტური
          ფაილის  სახელი ==>' );
    ReadLn(FileName);
```

```
      assign(TF,FileName);
      reset(TF);
    until (IOResult=0) or  (FileName='Title');
  {$I+}
    while  not SeekEof (TF)  do
       begin
          ReadLn(TF,st); { ფაილიდან წაკითხვა }
          WriteLn(st)
       end
  end; {OTKT}
  . . .
```

ამოცანა 9.3. ჩვენს მიერ შექმნილ ფაილს დავამატოთ ახალი კომპონენტები. როდესაც სტრიქონის მნიშვნელობა ტოლი იქნება სიტყვის „***Title***“, მაშინ შეწყვიტეთ სტრიქონის ჩაწერა ფაილში. დისკზე პროგრამა შეინახეთ ***RASHT*** პროცედურის სახელით.

```
  Procedure RASHT;
  begin { RASHT }
    { $I- } { შეტანა-გამოტანის ოპერაციის
              კონტროლის გაუქმება }
    repeat
    Write('საკორექტირებელი ფაილის სახელია ==> '
);
       ReadLn(FileName);
       assign(TF, FileName);
       append(TF);
    until IOResult=0;
    { $I+ } { შეტანა-გამოტანის ოპერაციის
          კონტროლის ჩართვა }
  while TRUE do
     begin
```

```
        WriteLn(' შემოიტანეთ სტრიქონი ==>');
        ReadLn(st);
        if st=' Title' then
                        begin
                          close(TF);
                          exit
                        end;
        Write(TF , st)
    end
end;{ RASHT }
. . .
```

თუ თითოეულ ზემოთ ჩამოთვლილ პროგრამის ფრაგმენტს გააფორმებთ, როგორც დამოუკიდებელ პროგრამებს, გამოძახებისას შეიძლება შეგექმნათ მრავალი პრობლემა. ამ პრობლემის თავიდან ასაცილებლად ტექსტურ ფაილებთან მუშაობისას შეგიძლიათ თითოეული ეს ფუნქციონალური ბლოკი გააფორმოთ პროცედურის სახით და ისინი შეიძლე-ბა გამოიძახოთ ძირითადი პროგრამიდან შესაბამისი რეჟიმის არჩევის შემდეგ.

ამოცანა 9.4. ერთ პროგრამაში გააერთიანეთ შექმნის, გახსნის და გაფართოების ტექსტური ფა-ილების ***(SOZDT, OTKT, RASHT)*** ფუნქციონალური ბლო-კები.

```
Program TextFiles;
Type  d:string[60];
var
  TF: text;
  st: string;
  rej: char;
  { $I SOZDT.PAS }
  { $I OTKT.PAS }
```

```
  { $I RASHT.PAS }
begin
  while TRUE do
    begin
      ClrScr;
      WriteLn(' მიუთითეთ რეჟიმი:' );
      WriteLn(' 1: ტექსტური ფაილის შექმნა'
);
      WriteLn(' 2: ტექსტური ფაილის ბეჭდვა'
);
      WriteLn('3: ტექსტურ ფაილში დამატება'
);
      WriteLn('4: პროგრამიდან გამოსვლა' );
      Read(rej);
      WriteLn;
      case rej of
        ' 1 ' : SOZDT;
        ' 2 ' : OTKT;
        ' 3 ' : RASHT;
        ' 4 ' : HALT;
     else WriteLn(' გაიმეორეთ რეჟიმის
ნომერი!')
      end {Case}
    end
end.
```

ლ ი ტ ე რ ა ტ უ რ ა

1. Абуладзе И.Б., Буадзе Г.А. Турбо Паскаль. Тбилиси, Технический университет, 2002.
2. Алексеев В.Е., Ваулин А.С., Петрова Г.Б. Вычислительная техника и программирование. М .: Высшая школа, 1991.