Увод

Тези уроци има за цел да подпомогнат профилираното обучение по програмиране на езика Паскал. Те може да са полезни както за начинаещи, така и за средно напреднали.

Езикът носи името на Блез Паскал – френски философ, математик, физик, писател. Автор на езика е швейцарският професор Никлаус Вирт.

I. Основни означения

Пример за програма на езика Паскал:

Задача 1. Да се напише програма, която въвежда стойности на реалните променливи a и b и извежда тяхната сума s.

```
Program Suma;
  var a, b, s: real;
Begin
  write('vavedete a: '); readln(a);
  write('vavedete b: '); readln(b);
    s:=a + b;
      writeln(a, '+' ,b, '=' , s:4:2);
  readln;
 end.
```

Първият ред на програмата се нарича **заглавие**. С него започва всяка програма. Името на програмата (в случая Suma) може да съчетава букви, цифри и знака за подчертаване _ , но започва винаги с буква.

Променливите, използвани при описанието на алгоритмите в езика Паскал задължително се описват в раздела var (от variable - променлив). В горния случай това са a, b и s. И трите променливи са от тип real, т.е. могат да приемат стойности от множеството на реалните числа. Декларирането на променливите води до отделяне на памет за тях.

След раздела на променливите в програмата е разделът на операторите. Този раздел започва с думата begin и завършва с end. В примера са използвани няколко оператора. Те описват алгоритъма на програмата. Отделят се със символа ; .

- оператора **write('текст')** извежда върху екрана в позицията на курсора текста, ограден в апострофи.

- оператора **write(a)** извежда на екрана стойността на параметъра a. Горните два оператора могат да се съчетават (например write('a=', a)).

Ако след думата write се постави ln (writeln) операторите отново извеждат информацията на екрана, но на нов ред.

- операторът read записва въведената стойност в оперативната памет. Той действа по следния начин: след write компютърът чака да се въведе стойност (например променливата a). Въвежда се стойност, която задължително трябва да е от декларирания тип. След натискането на клавиша Enter въведената стойност се записва в оперативната памет.

- операторът **readln** без параметри означава преминаване за четене на нов ред, чрез натискането на Enter. Readln в края на програмата, непосредствено преди заключителния end задържа екрана до натискане на enter, за да може да се види резултата от изпълнението на програмата.

- операторът s:= a + b е оператор за присвояване на стойност. Двойката символи := се нарича знак за присвояване. Променливата s присвоява стойността на израза a + b.

Идентификатори. Идентификаторите в езика паскал се използват за означаване на имена на програми, променливи, процедури, функции и др. Както обикновени идентификатори, въведени от програмиста, съществуват и идентификатори, които са запазени. Те се използват в програмирането по стандартен, предварително определен начин и не могат да се използват по друг начин. В стандарта на езика паскал има 35 запазени думи, като в програмата Suma са използвани 4 – program, var, begin и end.

В езика паскал програмистите могат да въвеждат **коментари.** Коментарите са редица от символи, които са заградени в скобите { и }. Предназначени са за програмистите и се игнорират при изпълнение на програмата. Чрез тях програмата става по-ясна и разбираема, като програмистът може да въвежда всякакви коментари и при това на произволни места. Препоръчително е обаче, коментарите да не се използват вътре в операторите, а между тях. Ето пример за програма, в която са включени коментари:

```
Program Suma;
    var a,b,s: real;
Begin
        {въвеждане на стойност на променливата a}
    write('vavedete a: ')readln(a);
        {въвеждане на стойност на променливата b}
    write('vavedete b: ')readln(b);
        {намиране на сумата на a и b}
```

```
    s:= a + b;
        {извеждане на резултата}
    writeln(a, '+' ,b, '=' ,s:4:2);
  readln;
End.
```

II. Структура на паскалската програма

Всяка програма на езика Паскал има точно определена структура. Тя се състои от заглавие (виж урок №1) и блок и завършва със символа точка. Блокът се състои от шест раздела, всеки от които може да бъде празен, като изключение прави последния. Това са резделите на етикетите, константите, типовете данни, променливите, функциите и процедурите, операторите. Разделите задължително трябва да са в горния порядък. Първите пет раздела са декларации. Чрез тях програмистът обявява различните обекти на програмата.

Раздел на етикетите – етикетът е цялочислено число без знак. Служи за отбелязване на място в програмата, където да бъде предадено управлението. Етикетът се поставя пред оператора, към който ще се предава управлението, като за разделител се използва двуеточие. <етикет> : <оператор>

Всички етикети, които се използват в програмата се декларират в раздела на етикетите: label <етикет>

Раздел на константите – често в програмите се използва една и съща стойност няколко пъти. В такъв случай се препоръчва константата да бъде декларирана предварително в раздела на константите. В долния пример можете да видите как се декларира и използва дадена константа.

Задача 2: Да се напише програма, която по даден радиус намира P и S на окръжност, кодето P=2.p.r и S=p.r.r, а p=3,142857.

```
Program Zad_2;
  const Pi=3.142857;
  var r, p, s: real;
Begin
  write('vavedete radius: ');
    readln(r);
  p:=2*pi*r;
  s:=pi*r*r;
  write('obikolkata e: ',p:2:4);
  write('litseto e: ',s:2:4);
    readln;
end.
```

Раздел на типовете данни. В езика Паскал има точно определен стандартен набор он типове данни – integer, real, boolean, char и др. В един език за програмиране не могат да се приведат всички типове данни, които могат да са необходими на програмиста. По тази причина езикът дава възможност на програмиста сам да дефинира нови типове данни. Това се извършва в раздела на типовете

type <идентификатор> = <име на тип>

В следващите параграфи ще бъдат дадени различни описания наконструкцията <име на тип>.

Раздел на променливите. Всяка променлива, която се използва в програма на езика Паскал, трябва да се опише. Това се извършва в раздела на променливите:

var <променлива> : <тип>

За всяка декларирана променлива се отделя точно количество памет, което зависи от типа на променливата.

Раздел на процедурите и функциите – в езика Паскал има определен набор от оператори и вградени функции. Чрез дефинирането на процедури се разширява наборът от оператори, а чрез дефинирането на функции – наборът от функции в езика Паскал. Конструирането на процедури и функции ще разгледаме в отделен урок.

Раздел на операторите – докато първите пет раздела имаха декларативен характер и някои от тях могат да бъдат празни, разделът на операторите описва алгоритъмът за решаване на задачата и задължително трябва да участва в програмата.

```
Begin
    <оператор>
    <оператор>
        ....
    <оператор>
End.
```

III. Типове данни в езика Паскал

В езика Паскал съществуват следните типове данни:

- **скаларни** – цял, реален, символен, булев, изборен, подобласт;
- **съсатвни** – масив, запис, множество, файл;
- тип **указател**.

Всеки от тези типове данни се характеризира с множество от стойностти и операции и вградени функции, които могат да се приложат над елементите му. Съставните типове данни се наричат онези данни, компонентите на които са редици от елементи. Типът

указател дава средства за динамично разпределение на паметта и се използва за реализиране на различни динамични структури от данни. В този урок ще разгледаме скаларните типове булев, цял, реален и символен.

3.1 Булев тип данни

За означаването и дефинирането на този тип се използва стандартната дума **boolean**. Множеството от стойностите на този език се състои от два елемента – стойтостите **true (истина)** и **false (лъжа)**. Тези стойности се наричат булеви константи. Деклариране на булеви променливи стьвь по познатия начин:

var b,x: boolean;

Над данни от булев тип е възможно да се прилагат някои операции и вградени функции. Операторите **not, and** и **or** (съответно логическо отрицание, логическо умножение и логическо събиране) се прилагат над операнди от булев тип. Резултатът е булева константа или променлива. На долната таблица са дадени стойности на променливите a и b, както и операции с тях:

a	**b**	**not a**	**a and b**	**a or b**
false	false	true	false	false
false	true	true	false	true
true	false	false	false	true
true	true	false	true	true

Операторите за сравнение също могат да се прилагат над данни от булев тип

= равно	<= по-малко или равно
<> различно	> по-голямо
< по-малко	>= по-голямо или равно

Сравняването се извършва, като се сравнят кодовете на булевите константи. Например true<=false получава стойност false, false=true също получава стойност false, true>=false получава true и така нататък.

Стандартни функции, даващи булев резултат:

Odd(x) – установява дали цяло цисло е четно или не

Odd(x) = true ако x е нечетно и обратно.

Eoln(x) – булева функция, установяваща дали е достигнат края на реда на даден текстов файл. Има стойност true за достигнат край на реда и false за недостигнат.

Eof(x) – булева функция, установяваща дали е достигнат края на даден текстов файл. Отново приема стойност true за достигнат и false за недостигнат край.

Функциите eof и eoln ще разгледаме по-подробно в един от следващите уроци (работа с файлове).

Променливите от типа булев не се въвеждат. На тях може да се дава стойност с оператора := (например b:=1 или b:=true), като 0 се счита за false а 1 - за true. Извеждането на този тип данни става по стандартния начин – с write и writeln.

3.2 Булеви изрази

Булевите изрази са правила за получаване на булева константа. Булевите константи true и false, както и булевите променливи са булеви изрази. Прилагането на булевите оператори not, and и or над булеви изрази също е булев израз. Например **[not(b and a)] or (b or a)=** е булев израз. Прилагането на оператори за сравнение също е булев израз.

3.3 Тип цял

За означаването и дефинирането на този тип се използва стандартната дума **integer**. Множеството от стойностите на променливите от този тип са всички цели числа от [-32768;32767]. Декларирането става по стандартния начин:

var x, y: integer;

Операции и вградени функции:

- **унарни операции –** + и - - потвърждават или променят знака, към който са приложени;

- бинарни операции – прилагането на следните операции дава цяла стойност:

+ - събиране * - умножение div – частното от деление
- - изваждане mod – остатък от деление

Пример 12 / 5 = 2.4 12 div 5 = 2 12 mod 5 = 4

- следните функции, приложени над ця аргумент връщат цял резултат:

abs(x) – връща абсолютната стойност на x - |x|
sqr(x) – дава квадрата на x – x*x
succ(x) – дава наследника на x – x+1
pred(x) – дава предшественика на x – x -1

- функции, приложени над реален аргумент и даващи цял резултат:

trunc(x) – отрязва дробната част и запазва цялата – trunc 7,9=7
round(x) – закръгля x.

Въвеждането на данни от типа integer става с read и readln, а извеждането – с write и writeln.

Освен integer в езика Паскал са дефинирани и следните цилочислени типове, различаващи се само от своето множество от стойности.

Shortint[-128,127] integer[-32768,32767] word[0,65636]
Longint[-2147483648,2147483647] byte[0,255]

3.4 Тип реален

За деклариране на този тип се използва стандартната дума **real**. Множеството от стойностите на този език се състои от всички реални числа от -1,7E38 до 1,7E38. Елементите на множеството от тип real се наричат реални константи. Деклариране на променливите става по познатия начин:

```
var
  x,y: real;
```

За стойностите от този тип са възможни следните операции и вградени функции:

- унарни операции

+ и - – потвърждаватили отменят знака на операнда, към който са приложени;

- бинарни операции – при условие, че поне един от операндите е цял, следните операции връщат реален резултат:

+ - събиране
* - умножение
- - изваждане
/ - деление

При два цели операнда операцията / връща реален резулат.

- при реален аргумент следните функции връщат реален резултат

abs(x) – намира абсолютната стойност на x - |x|

sqr(x) – намира квадрата на x

- при реален или цял аргумент следните функции връщат реален резултат

sin(x) – намира синуса на даден ъгъл

cos(x) – намира косинуса на даден ъгъл

! ъглите трябва да са зададени в радиани

in(x) – намира неперов логаритъм (x>0)

sqrt(x) – намира корен квадратен от x (x>0)

Въвеждането на реални променливи се осъществява с read и readln, а извеждането с write и writeln.

3.5 Аритметични изрази

Аритметичните изрази в Паскал са правила за получаване на числови константи. Те са цели и реални.

Цели аритметични изрази – към тях спадат целите константи или променливи. При прилагането на унарните и бинарните операции, като и функциите, като succ, abs и др. изразите също са цели или аритметични.

Реални аритметични изрази: както при целите аритметични изрази, реалните променливи и прилаганите им унарни или бинарни функции също са реални аритметични изрази.

Приоритет на операциите и вградените функции:

- вградени функции (най-висок приоритет)
- + , - – унарни
- * , / , div , mod
- + , - – бинарни

За нарушаване на този приоритет (ако е необходимо) се използват скобите. Приоритета на скобите е по-малък от приоритета на вградените функции и по-голам от всички останали операции.

Намирането на стойността на аритметичен израз изисква всички променливи, които участват в израза, предварително да бъдат обвързани със стойностти.

Целите и реалните аритметични изрази могат да се сравняват. В резултат на това се полечава булев езраз-

Приоритет на операциите и вградените функции в булевите изрази: булевият израз е правило за получаване на булева стойност. За намирането на тази стойност се използва следния приоритет:

- вградени функции
- скоби
- not, +, - (унарни)
- * , / , div , mod , and
- + , - , or
- = , < , > , <> , <= , >=

Операциите, които са с еднакъв приоритет се изпълняват от ляво на дясно.

3.6 Тип символен

За означаването на типа се използва запазената дума char. Множеството от стойности на типа се състои от елементите на крайно и наредено множество от символи. Символите биват графични и управляващи. Графичните символи имат графично (видимо) представяне – ‘@’ ‘A’ ‘a’ ‘?’ ‘1’ и др. Означават се, като символа се загражда в апострофи.

Управляващите символи се въвеждат и съхраняват в паметта на компютъра, но нямат графично представяне във вид на символ. Означават се с идентификатори, например Nul, EOT, CR, LF и т.н. Наредбата на символите е следната:

- от 0 до 31 – управляващи символи
- от 32 до 255 – графични символи (40-57 – графични символи, 65-90 – букви)

Тази наредба представя знаците, цифрите, главните и малките букви с последователни кодове.

Променливите от вида се декларират по познатия начин:

```
var
    ch: char;
```

Операции и вградени функции

ord(ch) – намира кода на променливата ch

chr(i) – намира символа, чиито код е i.

pred(ch) – намира символа, чиито код е с 1 по-малък от кода на променли ch

pred(ch)=chr(ord(ch) – 1)

succ(ch) – намира символа, чиито код е с 1 по-голям от кода на променливата ch

succ(ch)=chr(ord(ch) +1)

Оператори за сравнение - операторите за сравнение са същите, които се използват и в булевите изрази - <, >, =, <>, <=, >=. Резултатът от сравнението е от булев тип.

Данните от този тип се въвеждат с read и readln, както и с :=, и се извеждат с write и writeln.

3.7 – Символни изрази

Символният израз е правило за получаване на символни константи.

- константите от символен тип са символни изрази - ‘@’ ‘A’ ‘a’ ‘?’ ‘1’ и др.
- прилаганито на pred и succ към символни изрази е символен израз.
- Прилагането на функцията chr към цял израз е символен израз.

IV. Основни структури за управление на изчислителния процес

4.1 – Оператор за присвояване на стойност

Това е един от най-важните оператори в езика Паскал. Синтаксисът му е променлива := израз. Величината, която е вляво на знака := задължително е променлива. Тя може да е с поизволен тип с

изключение на типа файл. От дясно на оператора задължително стои израз, който е от същия тип, от който е променливата в лявата страна. Например a:=5, където a е от тип integer. Изключение: на променлива от тип реален може да се присвои стойтост от тип цял (тоест реалният тип припокрива типа целочислен).

4.2 – Съставен оператор

Съставният оператор обединява в логическо цяло няколко оператора. Често синтаксисът на някоя структура за изчислителния процес изисква да се използва точно един опреатор. Логиката на решаваната задача обаче изисква използването на няколко оператора. Тези няколко оператора могат да се вложат един в друг, така че да се запишат сато съставен оператор, като съставящите го оператори се изпълняват последователно.

4.3 – Празен оператор

Това е оператор, който не съдържа никакви действия и не съдържа никакви символи. Отделя се от останалита оператори с ; . Празният оператор може да бъде поставен след който е да е оператор.

Пример

```
begin
    x:=1; ; ; ; ;
    y:=14;
end;
```

Горният пример съдържа 7 оператора – оператор за присвояване x := 1, следван от четири празни оператора и отново оператор за присвояване.

Приложението на този оператор е за забавяне на изчислителния процес (for i := 1 to 10000 do ;).

4.4 – Условен оператор

Условен оператор – това е оператор, който дава възможност да се изпълни или да не се изпълни друг оператор в зависимост то някакво уславие. В езика Паскал условните оператори са в две форми – кратка и пълна.

Кратка форма на условния оператор: синтаксисът на тази форма е **if B then S**, където

- if и then са запазени думи

- B – булев израз
- S – оператор

Условният оператор се изпълнява по следния начин – пресмята се стойността на булевия израз B. В резултат се получава булева константа True или False. Ако стойността е True се изпълнява оператора S. Ако обаче тази стойност е False – S не се изпълнява.

След запазената дума then трябва да стои точно един оператор. Ако логиката на задачата налага използването на два или повече оператора, необходимо е те да се обединят в един съставен оператор.

Пример за програма, използваща оператора if - then.

Зад 2. Да се напише програма, която намира лицето на кръг и дължината на окръжност с радиус r. Пресмятането да се осъществи в случай, че r > 0.

```
Program Zad_2;
  const pi=3.142857;
  var r, p, s: real;
Begin
  write('vavedete radius r:  ');
    readln(r);
   if R > 0 then begin
     p:=2 * pi * r;
     s:=pi * r * r;
       write('daljinata e  ', p:2:4);
       write('litseto e ', s:2:4);
     end;
   readln;
End.
```

Изпълнение на оператора: Пресмята се булевият израз R > 0. Щом стойността му е равна на true, т.е. въведеният радиус е по-голям от 0 се изпълнява съставният оператор след then. Ако въведената стойност за R е по-малка или равна на 0, то операторът then не се изпълнява.

Пълна форма на условния оператор.

Синтаксисът на тазе форма е if B then S1 else S2, където

- if (ако), then (тогава), else(иначе) са запазени думи.
- B е булев израз.
- S1 и S2 са оператори.

Операторът се изпълнява по следния начин: Пресмята се стойността на булевия израз B. В резултат на това се получава true или false. Ако стойността на B е true се изпълнява S1 (S2 не се изпълнява). Ако стойността на на B е false S1 се прескача и се изпълнява S2.

Във всички случаи след изпълнението на S1 или S2 се преминава към следващите оператори.

Забележка: след then и след else трябва да стои точно един оператор.

Пример за програма, използваща оператора if-then-else.

За пример ще преустроим малко горната програма. Отново по даден радиус програмата ще трябва да изчисли S и P на окръжност, но ако радиусът R е по.малък от 0 ще се извежда “error !”

```
Program Zad_3;
  const pi=3.142857;
  var r, p, s: real;
Begin
  write(‘vavedete radius r:  ‘);
    readln(r);
  If R > 0 then begin
   p:=2 * pi * r;
   s:=pi * r * r;
     write(‘daljinata e  ‘, p:2:4);
     write(‘litseto e ‘, s:2:4);
   end;
 else writeln(‘error!’);
readln;
End.
```

Вложени условни оператори. В условният оператор if B then S1 else S2 S1 и S2 са произволни оператори. В този случай говорим за вложени условни оператори.

При влагането на един условен оператор в друг е възможно да възникнат логически проблеми. Затова в езика Паскал има правило, което определя начина, по който се изпълняват вложените оператори: Всяко else се съчетава в един условен оператор с най-близкото, несъчетано преди него then. Така текста се преглежда от ляво надясно.

Препоръчително е условен оператор да се влага в друг условен оператор само след else, а не след then. Ако след then трябва да се вложи оператор, условният оператор да се направи съставен.

4.5 – Оператори за цикъл

Операторите за цикъл се използват за реализиране на циклични изчислителни процеси. Това са процеси, при които

оператор или група от оператори се изпълнява многократно за различни стойности на техни параметри.

Цикличните процеси биват два вида – индуктивни или интеративни.

Индуктивен цикличен процес е такъв, при който броят на повторенията е предварително известен. Например: да се намери стойността на израза S = x + 2x + 3x + nx.

За да се намери сумата на израза е необходимо да се повтори действието n*x n пъти (n=1,2,3....), като всеки път стойността му се прибавя към текущата стойност на S (S:= S + n*x).

Цикличен процес, при който броят на повторенията не е известен, се нарича интеративен цикличен процес. При тези процеси някакво условие определя броя на повторенията.

В езика Паскал има три оператора за цикъл – оператор for (реализира индуктивни циклични процеси), оператор while и оператор repeat (реализират всякакви процеси).

4.5.1 – Оператор for

Този оператор се нарича оператор за цикъл с параметър. Предназначението му е да реализира индуктивни циклични процеси. В езика Паскал съществуват две форми – for-to-do и for-downto-do.

Пример за програма, използваща **оператора for-to-do**: Да се напише програма, която пресмята n! (това е функцията фактуриел. 5!=1*2*3*4*5). N е естествено число, по-голямо от 0.

```
Program facturiel;
    var n, i, fac: integer;
Begin
    write('vavedi n: ');
        readln(n);
    fac:=1;
    for i:=1 to n do fac:=fac*i;
        writeln(n ,'!=', fac);
    readln;
End.
```

Операторa for i:=1 to n do fac:=fac*I е оператор за цикъл. Той се изпълнява по следния начин: запазените думи в състава му означават следното: за (for) i:=1 ot n повтаряй (do) fac:=fac*i.

Ако стойността на n по време на изпълнение на оператора for i:=1 to n do... е 4 променливата първо присвоява стойност 2. Тогава изразът fac:=fac*i приема стойност 2 (на fac първоначално е дадена стойност 1). След това на i се дава стойност 3. Изразът fac също

присвоява новата стойност fac*3. След това i присвоява 4 и т.н., в зависимост каква е стойността на n, която сме въвели.

Променливата i може да бъде от различни типове. Следните оператори за правилно заисани For-to-do:

```
For i:=1 to 100 do writeln(i);
For i:=-10 to 20 do writeln(i);
For b:=false to true do writeln(i);
For ch:='a' to 'z' do writeln(i);
```

При случай, че стойността на i е различна от тази, зададена в оператора (например в горния оператор – тя е по-малка или по-голяма от N) оператора не се изпълнява.

Оператор for-downto-do

Синтаксисът на този оператор е for i := V1 to V2 do S, където

- for (за), downto (надолу до), do(прави) са запазени думи
- i променлива величина, наричана още управляваща променлива на цикъла. Възможните тирова за i са char, boolean, integer, изборен и подобласт
- V1 и V2 са изрази от тип, съвпадащ с типа на управляващата променлива i. V1 е началната стойност, а V2 – крайната.
- S е произволен оператор, наричан тяло на цикъла. Управляващата променлива приема последователно всички стойности между началната и крайната. Тези стойности тя приема автоматично и затова е забранено нейната стойност да бъде променяна в тялото на цикъла.

Операторът for i := V1 to V2 do S се изпълнява по следния начин: Първо се намират стойностите на V1 и V2 и се сравняват. Ако стойността на V1 е по-голяма или равна на стойността на V2 управляващата променлива получава последователно всички стойности от V1 до V2 и за всяка стойност се изпълнява операторът S. Ако стойността на V1 е по-малка от стойността на V2 операторът не се изпълнява нито веднъж.

Вложени оператори for

Тялото на оператора for е произволен оператор. В частни случаи той може да бъде оператор за цикъл. В такива случаи става дума за вложени цикли.

Пример за вложен цикъл:

```
var i, j : integer;
……
```

```
for i := 1 to 5 do
    for j := 1 to 3 do S
```

Тук S е произволен оператор, тяло на вложен цикъл. Този вложен цикъл се изпълнява по следния начин: Управляващата променлива i получава последователно стойности 1, 2 .... 5, като за всяка от тези стойности j получава стойност 1, 2 или 3 и за тези стойности се изпълнява операторът S.

4.5.2 – Оператор while

Чрез този оператор може да се реализира произволен цикличен процес. Препоръчително е обаче този оператор да бъде използван предимно за реализиране на интеративни циклични процеси.

Пример за програма, използваща цикъла while: Да се напише програма, която пресмята сумата $e^x = 1 + (x/1!) + (x^2 / 2!) + (x^3 / 3!)$….. докато абсолютната стойност на последното събираемо стане по-малка от стойността на променливата eps. (eps и x се въвеждат от клавиатурата).

```
program Zad_4_5_2;
   var x, x1, s, eps: real;
                  i: integer;
begin
   write('x= '); readln(x);
   write('eps= '); readln(eps);
   x1:=1;  S:=1;  I:=0;
   while abs(x1) >= eps do
      begin
         i:= I + 1;
         x1:= x1 * x/i!;
         s:= s + x1;
       end;
    writeln('S= ');
  readln;
end.
```

Изпълнение на програмата: След изпълнението на операторите x1:=1; S:=1; I:=0; имаме x1=1, S=1 и I=0. Нека въведем стойности за x и esp, например x=1 и esp=0.5.

Операторът while се изпълнява по следния начин: намира се стойността на булевия израз abs(x1)>=eps за текущите стойности на x и eps. Тъй като условието е изпълнено и стойността на булевия израз е true се изпълнява тялото на цикъла. След това отново се намира стойността набулевия израз abs(x1)>=eps. Тя отново е true и

тялото на цикъла се изпълнява още веднъж. Това се повтаря докато стойността на булевия израз abs(x1)>=eps не стане false, с което приключва изпълнението на цикъла.

В оператора while B do S

- while(докато) и do(повтаряй) са запазени думи.
- B е булев израз или условие, което трябва да има стойност при влизането в цикъла.
- S – произволен оператор.

Ако е необходимо да се изпълнят многократно няколко оператора, те трябва да се оформят като съставен оператор (с begin и end).

Върху стойността на булевия израз B трябва да влияе поне един от операторите, изграждащи тялото на цикъла, за да може това условие да се проверява всеки път. В противен случай този оператор не може да завърши изпълнението си.

4.5.3 – Оператор repeat

Предназначението на този оператор е за реализация на произволен цикличен процес. Препоръчително е използването му за реализация на интеративни циклични процеси.

Пример за програма, която използва оператора repeat: като пример ще разгледаме програмата от миналата точка, но решена не с оператор while-do а с оператор repeat-until.

```
program zad_4_5_3;
   var x, x1, s, eps: real;
                    i: integer;
begin
   write('x= '); readln(x);
   write('eps= '); readln(eps);
   x1:=1;  S:=1;
 if abs(x1) >= eps then
     begin
        i:=0;
         repeat
              i := i + 1;
              x1 := x1*x / i;
              s:=s+x1;
           until abs(x)<eps;
       end;
    writeln('s= ',s);
  readln;
end.
```

Тази програма използва оператора repeat-until. Той започва със запазената дума repeat, която обуславя началото на оператора. Означава повтаряй следното. След тази дума стои тялото на цикъла. В горния случай то се състои от три оператора за присвояване. След тези оператори, които съставят тялото на оператора, следва запазената дума until, която означава докато. Операторът завършва с булевия израз abs(x1) < eps.

Като цяло горният цикъл означава следното: повтаряй операторите...докато се изпълни условието.... .Когато зададеното условие е привършено цикълът приключва.

!!! Използването на оператора if abs(x1) >= eps then … е наложително, тъй като е възможно условието abs(x1) < eps да е в сила от началните стойности на x1 и eps.

!!! Между двете запазени думи repeat и until се състои тялото на цикъла. То се състои поне от един оператор. Ако логиката на решаваната задача изисква повече от един оператор се прави съставен оператор, но не се огражда с begin и end. За това служат Repeat и until.

И в този цикъл, както и в while-do поне един оператор от тялото на цикъла трябва да влияе на стойността на булевия израз, защото изпълнението на оператора може да не завърши.

Тялото на цикъла се изпълнява поне веднъж.

4.6 – Оператор за избор на вариант

В езика паскал е реализиран оператора за избор на вариант case, който дава възможност да се избере една от няколко възможности. Пример за програма, използваща оператора case:

Да се напише програма, извежда въведената цифра с думи (например за 5 - пет).

```
Program zad_4_6;
  var n: integer;
Begin
  write('n= '); readln(n);
    case n of
        0 : write('zero');
        1 : write('one');
        2 : write('two');
        3: write('three');
        4 : write('four');
        5 : write('five');
        6 : write('six');
        7 : write('sevem');
        8 : write('eight');
```

```
        9 : write('nine');
      end;
    writeln;
    readln;
end.
```

В горната програма е използван операторът case. Този оператор започва със запазената дума case (случай), след което следва израз. В горния случяй това е стойността на n. След израза стои запазената дума of (от), след която следват оператори, всеки от които се предхожда от цифра, наречена етикет. За разделител се използва символът : . Операторът case завършва със запазената дума end.

По време на изпълнение на програмата, операторът case се изпълнява, като се пресметне стойността на израза сред запазената дума case. След това получената константа се търси сред етикетите на оператора. Ако такъв етикет съществува се изпълнява оператора, с който е свързан. Ако етикет с тази стойност не съществува, оператора не се изпълнява.

Изразът след запазената дума case може а бъде от тип цял, символен, булев, изброен или подобласт.

Редът на операторите, съставящи case е без значение, като всяка етикетна константа може да се среща само веднъж.

V. Скаларни типоев изброен и подобласт

5.1 – Тип изброен

Този тип е скаларен. Дефинира се от програмиста. Дефинирането му се извършва или в раздела на типовете или в раздела на променливите.

Например:

Type week=(mon, tue, wen, thu, fri, sat, sun)

По този начин множеството от стойносттите се състои от всички идентификатори, които са изброени в дефиницията.

Операции и вградени функции

Оператор за присвояване: възможно е на променливи от тип изброен да се присвои като стойност коя да е константа от множеството от стойностите на типа, от които е променливата. По този начин променлива от отипа изброен се свързва с текуща стойност. Например:

d1:=sat;
d7:=mon;

Оператори за сравнение: константите от изброен тип могат да се сравняват с познатите оператори за сравнение: <, >, <>, <=, >=, =. При самото сравняване се сравняват кодовете им, като резултатът е булева константа.

mon<sun tue>fri

Вградени функции

- ord(x) – намера кода на x, където x е константа от тип изброен: ord(sun)=6
- pred(x) - получава константа, чиито код е с едно по-голям от кода на x: pred(sun) = sat
- succ(x) – получава константа, чиито код е с едно по-голям от този на x.

Функциите pred и succ не са определени съответон за първия и последния елемент на даден изброен тип.

Данните от тип изброен нямат гряфично представяне и операторите read и write са недопустими за тях. Те не могат да бъдат въвеждани и извеждани. И въпреки това използването им прави програмите по-ясни, четливи и бързи.

5.2 – Тип подобласт

Много често при програмирането не е необходимо цялото множество от стойности на даден тип данни. В тези случаи и цилисъобразно да се използва типът подобласт. Този тип също е скаларен и се дефинира от програмиста. Важно е да се знае, че константите, които използва програмиста, за да дефинира типа трябва да са от един и същи тип. Възможните типове на константите са булев, цял, символен, изброен. Константите, участващи в дефиницията на този тип се наричат горна и долна граница. Задължително е стойността на долната граница да е по-малка или равна на стойността на горната граница. Типа, от който са константите се нарича базов тип на типа подобласт.

Пример за деклариране на тип подобласт:

```
Type
    Nom = 1..35;
    Symb = 'a'..'m';
```

Така множеството от стойности на този тип се състои от всички константи от долната до горната граница.

Декларацията на променливи от тип подобласт води до отделянето на по-малко оперативна памет, отколкото за самия тип. Затова програмистите препоръчват използването на този тип където е възможно.