3. ПРОГРАМИ С РАЗКЛОНЕНА СТРУКТУРА

Алгоритмите много често са разклонени. Разклоненията в програмите се описват чрез условния оператор и оператора за избор на вариант, които ще разгледаме по-долу.

3.1. Условен оператор

Условният оператор има две форми - пълна и кратка.

3.1.1. Пълна форма на условния оператор

Тя има вида

If *Логически израз*

then *Оператор1*

else *Оператор2;*

Тази форма на условния оператор се изпълнява по следния начин:

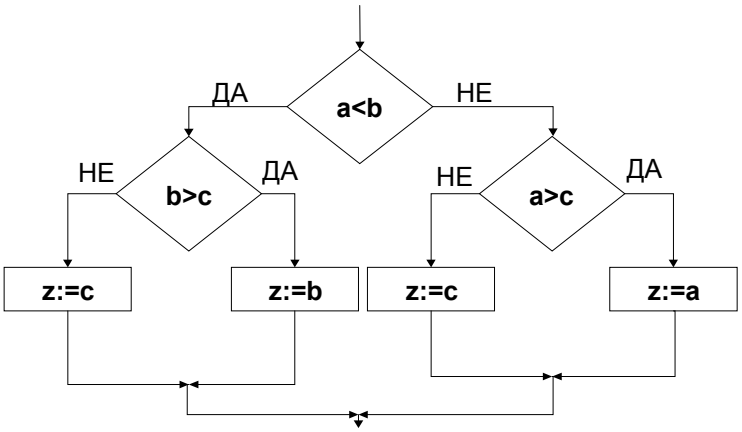
- намира се стойността на логическия израз;
- ако намерената логическа стойност е истина тогава (**then**) се изпълнява *Оператор1*, а в противен случай (**else**) се изпълнява *Оператор2*.

Един пример за условен оператор е следният:

```
If A > B
  then  Z := A
  else  Z := B;
```

Ако стойността на A е по-голяма от стойността на B, на Z се присвоява стойността на A, а ако стойността на A е по-малка или равна на стойността на B, на Z се присвоява стойността на B.

Нека да разгледаме още един пример за условен оператор. Той описва блок-схемата, показана на и присвоява на Z най-голямата от стойностите на A, B и C.



Фиг.3.1

```
If A < B
  then  If B > C
          then   Z := B
          else   Z := C
  else  If A > C
          then   Z := A
```

```
else   Z := C;
```

От примера се вижда, че *Оператор1* и *Оператор2* могат да бъдат също условни оператори. *Оператор1* и *Оператор2* могат да бъдат и съставни оператори, например:

```
If X>Y
  then  begin
          A := X; B := sqr(X); C := sqrt(X)
        end
  else  begin
          A := exp(X); B := exp(-X); C := sin(X)
        end;
```

3.1.2. Кратка форма на условния оператор

Тя има вида

If *Логически израз* **then** *Оператор;*

Изпълнява се по следния начин:

- намира се стойността на логическия израз;
- изпълнява се *Оператор*, само ако стойността на логическия израз е true.

Както и по-горе *Оператор* може да бъде и съставен оператор.

Например:

```
If X>0
  then begin
         A := X;  B := sqr(X);  C := sqrt(X)
       end;
```

Ако X>0, ще се изпълни съставния оператор и с това ще приключи изпълнението на условния оператор. Ако X<=0, изпълнението на оператора ще приключи без да бъде изпълнен съставния оператор.

Нека да разгледаме сега и следните два оператора:

If *Логически израз 1*

then If *Логически израз 2*

then *Оператор 1*

else *Оператор 2;*

If *Логически израз 1*

then If *Логически израз 2*

then *Оператор 1*

else *Оператор 2;*

Ние ги възприемаме като различни, защото първият оператор се състои от външен **if** в съкратена форма и вътрешен **if** в пълна форма. При втория оператор е точно обратното. Компилаторът на Паскал обаче ги третира като два еднакви оператора, защото при интерпретацията им съблюдава следното правило за вложените оператори **if**: “Всяка служебна дума **else** се отнася към най-близката преди нея служебна дума **if**, с която може да образува условен оператор в пълната форма.” Съгласно това правило двата оператора ще бъдат възприети така както е написан първият. Вторият оператор, за да се възприеме като различен от първия, трябва да бъде записан по следния начин:

If *Логически израз 1*

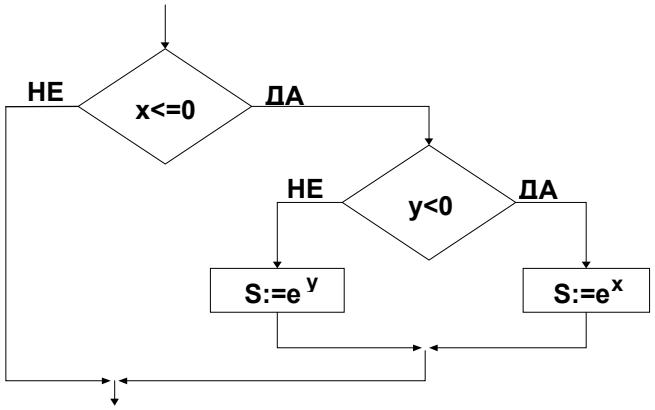
then ***begin***

If *Логически израз 2*

then *Оператор 1*

end

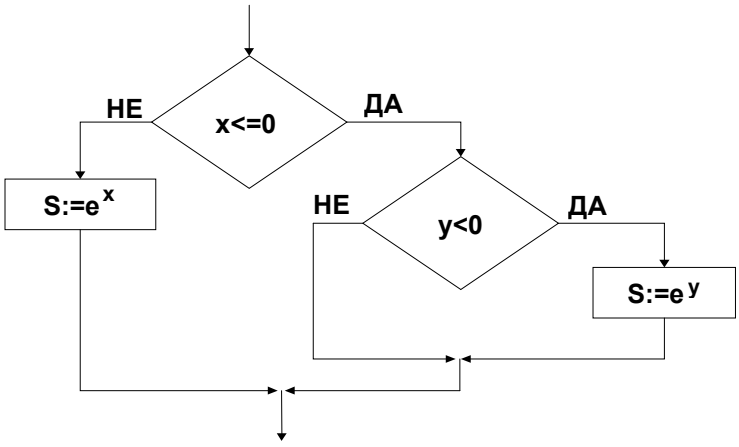
else *Оператор 2;*



Фиг.3.2

Например на блок-схемата на Фиг.3.2 съответства оператора:

```
If X <= 0
  then  If Y < 0
      then S := exp(X)
      else S := exp(Y);
```



Фиг.3.3

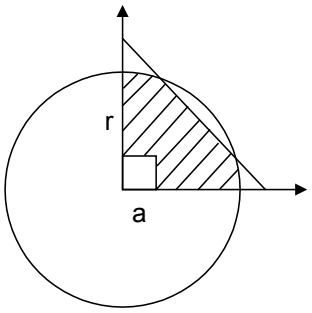
Докато на блок-схемата на Фиг.3.3 съответства оператора:

```
If X <= 0
   then  begin
            If Y < 0 then S := exp(Y)
         end
   else S := exp(X);
```

Пример 3.1. Програма, която проверява дали три отсечки със зададени дълж ини образуват триъгълник.

```
Var
  A, B, C:real; L:boolean;
Begin
  Write('Задайте дължината на първата отсечка: '); Readln(A);
  Write('Задайте дължината на втората отсечка: '); Readln(B);
  Write('Задайте дължината на третата отсечка: '); Readln(C);
  L:=(A+B>C ) and (B+C>A) and (C+A>B);
  If L then Writeln('Отсечките образуват триъгълник.')
     else Writeln('Отсечките не образуват триъгълник.')
End.
```

Пример 3.2. Програма, която проверява дали точка със зададени координати x и y лежи в защрихованата област:



Една точка лежи в защрихованата област, когато са изпълнени следните условия:

1. Лежи в кръга с радиус r, т.е. координатите й удовлетворяват условието sqr(x)+sqr(y)<=sqr(r).

2. Лежи в първия квадрант, т.е. координатите й удовлетворяват условията x>=0 и y>=0.

3. Лежи под правата с уравнение x+y=2, т.е. координатите й удовлетворяват условието y<=2-x.

4. Не лежи в квадрата със страна a, т.е. координатите й удовлетворяват условието **not** ((x>0) **and** (x<a) **and** (y>0) **and** (y<a)).

Горните четири условия ще са удовлетворени, ако е удовлетворено следното съставно условие:

```
(sqr(x)+sqr(y)<=sqr(r)) and (y<=r-x) and (x>=0) and (y>=0)
and not((x>0) and (x<a) and (y>0) and (y<a))
```

```
Var
  x,y,r,a:real;
Begin
  r:=1.5;
  a:=0.5;
  Write('x=');Readln(x);
  Write('y=');Readln(y);
  If (sqr(x)+sqr(y)<=sqr(r)) and (y<=r-x) and (x>=0) and (y>=0) and
       not ((x>0) and (x<a) and (y>0) and (y<a))
    then Writeln('Точката лежи в защрихованата област')
    else Writeln('Точката не лежи в защрихованата област');
  Readln
End.
```

Пример 3.3. Програма, която изчислява разликата между два ъгъла, като се знае, че G1, M1 и S1 са градусите, минутите и секундите на по-малкия ъгъл, G2, M2 и S2 са градусите, минутите и секундите на по-големия ъгъл, а G, M и S са градусите, минутите и секундите на разликата.

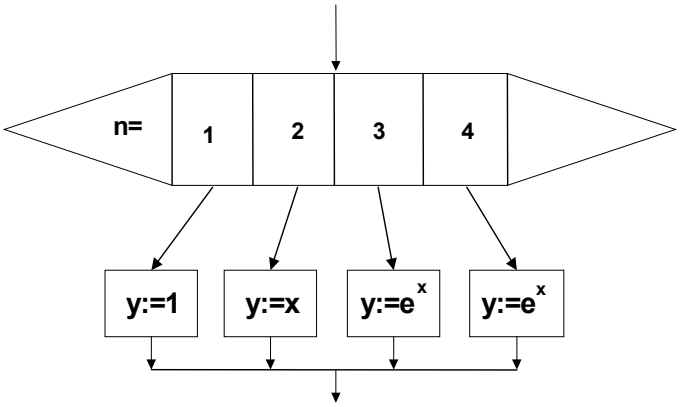
```
Var
  G1, M1, S1, G2, M2, S2, G, M, S:integer;
Begin
{Въвеждане на данните}
  Writeln('Задайте по-малкия ъгъл:');
  Write('Градуси: '); Readln(G1);
  Write('Минути: '); Readln(M1);
  Write('Секунди: '); Readln(S1);
  Writeln('Задайте по-големия ъгъл:');
  Write('Градуси: '); Readln(G2);
  Write('Минути: '); Readln(M2);
  Write('Секунди: '); Readln(S2);
{Изчисляване на разликата}
  If S1>S2
    then begin {Превръщане в секунди на една минута от по-големия ъгъл}
      M2:=M2-1;
      S2:=S2+60
    End;
  S:=S2-S1;
  If M1>M2
    then begin { Превръщане в минути на един градус от по-големия ъгъл }
      G2:=G2-1;
      M2:=M2+60
    End;
  M:=M2-M1;
  G:=G2-G1;
{Извеждане на резултата}
  Writeln('Разликата между ъглите е:');
  Writeln('Градуси: ', G);
  Writeln('Минути: ', M);
  Writeln('Секунди: ', S);
End.
```

3.2. Оператор за избор на вариант

Условният оператор е удобен за описване на разклонения с два клона. В алгоритмите обаче се срещат и разклонения с повече от два клона. Например алгоритъмът за пресмятане на функцията:

$$y(x,n)=\begin{cases}1, & n=1\\ x, & n=2\\ e^{x}, & n=3\\ e^{-x}, & n=4\end{cases}$$

при различни стойности на аргументите X и n е разклонение с четири клона (Фиг.3.4).



Фиг.3.4

Такова разклонение може да се опише с помощта на условния оператор например по един от следните два начина:

```
If N = 1 then Y := 1;
If N = 2 then Y := X;
If N = 3 then Y := exp(X);
If N = 4 then Y := exp(-X);
```

или

```
If N = 1
  then Y := 1
  else If N = 2
         then Y := X
         else If N = 3
                        then Y := exp(X)
                        else If N = 4
                        then Y := exp(-X);
```

И двете описания обаче не са задоволителни от гледна точка на прегледност и бързодействие. Такива разклонения се описват много по-удобно с помощта на *оператор за избор на вариант*. Той има вида:

Case *Селектор* ***of***

C1 : *Оператор 1;*

C2, C3 : *Оператор 2;*

. . . . . .

Cn : *Оператор n;*

else *Оператор*

End*;*

Селектор е израз от дискретен тип. *C1, C2,..., Cn* са константи от същия дискретен тип, от който е и селектора. *Оператор , Оператор 1, . . . ,Оператор n* могат да бъдат и съставни оператори. Изпълнението на оператора се заключава в следното:

- изчислява се стойността на *селектор;*
- изчислената стойност се сравнява с константите C1, C2, ... , Cn;
- изпълнява се онзи оператор, който е предшестван от константа, съвпадаща със стойността на селектора или оператора от клона **else,** ако

селекторът не съвпада с нито една от константите C1. . Cn.

Клонът **else** не е задължителен и ако той отсъства и нито една константа C1, C2, ... , Cn не съвпада със стойността на селектора, се преминава към следващия оператор след оператора **Case**.

Очевидно сред константите C1, C2, C3, ..., Cn не бива да има еднакви. Примерът, който разгледахме по-горе, сега вече можем да опишем по следния начин:

```
Case N of
  1 : Y := 1;
  2 : Y := X;
  3 : Y := exp(X);
  4 : Y := exp(-X)
end;
```

Този начин очевидно е много по-прегледен и по-ефективен при изпълнение.

Пример 3.4. Програма, която дава броя на дните в месец по зададен номер на същия и годината, когато номерът на месеца е 2. Програмата отчита, че високосни са годините, които се делят без остатък на 400 или се делят на 4 без остатък, но се делят на 100 с остатък.

```
Var
  BrDni, NomMes, God : integer;
Begin
  Write('Въведете номера на месеца: '); Readln(NomMes);
  Case NomMes of
  1,3,5,7,8,10,12:   BrDni:=31;
  4,6,9,11:          BrDni:=30;
  2:  begin
        Write('Въведете годината: '); Readln(God);
        If (God mod 400=0) or (God mod 4=0) and (God mod 100<>0)
          then BrDni:=29
          else BrDni:=28
      end
  end;
  Writeln('Месецът има ', BrDni,' дни.');
  Readln;
End.
```

3.3. Оператор за безусловен преход

Операторът за безусловен преход има вида:

goto *етикет;*

като етикетите се описват в раздела **Label** на описателната част.

След този оператор се изпълнява оператора с посочения етикет, а не следващият след него. Например след оператора

goto 25;

се преминава към оператора с етикет 25. Етикетът се поставя пред оператора, за който е предназначен, и се отделя от него с двоеточие (:).

Структурните алгоритми се описват на Паскал, без да се налага да се използва оператора **goto**. Ето защо се смята, че той е спасителна сламка за програмистите, които не могат да създават структурни алгоритми и структурни програми. Програмите, използващи **goto**, като правило, са много непрегледни.