UNIVERSITATEA TEHNICĂ A MOLDOVEI

PROGRAMAREA CALCULATOARELOR
FUNCȚII ÎN LIMBAJUL C
Prelegere

Kulev Mihail, dr., conf. univ.

Stimați studenți și stimată audiență!

Mă numesc Kulev Mihail, sunt doctor, conferențiar universitar la Universitatea Tehnică a Moldovei. Din cadrul cursului PROGRAMAREA CALCULATOARELOR Vă propun spre atenția Dumneavoastră prelegerea cu tema:

”FUNCȚII ÎN LIMBAJUL C”

UNIVERSITATEA TEHNICĂ A MOLDOVEI

PROGRAMAREA CALCULATOARELOR
FUNCȚII ÎN LIMBAJUL C
Prelegere

Kulev Mihail, dr., conf. univ.

UNIVERSITATEA TEHNICĂ A MOLDOVEI

FUNCȚII ÎN LIMBAJUL C

Conținutul prelegerii

1. Noțiune de subprogram și exemple de funcții în limbajul C.
2. Codul, apelul și prototipul unei funcții.
3. Interschimbarea datelor între două funcții.

   Exemple de cod.

UNIVERSITATEA TEHNICĂ A MOLDOVEI

FUNCȚII ÎN LIMBAJUL C

Conținutul prelegerii

1. Noțiune de subprogram și exemple de funcții în limbajul C.

Vom defini noțiunea de subprogram și vom afla ce reprezintă funcțiile în limbajul C.

2. Codul, apelul și prototipul unei funcții.

Cunoașterea acestor trei atribute care caracterizează o funcție este necesară pentru utilizarea și elaborarea funcțiilor în C. Vom studia ce reprezintă aceste trei elemente, cum se utilizează și prin ce se diferă.

3. Interschimbarea datelor între două funcții.

Această întrebare este foarte importantă pentru utilizarea și implementarea corectă a funcțiilor în limbajul C. Vom studia procese de interacțiune a două funcții – trimiterea și obținerea datelor.

4. Exemple de cod.

Pe parcursul studierii întrebărilor menționate vom prezenta exemple de cod în limbajul C cu utilizarea funcțiilor.

1. Noțiune de subprogram

În Programarea Calculatoarelor alături de noțiunea de *program* se utilizează și noțiunea de *subprogram*.

Subprogramul (sau procedura) reprezintă o parte independentă și reutilizabilă a programului, destinată rezolvării unei anumite ***subprobleme*** concrete a ***problemei*** mai mari supuse spre rezolvare. Programul poate avea unu sau mai multe subprograme în dependență de numărul de subprobleme în problema mai mare supusă spre rezolvare. Rezolvarea problemei mai mari prin divizarea ei în mai multe subprobleme și elaborarea ulterioară a subprogramelor respective constitue ideea principală a *Paradigmei programării procedurale.*

De menționat că *Paradigma programării procedurale* este paradigma de bază în Programarea Calculatoarelor și limbajul de programare C este un limbaj procedural de programare.

Exemple de subprograme: Subprogramul de introducere a valorilor unui tablou de la tastatură și/sau subprogramul de afișare a valorilor unui tablou pe ecran pot fi părțile componente și independente a diferitor programe de prelucrare a tablourilor și pot fi reutilizate în aceste programe.

Exemple de funcții în C

În limbajul C atât programul principal **main()** cât și toate subprogramele sunt numite funcții. Orice program în C poate conține și poate utiliza una sau mai multe funcții, dar în mod obligatoriu trebuie să conțină funcția **main()**.

Din punct de vedere al utilizării funcțiile pot fi divizate în:

- **funcții standard** predefinite în biblioteca limbajului;
  Ex.: **scanf(), printf()** (se includ în program utilizând directiva preprocesorului **#include <stdio.h>**), etc**.**
- **funcții utilizatorului** definite de dezvoltatorul programului; Ex.: funcția principală **main(),** funcțiile **sum(), max() swap(), readArray(), showArray(),** care vom studia ulterior în cadrul lecției, etc. *Din punct de vedere al interacțiunii între ele funcțiile pot fi divizate în:*
- **funcții apelante** care lansează în execuție alte funcții. Orice funcție poate fi apelantă - poate lansa în execuție o altă funcție;
  Ex.: funcția principală **main(), readArray(), showArray(),** etc.
- **funcții apelate** care sunt lansate în execuție de alte funcții.
  Ex.: **scanf(), printf(), sum(), max(), swap(), readArray, showArray()** pot fi lansate în execuție de funcția **main()** sau de alte funcții. De menționat, că funcția **main()** este o singură funcție, care se lansează în execuție de sistemul operațional.

2. Codul, apelul și prototipul unei funcții

Pentru utilizarea şi elaborarea funcţiilor în C, trebuie să cunoaştem trei elemente importante și distincte:

- **Codul (sau definiția) funcţiei**
- **Apelul funcţiei**
- **Prototipul (sau declarația) funcției**

Codul (sau definiția) unei funcții

Forma generală a codului unei funcții este următoarea:

```
tip_returnat  nume_functie (lista_parametri)      // antetul funcției
   {                                              // corpul funcției
     // lista_de_declaratii;
    // lista_de_instructiuni;     implementează algoritmul funcției
     return expresie;  //  sau return;  (dacă tip_returnat este void )
   }
```

Antetul funcției conține trei elemente:

- **tip_returnat** - reprezintă tipul valorii returnabile de funcţie sau tipul funcției - coincide cu tipul expresiei a instrucțiunii return;
- **nume_functie** - denumirea funcției;
- **lista_parametri** - reprezintă lista tipurilor de parametri ai funcției (variabilelor locale folosite în cadrul funcţiei) cu denumiri generice, formale (parametrii formali). De menționat, că denumirile parametrilor nu trebuie neapărat să coincidă cu denumirile variabilelor respective folosite în funcția apelantă. Lista parametrilor funcției poate fi vidă (void).

Exemplu: Codul (sau definiția) funcției `suma()`:

```
int suma(int num)   // antetul funcției
{          // corpul funcției
   int s = 0;
   int i;
   for(i=1; i<=num; ++i)
   {s += i;}
   return s;
}
```

Exemplu: Codul (sau definiția) funcției `max()`:

```
int max(int a, int b) // antetul funcției
{                     // corpul funcției
   if (a>b)
   return a;
   // else
   return b;
}
```

Exemplu: Codul (sau definiția) funcției `readArray()`:

```
void readArray(int *arr, int n) //antetul funcției
{                               // corpul funcției
   int i;
   printf ("Dati elementele tabloului: ");
   for (i = 0; i < n ; i ++ )
    { scanf("%d", &arr[i]); }
   // return;
}
```

Exemplu: Codul (sau definiția) funcției **showArray**():

```
void showArray(int *arr, int n) //antetul funcției
{                               // corpul funcției
   int i;
   printf ("Elementele tabloului: \n");
   for (i = 0; i < n ; i ++ )
     { printf ("%d\t", arr[i]); }
   printf("\n");
   // return;
}
```

Exemplu: Codul (sau definiția) funcției `swap()`:

```
void swap(int* x, int* y) //antetul f.
{              // corpul funcției
 int temp;
 temp = *x; *x = *y; *y = temp;
 // return;
}
```

Exemplu: Codul (sau definiția) funcției `showMessage()`:

```
void showMessage(void) //antetul funcției.
{           // corpul funcției
 printf("Orice mesaj de afisat \n");
 // return;
}
```

Apelul unei funcţii

Forma generală a apelului funcției este următoarea :

nume_functie (arg1, arg2, …)

Argumentele apelului care se mai numesc parametrii actuali (reali sau efectivi) pot fi expresii, variabile sau valori ce substitue parametrii formali respectivi ai codului funcției la momentul apelului. Cu alte cuvinte, parametrii formali ai funcţiei fiind variabile locale ale funcției sunt iniţializaţi cu valorile argumentelor la momentul apelului. Numărul, tipul și ordinea argumentelor (parametrilor actuali) și parametrilor formali trebuie să coincidă.

De menționat, că există două feluri de apel la funcție: apel-valoare și apel-instrucțiune.

Exemplu de program cu apel la funcția utilizatorului (apel-valoare)

```
#include <stdio.h>
int suma(int num)    // codul funcției suma()
{
  int s = 0;
  int i;
  for(i=1; i<=num; ++i)
    s += i;
  return s;
}
int main()           // codul funcției main()
{
    int v, n= 100;
    v=suma(n)/2;     // apelul funcției suma()  apel-valoare
    printf(„v = %d\n",v);
    return 0;
}
```

Exemplu de program cu apel la funcția utilizatorului (apel-instrucțiune)

```
#include <stdio.h>
void showArray(int* arr, int n)      // codul funcției showArray()
{ int i;
  printf ("Elementele tabloului: \n");
  for (i = 0; i < n ; i ++ )
   { printf ("%d\t", arr[i]); }
  printf("\n");
  return;
}
int main()                           // codul funcției main()
{    int n=10;
     int tab[]={3,-7,0,35,-102,67,89,-11,49,-25};
     showArray(tab, n);      // apelul funcției showArray()  apel-instrucțiune
     return 0;
}
```

Prototipul (sau declarația) unei funcţii

Prototipul unei funcţii are următoarea formă generală (analogică cu prima linie a codului – antetul funcției):

tip_returnat nume_funcţie (lista_parametri) ;

(De menționat, că prototipul funcției se termină cu punct și virgulă și nu conține corpul funcției)

- **tip_returnat** – reprezintă tipul valorii returnate de funcţie;
- **nume_functie** – denumirea funcției;
- **lista_parametri** – reprezintă lista tipurilor de parametri ai funcției (variabile locale folosite în cadrul funcţiei), dar care *pot avea opțional* denumiri generice, formale (parametri formali). De menționat, că anologic cu antetul funcției denumirile parametrilor nu trebuie neapărat să coincidă cu denumirile variabilelor respective folosite în funcția apelantă. Astfel, atunci când se apelează o funcţie folosind ca variabile parametrii actuali, numele variabilelor utilizate nu au nici o legătură cu numele parametrilor formali.

Prototipul (sau declarația) unei funcții

Pentru funcţionarea corectă a programului funcţia utilizatorului trebuie definită (prin codul funcției) sau declarată (prin prototipul funcției) anterior apelului funcției.

Declararea funcției prin prototip este necesară dacă funcţia este definită (prin codul funcției) în altă parte decât în fişierul în care este apelată, sau dacă este definită în acelaşi fişier, dar după apelare la funcție.

Exemplu de program (utilizarea prototipului funcției)

```
#include <stdio.h>
int suma(int num); // sau  int suma(int );  2 forme prototipului functiei suma()
int main()         // codul funcîiei main()
{
    int v, n= 100;
    v=suma(n)/2;    // apelul functiei suma()
    printf("v = %d\n",v);
    return 0;
}
int suma(int num)  // codul functiei suma()
{
  int s = 0;
  int i;
  for(i=1; i<=num; ++i)
    s += i;
  return s;
}
```

3. Interschimbarea datelor între două funcții

Intershimabarea datelor între două funcții constă din 2 procese diferite:

1. Trimiterea datelor de la funcția apelantă la funcția apelată în momentul apelului
2. Obținerea datelor (rezultatelor) de la funcția apelată la funcția apelantă după executarea funcției apelate

Interschimbarea datelor între două funcții

Pentru trimiterea datelor de la funcția apelantă la funcția apelată există 2 modalități:

1. Utilizarea variabilelor globale (nu se recomandă)
2. Utilizarea parametrilor funcției de tip regulat sau/și de tip pointer

(! Pentru tablouri trimitem în funcția apelată adresa tabloului în memorie, ci nu întreg tablou)

Pentru obținerea datelor și rezultatelor executării de la funcția apelată la funcția apelantă există 3 modalități:

1. Utilizarea variabilelor globale (nu se recomandă)
2. Utilizarea valorii returnabile ale funcției (! o singură valoare de tip regulat sau de tip pointer)
3. Utilizarea parametrilor funcției de tip pointeri (! mai multe valori și rezultate corecte)

Exemple de cod

Exemplu: Utilizarea parametrilor de tip regulat și valorii returnabile

```
#include <stdio.h>
int max(int a, int b); // sau  int max(int, int ); prototipul functiei max()
int main()          // codul functiei main()
{
    int x= 2, y= 3;
    printf("max = %d\n", max(x,y)); // apelul functiei max()
    return 0;
}
int max(int a, int b) // codul functiei max()
{
  if(a>b)
  return a;
  // else
  return b;
}
```

Exemplu: Utilizarea parametrilor de tip regulat (varianta greșită)

```
#include <stdio.h>
void swap(int x, int y)
{
  int temp;
  temp = x; x = y; y = temp;
  return;
}

int main()
{
    int a = 2, b = 3;
    swap(a, b);   // apel la funcția swap()
    printf("a = %d, b = %d\n",a, b); // a = 2, b = 3
    return 0;
}
```

Exemplu: Utilizarea parametrii de tip pointeri (varianta corectă)

```
#include <stdio.h>
void swap(int *x, int *y)
{
  int temp = *x; *x = *y; *y = temp;
  printf("*x = %d, *y = %d\n", *x, *y); // *x = 3, *y = 2  afișare în funcția swap()
  return;
}
int main()
{
    int a = 2, b = 3;
    swap(&a, &b);        // apel la funcția swap()
    printf("a = %d, b = %d\n",a, b);  // a = 3, b = 2      afișare în funcția main()
    return 0;
}
```

Probleme propuse spre rezolvare cu utilizarea funcțiilor în limbajul C

1. Să se scrie un program pentru căutarea valorii date într-un tablou de valori aleatoare de tip întreg cu definirea funcțiilor utilizatorului necesare.

2. Pentru tabloul unidimensional care conține elementele cu valori aleatoare de tip real să se determine suma elementelor negative mai mari ca elementul minimal și produsul elementelor pozitive mai mici ca elementul maximal în tablou folosind functiile utilizatorului.

3. Să se scrie un program care determină valorile și pozițiile elementului maximal și minimal întru-un tablou cu valori aleatoare de tip real cu definirea funcțiilor utilizatorului necesare.

4. Sa se scrie un program care determina pozitia si valoarea elementului maximal, precum si numarul elementelor pozitive ce se află la dreapta de elementul maximal într-un tablou unidimensional de valori aleatoare de tip real folosind funcțiile utilizatorului.

5. Să se realizeze un program care să rezolve o ecuaţie de gradul 2 cu coeficienţii citiţi de la tastatură, folosind o funcţie care să returneze două valori - rădăcinile, şi să precizeze dacă ecuația are sau nu rădăcini reale.

Tutoriale online pentru tema prelegerii:

1. https://ocw.cs.pub.ro/courses/programare/laboratoare/lab04
2. http://www.aut.upt.ro/~rraul/PC/2009-2010/PC_lab8.pdf
3. http://andrei.clubcisco.ro/cursuri/1pc/curs/1/Curs%204_5%20Doc.pdf
4. https://igotopia.ro/ce-sunt-functiile-in-c-ce-parametri-pot-primi-si-care-e-diferenta-dintre-variabilele-locale-si-cele-globale/
5. http://apollo.eed.usv.ro/~roscaneanu.alexandru/teaching/pclp1/Lab_functii_C.pdf
6. https://cester.utcluj.ro/lectures/ComputerProgrammingCLanguage/Laborator%2010%20C.pdf

VĂ MULȚUMESC PENTRU ATENȚIE!

MULTĂ SĂNĂTATE ȘI SUCCESE!