RECURSIVITATE

Recursivitatea inseamna in general proprietatea intrinseca a unei entitati (proces, obiect, fenomen) de a putea fi descrisa, prelucrata, analizata pe baza unor entitati de acelasi tip. Mai simplu spus, in definirea sau prelucrarea unei entitati recursive se fac apeluri sau referiri la insasi entitatea respectiva.

Exemple de definitii matematice recursive:

1) n! = $\begin{cases} 1, dacă\ n = 0 \\ n * (n-1)!, dacă\ n > 0 \end{cases}$

2) $x^n = \begin{cases} 1, dacă\ n = 0 \\ x * x^{n-1}, dacă\ n > 0 \end{cases}$

3) **a) cmmdc(a,b) - algoritmul lui Euclid**

cmmdc(a,b)=$\begin{cases} a, dacă\ b = 0 \\ \text{cmmdc}(b, a\ mod\ b), dacă\ b > 0 \end{cases}$

b) cmmdc(a,b) - scăderi succesive

cmmdc(a,b)= $\begin{cases} a, dacă\ a = b \\ cmmdc(a-b, b), dacă\ a > b \\ cmmdc(a, b-a),\ dacă\ b > a \end{cases}$

4) **suma cifrelor unui numar natural n**

sumcif(n)= $\begin{cases} n, dacă\ n < 10 \\ n\%10 + sumcif\,(n/10), dacă\ n \geqslant 10 \end{cases}$

Exemplu:
s(123) =s(12) +3 = 1+2+3
s(12) = s(1) + 2 = 1+2
s(1) = 1

5) **şirul lui Fibonacci**

f(n) = $\begin{cases} 1, dacă\ n = 1\ sau\ n = 2 \\ f(n-1) + f(n-2), dacă\ n > 2 \end{cases}$

Obs. Nerecomandata, fiind recursivitate “in cascada” !

6) **al n-lea termen al unei progresii aritmetice de raţie r si cu primul termen a1**

a(n) = $\begin{cases} a1, dacă\ n = 1 \\ a(n-1) + r, dacă\ n > 1 \end{cases}$

Recursivitatea este si un mecanism general de elaborare a programelor. Ea a aparut din necesitatea transcrierii directe a formulelor matematice recursive.

Definitie. **Recursivitatea** este acel mecanism prin care un subprogram (functie) se autoapeleaza.

Exista doua tipuri de subprograme recursive:

a) direct recursive
b) indirect recursive (mutual recursive)

Un subprogram este *direct recursiv*, daca in cadrul corpului sau se intalneste un apel la el insusi (**se autoapeleaza in corpul sau**).

Daca subprogramul P face apel la subprogramul Q, care la randul sau apeleaza subprogramul P, atunci cele doua se numesc *indirect recursive*. Cu alte cuvinte subprogramul P face apel la el insusi prin intermediul subprogramului Q.

Recursivitate directa

Exemple de subprograme direct recursive:

1. Functie recursiva pentru calculul lui n!.

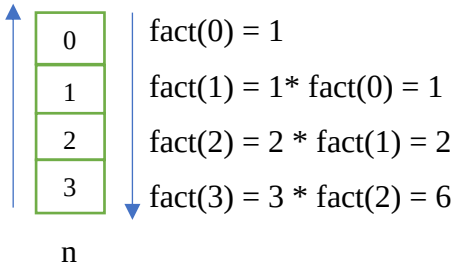
```
int fact(int n)
{
  if (n==0) return 1;
  else return n*fact(n-1)
}
```

Exemplu: apel cu fact(3)

a) Mecanismul recursivitatii - cu lant de autoapeluri

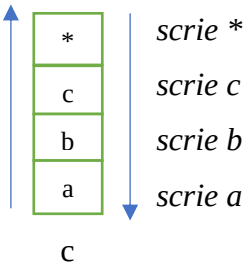
fact(3) = 3 * fact(2) = 6
fact(2) = 2 * fact(1) = 2
fact(1) = 1* fact(0) = 1
fact(0) = 1

b) Mecanismul recursivitatii - cu stiva



2. Sa se scrie o functie recursiva care citeste caractere si le afiseaza in ordinea inversa citirii, stiind ca citirea se incheie cu caracterul '*'. Nu se va lucra cu string-uri sau tablouri.

```
void inversare()
{ char c;
  cin>>c;
  if  (c!='*')  inversare();
  cout<<c;
}
```



Observatii:

1. Un algoritm recursiv corect trebuie contina obligatoriu o *conditie de terminare*, care opreste procesul recursiv. De exemplu, pentru calculul lui n! conditia de terminare este cand n=0.
2. In general structura unui subprogram recursiv contine:
3. *cazul de baza* – o operatie care rezolva problema pe un caz particular, cand s-a ajuns la conditia de terminare. Obigatoriu, el nu va contine alt autoapel.
4. *cazul general* – care consta din toate prelucrarile necesare reducerii dimensiuniii problemei (pentru a se ajunge la cazul de baza). Cea mai importanta operatie o reprezinta autoapelul.
5. Intreruperea autoapelurilor si trecerea la cazul de baza este posibila numai la indeplinirea conditiei de oprire.
6. Pentru orice algoritm iterativ exista un algoritm recursiv echivalent si invers.
7. Se numeste *adancimea recursivitatii*, numarul de autoapeluri succesive ale unui subprogram. Marimea adancimii recursivitatii determina eficienta unui subprogram recursiv.

Mecanismul recursivitatii

Se stie ca executia "linie cu linie" a unui bloc este intrerupta la intalnirea unei instructiuni de apel. Ea conduce la parasirea blocului curent pentru efectuarea tuturor instructiunilor cuprinse in subprogramul apelat. La finalul acestei operatii se revine in blocul initial si se continua cu instructiunea imediat urmatoare celei de apel.

Cum se realizeaza controlul acestui proces la nivelul memoriei interne a calculatorului?

In memorie exista o zona numita *stiva*, care permite salvarea temporara a instantei programului/subprogramului apelat. Stiva este o structura de date in care toate operatiile (adaugare/extragere) se fac la un singur capat , numit varful stivei. Ea se mai numeste lista LIFO (**L**ast **I**n **F**irst **O**ut).

Instanta unui program/subprogram cuprinde:

- *adresa de revenire*
- *contextul programului/subprogramului.*

Adresa de revenire reprezinta adresa primei instructiuni care trebuie executata la revenirea din apel, adica dupa incheierea executiei subprogramului apelat.

Contextul subprogramului cuprinde:

- valorile parametrilor de tip valoare
- adresele parametrilor de tip adresa
- valorile tuturor variabilelor locale.

Si in cazul unui subprogram recursiv, stiva va servi la salvarea temporara a instantei subprogramului, ori de cate ori se intalneste un autoapel. Astfel, la fiecare nou autoapel, in stiva se creaza un nou nivel in care se depun valorile parametrilor valoare, adresele parametrilor variabila si valorile variablilelor locale (are loc incarcarea stivei). Odata indeplinita conditia de terminare se revine din autoapeluri la instructiunea care urmeaza acestora (adresa de revenire) si incepe descarcarea stivei. Aceasta instructiune va lucra cu valorile retinute in stiva la momentul autoapelului, la care se adauga valorile returnate de functie.

Avantaje si dezavantaje ale recursivitatii:

- subprogramele scrise recursiv sunt mai concise si mai clare
- subprogramele scrise recursiv sunt mai greu de controlat si depanat
- subprogramele scrise recursiv necesita spatiu de memorie si timp de executare mai mare.

Tema: de implementat cele 6 exemple de la inceputul documentului!!!

Implementarile la exemplele date

```
int r, a1, n;

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

double putere(double x, int n)
{
   if(n==0) return 1;
   else return x*putere(x,n-1);
}

int cmmdc(int a, int b)
{
   if(b==0) return a;
   else return cmmdc(b, a%b);
}

int cmmdc2(int a, int b)
{
   if(a==b) return  a;
   else if(a>b) return cmmdc2(a-b,b);
   else return cmmdc2(a,b-a);
}

int sumcif(int n)
{
  if(n<10) return n;
  else return n%10+sumcif(n/10);
}

int f(int n)
{ if(n==1 || n==2) return 1;
else return f(n-1)+f(n-2);
}

int progresie(int n, int a1, int r)
{
   if(n==1) return a1;
   else return progresie(n-1,a1,r)+r;
}
```

Aplicatii cu cifrele unui numar natural n

- **Numarul de cifre ale lui n**

$$\text{nrcif(n)}=\begin{cases}1, dacă\ n<10\\ 1+nrcif(n/10), dacă\ n\geqslant 10\end{cases}$$

Exemplu: n=239

nrcif(239) = 1+nrcif(23) = 1+2 =3
nrcif(23) = 1+nrcif(2) = 1+1 = 2
nrcif(2) = 1

- **Numarul de cifre pare ale lui n**

$$\text{nrcif(n)}=\begin{cases} 1, dacă\ n<10\ si\ n\%2=0 \\ 0, dacă\ n<10\ si\ n\%2\neq 0 \\ 1+nrcif(n/10), dacă\ \ n\%2=0 \\ nrcif(n/10),\ \ altfel \end{cases}$$

- **Cifra maxima a lui n**

$$\text{Max(n)} = \begin{cases} n, dacă\ n<10 \\ n\%10, dacă\ n\%10>Max(\text{n}/10) \\ Max(n/10),\ \ alfel \end{cases}$$

- **Afisarea cifrelor lui n de la dreapta la stanga**

```
void afis (int n)
{  if (n>0)
      { cout<<n%10<<" ";
        afis(n/10);
       }
 }
```

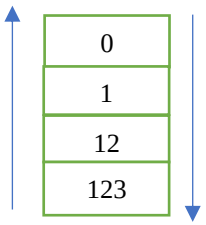
- **Afisarea cifrelor pare ale lui n de la dreapta la stanga**

```
void afis (int n)
{  if (n>0)
      { if (n%2==0) cout<<n%10<<" ";
        afis(n/10);
       }
 }
```

- **Afisarea cifrelor lui n de la stanga la dreapta**

```
void afis (int n)
{  if (n>0)
      {   afis(n/10);
          cout<<n%10<<" ";


       }
 }
```



scrie 1
scrie 2
scrie 3

Aplicatii vectori

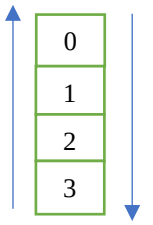
- **Citirea elementelor vectorului v cu n elemente**

```
void citire(int i)
{ if (i<=n)
      { cin<<v[i];
        citire(i+1);
       }
 }
```

Apel: citire(1);

- **Afisare de la primul la ultimul a elementelor vectorului v cu n elemente**

```
void afisare(int n)
{ if (n>0)
      {
        afisare(n-1);
        cout<<v[n]<<" ";
       }
 }
```

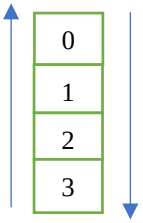


scrie v[1]
scrie v[2]
scrie v[3]

Apel: afisare(n);

- **Afisare de la ultimul la primul a elementelor vectorului v**

```
void afisare_inversa(int n)
{ if (n>0)
      {
        cout<<v[n]<<" ";
        afisare(n-1);
       }
 }
```



scrie v[1]
scrie v[2]
scrie v[3]

- **Afisare a elementelor pare din vectorul v cu n elemente**

```
void afisare(int n)
{ if (n>0)
      {
        afisare(n-1);
        if (v[n] % 2 ==0) cout<<v[n]<<" ";
       }
 }
```

- **Suma elementelor din vectorul v cu n elemente intregi**

Notam: suma(n) = suma primelor n elemente din vector

$$\text{suma(n)} = \begin{cases} 0, dacă\ n = 0 \\ suma(n-1) + v[n], dacă\ n > 0 \end{cases}$$

suma(3) = suma(2) + v[3] = v[1] + v[2] + v[3]
suma(2) = suma(1) + v[2] = v[1] + v[2]
suma(1) = suma(0) + v[1] = v[1]
suma(0) = 0

```
int suma (int n)
{ if (n==0) return 0;
  else return suma(n-1) +v[n];
}
```

- **Suma elementelor pare din vectorul v cu n elemente intregi**

Notam: suma(n) = suma elementelor pare din primele n elemente din vector

$$\text{suma(n)} = \begin{cases} 0, dacă\ n = 0 \\ suma(n-1) + v[n], dacă\ v[n]\ este\ par \\ suma(n-1),\ alfel \end{cases}$$

v=(4, 5, 6, 9), n=4
suma(4) = suma(3) = 4 + 6 =10
suma(3) = suma(2) + 6 = 4 + 6
suma(2) = suma(1) = 4
suma(1) = suma(0) + 4 = 4
suma(0) = 0

```
int suma (int n)
{ if (n==0) return 0;
  else if (v[n]%2 == 0) return suma(n-1) +v[n];
       else suma(n-1);
}
```

- **Maximul din vectorul v cu n elemente intregi**

$$\text{Max(n)} = \begin{cases} v[n], dacă\ n = 1 \\ v[n], dacă\ v[n] > Max(\text{n} - 1) \\ Max(n-1),\ alfel \end{cases}$$

```
int Max (int n)
{ if (n==1) return v[n];
  else if (v[n]>Max(n-1)) return v[n];
       else return Max(n-1);
}
```