13. РЕКУРСИВНИ ПОДПРОГРАМИ

Рекурсивни са подпрограмите, които се обръщат към себе си, т.е. съдържат в тялото си обръщение към себе си. Обръщението в този случай е пряко. Възможно е и непряко обръщение, например подпрограмата P се обръща към подпрограмата Q, а Q се обръща към P.

Нека да проследим как работи дадената по-долу програма с рекурсивна процедура. Това ще ни помогне да разберем как изобщо работи програма, съдържаща рекурсивна подпрограма, а като разберем това, значително по-лесно ще можем да съставяме рекурсивни подпрограми.

Програма 13.1.

```
Var
  M:byte;
Procedure Recurs(N:byte);
  Begin
    Writeln('Донесете ',N,' бири, моля!');
    If N > 1 then Recurs(N -1);
    Writeln('   Плащам ',N,' бири, моля!');
  End;
Begin
  Write('M='); Readln(M);
  Recurs(M);
  Readln
End.
```

В резултат на изпълнение на тази програма при M=3, на екрана ще се появи следният текст:

```
Донесете 3 бири, моля!
Донесете 2 бири, моля!
Донесете 1 бири, моля!
   Плащам 1 бири, моля!
   Плащам 2 бири, моля!
   Плащам 3 бири, моля!
```

При компилирането на програмата в областта на статичните обекти от ОП се определя клетка за целочислената променлива M.

Изпълнението на програмата започва от първия оператор на главната програма. Въвежда се стойността на M. За определеност ще приемем, че тя е 3.

Следващият оператор в главната програма представлява обръщение към процедурата Recurs. В изпълнение на това е **първо** обръщение към Recurs в системния стек се обособява “**първи етаж**”, където се определя клетка за фиктивния параметър N и в нея постъпва стойността на действителния аргумент M, т.е. числото 3. Започва **първото** изпълнение на операторите от изпълнимата част на процедурата Recurs. Първият оператор извежда съобщението "Донесете 3 бира, моля". Вторият оператор предизвиква **второ** обръщение към Recurs, защото N >1. С това в системния стек се обособява “**втори етаж**”, където отново се определя клетка за фиктивния параметър N. В клетката на N постъпва стойността на израза N-1, която е 2, защото в този израз участва N от “първия етаж”, което има стойност 3.

Започва ново (**второ**) изпълнение на операторите от изпълнимата част на процедурата Recurs, но сега с N=2. Първият оператор извежда съобщението "Донесете 2 бири, моля", а вторият оператор предизвиква **трето** обръщение към Recurs, защото и сега N>1. С това в системния стек се обособява “**трети етаж**”,

където отново се определя клетка за фиктивния параметър N. В клетката на N постъпва стойността на израза N-1, която е 1, защото в този израз участва N от “втория етаж”, чиято стойност е 2.

Започва ново (**трето**) изпълнение на операторите от изпълнимата част на процедурата Recurs, но сега с N=1. Първият оператор извежда съобщението "Донесете 1 бири, моля". Вторият оператор този път няма да предизвика ново обръщение към Recurs, защото и сега N=1. С това се прекратяват рекурсивните обръщения и започва довършването на направените до тук обръщения. Те са недовършени, защото последният оператор "Write('Плащам ',N,' бири, моля!');" не е изпълнен при нито едно от тях.

Най-напред се довършва третото обръщение, т.е. извеждане на съобщението “Плащам 1 бири, моля”, излизане от процедурата и освобождаване на третия етаж от стека.

Следва довършване на второто обръщение, т.е. извеждане на съобщението “Плащам 2 бири, моля”, излизане от процедурата и освобождаване на втория етаж от стека.

Следва довършване на първото обръщение, т.е. извеждане на съобщението “Плащам 3 бири, моля”, излизане от процедурата и освобождаване на първия етаж от стека.

Следва довършване на главната програма.

Проследяването на изпълнението на горната програма дава основание да направим следните важни констатации:

1. Изпълнението на рекурсивна подпрограма прилича на изпълнението на цикъл, но има някои съществени разлики:

- вместо условие за излизане от цикъла, тук има условие за прекратяване обръщенията към рекурсивната подпрограма;
- докато условието за излизане от цикъла може да се намира пред или след тялото на цикъла, обръщението към рекурсивната подпрограма може да е на всяко място в изпълнимата й част.
- при циклите цялото тяло на цикъла се изпълнява определен брой пъти, докато изпълнението на рекурсивна подпрограма може да се раздели на две фази:

I-ва фаза - тя се състои в последователни обръщения на подпрограмата към себе до удовлетворяване на условието за прекратяване на тези обръщения. При всяко обръщение се изпълняват само операторите й до условието за спиране на обръщенията.

II-ра фаза - тя започва след като се удовлетвори условието за спиране на обръщенията и се състои в последователно довършване на направените (започнатите, но недовършени) обръщения. Те обаче се довършват в ред, обратен на реда, в който са започнати.

2. Рекурсивната подпрограма трябва да съдържа фиктивен параметър-стойност, който участва в условието за прекратяване на обръщенията и в израза действителен параметър на рекурсивното обръщение.

3. При всяко обръщение към рекурсивната подпрограма в системния стек се формира нов “етаж”, в който се отделят клетки за локалните променливи и фиктивните параметри на рекурсивната подпрограма.

4. Когато за изпълнението на даден оператор от подпрограмата е потребна стойността на дадена променлива, тя се търси в стека, като търсенето започва от върха на стека. Ако не бъде намерена в стека, търсенето продължава в областта на статичните обекти.

Като имаме предвид направените изводи, може да заменим всеки цикъл с рекурсия. Това обаче значително се улеснява, ако самото решение се формулира не като итеративен процес, а като рекурсия. Ще разгледаме няколко примера, от които може да се види как става това.

Програма 13.2. Програма с рекурсивна функция за изчисляване на n!.

Дефиницията

$$n! = \begin{cases} 1 & \text{, ако } n=0 \\ n.(n-1).(n-2). \ ... \ .2.1 & \text{, ако } n>0, \end{cases}$$

описва функцията факториел (n!) като итеративен изчислителен процес, защото получаването на n! предполага многократно повтаряне на едни и същи изчислителни операции (P:=P*i и i:=i+1).

По-особена е следната дефиниция за n!

$$n! = \begin{cases} 1 & \text{, ако } n=0 \\ n.(n-1)! & \text{, ако } n>0, \end{cases}$$

защото в дефиницията на функцията факториел участва самата функция факториел, но за по-прост случай ((n-1)! е по-прост случай от n! на една и съща задача). А дефиниция, чрез която един обект се пределя чрез по-прост подобен на себе си обект се нарича ***рекурсивна дефиниция***.

В съответствие с тази рекурсивна дефиниция за n! можем да напишем следната програма с рекурсивна функция:

```
Var
  M:integer;
Function Fact ( N:integer):integer;
  Begin
    If N=0
    then Fact :=1
    else Fact :=N*Fact(N-1)
  End;
Begin
  Write('Въведете число: ');Readln(M);
  Writeln('   ',M,'!=',Fact(M));
  readln
End.
```

Програма 13.3. Програма с рекурсивна функция на Паскал за изчисляване на функцията $f(n) = x^n$, където n е цяло, а x>0.

Рекурсивната дефиниция на дадената функция има вида:

$$f(x,n) = \begin{cases} 1 & \text{при } n=0 \\ x \ . \ f(x,n-1) & \text{при } n>0. \end{cases}$$

Съответната програма с рекурсивна функция е:

```
Var
  a:real; k:integer;
Function Stepen(X:real;N:integer):real;
  Begin
    If N=0
    then Stepen:=1
    else Stepen:=X * Stepen(X,n-1)
```

```
  End;
Begin
  Write('Въведете основата: ');Readln(a);
  Write('Въведете степенния показател: ');Readln(k);
  Writeln('Резултат: ',Stepen(a,k):8:3);
  readln
End.
```

Програма 13.4. Програма с рекурсивна функция за получаване на n-тото число от редицата на Фибоначи (n=0,1, 2, . . .).

Редицата от числа 0, 1, 1, 2, 3, 5, 8, 13, . . . е известна под името редица на Фибоначи. Трябва да се състави програма с рекурсивна функция за получаване на n-тото число от редицата.

Можем да опишем рекурсивен процес за получаване на n-тото число от редицата по следния начин:

$$f(n) = \begin{cases} n & \text{ако } n=0 \text{ или } n=1, \\ f(n-2) + f(n-1) & \text{ако } n>=2. \end{cases}$$

Сега вече лесно можем да напишем следната програма с рекурсивна функция за получаване на n-тото число от редицата на Фибоначи.

```
Var
  k :integer;
Function Fibonaci (N:integer):integer;
Begin
    If (N=0) or (N=1)
    then Fibonaci:=N
    else Fibonaci:=Fibonaci (N-2) + Fibonaci (N-1)
  End;
Begin
  Write('Въведете номера на търсеното число: ');Readln(k);
  Writeln('   ',k,'-тото число на Фибоначи е: ',Fibonaci(k));
  readln
End.
```

От разгледаните примери се вижда, че рекурсията е чудесно средство за създаване на много кратки, много прегледни и много изящни функции. Те обаче имат и някои недостатъци.

Нека да си припомним, че при обръщане към подпрограма действителните параметри и локалните обекти се настаняват в системния стек и остават там до излизане от подпрограмата. При рекурсивните подпрограми обаче могат се реализират твърде много обръщения, системният стек да се препълни и да се провали решението на задачата.

Програма 13.5. Програма с рекурсивна функция за намиране сумата на елементите на едномерен масив.

Можем да опишем рекурсивния процес за получаване сумата на елементите на едномерен масив по следния начин:

$$S(n)= \begin{cases} A[1] & \text{при } n=1; \\ A[n]+S(n-1) & \text{при } n>1. \end{cases}$$

Сега вече лесно можем да напишем на Паскал следната рекурсивна функция за получаване на сумата на елементите на едномерен масив.

```
Function SumaRec(n:integer):real;
```

```
Begin
  If n=1
    then SumaRec:=A[1]
    else SumaRec:=A[n]+SumaRec(n-1)
End;
```

В тази рекурсивна функция A е несобствена променлива от тип масив. Това означава, че можем да я използване за намиране сумата на елементите само на масив с име A. Това е неудобство, от което можем да излезем, като включим A в списъка на фиктивните параметри, защото ще можем да заместим A с всеки масив от същия тип. Но тогава се натъкваме пък на друго неудобство - при всяко обръщане към функцията масивът, като действителен параметър, се настанява в системния стек и бързо може да го препълни. За да се избегне и този проблем рекурсивните подпрограми се правят на две нива както е показано в следващата програма.

```
Type
  Mas=array[1..10] of real;
Var
  i,n:byte; S:real; A:mas;
Function SumElMas(n:integer;A:mas):real;
  Function SumaRec(n:integer):real;
    Begin
      If n=1
      then SumaRec:=A[1]
      else SumaRec:=A[n]+SumaRec(n-1)
  End;
  Begin
    SumElMas:=SumaRec(n)
  End;
Begin
  Write('n=');Readln(n);
  For i:=1 to n do readln(A[i]);
  S:=SumElMas(n,A);
  Writeln(S:7:2);
  Readln
End.
```

Програма 13.6. Програма с непряка рекурсия за намиране на най-големия елемент в едномерен масив.

Можем да опишем рекурсивния процес за намиране на най-големия елемент в едномерен масив по следния начин:

$$\text{MaxElem}(n)=\begin{cases} A[1] & \text{при } n=1; \\ \max2(A[n], \text{MaxElem}(n-1)) & \text{при } n>1, \end{cases}$$

където max2 означава по-голямата от двете стойности в скобите. В съответствие с това описание можем да напишем следната програма:

```
Type
  Mas=array[1..10] of real;
Var
  i,n:byte;P:real;A:mas;
Function MaxElem(n:integer;A:mas):real;
  Function MaxRec(n:integer):real;
    Function Max2(X,Y:real):real;
      Begin
```

```
        If X>Y
        then Max2:=X
        else Max2:=Y
      End;
  Begin
    If n=1
      then MaxRec:=A[1]
      else MaxRec:=Max2(A[n], MaxRec(n-1))
  End;
  Begin
    MaxElem:=MaxRec(n)
  End;
Begin
  Write('n=');Readln(n);
  For i:=1 to n do readln(A[i]);
  P:=MaxElem(n,A);
  Writeln('Максималната стойност в масива е ',P:7:2);
  Readln
End.
```

По причините, които изяснихме при предната програма, рекурсивната функция е вложена в нерекурсивна. Освен това сега рекурсията е непряка. Защо?

Програма 13.7. Програма с рекурсивна функция за намиране на най-големия елемент в едномерен масив.

```
Type
  Mas=array[1..10] of integer;
Var
  i,m:integer;B:mas;
Function MaxElMas(n:integer;Var A:mas):integer;
  Var
    P:integer;
  Function MaxElRec(n:integer):integer;
    Begin
      If n=1
         then MaxElRec:=A[1]
         else begin
              P:=MaxElRec(n-1);
              If A[n]>P
                 then MaxElRec:=A[n]
                 else MaxElRec:=P
            end;
    End;
  Begin
    MaxElMas:=MaxElRec(m)
  End;
Begin
  Write('m=');Readln(m);
  For i:=1 to m do begin Write('i=',i,'  '); Readln(B[i]) end;
  Writeln(MaxElMas(m,B));
  Readln
End.
```

Програма 13.8. Рекурсивна функция на Паскал за изчисляване стойността на полином от n-та стрепен

$$P(n) = a_o x^n + a_1 x^{n-1} + a_2 x^{n-2} + \ldots + a_{n-1} x + a_n.$$

Този полином може да се представи по следния начин:

$$P(n) = (\ldots ((a_o x + a_1)x + a_2)x + \ldots + a_{n-1})x + a_n.$$

На това представяне на полинома (то е на Хорнер) съответства следната схема за изчисляването му:

$$P(0) = a_o,$$
$$P(i) = x.P(i-1) + a_i \qquad \text{за } i=1,2,\ldots,n.$$

Рекурсивният процес може да се опише по следния начин:

$$Pn = \begin{cases} a_0 & \text{при } n=0, \\ x.P_{n-1} + a_n & \text{при } n>0. \end{cases}$$

Програмата с рекурсивна функция за изчисляване стойността на полинома от n-та стрепен ще бъде следната:

```
Type
  msv=array[0..10] of real;
Var
  i,m:integer; X:real;B:msv;
Function PolHorn(x:real;n:integer;var A:msv):real;
  Function P(n:integer):real;
    Begin
      If n=0
         then P:=A[0]
         else P:=P(n-1)*X+A[n]
    End;
Begin
    PolHorn:=P(n)
End;
Begin
  Write('m='); readln(m);
  For i:=0 to m do
    begin Write('b',i,'='); readln(B[i]) end;
  Write('X='); readln(X);
  Writeln(PolHorn(X,m,B):10:5);
  Readln
End.
```

Тук X е аргумент, а A[0], A[1], . . . , A[n] са коефициентите на полинома.

Разгледахме достатъчно рекурсивни функции, които написахме след като най-напред формулирахме процеса като рекурсивен. Това обаче не винаги е лесно. В такива случаи трябва да се възползваме от изводите, формулирани след програма 13.1.

Програма 13.9. Рекурсивна процедура, която създава масив, съдържащ цифрите на числото.

Нека да започнем с отделянето на цифрите на едно конкретно число, например 25387. При последователно целочислено деление на 10 ще получим следната картина за частните и за остатъците:

```
25387  →  2538  →  253  →  25  →  2  →  0
  ↓        ↓        ↓       ↓      ↓
  7        8        3       5      2
```

Както се вижда от примера, първата цифра (2) се получава като остатък при последното целочислено деление на 10, втората цифра (5) - като остатък при предпоследното деление и т. н.

Очевидно, за да получим цифрите в реда, в който те присъстват в числото, трябва най-напред да намерим и запомним частните и едва след това от тях в обратен ред да получим цифрите.

При тези уговорки програмата има вида:

```
Type
  Mas=array[1..10] of byte;
Var
  P:longint; i,n:byte; A:mas;
Procedure RekCif(Cislo:longint;Var BrCif:byte;Var Cifri:mas);
  Procedure Rec(Cislo:longint);
    Begin
      If Cislo>9 then Rec(Cislo div 10);
      BrCif:=BrCif+1; Cifri[BrCif]:=Cislo mod 10
    End;
Begin
  BrCif:=0; Rec(Cislo)
End;
Begin
  Repeat
    Write('Въведете число /0-за край/: ');Read(P);
    Rekcif(P, n, A);
    For i:=1 to n do Write(A[i],' '); writeln;Readln
  until P=0
End.
```

Процедурата Rec се състои само от три оператора. Първият (**If** Cislo ...) реализира последователни обръщения на процедурата към себе си докато числото стане едноцифрено. При всяко от тези обръщения в системния стек постъпва като стойност на фиктивния параметър Cislo - при първото обръщение - първоначалното число, при второто обръщение - първоначалното число без последната цифра, при третото обръщение - първоначалното число без последните две цифри, ... и накрая само първата цифра на първоначалното число. Следващите два опратора дават номер на цифрата, изчисляват я, и я присвояват на елемент на масива.

Преместете операторите BrCif:=BrCif+1; Cifri[BrCif]:=Cislo **mod** 10 на процедурата Rec пред оператора **If** Cislo>9 **then** Rec(Cislo **div** 10). Вижте резултата от промяната. Обяснете го.

Програма 13.10. Програма с рекурсивна функция за обръщане реда на цифрите в дадено цяло число

```
Var
  K,L:longint;
Function ObrCislo(A:longint):longint;
  Var
    N:longint;
  Function ObrRed(M:longint):longint;
    Var
      C:byte;
    Begin
      C:=M mod 10;
      N:=N*10+C;
```

```
      If M>9
        then ObrRed:=ObrRed(M div 10)
        else ObrRed:=N
    End;
  Begin
    N:=0;
    ObrCislo:=ObrRed(A)
  End;
Begin
  Write('Задайте числото: ');Readln(L);
  K:=ObrCislo(L);
  Writeln('Обърнатото число е ',K);
  Readln
End.
```

Програма 13.11. Програма с рекурсивна функция за разделяне цифрите на дадено цяло число и възстановяване на числото

```
Var
  K,L:longint;
Function ObrCislo(A:longint):longint;
  Function ObrRed(M:longint):longint;
    Var
      C:byte;
    Begin
      C:=M mod 10;
      If M>9
        then ObrRed:=ObrRed(M div 10)*10 + C
        else ObrRed:=M;
    End;
  Begin
    ObrCislo:=ObrRed(A)
  End;
Begin
  Write('Задайте числото: ');Readln(L);
  K:=ObrCislo(L);
  Writeln('Обърнатото число е ',K);
  Readln
End.
```

Програма 13.12. Програма за решаване задачата, известната като задача за Ханойските кули. Задачата е следната:

Дадени са N диска с различни диаметри и три стълба. Началното подреждане на дисковете при N=5 е дадено на Фиг.13.1. Задачата е да се преместят дисковете от първия на последния стълб, като се използва средния стълб като междинен, и се спазват следните ограничения:

а) във всеки момент в процес на преместване може да бъде само един диск и този диск трябва да бъде най-горен за някой от стълбовете. Всеки от останалите дискове трябва да се намира на някой от стълбовете.

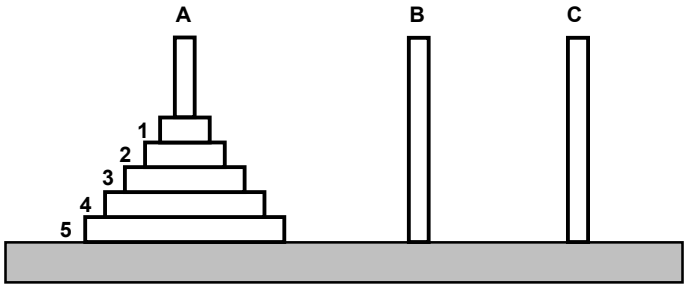
б) диск с даден диаметър не може да бъде поставен върху диск с по-малък диаметър.

Да означим с A, B и C съответно началния, междинния и крайния стълб. Ако е възможно преместването на N диска от стълб A на стълб C, като се използва стълб B като междинен, то трябва да е възможно и преместването на N-1 диска от стълб A на стълб B, като се използва стълб C като междинен. Това е основание да предложим следната стратегия за решаването на задачата:

1. Преместваме горните N-1 диска от стълб A на стълб B, като използваме стълб C като междинен.

2. Преместваме N-ия (най-долния) диск от стълб A на стълб C.

3. Преместваме горните N-1 диска от кула B на стълб C, като използваме стълб A като междинен.



Фиг.13.1

Самият факт, че предложената стратегия предполага свеждане на по-сложния случай (N диска) до по-прост такъв (N-1 диска) подсказва, че програмата трябва да се напраяи с рекурсивна процедура. Такава е програмата по-долу, където с фиктивните параметри A, B, C и променливите A1, B1, C1 са имената на стълбовете.

```
Var
  A1,B1,C1:char; N:integer;
Procedure Kula(A,C,B:char;N:integer);
  Begin
    If N=1
    then Writeln('Премести диск 1 от ',A,' на ',C)
    else begin
       Kula(A,B,C,N-1);
       Writeln('Премести диск ',N:1,' от ',A,' на ',C);
       Kula(B,C,A,N-1)
    end
  End;
Begin
  Repeat
    Write('Въведете броя на дисковете: ');Readln(N)
  until N<10;
  Writeln('Въведете имената на стълбовете:');
  Write('Начален стълб: '); Readln(A1);
  Write('Краен стълб: ');   Readln(C1);
  Write('Междинен стълб: ');Readln(B1);
  Kula(A1,C1,B1,N);Readln
End.
```