Laborator 4. Programare modulară. Funcţii în limbajul C. Dezvoltarea algoritmilor folosind funcţii

Antetul funcției

Funcţiile împart taskuri complexe în bucăţi mici mai uşor de înţeles şi de programat.
Crearea funcţiilor trebuie să se bazeze pe următoarele principii: claritate, lizibilitate, uşurinţă în întreţinere, reutilizabilitate.
Antetul unei funcţii este:

```
tip_returnat nume_functie (tip_param1 nume_param1
    , tip_param2 nume_param2, ...);
```

Definirea funcției

```
tip_returnat nume_functie(tip_param1 nume_param1,
    tip_param2 nume_param2, ...) {
  declaratii de variabile si instructiuni;

  return expresie;
}
```

Limbajul C permite separarea declaraţiei unei funcţii de definiţia acesteia (codul care o implementează). Pentru ca funcţia să poată fi folosită, este obligatorie doar declararea acesteia înainte de codul care o apelează.
Definiţia poate apărea mai departe în fişierul sursă, sau chiar într-un alt fişier sursă sau bibliotecă.
Tipul returnat de o funcţie poate fi orice tip standard sau definit de utilizator, inclusiv tipul void (care înseamnă că funcţia nu returnează nimic).
Orice funcţie care întoare un rezultat trebuie să conţină instrucţiunea:

```
 return expression;
```

```
int min(int x, int y); //declarare

int min(int x, int y) { //definire
  if (x < y) {
    return x;
  }

  return y;
}
```

Apelul funcțiilor

Apelul unei funcţii se face specificând parametrii efectivi (parametrii care apar în declararea funcţiei se numesc parametri formali).

```
int main() {
  int a, b, minimum;
  //..........
  x = 2;
  y = 5;
  minimum = min(x, 4);
  printf("Minimul␣dintre␣%d␣si␣4␣este:␣%d", x,
      minimum);
  printf("Minimul␣dintre␣%d␣si␣%d␣este:␣%d", x, y,
       min(x, y));
}
```

Transmiterea parametrilor

Transmiterea parametrilor prin valoare
Funcţia va lucra cu o copie a variabilei pe care a primit-o şi orice modificare din cadrul funcţiei va opera asupra aceste copii. La sfârşitul execuţiei funcţiei, copia va fi distrusă şi astfel se va pierde orice modificare efectuată.

```
min(x, 4);  // se face o copie lui x
```

Funcții recursive

O funcţie poate să apeleze la rândul ei alte funcţii. Dacă o funcţie se apelează pe sine însăşi, atunci funcţia este recursivă. Pentru a evita un număr infinit de apeluri recursive, trebuie ca funcţia să includă în corpul ei o condiţie de oprire, astfel ca, la un moment dat, recurenţa să se oprească şi să se revină succesiv din apeluri.

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

sau

int fact(int n) {
  return (n >= 1) ? n * fact(n - 1) : 1;
}
```

Funcția main

Orice program C conţine cel puţin o funcţie, şi anume cea principală, numită main(). Aceasta are un format special de definire:

```
int main(int argc, char *argv[])
{
    // some code
    return 0;
}
```

Primul parametru, argc, reprezintă numărul de argumente primite de către program la linia de comandă, incluzând numele cu care a fost apelat programul.
Al doilea parametru, argv, este un pointer către conţinutul listei de parametri al căror număr este dat de argc.
În mod normal, orice program care se execută corect va întoarce 0, şi o valoare diferită de 0 în cazul în care apar erori.

Tipul de date void

Tipul de date void are mai multe întrebuinţări.
Atunci când este folosit ca tip returnat de o funcţie, specifică faptul că funcţia nu întoarce nici o valoare.
Exemplu:

```
void print_nr(int number) {
  printf("Numarul␣este␣%d", number);
}
```

Atunci când este folosit în declaraţia unei funcţii, void semnifică faptul că funcţia nu primeşte nici un parametru.
Exemplu:

```
int init(void) {
  return 1;
}
```