SUBPROGRAME

Un subprogram este un ansamblu ce poate conţine tipuri de date, variabile şi instrucţiuni destinate unei anumite prelucrări (calcule, 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: funcţii şi proceduri. În C/C++, este utilizată doar noţiunea de funcţie, o procedură din Pascal fiind asimilată în C/C++ unei funcţii 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:

- matematice: *sin*, *cos*, *exp*, *log*;
- de citire/scriere: *read*/*write* (Pascal), *scanf*/*printf* (C/C++)
- de prelucrare a şirurilor 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 conţine:

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

Soluţia 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);
                  //ieşire fara eroare
                  exit(0);
            }
      //afisare rezultat
      printf("proprietatea este indeplinita pentru numere >2 si
             <=%d\n", n);
}
```

Observaţie: Problema a fost descompusă în altele mai mici. Rezolvarea unei probleme complexe este mai uşoară 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 soluţii ale problemei iniţiale;
- reducerea numărului de erori care pot apărea la scrierea programelor şi despistarea cu uşurinţă a acestora.

Elementele unui subprogram sunt prezentate în continuare, cu referire la noţiunea de funcţie din C/C++. Pentru funcţiile şi procedurile din Pascal, aceste elemente se pot distinge în mod similar.

- Antetul funcţiei *prim* este: `bool prim(int n)`
- Numele funcţiei este `prim`
- Funcţia *prim* are un parametru *n*.
- Funcţia are un tip, care reprezintă tipul de date al rezultatului său. Tipul funcţiei *prim* este *bool*.

În cazul programelor *C*/*C++*, dacă tipul funcţiei este *void* înseamnă că funcţia nu returnează un rezultat prin nume.

- Funcţia are variabila proprie (locală) *i*.
- Funcţia întoarce un anumit rezultat (în cazul funcţiei *prim*, o valoare booleană), prin intermediul instrucţiunii *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.

Transmiterea parametrilor

Parametrii actuali trebuie să corespundă celor formali, ca număr şi 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 transmişi şi memoraţi în stivă sunt variabile. Numele lor este cel din lista parametrilor formali.
- Variabilele obţinute în urma memorării parametrilor transmişi sunt variabile locale.
- La revenirea în blocul apelant, conţinutul variabilelor memorate în stivă se pierde.

Transmiterea parametrilor se poate realiza prin intermediul a două mecanisme:

- prin valoare
- prin referinţă.

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 reţină valoarea modificată în subprogram.

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

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

Exemplu: Să se scrie o funcţie 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;
}
```

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

Transmiterea prin valoare a tablourilor permite ca funcţiile să returneze noile valori ale acestora (care au fost modificate în funcţii). Explicaţia 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 funcţie care iniţializează 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 referinţă este utilizată atunci când dorim ca la revenirea din subprogram variabila transmisă să reţină valoarea stabilită în timpul execuţiei programului.

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

La compilare, orice referinţa 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 reţine adresa ei.

Exemplu: Să se scrie o funcţie care interschimbă valorile a două variabile.

```
#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;}
```

Supraîncărcarea funcţiilor

În C++ există posibilitatea ca mai multe funcţii să poarte acelaşi nume. În această situaţie, funcţiile trebuie să fie diferite fie ca număr de parametri, fie ca tip. În acest din urmă caz, este necesară o condiţie suplimentară: parametrii efectivi să nu poată fi convertiţi implicit către cei formali.

Exemplu: Să se scrie o funcţie supraîncărcată care afişează aria pătratului, respectiv a dreptunghiului, în funcţie 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);
}
```

Vectori alocaţi dinamic

Structura de vector are avantajul simplităţii şi economiei de memorie faţă de alte structuri de date folosite pentru memorarea unei colectii de informaţii între care există anumite relaţii. Între cerinţa de dimensionare constantă a unui vector şi generalitatea programelor care folosesc astfel de vectori există o contradicţie. De cele mai multe ori programele pot afla (din datele citite) dimensiunile vectorilor cu care lucrează şi deci pot face o alocare dinamică a memoriei pentru aceşti vectori. Aceasta este o solutie mai flexibilă, care foloseşte mai bine memoria disponibilă şi nu impune limitări arbitrare asupra utilizării unor programe.

În limbajul C nu există practic nici o diferenţă între utilizarea unui vector cu dimensiune fixă şi utilizarea unui vector alocat dinamic, ceea ce încurajează şi mai mult utilizarea unor vectori cu dimensiune variabilă.

Exemplu:

```
int main() {
  int n,i;
  int *a; // Adresa vector alocat dinamic

  printf("n = ");
  scanf("%d", &n); // Dimensiune vector

  a = calloc(n, sizeof(int));
// Alternativ: a = malloc(n * sizeof(int));
  printf("Componente vector: \n");

  for (i = 0; i < n; i++) {
    scanf("%d", &a[i]);
  }
  for (i = 0; i < n; i++) {    // Afisare vector
    printf("%d ",a[i]);
  }

  free(a);    // Eliberam memoria !!!

  return 0;
}
```

Există şi cazuri în care datele memorate într-un vector rezultă din anumite prelucrări, iar numărul lor nu poate fi cunoscut de la începutul execuţiei. Un exemplu poate fi un vector cu toate numerele prime mai mici ca o valoare dată (*Goldbach*). În acest caz se poate recurge la o realocare dinamică a memoriei.

Exemplu: se citeşte un număr necunoscut de valori întregi într-un vector care poate fi extins

```
#define INCR 100 // cu cat creste vectorul la fiecare realocare

int main() {
  int n, i, m;
  float x, *v; // v = adresa vector
  n = INCR;  i = 0;

  v = malloc(n * sizeof(float)); // Dimensiune initiala vector

  while (scanf("%f", &x) != EOF) {
    if (++i == n) {              // Daca este necesar...
      n = n + INCR;              // ... creste dimensiune vector
      v = realloc(vector, n * sizeof(float));
    }

    v[i] = x;        // Memorare in vector numar citit
  }

  m = i;

  for (i = 0; i < m; i++) {       // Afisare vector
    printf("%f ",v[i]);
  }

  free(vector);

  return 0;
}
```

Exerciţii:

NOTĂ: Pentru toate funcțiile de mai jos, determinați și argumentați complexitatea lor.

1. Definiți următoarele 5 funcții:
- *read* – citește de la tastatură un tablou unidimensional;
- *positives* – returnează un tablou unidimensional alocat dinamic format din valorile strict pozitive dintr-un tablou unidimensional cu elemente de tip întreg, precum și numărul acestora;
- *multiply* – înmulțește cu x fiecare element al tabloului unidimensional v format din n numere întregi;
- *sort* – sortează crescător tabloul unidimensional v format din n numere întregi;
- *display* – afișează pe ecran tabloul unidimensional v format din n numere întregi.

Folosind doar apeluri utile ale celor 5 funcții definite anterior, scrieți un program care să afișează în ordine descrescătoare numerele strict negative dintr-un tabloul unidimensional format din numere întregi.

2.
- Scrieți o funcție care să citească numărul de elemente ale unui tablou unidimensional, să aloce dinamic tabloul respectiv și apoi să citească valorile elementelor sale.
- Scrieți o funcție care să returneze poziția primului element strict mai mare decât un număr real x dintr-o secvență cuprinsă între doi indici i și j ($0 \leq i \leq j < n$) a unui tablou unidimensional format din n numere reale sau -1 dacă nu există nici un astfel de element.
- Scrieți o funcție care, folosind apeluri utile ale funcției definite anterior, afișează mesajul "Da" în cazul în care un tablou de numere reale, primit ca parametru, este sortat strict descrescător sau mesajul "Nu" în caz contrar.

3.
- Scrieți o funcție care să rearanjeze în ordine inversă elementele unei secvențe cuprinse între doi indici i și j ($0 \leq \mathrm{i} \leq \mathrm{j} < \mathrm{n}$) dintr-un tablou unidimensional de numere întregi, fără a folosi un tablou suplimentar.
- Scrieți o funcție care, pentru un număr natural k, rearanjează elementele unui tablou unidimensional t format din n numere întregi $\mathrm{t}[0], \mathrm{t}[1], \ldots, \mathrm{t}[\mathrm{n}-1]$ în ordinea $\mathrm{t}[\mathrm{k}+1], \ldots, \mathrm{t}[\mathrm{n}-1], \mathrm{t}[0], \ldots, \mathrm{t}[\mathrm{k}]$ ($0 \leq \mathrm{k} < \mathrm{n}$), folosind doar apeluri utile ale funcției definite anterior.

4.
- Scrieți o funcție care să calculeze frecvența de apariție a unui număr întreg x într-o secvență cuprinsă între doi indici i și j ($0 \leq \mathrm{i} \leq \mathrm{j} < \mathrm{n}$) dintr-un tablou unidimensional format din n numere întregi.
- Scrieți o funcție care, folosind apeluri utile ale funcției definite anterior, afișează mesajul "Da" în cazul în care un tablou unidimensional de numere întregi, primit ca parametru, are elemente distincte sau mesajul "Nu" în caz contrar.

5.

- Se da un sir de numere intregi. Se cauta un subsir cu lungimea cuprinsa intre l si u, format din elemente consecutive ale sirului initial, cu suma elementelor maxima.

6.

- 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.

7.

- Se citesc de la tastatura un numar natural n si un sir de numere naturale. Sa se scrie un program care aafiseaza indicii i si j care indeplinesc urmatoarele conditii:
  $a)\ 1 \le i < j \le n;$
  $b)\ a_i > a_k, a_j > a_k, \forall k, i+1 \le k \le j-1;$
  $c)\ diferenta\ \ j-i\ este\ \max ima.$

Probleme propuse:

8.

- Se considera N numere intregi care trebuie repartizate in p grupuri. Grupurile sunt identificate prin numere naturale cuprinse intre 1 si p. Repartizarea in 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 intre i si p.

9.

- Consideram ca toate punctele de coordonate intregi din plan (coordonate 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.

10.

- Se considera o matrice binara de dimensiune m x 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	*out.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	23