Programarea Calculatoarelor

Cursul 4: Funcții

Ion Giosan
Universitatea Tehnică din Cluj-Napoca
Departamentul Calculatoare

Introducere

- Funcții
  - Permit modularizarea programelor
  - Variabilele declarate în interiorul funcțiilor – variabile locale (vizibile doar în interior)
- Parametrii funcțiilor
  - Permit comunicarea informației între funcții
  - Sunt variabile locale funcțiilor
- Avantajele utilizării funcțiilor
  - Divizarea problemei în subprobleme
  - Managementul dezvoltării programelor
  - Utilizarea/Reutilizarea funcțiilor scrise în alte programe
  - Abstractizare – ascunderea informației interne (funcții în biblioteci)
  - Elimină duplicarea codului scris

Structura unei funcții

```
tip_returnat nume(lista_parametri_formali)
{
   declaratii
   instructiuni
}
```

- **`nume`** – un identificator valid
- **`tip_returnat`** – tipul de dată al rezultatului (implicit este **`int`**)
- Primul rând se numește antetul funcției (*header*)
- Lista de parametri formali poate conține
  - Nici un parametru:

    ```
    tip_returnat nume()
    tip_returnat nume(void)
    ```

  - Unul sau mai mulți parametri separați prin virgulă. Un parametru formal este specificat prin **`tip identificator`**

Valoarea returnată

- Două categorii de funcții
  - Care returnează o valoare: prin utilizarea instrucțiunii

    `return expression;`
  - Care nu returnează nicio valoare: prin instrucțiunea

    `return;`

    În acest caz **`tip_returnat`** este înlocuit cu **`void`**
- Returnarea valorii
  - Declarațiile și instrucțiunile din funcții sunt executate până se întâlnește
    - Instrucțiunea **`return`**

      sau
    - Până execuția atinge finalul funcției – acolada închisă **`}`**

Valoarea returnată

```
int adunare(int a, int b)
{
    printf("Functie care calculeaza si returneaza suma a
            doi intregi\n");
    int suma;
    suma = a+b;
    return suma;
    printf("Aceasta instructiune nu se mai executa\n");
}

int max_int()
{
    printf("Functie care returneaza cel mai mare intreg
           pozitiv\n");
    return 0x7FFFFFFF;
}
```

Valoarea returnată

```
void impar(int x)
{
    printf("Functie care afiseaza daca un numar este par
            sau impar\n");
    if (x%2==1)
        printf("%d este numar impar\n",x);
    else
        printf("%d este par\n",x);
    return;
    printf("Aceasta instructiune nu se mai executa\n");
}

void patrat(int a)
{
    printf("Functie care afiseaza patratul unui numar\n");
    printf("%d^2=%d\n",a,a*a);
}
```

Prototipul și argumentele funcțiilor

- **Prototipul** unei funcții constă în specificarea antetului urmat de caracterul `;`
  - Nu este necesară specificarea numelor parametrilor formali

```
int adunare(int, int);
```

  - Este necesară inserarea prototipului unei funcții înaintea altor funcții în care este invocată dacă implementarea ei este localizată după implementarea acelor funcții
- **Parametrii** apar în definiții
- **Argumentele** apar în apelurile de funcții
  - În limbajul C - trimiterea argumentelor se face prin valoare
- Regula de **conversie a argumentelor**
  - În cazul în care diferă, tipul fiecărui argument este convertit automat la tipul parametrului formal corespunzător
    - Ca și în cazul unei simple atribuiri

Prototipul și argumentele funcțiilor

```c
double f(double t)
{
    return t-1.5;
}

float g(int);

int main()
{
    float a=11.5f;
    printf("%f\n",f(a));
    printf("%f\n",g(a));
    return 0;
}

float g(int z)
{
    return z+2.f;
}
```

Rezultat afișat

10.000000

13.000000

Argumente de tip șiruri

- Lungimea șirului unidimensional poate să nu fie specificată în lista de parametri
  - Nu se poate determina lungimea acestuia nici măcar utilizând funcția **sizeof**

```
int f(int a[]){ // nu este specificata dimensiunea
    …
}
```

  - Valoarea **sizeof(a)** arată dimensiunea adresei de memorie a primului element din tablou
- Lungime variabilă specificată (începând cu C99)
  - Trebuie ca n să fie un parametru anterior

```
int f(int n, int a[n]){
  …
}
```

- În cazul în care argumentul este numele unui tablou
  - Acesta are ca valoare adresa primului element
  - Se transmite adresa ca valoare pentru parametrul formal corespunzător
  - Funcţia respectivă poate modifica elementele tabloului al cărui nume s-a folosit ca parametru actual

Argumente de tip șiruri

```
#include <stdio.h>
#define N 10
void citire_sir(int n, int a[n])
{
    for (int i=0; i<n; i++)
    {
        printf("a[%d]=",i);
        scanf("%d",&a[i]);
    }
}
void afisare_sir(int n, int a[])
{
    for (int i=0; i<n; i++)
        printf("a[%d]=%d\n",i,a[i]);
}
int main()
{
    int nr, a[N];
    printf("Nr. elemente (maxim 10)=");
    scanf("%d", &nr);
    citire_sir(nr, a);
    afisare_sir(nr, a);
    return 0;
}
```

Fișiere *header* – cu extensia *.h*

- Conțin prototipuri de funcții
- Bibliotecile standard
  - Conțin prototipuri de funcții standard regăsite în fișierele *header* corespunzătoare (ex. **`stdio.h, stdlib.h, math.h, string.h`**)
  - Exemplu – biblioteca **`stdio.h`** care conține și prototipul funcției **`printf`**:

```
int printf(const char* format, ...);
```

  - Se încarcă cu **`#include <filename.h>`**
- Biblioteci utilizator
  - Conțin prototipuri de funcții și macrouri
  - Se pot salva ca fișiere cu extensia *.h* : ex. **`filename.h`**
  - Se încarcă cu **`#include "filename.h"`**

Apelul funcțiilor

- Funcție care nu returnează nici o valoare

```
nume(lista_parametri_actuali);
```

- Funcție care returnează o valoare
  - Ca și mai sus, valoarea returnată fiind pierdută
  - Ca și un operand într-o expresie, valoarea returnată fiind utilizată în evaluarea expresiei
    - Exemplu de memorare a valorii returnate într-o variabilă:

```
variabila=nume(lista_parametri_actuali);
```

- Corespondența între parametrii formali și actuali este **pozițională**
  - În cazul în care tipul unui parametru actual este diferit de tipul parametrului formal corespunzător, acesta este convertit automat la tipul parametrului formal

Apelul funcțiilor

- Utilizat la invocarea funcțiilor

- În limbajul C apelul se poate face doar prin **valoare**
  - O copie a argumentelor este trimisă funcției
  - Modificările în interiorul funcției nu afectează argumentele originale

- În limbajul C++ apelul se poate face și prin **referință**
  - Argumentele originale sunt trimise funcției
  - Modificările în interiorul funcției afectează argumentele trimise

Apel prin valoare

Valorile argumentelor **a** și **b** **nu** sunt interschimbate după apelul funcției

```
#include <stdio.h>
#include <stdlib.h>
/* interschimbarea valorilor a si b */
void interschimba(int a, int b)
{
   int aux;
   printf("\nLa intrarea in functie: a=%d b=%d\n", a, b);
   aux = a; a = b; b = aux;
   printf("\La iesirea din functie: a=%d b=%d\n", a, b);
}
int main()
{
   int a=3, b=2;
   printf("\nIn main inainte de apelul functiei: a=%d b=%d\n", a, b)
   interschimba(a, b);
   printf("\nIn main dupa apelul functiei: a=%d b=%d\n", a, b)
   return 0;
}
```

Apel prin valoare

Valorile lui **a** și **b** **sunt** interschimbate după apelul funcției – funcției îi sunt trimise adresele de memorie ale lui **a** și **b** (pointerii corespunzători)

```
#include <stdio.h>
#include <stdlib.h>
/* interschimbarea valorilor a si b */
void interschimba(int *a, int *b)
{
   int aux;
   printf("\nLa intrarea in functie: a=%d b=%d\n", *a, *b);
   aux = *a; *a = *b; *b = aux;
   printf("\La iesirea din functie: a=%d b=%d\n", *a, *b);
}
int main()
{
   int a=3, b=2;
   printf("\nIn main inainte de apelul functiei: a=%d b=%d\n", a, b)
   interschimba(&a, &b);
   printf("\nIn main dupa apelul functiei: a=%d b=%d\n", a, b)
   return 0;
}
```

Apel prin referință – numai în C++

Valorile lui **a** și **b** **sunt** interschimbate după apelul funcției

```
#include <stdio.h>
#include <stdlib.h>
/* interschimbarea valorilor a si b */
void interschimba(int &a, int &b)
{
   int aux;
   printf("\nLa intrarea in functie: a=%d b=%d\n", a, b);
   aux = a; a = b; b = aux;
   printf("\nLa iesirea din functie: a=%d b=%d\n", a, b);
}
int main()
{
   int a=3, b=2;
   printf("\nIn main inainte de apelul functiei: a=%d b=%d\n", a, b);
   interschimba(a, b);
   printf("\nIn main dupa apelul functiei: a=%d b=%d\n", a, b);
   return 0;
}
```

Calitatea unei funcții

- Este apelată de foarte multe ori
- Face codul care o apelează mult mai compact și mai ușor de citit
- Rezolvă o anumită problemă bine specificată și o rezolvă foarte bine
- Interfața cu restul programului este clară și scurtă

Apelul funcției și procesul de revenire din apel

- Etapele principale ale apelului unei funcției și a revenirii din acesta în funcția de unde a fost apelată
  - Argumentele apelului sunt evaluate și trimise funcției
  - Adresa de revenire este salvată pe stivă
  - Controlul trece la funcția care este apelată
  - Funcția apelată alocă pe stivă spațiu pentru variabilele locale și pentru cele temporare
  - Se execută instrucțiunile din corpul funcției
  - Dacă există valoare returnată, aceasta este pusă într-un loc sigur
  - Spațiul alocat pe stivă este eliberat
  - Utilizând adresa de revenire controlul este transferat în funcția care a inițiat apelul, după acesta

Stiva în C

- La execuția programelor C se utilizează o structură internă numită **stivă** (*stack*) și care este utilizată pentru alocarea memoriei și manipularea variabilelor temporare

- Pe stivă sunt alocate și memorate:
  - Variabilele locale din cadrul funcțiilor
  - Parametrii funcțiilor
  - Adresa de revenire din apelul funcțiilor

- Dimensiunea implicită a stivei este redusă
  - În timpul execuției programele trebuie să nu depășească dimensiunea stivei
  - Dimensiunea stivei poate fi modificată în prealabil din setările editorului de legături (*linker*)

Stiva în C – depășirea dimensiunii

Ambele programe eșuează în timpul execuției din cauza depășirii dimensiunii stivei

```c
int f()
{
    int a[10000000]={0};
    return 2020;
}

int main()
{
    f();
    return 0;
}
```

```c
int f(int a,int b)
{
    if (a<b)
        return 1+f(a+1,b-1);
    else
        return 0;
}

int main()
{
    printf("%d",f(0,1000000));
    return 0;
}
```