Ce sunt *Subprogramele recursive ?*

ÎNVĂŢARE PRIN PROIECTE

“Să învăţăm inteligent”

Să ne amintim

- Ce este un *subprogram* ?
- Care sunt *avantajele* folosirii subprogramelor ?
- Ce sunt *parametrii* unui subprogram ?
- Care sunt *metodele de transfer* al parametrilor ?
- Ce înseamnă *apelul unui subprogram* ?
- Care sunt *clasele de variabile* folosite într-un program ?

Ce vom învăţa

- Ce este *recursivitatea* ?

Recursivitatea este un concept matematic care implică definirea unui concept prin referirea la acelaşi concept.

Astfel, mulţimea numerelor naturale se poate defini:

- 1 este *număr natural*
- orice succesor al unui număr natural este de asemenea un *număr natural*
- În definiţia de mai sus, observăm că avem o valoare iniţială (1), iar restul valorilor se obţin adunând 1 la valoarea anterioară

Definiţii recursive

- Unul dinte cele mai simple exemple de suprograme recursive este cel scris pentru funcţia factorial. Factorialul lui n poate fi definit *recursiv* astfel:

  0! = 1

  n! = n· (n-1)! , pentru n>0

- Dacă notăm cu $f(n)$=n! funcţia are următoarea definiţie recursivă:

$$f(n)=\begin{cases} 1, dacă\ n = 0 \\ n \cdot f(n-1), pentru\ n > 0 \end{cases}$$

Funcţia factorial

- Funcţia factorial este scrisă după definiţia recursivă:

```
int f(int n)
{
    if (n==0) return 1;
   else return n*f(n-1);
}
```

- *apel recursiv*

```
int main()
{  int n;
   cout<<"n="; cin>>n ;
   cout << n <<"! = "<<f(n));
}
```

- *apel iniţial*

n= 3
f(3)=

-fiecare apel adaugă un context nou în stivă

-apelurile recursive se termină când ajungem la rezolvarea directă

-la revenirea din apelul recursiv se goleşte stiva

Recursiv şi iterativ

- Să reluăm funcţia factorial în varianta
  - *recursivă*

```
int f (int n)
{
    if (n==0) return 1;
   else return n*f(n-1);
}
```

  - *iterativă*

```
int f (int n)
{ int i,P=1;
  for (i=1; i<= n; i++) P=P*i;
    return P;
}
```

Ce diferenţe observaţi ?

- Ce *parametri* apar în antet ?
- Ce *variabile locale* sunt declarate ?
- Ce *instrucţiuni* se folosesc ?

O procedură recursivă

- Cum afişăm numerele naturale de la 1 la n printr-o procedură recursivă ?
- *Rezolvare:* În acest caz, nu avem o formulă de recurenţă pentru scrierea unui subprogram recursiv. Pentru a rezolva problema o descompunem în mai multe subprobleme de acelaşi tip:
- *Subproblema p(i)*: pentru valoarea *i* a parametrului vom tipări *i* după care apelăm (recursiv) *p* pentru *i*+1
- *Apelul iniţial* este p(1) : începem cu 1
- *Condiţia de oprire* este să ajungem la n, când tipărim doar n fără alte apeluri

Lista primelor n numere naturale

```
int n ;                          // câte numere tipărim
void p(int i)                    // procedura p cu parametrul i
{
   if (i==n)  cout(n) ;          // rezolvarea directă (condiţia de STOP)
    else {
          cout<<i<<' ';          // Subproblema p(i): tipărim i
           p(i+1);               // trecem la subproblema p(i+1)
           }
}
int main()
{  cout<<“n=“;  cin>>n;          // citim valoarea lui n
  p(1);                          // apelul iniţial
}
```

Ce este un subprogram recursiv ?

- Un ***subprogram recursiv*** se caracterizează prin proprietatea că *se auto-apelează*, adică din interiorul lui se apelează pe el însuşi. Din afara subprogramului facem un *prim apel* al acestuia, după care subprogranul se auto-apelează de un anumit număr de ori: la fiecare nouă auto-apelare a subprogramului, se execută din nou secvenţa de instrucţiuni ce reprezintă corpul său, eventual cu alte date, creîndu-se un aşa-numit *„lanţ de auto-apeluri recursive”*.
- Putem spune că un subprogram recursiv are acelaşi efect ca şi un ciclu: repetă execuţia unei anumite secvenţe de instrucţiuni. Dar, la fel ca în cazul unui ciclu, este necesar ca repetarea să nu aibă loc la infinit. De aceea în corpul subprogramului trebuie să existe cel puţin o testare a unei *condiţii de oprire*, la îndeplinirea căreia se întrerupe lanţul de auto-apeluri.

Cum scriem un subprogram recursiv ?

1. Trebuie să formulăm problema în termeni recursivi
   - *stabilim formula de recurenţă*
   - *identificăm soluţia în cazul rezolvării directe*, dată de obicei sub forma *condiţiilor iniţiale*
   - *formulăm subproblemele* de rezolvat în cazul în care nu dispunem de o formulă de recurenţă
2. Scriem subprogramul recursiv:
   - soluţia directă se scrie sub forma *condiţiei de oprire*
   - *apelul recursiv* rezultă din formula de recurenţă
   - la fiecare apel *problema parţială trebuie rezolvată complet*

Răspundeţi la întrebări

1. Se consideră subprogramul f definit alăturat.
Ce valoare are f (250) ?

```
int f (int x)
{ if (x % 3== 0) return 0;
  else return 1+f(x/3);
}
```

a. 5 b. 2 c. 3 d.4

2. Subprogramul f are definiţia alăturată.
Ce valoare are f (3) ? Dar f (10) ?

```
int f (int x)
{ if (x == 0) return 0;
  else return f(x-1)+2;
}
```

3. Subprogramul f are definiţia alăturată.
Ce valoare are f (4) ? Dar f (11) ?

```
int f (int x)
{ if (x < 1) return 1;
  else return f(x-3)+1;
}
```

Răspundeţi la întrebări

4. Pentru definiţia alăturată a subprogramului `sc`, stabiliţi ce valoare are `sc(10)`.
Dar `sc(901324)` ?

```
int sc (int x)
{ if (x < 10) return x;
  else return sc(x/10)+x%10;
}
```

5. Pentru definiţia alăturată a subprogramului `g`, ce valoare are `g(3)` ? Dar `g(8)` ?

```
int g (int x)
{ if (x <=4) return x*x-3;
  else return g(x-3)+4;
}
```

6. Subprogramul `h` are definiţia alăturată.
Ce valoare are `h(7)` ? Dar `h(100)` ?

```
int h (int x)
{ if (x%6==0) return x;
  else return h(x-1);
}
```

Aţi înţeles ? Încercaţi să rezolvaţi

- Scrieţi subprogramele recursive :
- Calculaţi recursiv sumele:
  a) $S_n = 1 + 2 + \cdots + n$
  b) $1 \cdot 2 + 2 \cdot 3 + \ldots + n \cdot (n+1)$
  c) $1 + 1/2 + \ldots + 1/n$
  d) $1/(2 \cdot 3) + 2/(3 \cdot 4) + \ldots + n/((n+1) \cdot (n+2))$

- Afişarea primelor n numere în ordine descrescătoare