**Programare Procedurala (Funcții în C)**

**Noțiuni de bază**

* O funcție este un subprogram utilizat pentru efectuarea anumitor operații și

care eventual returnează un rezultat. Fiecare program C este de fapt alcătuit

dintr-o succesiune de funcții, printre care se află  funcția principală  main().

Structura generală a unei funcții în C este:

**tip nume (lista\_parametrii\_formali)**

**{**

**declaratii variabile locale**

**instructiuni**

**}**

Unde:

* tip - rezultatul returnat de funcție
* nume - numele funcției
* lista\_parametrii\_formali - listă de declarații de parametri separați

prin virgulă

Observații:

* Dacă tipul returnat de funcție este void atunci nu se returnează nimic,

doar se efectuează anumite operații (echivalentul unei proceduri în pascal).

* Dacă tipul returnat este diferit de void, atunci este necesară o instrucțiune de tip return valoare; unde valoare are tipul tip.
* Pot exista mai multe instrucțiuni de tip return. Execuția funcției se termină atunci când s-a executat ultima instrucțiune sau s-a ajuns la o instrucțiune de tip return.
* Lista de parametrii poate fi vidă, în acest caz se utilizează  void.
* Dacă  tipul returnat nu este specificat se consideră implicit tipul int.

Apelul unei funcții: poate avea loc în cadrul funției principale main() sau

al altei funcții și este de forma:

**nume(lista\_parametri\_actuali);**

**sau**

**variabila=nume(lista\_parametri\_actuali);**

**(dacă  tipul returnat este diferit de void).**

Numărul și tipul parametrilor actuali trebuie să fie aceleași ca și cazul parametrilor

formali.

Declararea functiilor

**Exemple:**

* **void f1 (int, int);**

Functia f1 () are doi parametri de tip int si nu intoarce nici un rezultat.

* **void clrscr ();**

Functia clrscr () este 0 functie declarata in fisierul antet conio. h si are rolul de a sterge fereastra text curenta si de a plasa cursorul in coltul din stanga sus al ferestrei. Este o functie care nu are nici un parametru si nu intoarce nici un rezultat.

**Variabilele globale:**

* pozitie: sunt declarate in exteriorul oricarei functii;
* clasa de memorare: li se aloca memorie in segmentul de date al programului, memoria alocata fiind automat initializata cu 0;
* durata de viata: memoria ramane alocata pana la sfarsitul executiei programului;
* domeniu de vizibilitate: sunt vizibile din momentul declararii pana la sfarsitul programului, in toate modulele acestuia, chiar si in alte fisiere sursa.

**Variabilele locale:**

* pozitie: sunt declarate in blocul unei functii;
* clasa de memorare: Ii se aloca memorie pe stiva, memoria alocata nefiind initializata automat;
* durata de viata: memoria ramane alocata pana la sfarsitul executiei blocului in care este declarata variabila;
* domeniu de vizibilitate: sunt vizibile numai in blocuJ in care sunt declarate.

**Exemplu:**

* Scriem un program care citeste un vector a cu n elemente intregi si un vector b cu m elemente intregi, apoi afiseaza pe ecran cei doi vectori cititi. Folosim functie de citire si de afisare.

#include <stdio.h>

#include<conio.h>

void citire\_vector(int \* nr, int x[10])

{

int i;

printf("Nr. elemente = ");

scanf\_s("%d", nr);

printf("Elementele vectorului: ");

for (i = 0; i < \*nr; i++)

scanf\_s("%d", &x[i]);

}

void afisare\_vector(int nr, int a[10])

{

int i;

printf("Elementele vectorului sunt: ");

for (i = 0; i < nr; i++)

printf("%d", a[i]);

}

void main()

{

int n, a[10], m, b[10];

citire\_vector(&n, a);

citire\_vector(&m, b);

afisare\_vector(n, a);

afisare\_vector(m, b);

\_getch();

}

Transmiterea parametrilor prin valoare

La apelul unei functii valorile parametrilor actuali sunt atribuite, în ordine, parametrilor corespunzatori. Pentru aceasta se aloca spatiu pentru valorile parametrilor actuali, memorie care va fi eliberata la iesirea din functie.

Atentie! Acest lucru are ca urmare revenirea parametrilor actuali la valorile dinainte de apelul functiei, chiar daca în functie au fost modificati.

**Acest tip de apel se numeste apel prin valoare!!**

Observatie! Deoarece numele de tablou (vector, matrice etc.) retine adresa de început a zonei de memorie alocata pentru elementele tabloului, modificarile produse asupra vectorului se pastreaza si la iesirea din functie (se efectuaza la adresele corespunzatoare!).

Transmiterea parametrilor prin adresa

Daca dorim ca modificarile efectuate în functie asupra unui parametru sa se **pastreze si dupa terminarea functiei** este necesar sa  transmitem acei parametrii prin adresa. Pentru aceasta parametrul formal va fi de tip pointer (va contine o adresa) iar parametrul actual va fi adresa variabiei pe care dorim sa o modificam.

**Exemplu:** Interschimbarea valorii a doua variabile prin intermediul unei functii.

void schimba(int \*x, int \*y)

{

int aux;

aux=\*x;

\*x=\*y;

\*y=aux;

}

void main();

{

int a,b;

printf("a="); scanf("%d",&a);

printf("b="); scanf("%d",&b);

schimba(&a,&b);

printf("a=%d, b=%d",a,b);

}

**Tablou bidimensional**

1. Se da o matrice n linii si m coloane. Cate elemente din matrice au exact 2 cifre si sunt numere prime?

#include <stdio.h>

#include <malloc.h>

#include<conio.h>

int doua\_cifre(int n)

{

int nr = 0;

while (n != 0)

{

nr++;

n = n / 10;

}

return nr;

}

int nr\_prim(int n)

{

int OK = 0, i;

for (i = 2; i<n /2; i++)

if (n%i == 0)

OK = 1;

return OK;

}

void main()

{

int m, n, \*\*A, i, j, k = 0;

printf("Dati n: ");

scanf\_s("%d", &n);

printf("Dati m: ");

scanf\_s("%d", &m);

A = (int\*\*)malloc(n\*sizeof(int \*));

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

A[i] = (int\*)malloc(m\*sizeof(int));

for (int i = 0; i<n; i++)

for (j = 0; j<m; j++)

scanf\_s("%d", &A[i][j]);

for (int i = 0; i<n; i++)

for (int j = 0; j<m; j++) {

if (doua\_cifre(A[i][j]) == 2 && nr\_prim(A[i][j]) == 0)

k++;

}

printf("Numarul elementelor este: %d", k);

\_getch();

}

**2. Se da un sir de numere naturale. Se cere sa se adauge cate un 0 dupa fiecare numar care contine doar cifre pare.**

#include <stdio.h>

#include <malloc.h>

#include<conio.h>

int test\_cifre\_pare(int n)

{

int ok = 0;

while (n != 0)

{

if (n % 10 % 2 == 1)

ok = 1;

n = n / 10;

}

return ok;

}

void main()

{

int \*a, n, i;

printf("dimensiune sir=");

scanf\_s("%d", &n);

a = (int \*)malloc(n\*sizeof(int\*));

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

{

printf("a[%d]=", i);

scanf\_s("%d", &a[i]);

}

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

{

printf("Elementul testat: %d", a[i]);

if (test\_cifre\_pare(a[i]) % 2 == 0)

{

n++;

a = (int \*)realloc(a, n\*sizeof(int));

for (int j = n - 2; j >= i + 1; j--)

a[j + 1] = a[j];

a[i + 1] = 0;

i++;

}

printf(" n-ul este: %d \n", n);

}

printf(" Vectorul este: ");

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

printf(" %d", a[i]);

free(a);

\_getch();

}

**Avantajele utilizării functiilor:**

* elaborarea algoritmilor prin descompunerea problemei ce trebuie a fi rezolvată în mai multe probleme mai simple – în acest caz vom rezolva probleme mai simple
* modularizarea problemei (descompunerea in subprobleme)
* reutilizarea codului - subprogramul odată scris poate fi utilizat şi în alte programe
* reducerea numărului de erori care apar scrierea programului
* depistarea şi corectarea cu uşurinţă a erorilor
* realizarea unor programe uşor de urmărit.

**PROBLEME DE REZOLVAT LA LABORATOR**

1. Citirea elementelor unui vector de numere întregi și determinarea elementului maxim.
2. Sa se scrie un program care determina numerele prime care au cifrele identice.
3. Se da o matrice cu m linii si n coloane. Cate elemente din matrice sunt deasupra diagonalei principale si au exact 3 cifre pare?
4. Se da o matrice cu m linii si n coloane. Cate elem au ultima si penultima cifra > 3 si egale?
5. Se da un sir de numere naturale. Sa se elimine elementul cu cele mai multe cifre din sir.

**TEMA**

1. Sa se scrie un program care afiseaza media aritmetica a elementelor care au ultima cifra para si numarul de divizori egal cu 3.
2. Se da o matrice. Sa se numere cate elemente sunt in matrice cu cifrele in ordine crescatoare.
3. Se da un sir. Dupa fiecare element care are numarul de cifre pare egal cu numarul de cifre impare, sa se introduca media aritmetica a cifrelor.
4. Se da o matrice. Se cere sa se puna cate o coloana cu 0 (zero) dupa fiecare coloana care contine un numar prim.