

## SUBPROGRAME

Un subprogram este un ansamblu ce poate conține tipuri de date, variabile și instrucțiuni destinate unei anumite prelucrări (calcul, citiri, scrieri). Subprogramul poate fi executat doar dacă este apelat de către un program sau un alt subprogram.

În limbajul Pascal, subprogramele sunt de 2 tipuri: funcții și proceduri. În C/C++, este utilizată doar noțiunea de funcție, o procedură din Pascal fiind asimilată în C/C++ unei funcții care returnează *void*.

O altă clasificare a subprogramelor le împarte pe acestea în următoarele categorii:

- predefinite
- definite de către utilizator.

Dintre subprogramele predefinite, amintim subprogramele:

- matematic: *sin, cos, exp, log*;
- de citire/scriere: *read/write* (Pascal), *scanf/printf* (C/C++)
- de prelucrare a șirurilor de caractere: *substr, strlen, strcpy* (C/C++).

**Exemplu:** Conjectura lui Goldbach afirmă că orice număr par  $> 2$  este suma a două numere prime. Să se scrie un subprogram care verifică acest lucru pentru toate numerele pare mai mici decât  $N$  dat.

Vom rezolva această problemă cu ajutorul subprogramelor. Astfel, programul va conține:

- un subprogram *prim* care determină dacă un număr furnizat ca parametru este prim;
- un subprogram *verifica* prin care se obține dacă un număr furnizat ca parametru verifică proprietatea din enunț;
- programul principal, care apelează subprogramul *verifica* pentru toate numerele pare  $2 < k < N$ .

Soluția în C este următoarea:

```
#include "stdio.h"
#include "stdlib.h"

bool prim(int n)
{
    int i;
```

```

        for (i = 2; i<= n/2; i++)
            if(n%i==0) return false;
        return true;
    }

bool verifica(int n)
{
    int i;
    for (i = 2; i<= n/2; i++)
        if (prim(i) && prim(n - i))
            return true;
    return false;
}

void main(void)
{
    //declarare variabile
    int n, k;
    //citire valori de intrare
    printf("n=");
    scanf("%d", &n);
    //prelucrare: verificare daca este indeplinita conditia
    //pentru fiecare k par
    for(k=4; k<=n; k+=2)
        if(!verifica(k))
        {
            printf("%d nu verifica!\n", k);
            //ieșire fara eroare
            exit(0);
        }
    //afisare rezultat
    printf("proprietatea este indeplinita pentru numere >2 si
        <=%d\n", n);
}

```

**Observație:** Problema a fost descompusă în altele mai mici. Rezolvarea unei probleme complexe este mai ușoară dacă descompunem problema în altele mai simple.

Avantaje ale utilizării subprogramelor:

- reutilizarea codului (un subprogram poate fi utilizat de mai multe subprograme);
- rezolvarea mai simplă a problemei, prin descompunerea ei în probleme mai simple, care conduc la elaborarea de algoritmi ce reprezintă soluții ale problemei inițiale;
- reducerea numărului de erori care pot apărea la scrierea programelor și despistarea cu ușurință a acestora.

**Elementele unui subprogram** sunt prezentate în continuare, cu referire la noțiunea de funcție din C/C++. Pentru funcțiile și procedurile din Pascal, aceste elemente se pot distinge în mod similar.

- Antetul funcției *prim* este: `bool prim(int n)`
- Numele funcției este `prim`
- Funcția *prim* are un parametru *n*.
- Funcția are un tip, care reprezintă tipul de date al rezultatului său. Tipul funcției *prim* este *bool*.  
În cazul programelor C/C++, dacă tipul funcției este *void* înseamnă că funcția nu returnează un rezultat prin nume.
- Funcția are variabila proprie (locală) *i*.
- Funcția întoarce un anumit rezultat (în cazul funcției *prim*, o valoare booleană), prin intermediul instrucțiunii *return*.

**Parametrii unui subprogram** sunt de două tipuri:

- Parametri formali – cei ce se găsesc în antetul subprogramului;
- Parametri actuali (efectivi) – cei care se utilizează la apel.

De exemplu, în linia:

```
if (!verifica(k))
```

parametrul *k* este un parametru efectiv.

### Declararea variabilelor

- Sistemul de operare alocă fiecărui program trei zone distincte în memoria internă în care se găsesc memorate variabilele programului:
  - Segment de date
  - Segment de stivă
  - Heap.

Există și posibilitatea ca variabilele să fie memorate într-un anumit registru al microprocesorului, caz în care accesul la acestea este foarte rapid.
- O variabilă se caracterizează prin 4 atribute:
  - Clasa de memorare – locul unde este memorată variabila respectivă; o variabilă poate fi memorată în:
    - segmentul de date (variabilele globale)
    - segmentul de stivă (în mod implicit, variabilele locale)
    - heap
    - un registru al microprocesorului (în mod explicit, variabilele locale).
  - Vizibilitatea – precizează liniile textului sursă din care variabila respectivă poate fi accesată. Există următoarele tipuri de vizibilitate:
    - la nivel de bloc (instrucțiune compusă) (variabilele locale);
    - la nivel de fișier (în cazul în care programul ocupă un singur fișier sursă) (variabilele globale, dacă sunt declarate înaintea tuturor funcțiilor);
    - la nivel de clasă (în legătură cu programarea orientată pe obiecte).
  - Durata de viață – timpul în care variabila respectivă are alocat spațiu în memoria internă. Avem:
    - durată statică – variabila are alocat spațiu în tot timpul execuției programului (variabilele globale);

- durată locală – variabila are alocat spațiu în timpul în care se execută instrucțiunile blocului respectiv (variabilele locale);
- durată dinamică – alocarea și dealocarea spațiului necesar variabilei respective se face de către programator prin operatori și funcții speciale.
  - Tipul variabilei.
- În C++ variabilele pot fi împărțite în 3 mari categorii: locale, globale și dinamice.

### Transmiterea parametrilor

Parametrii actuali trebuie să corespundă celor formali, ca număr și tip de date. Tipul parametrilor actuali trebuie fie să coincidă cu tipul parametrilor formali, fie să poată fi convertit implicit la tipul parametrilor formali.

- Pentru memorarea parametrilor, subprogramele folosesc segmentul de stivă, la fel ca pentru variabilele locale.
- Memorarea se face în ordinea în care parametrii apar în antet.
- În cadrul subprogramului, parametrii transmiși și memorați în stivă sunt variabile. Numele lor este cel din lista parametrilor formali.
- Variabilele obținute în urma memorării parametrilor transmiși sunt variabile locale.
- La revenirea în blocul apelant, conținutul variabilelor memorate în stivă se pierde.

Transmiterea parametrilor se poate realiza prin intermediul a două mecanisme:

- prin valoare
- prin referință.

**Transmiterea prin valoare** este utilizată atunci când dorim ca subprogramul să lucreze cu acea valoare, dar, în prelucrare, nu ne interesează ca parametrul actual (din blocul apelant) să rețină valoarea modificată în subprogram.

În toate apelurile din exemplul precedent, transmiterea parametrilor se realizează prin valoare.

**Observație:** Dacă nu ar exista decât acest tip de transmitere, ar fi totuși posibil să modificăm valoarea anumitor variabile care sunt declarate în blocul apelant. Acest lucru se realizează în situația în care am lucra cu variabile de tip pointer.

**Exemplu:** Să se scrie o funcție care interschimbă valorile a două variabile. Transmiterea parametrilor se va face prin valoare.

```
#include <iostream.h>
void interschimba(int *a, int *b)
{
    int aux = *a;
    *a = *b;
    *b = aux;
}

main()
```

```

{
    int x = 2, y = 3;
    interschimba(&x, &y);
    cout<< x << " " << y;
}

```

**Observație:** În limbajul C, acesta este singurul mijloc de transmitere a parametrilor.

Transmiterea prin valoare a tablourilor permite ca funcțiile să returneze noile valori ale acestora (care au fost modificate în funcții). Explicația este dată de faptul că numele tabloului este un pointer către componentele sale. Acest nume se transmite prin valoare, iar cu ajutorul său accesăm componentele tabloului.

**Exemplu:** Să se scrie o funcție care inițializează un vector transmis ca parametru.

```

#include <iostream.h>

void vector (int x[10])
{
    for (int i = 0; i < 10; i++)
        x[i] = i;
}

main()
{
    int a[10];
    vector(a);
    for (int i=0; i < 10; i++)
        cout << a[i] << " ";
}

```

**Transmiterea prin referință** este utilizată atunci când dorim ca la revenirea din subprogram variabila transmisă să rețină valoarea stabilită în timpul execuției programului.

În acest caz, parametrii actuali trebuie să fie referințe la variabile. La transmitere, subprogramul reține în stivă adresa variabilei.

La compilare, orice referință la o variabilă este tradusă în subprogram ca adresare indirectă. Acesta este motivul pentru care în subprogram putem adresa variabila normal (nu indirect), cu toate că, pentru o variabilă transmisă, se reține adresa ei.

**Exemplu:** Să se scrie o funcție care interschimbă valorile a două variabile. Transmiterea parametrilor se va face prin referință.

```

#include <iostream.h>
void interschimba(int &a, int &b)
{
    int aux = a;

```

```

        a = b;
        b = aux;
    }

main()
{
    int x = 2, y = 3;
    interschimba(x, y);
    cout<< x << " " << y;
}

```

Acest program, scris în limbajul Pascal, are următoarea formă:

```

program transm_referinta;
var x, y: integer;
procedure interschimba(var a,b: integer);
var aux: integer;
begin
    aux := a;
    a := b;
    b := aux;
end;

procedure nu_interschimba(a,b: integer);
var aux: integer;
begin
    aux := a;
    a := b;
    b := aux;
end;

begin
    x := 31;
    y := 21;
    interschimba(x, y);
    writeln(x, ' ', y);
    writeln('-----');
    nu_interschimba(x, y);
    writeln(x, ' ', y);
end.

```

## Supraîncărcarea funcțiilor

În C++ există posibilitatea ca mai multe funcții să poarte același nume. În această situație, funcțiile trebuie să fie diferite fie ca număr de parametri, fie ca tip. În acest din urmă caz, este necesară o condiție suplimentară: parametrii efectivi să nu poată fi convertiți implicit către cei formali.

**Exemplu:** Să se scrie o funcție supraîncărcată care afișează aria pătratului, respectiv a dreptunghiului, în funcție de numărul de parametri.

```
#include <iostream.h>

void arie(double latura)
{
    cout << "aria patratului este "<< latura * latura;
}

void arie(double lung, double lat)
{
    cout << "aria dreptunghiului este "<< lung * lat;
}

main()
{
    arie(3);
    arie (4, 7);
    double d = 7;
    arie(&d);
}
```

Exerciții:

- 1) Se da un sir de numere intregi. Se cauta un subsir cu lungimea cuprinsa intre 1 si  $u$ , format din elemente consecutive ale sirului initial, cu suma elementelor maxima.
- 2) Se considera un numar natural  $n$ . Determinati cel mai mic numar  $m$  care se poate obtine din  $n$ , prin eliminarea unui numar de  $k$  cifre din acesta fara a modifica ordinea cifrelor ramase.
- 3) Se citesc de la tastatura un numar natural  $n$  si un sir de numere naturale. Sa se scrie un program care afiseaza indicii  $i$  si  $j$  care indeplinesc urmatoarele conditii:
  - a)  $1 \leq i < j \leq n$ ;
  - b)  $a_i > a_k, a_j > a_k, \forall k, i+1 \leq k \leq j-1$ ;
  - c) *diferenta  $j-i$  este maxima.*

Probleme propuse:

- 1) Se considera  $N$  numere intregi care trebuie repartizate in  $p$  grupuri. Grupurile sunt identificate prin numere naturale cuprinse între  $1$  si  $p$ . Repartizarea în grupuri trebuie sa se realizeze astfel incat suma numerelor din oricare grup  $i$  sa fie divizibila cu numarul total de numere care fac parte din grupurile identificate prin valori cuprinse între  $i$  si  $p$ .
- 2) Consideram ca toate punctele de coordonate intregi din plan (coordoate mai mici de 1000) sunt colorate in negru, cu exceptia a  $n$  puncte care sunt colorate in rosu. Doua puncte rosii aflate pe aceeasi linie verticala (au aceeasi ordonata sau aceeasi abscisa) pot fi unite printr-un segment. Coloram in rosu toate punctele de coordonate intregi de pe acest segment. Repetam operatia cat timp se obtin puncte rosii noi. Cunoscand coordonatele celor  $n$  puncte care erau initial rosii, aflati numarul maxim de puncte rosii care vor exista in final.
- 3) Se considera o matrice binara de dimensiune  $m \times n$  (elementele matricei sunt 0 sau 1). Se cere sa se determine aria maxima care poate fi acoperita de doua dreptunghiuri care contin numai elemente cu valoare 0 (dreptunghiurile nu se pot suprapune).

*in.txt*

6 8

1 0 0 0 0 0 0 0

1 0 0 0 0 0 0 0

1 1 1 0 0 0 1 1

0 0 1 0 0 0 1 1

0 0 1 0 0 0 1 1

0 0 1 1 1 1 1 1

*out.txt*

23