



Subprograme

în limbajul C



Subprograme

- Definiție. Clasificare
- Construcție și apel
- Transferul datelor între apelator și apelat
- Tratarea parametrilor din linia de comandă
- Subprograme cu număr variabil de parametri
- Pointeri spre subprograme
- Subprograme polimorfice
- Recursivitate



Definiție

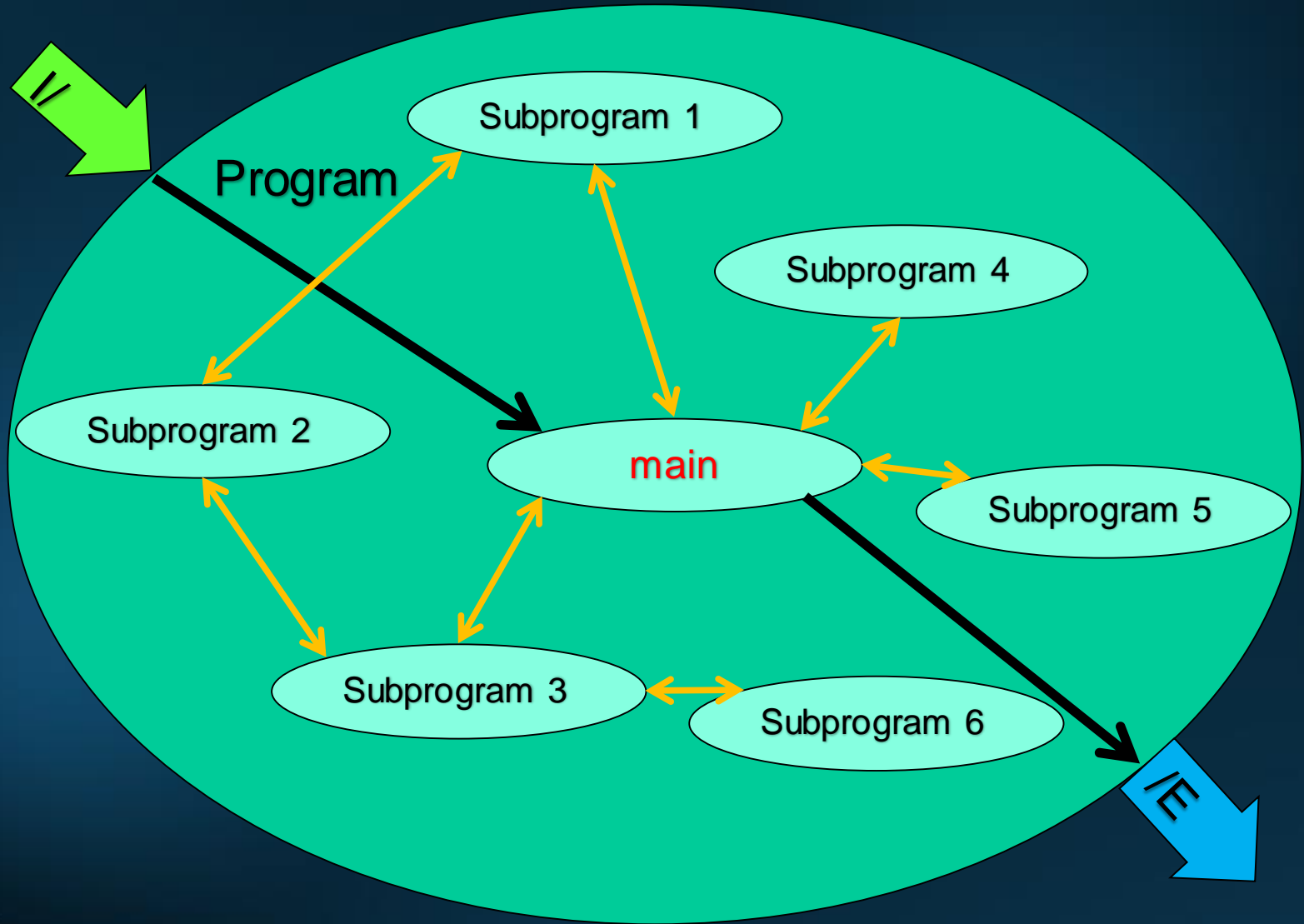
- Subprogramele sînt unități de program care:
 - au un algoritm propriu,
 - pot fi proiectate independent
 - pot fi scrise independent
 - pot fi compilate independent
 - nu se pot executa independent ci numai în cadrul unui program (apel)
- Avantaje
 - evitarea scrierii repetate a aceluiași set de instrucțiuni
 - creșterea eficienței, prin reutilizarea subprogramelor (biblioteci de subprograme)



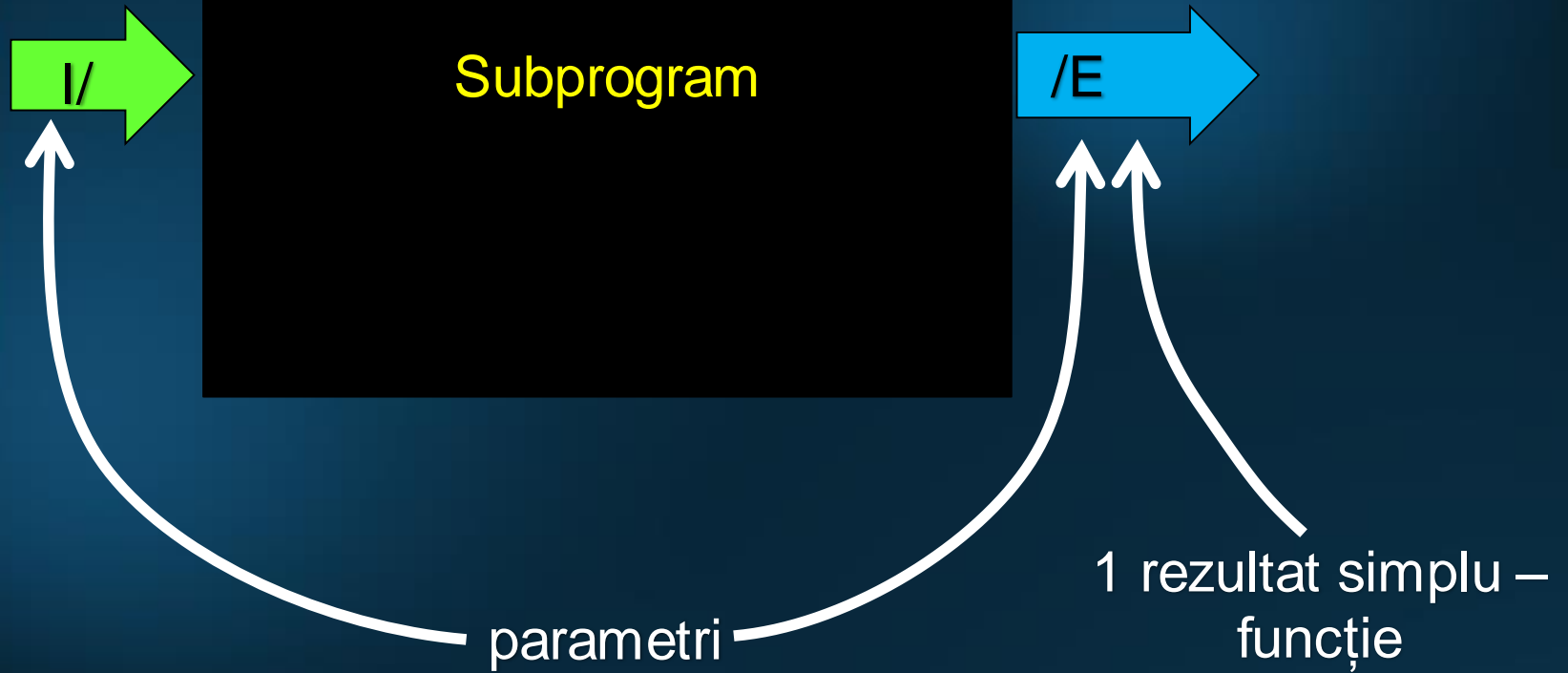
Clasificare

- Rol
 - apelator, apelat, programul principal
 - de calcul, de intrare/ieșire
- Construcție și utilizare
 - funcții, proceduri
- Localizare
 - interne, externe
- Aria de folosire
 - standard, ale utilizatorului

Program și subprogram



Subprogram



Construcție

- Forma generală

`antet`

`corp`

- Antet

`tip rez nume ([lista parametrilor formali])`

- Lista parametrilor formali

`declaratie1, declaratie 2 ...`

`tip1 nume1, tip2 nume2 ...`

- Corp

`o instrucțiune compusă { ... }`

- Instrucțiunea `return`

- Rol

- Forma generală

`return expresie;`



Exemple

```
int suma ( int a, int b )  
{ return a + b;  
}
```

```
float ecuatie ( float x )  
{ return x*x - 2*x + 1;  
}
```

```
void mesaj ( int cod )  
{ if ( cod == 3 ) printf ( "\n Mesajul  
    numarul 1");  
  else printf ( "\n Mesajul numarul 2");  
}
```




Construcție

- Subrograme imbricate: NU
- Prototip

antet;

- pot lipsi numele parametrilor (doar tipuri)

int suma (int, int);

- Apel
 - Transferul controlului: apel, context apel
nume (lista parametrilor reali)
 - Transferul datelor: parametri, variabile globale

Exemplu

- Să se scrie o funcție care calculează cel mai mare divizor comun dintre două numere întregi nenule (utilizând algoritmul lui Euclid) și un apelator pentru testare.

```
#include <stdio.h>
/*definirea functiei cmmdc*/
int cmmdc(int a, int b)
{ int r,d=a,i=b;
  do { r = d%i;
        d = i;
        i = r;
      }
  while(r!=0);
  return d;
}
void main()
{ int n1,n2;
  printf("\nNumerele pentru care se va calcula cmmdc:");
  scanf("%d%d",&n1,&n2);
  if(n1&& n2)
    printf("\ncmmdc=%d",cmmdc(n1,n2));
  else
    printf("Numerele nu sint nenule!");
}
```

Exemplu

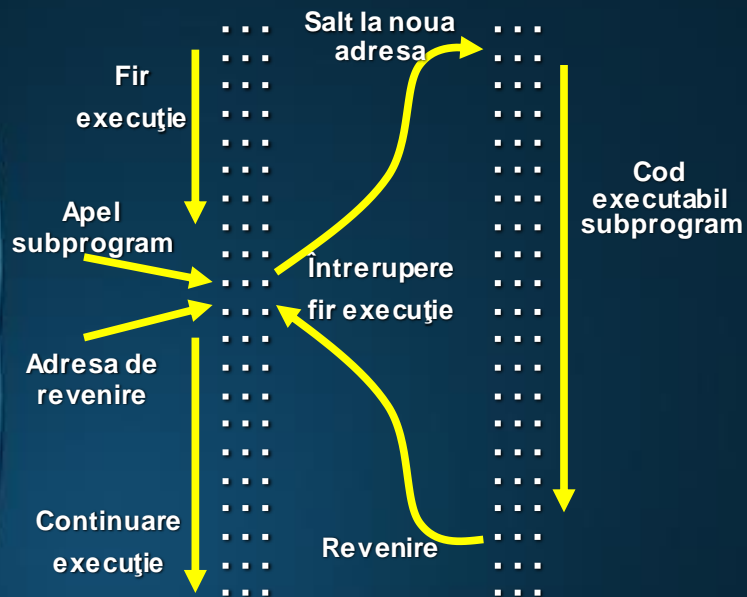
- Același exemplu folosind un prototip pentru funcția *cmmdc*:

```
#include <stdio.h>
/* prototipul functiei cmmdc*/
int cmmdc(int, int);
void main()
{ int n1,n2;
  printf("Numerele pentru care se va calcula cmmdc:");
  scanf("%d%d",&n1,&n2);
  if(n1&& n2)
    printf("\ncmmdc=%d",cmmdc(n1,n2));
  else
    printf("Numerele nu sint nenule! ");
}

/*definirea functiei cmmdc*/
int cmmdc(int a, int b)
{ int r,d=a,i=b;
  do { r=d%i;
      d=i; i=r;}
  while(r!=0);
  return d;
}
```

Ce se întâmplă?

Segment de cod



Stivă

n1
n2

Adr. revenire
rezultat

a
b
d
i
r

} parametri reali

} variabile locale

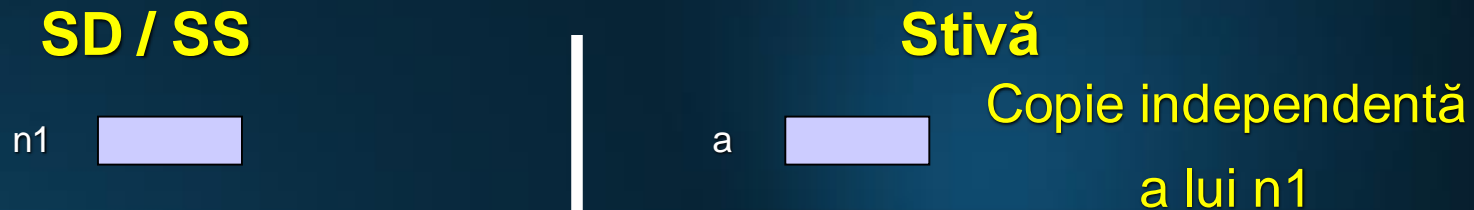


Transferul datelor între apelator și apelat

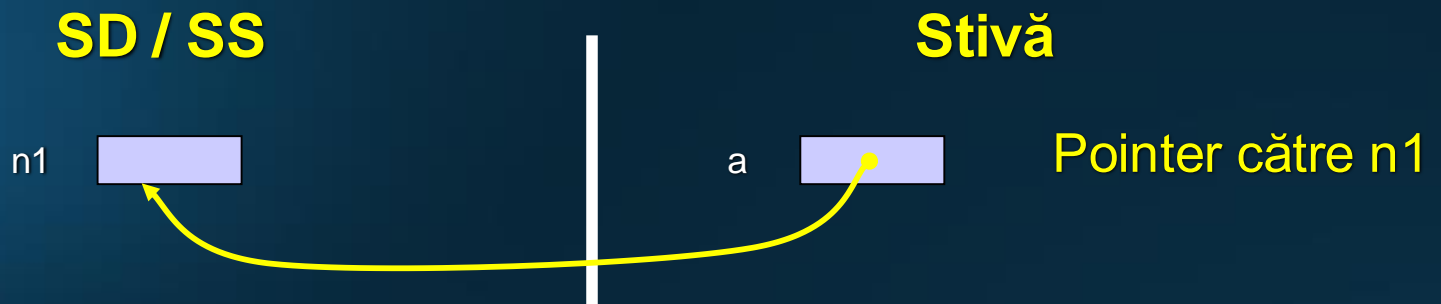
- Prin variabile globale (**nerecomandat**)
- Prin parametri
 - Transfer teoretic
 - prin valoare
 - prin adresă
 - Variabile simple
 - Simularea transferului prin adresă pentru variabile simple
 - Masive
 - Vectori
 - Matrice

Transferul prin valoare / adresă

- Transferul prin valoare: I/



- Transferul prin adresă: I/E



- **ANSI-C** : numai transferul prin valoare
- **C++** : ambele tipuri de transfer

Transferul variabilelor simple (ANSI-C)

- Date de intrare
- Date de ieșire (rezultate): simularea transferului prin adresă

I/ : a, b

/E: r1, r2

```
int numefunctie( int a, int b, int *r2 )  
{ ...  
  ... *r2 ...  
}
```

Exemplu de apel

```
int d1, d2, d3, d4; ← Context apel  
d3 = numefunctie(d1, d2, &d4); ← Apel
```

Transferul variabilelor simple (C++)

- Date de intrare
- Date de ieșire (rezultate): transfer prin adresă

I/ : a, b

/E: r1, r2

```
int numefunctie( int a, int b, int &r2 )  
{ ...  
    ... r2 ...  
}
```

Exemplu de apel

```
int d1, d2, d3, d4; ← Context apel  
d3 = numefunctie(d1, d2, d4); ← Apel
```

Transferul vectorilor

I/ : v[10], n

/E: -

`int *a`



`void sortare(int a[10], int n)`

`{ ...`

`... a[i] ...`

`}`

Transferul matricelor

I/ : a[10][10], m, n

/E: -

int **a



```
void min( int a[10][10], int m, int n )
```

```
{ ...
```

```
    ... a[i][j] ...
```

```
}
```

Matrice alocată dinamic



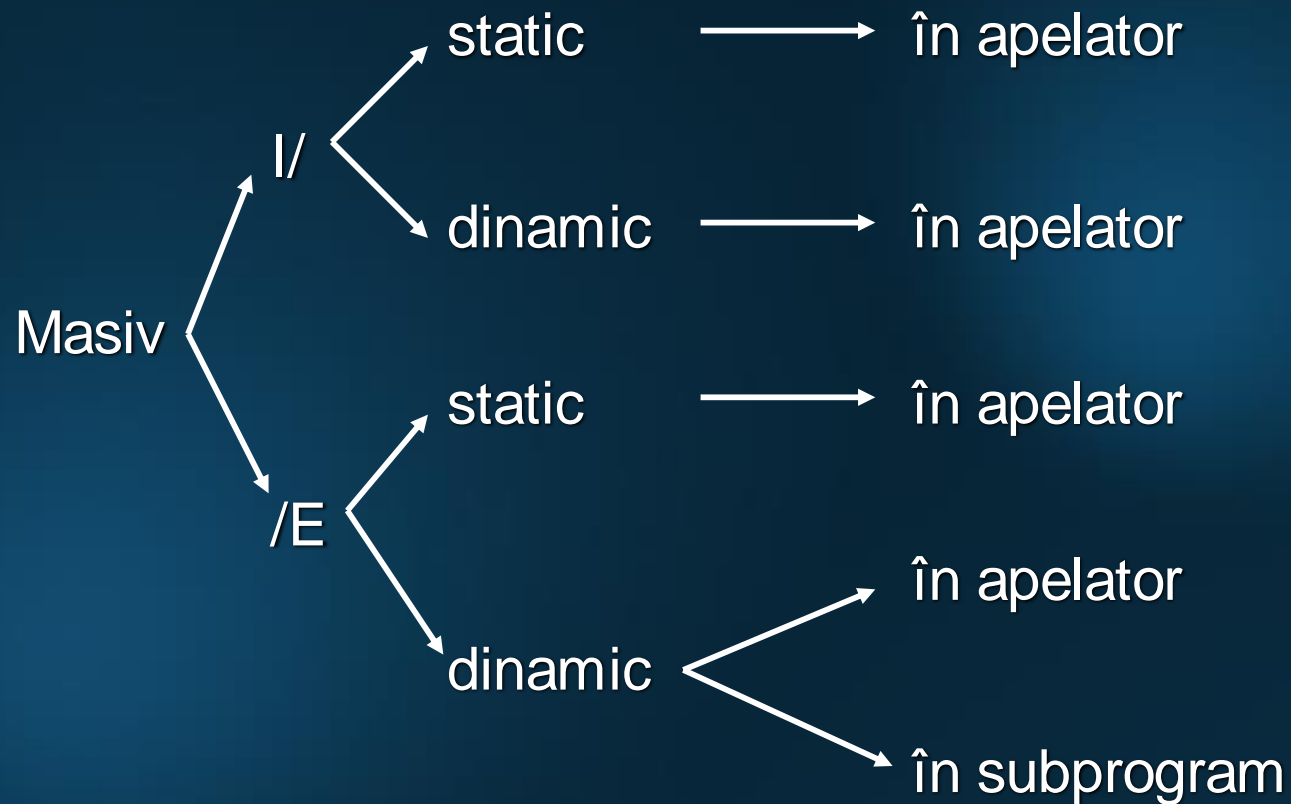
```
void min( int **a, int m, int n )
```

```
{ ...
```

```
    ... a[i][j] ...
```

```
}
```

Transferul masivelor





Exemple

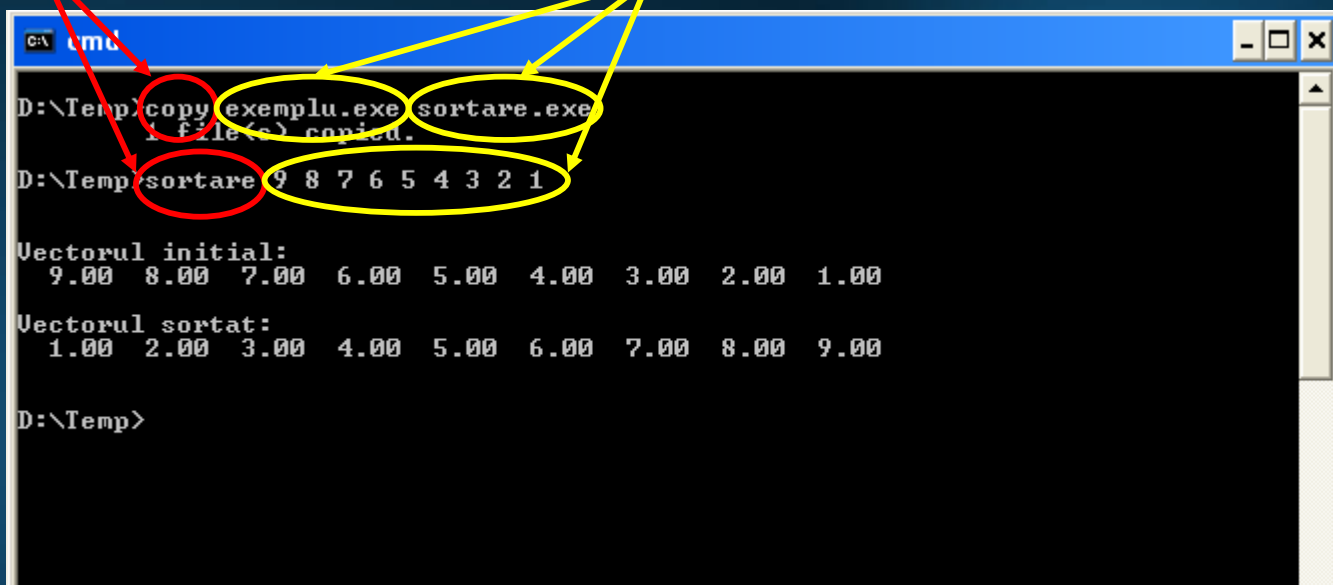
- Produs vectorial între doi vectori
 - Toate masivele alocate static
 - Toate masivele alocate dinamic în apelator
 - Toate masivele alocate dinamic, rezultatul alocat în subprogram
- Produsul dintre 2 matrice
 - Toate masivele alocate static
 - Toate masivele alocate dinamic în apelator
 - Toate masivele alocate dinamic, rezultatul alocat în subprogram

Tratarea parametrilor din linia de comandă

- Parametri în linia de comandă

Comanda

Parametri în linia de comandă



```
C:\> D:\Temp\copy exemplu.exe sortare.exe
1 file(s) copied.

D:\Temp> sortare 9 8 7 6 5 4 3 2 1

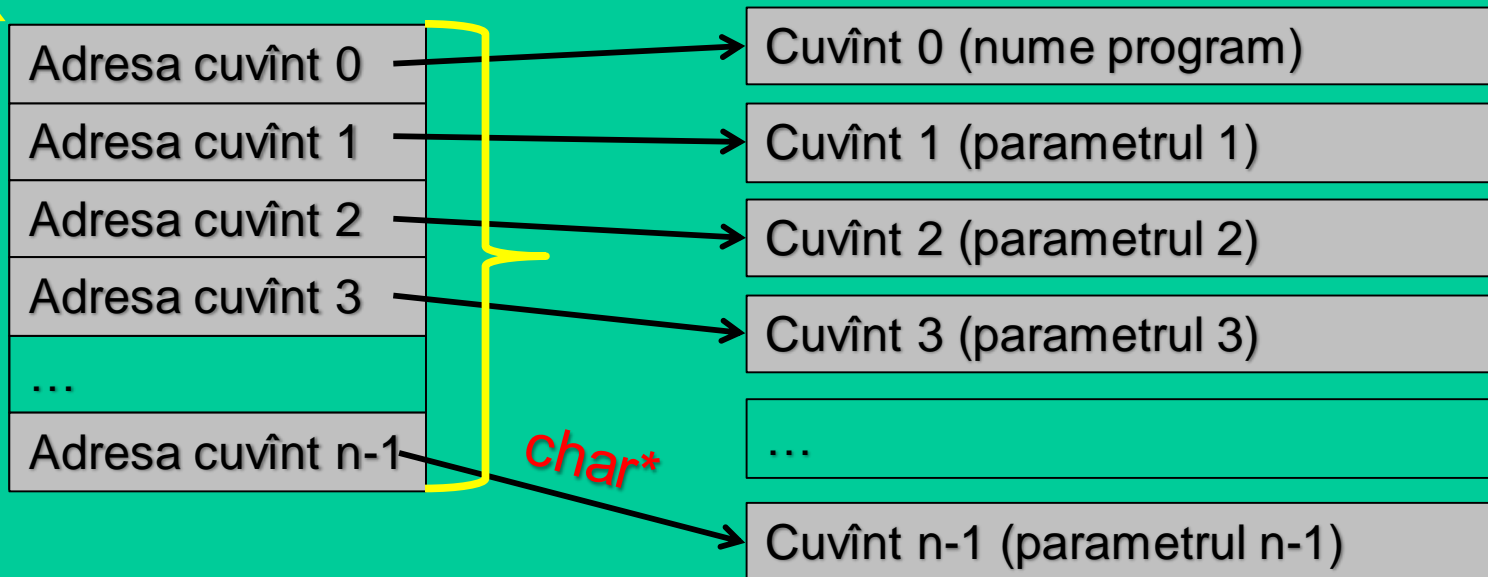
Vectorul initial:
 9.00 8.00 7.00 6.00 5.00 4.00 3.00 2.00 1.00

Vectorul sortat:
 1.00 2.00 3.00 4.00 5.00 6.00 7.00 8.00 9.00

D:\Temp>
```

Tratarea parametrilor din linia de comandă

Memoria internă



adresa
vectorului
char**

↑
Șiruri de caractere



Tratarea parametrilor din linia de comandă

- Parametri:

`int argc`

`char* argv[]`

`void main (int argc, char* argv[])`

- E sarcina programatorului să valideze lista de parametri și să o prelucreze
- Preluare ⇔ conversie din șir de caractere în formatul dorit (ex. sscanf)
- Algoritm: se parcurge vectorul de cuvinte și se preia fiecare parametru

Tratarea parametrilor din linia de comandă

- Adunarea a două numere reale preluate din linia de comandă

```
#include <stdio.h>
```

```
void main(int argc, char* argv[])  
{ float x,y;  
  if(argc!=3)printf("\nNumar incorect de  
    parametri!\n");  
  else  
  { sscanf(argv[1],"%f", &x);  
    sscanf(argv[2],"%f", &y);  
    printf("\nProgramul executabil:%s\n",argv[0]);  
    printf("\nx=   %6.2f\n",x);  
    printf("\ny=   %6.2f\n",y);  
    printf("\nx+y=%6.2f\n",x+y);  
  }  
}
```

Tratarea parametrilor din linia de comandă

- Sortarea unui șir de numere reale

```
#include <stdio.h>
void main(int argc, char* argv[])
{ float x[100],aux;
  int i,j,n;
  if(argc<2)printf("\nNumar incorect de parametri!\n");
  else
  { n=argc-1;
    for(i=1;i<argc;i++)
      sscanf(argv[i],"%f", &x[i-1]);
    printf("\n\nVectorul initial:\n");
    for(i=0;i<n;i++)
      printf("%6.2f",x[i]);
    if(argc<2)printf("\nNumar incorect de parametri!\n");
    else
    { n=argc-1;
      for(j=i+1;j<n;j++)
        if(x[i]>x[j])
        { aux=x[i];
          x[i]=x[j];
          x[j]=aux;
        }
      for(i=1;i<argc;i++)
        sscanf(argv[i],"%f", &x[i-1]);
      printf("\n\nVectorul sortat:\n");
      for(i=0;i<n;i++)
        printf("%6.2f",x[i]);
      printf("\n\n");
    }
  }
}
```


Subprograme cu număr variabil de parametri

- De ce?
- Prototip
 - Cu listă fixă de parametri

```
tip_rezultat nume(lista_parametri);
```

- Cu număr variabil de parametri lista variabilă de parametri

```
tip_rezultat nume(lista_parametri_ficși, ...);
```

nevidă



Subprograme cu număr variabil de parametri

- Este sarcina programatorului să scrie cod care știe să preia și să trateze corect toți parametrii din lista variabilă de parametri !
- Elemente definite în **stdarg.h**
 - va_list** - tip de dată
 - va_start** - funcție, inițializare lucru cu lista variabilă
 - va_arg** - funcție, extrage următorul parametru
 - va_end** - funcție, încheiere lucru cu lista variabilă



Subprograme cu număr variabil de parametri

`va_list`

- o variabilă locală de tip `va_list` care reține adresa listei de parametri variabili

`va_list p;`

`void va_start(va_list p, ultim);`

- inițializează adresa listei variabile
- se apelează prima
- `ultim` este numele ultimului parametru fix (uneori reprezintă numărul de parametri variabili)



Subprograme cu număr variabil de parametri

tip **va_arg(va_list p,tip);**

- obține valoarea următorului parametru din lista variabilă, ***conform tipului cerut!***

void **va_end(va_list p);**

- încheie lucrul cu lista variabilă de parametri



Subprograme cu număr variabil de parametri

- Algoritm
 - declarare variabilă locală de tip **va_list**
 - apelare **va_start**
 - buclă* de preluare/prelucrare a parametrilor cu **va_arg**
 - apelare **va_end**
- Detectare număr parametri în listă
 - *Precizare număr parametri la apel (de obicei ultimul parametru fix) ⇔ buclă cu număr cunoscut de pași*
 - *O valoare convențională care marchează terminarea listei variabile de parametri ⇔ buclă condiționată anterior*



Exemple

1. Media unui șir de elemente reale, de lungime necunoscută
2. Cel mai mare divizor al unui număr oarecare de numere întregi.
3. Concatenarea unui număr oarecare de șiruri de caractere la sfârșitul unui șir dat.

Media

```
double media(int nr, ...)
{double suma=0; int i;
  va_list p;
  va_start(p, nr);
  for(i=0;i<nr;i++)
    suma+=va_arg(p, double );
  va_end(p);
  return(suma/nr);
}
```

```
x=media(3,4.0,6.0,9.0);
y=media(2,5.0,8.0);
z=media(4,4.0,5.0,6.0,7.0);
```

```
double media(float unu, ...)
{double n,suma=unu; int cite=1;
  va_list p;
  va_start(p, unu);
  n=va_arg(p, double);
  while(n!=-1)
  { suma+=n;
    cite++;
    n=va_arg(p, double );
  }
  va_end(p);
  return(suma/cite);
}
```

```
x=media(4.0,6.0,9.0,-1.0);
y=media(5.0,8.0,-1.0);
z=media(4.0,5.0,6.0,7.0,-1.0);
```

Atenție la tipurile parametrilor și la cazurile speciale!

Concatenare

```
char* concat_var(char* sir, int
    nr,...)
{va_list p;
 char* s; int i;
 va_start(p,nr);
 for(i=0;i<nr;i++)
 {s=va_arg(p,char*);
  strcat(sir,s);
 }
 va_end(p);
 return sir;
}
```

```
char* concat_var(char* sir, ...)
{va_list p;
 char* s;
 va_start(p,sir);
 s=va_arg(p,char*);
 while(s)
 { strcat(sir,s);
  s=va_arg(p,char*);
 }
 va_end(p);
 return sir;
}
```

CMMDC

Temă!



Pointeri spre subprograme

- Numele unei funcții poate fi folosit ca pointer constant (asemănător masivelor)
- Semnificația:
 - adresa din memorie unde se află codul executabil al subprogramului respectiv
- Tipul:
 - Pointer către un subprogram care primește *o anumită listă de parametri* și întoarce *un anumit tip de rezultat*
- Utilizare:
 - Transmiterea subprogramelor ca parametri pentru alte subprograme

Pointeri spre subprograme

- Exemplu:

```
void sortare(float v[], int n);
```

sortare ⇔ pointer către o funcție care primește ca parametri *un vector cu elemente float și un întreg* și are rezultat de tip *void*

```
float minim(float v[], int n, int poz[], int* nr);
```

minim ⇔ pointer către o funcție care primește ca parametri *un vector cu elemente float, un întreg, un vector cu elemente întregi și un pointer către întreg* și are rezultat de tip *float*

Pointeri spre subprograme

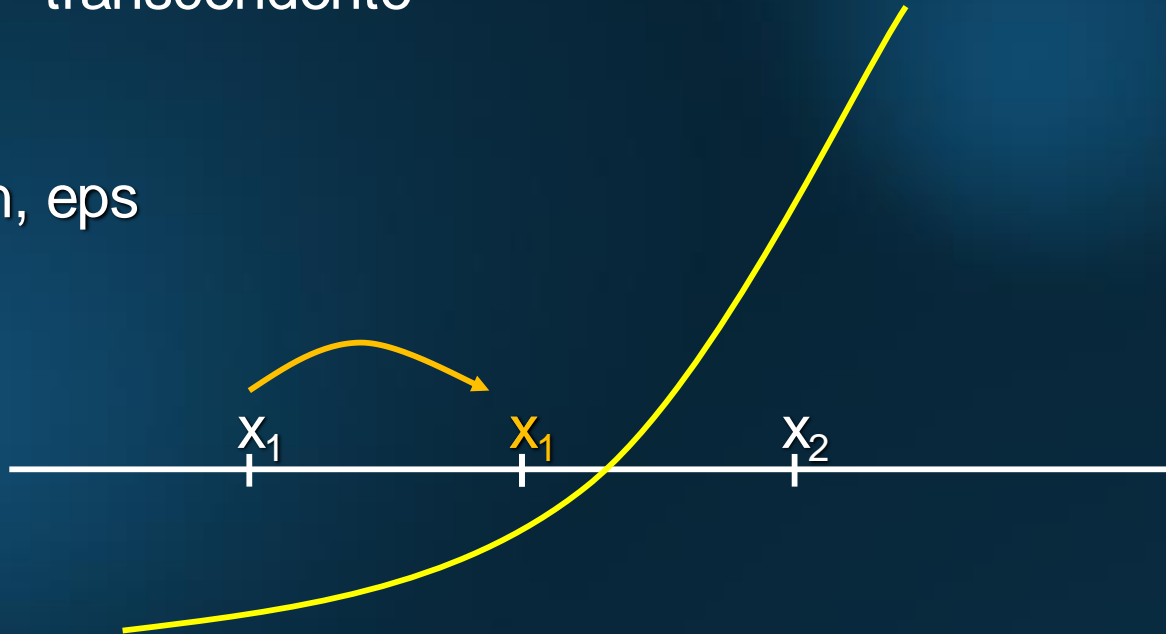
- Declarare variabilă/parametru tip pointer la funcție și utilizare

```
void sortare(float v[], int n);  
float minim(float v[], int n, int poz[], int* nr);  
  
void main()  
{ int n; float v[100];  
  void (*p)(float v[], int n);  
  float (*q)(float v[], int n, int poz[], int* nr);  
  ...  
  p = sortare;  
  q = minim;  
  sortare(v,n); // ⇔ (*p)(v, n);  
  ...  
}
```

Pointeri spre subprograme

- Exemplu
 - Metoda biseecției pentru rezolvarea unei ecuații transcendente

n, eps



Metoda biseecției

```
#include <stdio.h>

float ecuatie(float x)
{ return x*x - 7*x + 12;
}

int bisectie( float x1, float x2,
             float eps, int n,
             float (*f)(float),
             float *x)
{ int cod = 0;
  while ((n > 0) && (cod == 0))
  { *x = (x1 + x2) / 2;
    if ((*f) (*x) == 0) cod = 1;
    else if ((x2-x1) < eps) cod = 2;
    else if ((*f) (x1) * (*f) (*x) < 0)
        x2 = *x;
    else x1 = *x;

    n--;
  }
  return cod;
}
```

```
void main()
{ float a, b, sol, prec;
  int nr, cod;
  printf("\na="); scanf("%f", &a);
  printf("\nb="); scanf("%f", &b);
  printf("\nprecizia=");
  scanf("%f", &prec);
  printf("\nnr="); scanf("%d", &nr);

  cod =
    bisectie(a,b,prec,nr,ecuatie,&sol);

  switch(cod)
  { case 0: printf("\nFara solutie");
    break;
    case 1: printf("\nSolutia exacta este
                  %7.3f", sol);
    break;
    case 2: printf("\nSolutia aproximativa
                  este %7.3f", sol);
  }
}
```

Temă: metoda tangentei.



Subprograme polimorifice

- Algoritmii generali de prelucrare a datelor nu depind de tipul lor => pot fi proiectați independent
- Operațiile elementare depind de tipul datelor => trebuie implementate separat pentru fiecare tip
- Pentru a implementa un algoritm general trebuie cunoscut modul de realizare a operațiilor elementare => parametri de tip subprogram

Subprograme polimorfe

- SP: efectuează calcule indiferent de tipul parametrilor primiți
 - acceptă parametri de mai multe tipuri, într-o anumită măsură
- Pentru localizarea datelor folosesc pointeri fără tip:
- Pentru accesarea efectivă a datelor e necesară conversia

`void*`

`void *p;`

`*(tip*)p`

`*(float*)p *(int*)p *(char*)p`

```
p=malloc(sizeof(int));
```

```
scanf("%d", p); printf("%3d", *(int*)p );
```

```
.....
```

```
p=malloc(sizeof(float));
```

```
scanf("%f", p); printf("%6.2f", *(float*)p );
```



Subprograme polimorifice

- Elemente necesare (parametri)
 - **adresele** datelor de prelucrat, ca pointeri fără tip
 - **dimensiuni** date (în octeți)
 - parametri care **identifică tipurile** datelor de prelucrat
 - **adresele** unor **funcții** care efectuează operațiile elementare specifice pentru datele primite (atribuire, comparație, adunare, scădere etc.)



Subprograme polimorfe

- Operații elementare
 - Operația de atribuire

```
void memmove(void * dest, const void* sursa, unsigned n);
```

```
mem.h    string.h
```

- Funcții care realizează operațiile elementare

```
int comparafloat(const void *a, const void *b)
```

```
{ int rez;
```

```
  if(*(float *)a==*(float *)b) rez=1;
```

```
  else rez=0;
```

```
  return rez;
```

```
}
```

Subprograme polimorfe

- Citire vector, cu alocare memorie

```
int* citeste(int* n)
```

```
{    int i;
```

```
    int* p;
```

```
    printf("\nn=");
```

```
    scanf("%d", n);
```

```
    p = (int*) malloc(sizeof(int) * *n);
```

```
    for(i=0; i<*n; i++)
```

```
    {    printf("v[%d]=", i);
```

```
        scanf("%d", p+i);
```

```
    }
```

```
    return p;
```

```
}
```

Dimensiune: int d

void*

Operație
elementară

=> Cod care identifică
tipul datelor

Subprograme polimorifice

- Rezolvarea operației elementare
 - Parametru de tip întreg care codifică tipul de dată
 - 0 – int, 1 – float etc.
- ```
switch(c)
{ case 0: scanf("%d", (int*)p+i); break;
 case 1: scanf("%f", (float*)p+i);
 // etc.
}
```
- Parametru șir de caractere care definește conversia
    - “%d” – int, “%f” – float etc.
- ```
( ..., char* cod, ...)
```
- ```
scanf(cod, (char*)p+i*d);
```



# Subprograme polimorfice

```
void* citeste(int* n, int d, char* c)
{ int i;
 void* p;
 printf("\nn=") ;
 scanf("%d",n) ;
 p=malloc(*n*d) ;
 for(i=0;i<*n;i++)
 { printf("v[%d]=",i) ;
 switch(d, (char*)p+i*d) ;
 { case 1: scanf("%d", (int*)p+i) ; break;
 return p; case 2: scanf("%f", (float*)p+i) ;
 }
 }
 return p;
}
```



# Subprograme polimorfe

- Afișare elemente vector (!!!)

```
void afiseaza(void* p, int n, int d, char* cod)
{
 int i;
 for(i=0;i<n;i++)
 printf(cod, *((char*)p+i*d));
 printf("\n");
}
```



# Subprograme polimorfe

- Căutare element în vector

```
int cauta (void *v, void *k, int n, int dim,
 int(*compara)(const void *,const void *))
{
 int rez;
 rez=-1;
 for(int i=0;i<n;i++)
 if((*compara)((char *)k,(char *)v+i*dim))
 rez=i;
 return rez;
}
```

# Subprograme polimorfe

- Sortare vector

```
void sort(void *v, int n, int dim,
 int (*compara)(const void *x, const void *y))
{
 int i, j;
 void *aux;
 aux = malloc(dim);
 for (i = 0; i < n - 1; i++)
 for (j = i + 1; j < n; j++)
 if ((*compara)((char *)v + dim * i, (char *)v + dim * j))
 {
 memmove(aux, (char *)v + dim * i, dim);
 memmove((char *)v + dim * i, (char *)v + dim * j, dim);
 memmove((char *)v + dim * j, aux, dim);
 }
 free(aux);
}
```



# Subprograme recursive

- Algoritmi iterativi
- Algoritmi recursivi
  - Recursivitate simplă (algoritm unirecursiv)
  - Recursivitate multiplă (algoritm multirecursiv)
  - Formulă de start (una sau mai multe)
  - Formulă recursivă
- Exemple
  - Numărarea valorilor care îndeplinesc o condiție
  - Suma elementelor unui vector
  - Algoritmul lui Euclid
  - Șirul lui Fibonacci



# Subprograme recursive

- Un algoritm fie iterativ sau recursiv poate fi implementat printr-un subprogram fie iterativ, fie recursiv
- Subprogram recursiv: generează (cel puțin) un apel către el însuși
  - Recursivitate directă
    - Simplă
    - Multiplă
  - Recursivitate mutuală
- Implicații
  - Mod de construire a subprogramelor
  - Necesari de memorie





# Subprograme recursive

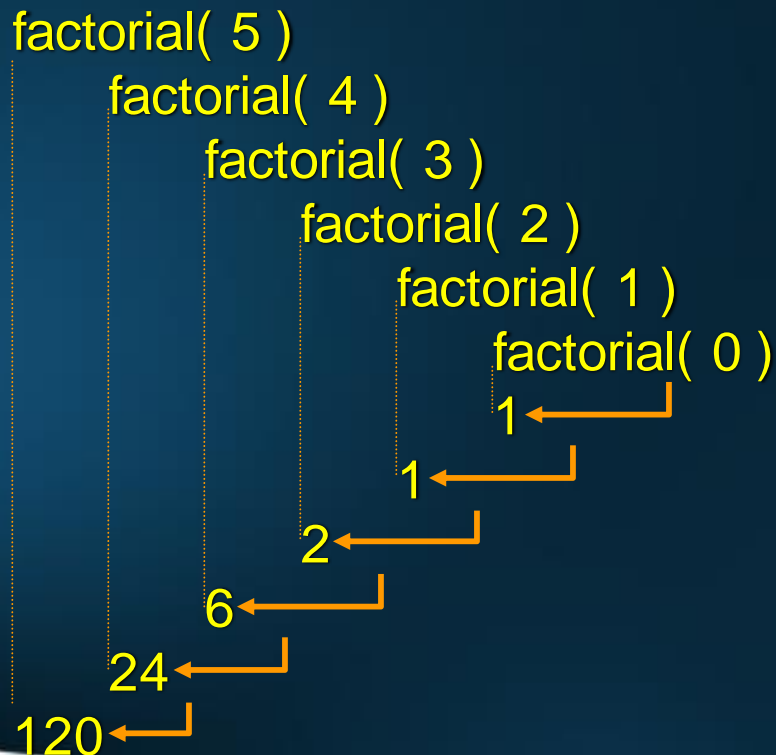
- Construcția subprogramelor recursive
  - Să asigure respectarea *finititudinii* algoritmului: ieșirea după un număr finit de pași
  - Condiție de oprire a generării de noi apeluri
    - Aplicarea formulei de start dacă e îndeplinită condiția
    - Aplicarea formulei recursive în caz contrar

```
long factorial(unsigned n)
{ long f;
 if (!n) f = 1;
 else f= n*factorial(n-1);
 return f;
}
```

# Subprograme recursive

- Necesarul de memorie
  - La fiecare apel se alocă spațiu în stivă pentru ...

```
n = factorial(5);
```



# Subprogram recursive

$\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2)$ ,  $\text{fib}(1) = \text{fib}(0) = 1$

fib(4)  
fib(3)  
fib(2)  
fib(1)  
1  
fib(0)  
1  
2  
fib(1)  
1  
3  
fib(2)

fib(1)  
1  
fib(0)  
1  
2  
6



# Subprograme recursive

- Când alegem subprograme iterative/recursive?
  - Avantaje și dezavantaje
    - Consum memorie
    - Timp de execuție
    - Ușurință în proiectare / implementare
      - Lungime cod sursă



# Subprograme recursive

- Considerații generale
  - Fiecare apel recursiv trebuie aplicat unei probleme mai simple decât în pasul anterior
  - Rezultă o metodă simplă de oprire a generării de noi apeluri
- Cum se transformă un subprogram iterativ în unul recursiv?
  1. instrucțiunea iterativă dispare
  2. condiția de la instrucțiunea iterativă devine (eventual modificată) condiție de oprire a generării de noi autoapeluri
  3. Repetarea este asigurată prin autoapel
    - Exemplu: metoda biseecției



# Subprogramme recursive - exemple

```
int bisectie(float x1, float x2, float eps, int n, float (*f)(float), float *x)
{
 int cod = 0;
 if (n == 0) cod = 0;
 while ((n > 0) && (cod == 0))
 {
 *x = (x1 + x2) / 2;
 if ((*f)(*x) == 0) cod = 1;
 else if ((x2 - x1) < eps) cod = 2;
 else if ((*f)(x1) * (*f)(*x) < 0)
 {
 x2 = *x;
 }
 else x1 = *x;
 n--;
 cod = bisectie(x1, x2, eps, n, f, x);
 }
 return cod;
}
```

Diagram illustrating the recursive call in the `bisectie` function:

- The function `bisectie` is called with parameters `x1`, `x2`, `eps`, `n`, `f`, and `x`.
- The function calculates the midpoint `*x = (x1 + x2) / 2`.
- The function checks if `(*f)(*x) == 0`. If true, `cod = 1`.
- The function checks if `(x2 - x1) < eps`. If true, `cod = 2`.
- The function checks if `(*f)(x1) * (*f)(*x) < 0`. If true, it updates `x2 = *x`.
- The function updates `x1 = *x` if the previous condition is false.
- The function decrements `n` and calls `bisectie` recursively with updated parameters.



# Subprograme recursive - exemple

- Calculul combinatorilor de  $n$  luate câte  $k$

$$C_n^k = \frac{n!}{k!(n-k)!}$$

$$C_n^k = C_{n-1}^k + C_{n-1}^{k-1}$$

$$C_n^0 = 1 \quad C_k^k = 1$$

```
long comb(unsigned n, unsigned k)
{
 long c;
 if (k>n) c = 0;
 else if ((k==0) || (k==n)) c = 1;
 else c = comb(n-1,k)+comb(n-1,k-1);
 return c;
}
```



# Subprograme recursive - exemple

- Suma elementelor unui vector
- $S(n) = x[n-1] + S(n-1)$ ,  $S(0) = 0$

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

- Produsul elementelor unui vector



# Subprograme recursive - exemple

- Cautarea binară

```
int cauta(float v[], int p, int u, float k)
{ int m;
 if (p > u) m = -1;
 else { m = (p + u) / 2;
 if(k < v[m]) m = cauta(v, p, m-1, k);
 else if(k > v[m]) m = cauta(v, m+1, u, k);
 }
 return m;
}
```



# Subprograme recursive - teme

- Numărul de elemente negative dintr-un vector
- Produs scalar între doi vectori
- Algoritmul lui Euclid
- Calculul cmmdc\*
- Metoda tangentei
- Problema turnurilor din Hanoi\*
- Sortare\*
  
- \* Se găsesc în manual



**Spor la învățat!**