Programarea calculatoarelor
Limbajul C

CURS 5

Funcții

Modularizarea programelor

- Un program mare poate fi mai uşor de scris, de înțeles şi de modificat dacă este modular (format din module funcționale relativ mici = funcții)
- Funcția main: funcția principală a unui program C
- Funcție (în C) = piesa de bază a conceptului de modularitate al programării

Funcție C:

- poate executa o sarcină precisă.
- poate fi reutilizată în mai multe aplicații
  - reduce efortul de programare al unei noi aplicații.
- poate fi scrisă şi verificată separat de restul aplicației
  - reduce timpul de punere la punct a unei aplicații mari (deoarece erorile pot apare numai la comunicarea între funcții corecte).
- modificările se fac numai în anumite funcții şi nu afectează alte funcții (care nici nu mai trebuie recompilate)
  - întreținerea unei aplicații este simplificată

Exemplu

- Care ar fi funcțiile pentru operații cu mulțimi de numere întregi necesare pentru o implementare cât mai completă a acestora?
  - inițializare mulțime vidă
  - verificare apartenență la o mulțime
  - adăugare element la o mulțime (dacă nu există deja)
  - afişare mulțime (între acolade)
  - reuniune
  - intersecție
  - diferență de mulțimi, etc

Declarare şi implementare funcții

- *tip_rezultat_returnat nume(lista_parametrii_formali)*

  {

  declarare variabile locale

  instrucțiuni funcție

  }

- Lista parametrilor formali cuprinde declarația parametrilor formali, separați prin virgulă:

  tip_p1 nume_p1, tip_p2 nume_p2, ..., tip_pn nume_pn

- funcțiile nu pot fi definite încuibat (ca in Pascal)!

Tipul unei funcții

- dacă *tip_rezultat_returnat* este:
  - int, float, etc - funcția este întreagă, reală, etc adică funcție cu tip
  - void - funcția este void sau funcție fără tip

- dacă funcția nu returnează nici un rezultat şi nu primeşte parametri, definiția va fi:

```
void nume_funcție (void){
    definirea variabilelor locale
    prelucrari  /*instrucțiuni*/
}
```

Numărarea şi afişarea numerelor prime mai mici ca un întreg dat n.

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

int prim (int numar){
    int div;
    for (div=2; div<=sqrt(numar); div++)
        if (numar % div ==0) return 0;
    return 1;
}
```

Numararea si afisarea numerelor prime mai mici ca un întreg dat n.

```
int main () {
      int n,m,contor ;
          /* contor de numere prime*/
      printf ("n= "); scanf ("%d",&n);
      contor=0;
      for (m=2;m<=n;m++)   /* incearca numerele m*/
          if ( prim(m) == 1 ){       /* dacă m este prim*/
            printf ("%d\n",m);
            contor++;
          }
      printf ("există  %d numere prime mai mici decat %d\n", contor,n);
      system(“pause”);
      return 0;
 }
```

Apel funcție

- pentru funcție fără tip (void):

  nume_funcție (lista_parametrii_efectivi);

- pentru funcție cu tip T, unde t este de tipul T:

  t = nume_funcție (lista_parametrii_efectivi);

  - *Sau poate apare într-o expresie în care se foloseşte rezultatul ei:*

  if(prim(m) == 1) …

- *parametri actuali=parametri efectivi*

Apel funcție

- La apelul unei funcții, se execută corpul său, după care se revine în funcția apelantă, la instrucțiunea următoare apelului.
- Parametrii efectivi sunt expresii, care trebuie să corespundă ca numar şi tipuri (eventual prin conversie implicită) cu parametrii formali.
- Parametrii efectivi sunt transmişi prin valoare, la apelul funcției (valorile lor sunt depuse pe stiva). Modificarea valorii lor de către funcție nu este vizibilă în exterior.

Prototipul unei funcții

- O funcție poate fi apelată, dacă în fața apelului există definiția sau cel putin declarația funcției (prototipul, antetul acesteia).
- Început fişier sursă sau fişier header:

  tip nume(lista_tipuri_parametrii_formali);
- În prototip, numele parametrilor formali pot fi omişi, apărând doar tipul fiecăruia.
- *Prototipul implicit* este:

  int nume_funcție(void);

Utilizare prototip funcții

```
#include <stdio.h>
#include <math.h>

int prim (int numar);
//sau: int prim (int);

int main(){
…
}

int prim (int numar){
      int div;
      for (div=2; div<=sqrt(numar); div++)
            if (numar % div ==0) return 0;
      return 1;
}
```

Apel funcție

- La apelul unei funcții, pe stivă se crează o *înregistrare de activare*, care cuprinde, de jos în sus:
  - adresa de revenire din funcție
  - valorile parametrilor formali
  - variabilele locale.

Return

- Revenirea dintr-o funcție se face la
  - întâlnirea instrucțiunii return,
  - terminarea execuției corpului funcției (funcția poate să nu conțină instrucțiunea return doar în cazul funcțiilor void - care nu returnează nici un rezultat).
- Forme ale instrucțiunii return:
  - *return;* //dacă funcția e void
  - *return expresie;* /*expresia e de acelaşi tip cu tip_rezultat (eventual prin conversie implicită) */

Parametri formali şi parametri efectivi

Exemplu:

```
#include<stdio.h>
#include<stdlib.h>

int factorial(int n)            //n este parametru formal de tip întreg
{
   int i, fact=1;
   for (i=2; i<=n; i++)
            fact=fact*i;
   return fact;   //funcția factorial returnează valoarea lui n!
}
```

Parametri formali şi parametri efectivi

```
int main(void)
{
   int v;
   printf("3!=%d\n",factorial(3));
            /*în funcția printf apelam funcția factorial având ca parametru efectiv
               valoarea 3*/
   printf("Introd o valoare:");
   scanf("%d",&v);
   printf("%d!=%d\n",v,factorial(v));
            /*funcția factorial este apelata având ca parametru efectiv valoarea citita
               în variabila v*/
   system(“pause”);
   return 1;
}
```

Domeniu de vizibilitate (scope)

- Un nume (variabilă, funcție) poate fi utilizat numai după ce a fost declarat.
- Domeniul de vizibilitate (scope) al unui nume este mulțimea instrucțiunilor (liniilor de cod) în care poate fi utilizat acel nume (numele este vizibil).
- *Regula de bază: identificatorii sunt accesibili doar în blocul în care au fost declarați; ei sunt necunoscuti în afara acestor blocuri.*

Domeniu de vizibilitate (scope)

- variabile globale –variabile ce sunt declarate în afara oricărui bloc

- variabile locale – variabilele ce sunt declarate: în funcții, în blocuri, ca parametri.

Variabile locale şi variabile globale

Exemplu:

```
#include<stdio.h>
#include<stdlib.h>

int fact=1;
void factorial(int n)
{ int i;
  fact=1;
  for(i=2;i<=n;i++) fact=fact*i;
}
```

Variabile locale şi variabile globale

```
int main(void) {
        int v;
        factorial(3);
        printf("3!=%d\n",fact);
        printf("Introd o valoare:");
        scanf("%d",&v);
        factorial(v);
        printf("%d!=%d\n",v,fact);
        system(“pause”);
        return 1;
      }
```

Observații

- Definiția unui identificator maschează pe cea a aceluiaşi identificator declarat într-un suprabloc sau în fişier (global)!
- Apariția unui identificator face referință la declararea sa în cel mai mic bloc care conține această apariție!

```
#include<stdio.h>
#include<stdlib.h>
int fact=1;
void factorial(int n)
{ int i;
  int fact=1;
  for(i=2;i<=n;i++) fact=fact*i;
}
```

Observații

```
int main(void)
{
  int v;
  factorial(3);
  printf("3!=%d\n",fact);
  printf("Introd o valoare:");
  scanf("%d",&v);
  factorial(v);
  printf("%d!=%d\n",v,fact);
  system(“pause”);
  return 1;
}
```

Atenție! Va afişa 3!=1. Rezultatul este 1 oricare ar fi valoarea lui v !!!

Probleme rezolvate

1. Să se calculeze şi să se afişeze valoarea expresiei $x^m+y^n+(xy)^{m\^n}$, x,y,m,n fiind citiți de la tastatură, astfel încât întregii m,n să fie pozitivi. Ridicarea la putere se va realiza printr-o funcție *putere* care primeşte baza şi exponentul ca parametri şi returnează rezultatul.

- Observație: apelul funcției putere (analog pow) apare într-o expresie şi de asemenea ca parametru actual într-un apel.

Rezolvare

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

double putere (double baza, int exp); // calculează baza^exp

int main(void){
    int m,n;
    double x,y;
    while( printf( “m(>=0):"), scanf("%d",&m), m<0);
    while( printf( “n(>=0):"), scanf("%d",&n), n<0);
    if ( m==0 && n==0 ) { //semnalare eroare 0^0
        puts("0^0 imposibil");
        return; // din main
    }
```

Rezolvare

```
    printf("valorile lui x,y:");
    scanf("%lf%lf",&x,&y);
    printf("%lf^%d+%lf^%d+(%lf*%lf)^%d^%d (cu putere ):%lf\n",
           x,m,y,n,x,y,m,n,  putere(x,m)+putere(y,n) +
           putere(x*y,putere(m,n)));
    printf("rezultat cu pow: %lf\n", pow(x,m) + pow(y,n) +
           pow(x*y,pow(m,n)));
    system("pause");
    return 1;
} // main

double putere (double baza, int exp){
      int i; float rez;
      for(i=rez=1;i<=exp;i++)rez*=baza;
      return rez;
} // putere
```

Observație

- while(printf("m(>=0): "),scanf("%d",&m),m<0);

- Echivalent cu:

```
printf("m(>=0): ");
scanf("%d",&m);
while(m<0){
        printf("m(>=0): ");
        scanf("%d",&m);
}
```

Exerciții propuse

1. Se citesc numere naturale până la întâlnirea numărului 0. Să se afişeze toate perechile de numere citite consecutiv, cu proprietatea că al doilea număr reprezintă inversul primului număr. Se va rezolva în două variante:

a) fără funcții auxiliare

b) se va utiliza o funcție de calcul a inversului unui număr.

Exemplu: 12 3 43 34 43 56 876 678 9 9 0

Va afisa:

43 34

34 43

876 678

9 9

Rezolvare 5.1a

```
#include<stdio.h>
#include<stdlib.h>

int main(){
    int x,y,z,aux;
    scanf("%d",&x);
    while (x)
    {
        scanf("%d",&y);
        if(!y) break;
        z=0;
        aux=x;
```

Rezolvare 5.1a

```
        while(aux){
            z=z*10+aux%10;
            aux/=10;
        }
        if(z==y) printf("%d %d\n",x,y);
        x=y;
    }

    system("pause");
    return 0;
}
```

Rezolvare 5.1 b

```
#include<stdio.h>
#include<stdlib.h>

int inv (int x){
    int y=0;
    while(x){
            y=y*10+x%10;
            x/=10;
            }
    return y;
}
```

Rezolvare 5.1 b

```
int main(){
    int x,y;
    scanf("%d",&x);
    while (x)
    {
        scanf("%d",&y);
        if(!y) break;
        if(x==inv(y)) printf("%d %d\n",x,y);
        x=y;
    }

    system("pause");
    return 0;
}
```

Exerciții propuse

2. Se citeşte de la tastatură un număr natural n (1<=n<=10). Să se determine toate numerele prime x care au n cifre utilizând o funcție care spune dacă un număr x este prim sau nu. NU SE VA UTILIZA FUNCTIA pow DIN C!

Exemplu: n=2, x={11,13,17,19,23,29,31,37,41,43,47, 53, 59, 61,67,71, 73, 79, 83,89,97}.

3. Să se determine primele n numere perfecte. X este numar perfect daca X este egal cu 1+suma tuturor divizorilor săi, mai puțin el însuşi. Se va utiliza o funcție care spune dacă un număr x este perfect sau nu.

Exemplu: n=10000, X={6, 28, 496, 8128}

Exerciții propuse

4. Se citeşte de la tastatură un număr natural n. Să se determine toate numerele prime mai mici decât n cu proprietatea că x = invers(x) (palindrom).

Ex: n=1000, x={2,3,5,7,11,101, 131,etc}

5. Se citeşte de la tastatură un număr natural n (1<=n<=10). Să se determine pentru toate numerele x de n cifre suma şi produsul divizorilor săi primi (luați o singură dată, fără 1).

NU SE VA UTILIZA FUNCTIA pow DIN C!

Exemplu: n=2

x=10, s=2+5=7, p=2*5=10;

x=11, s=11, p=11;

x=12, s=5, p=6...etc

Rezolvare 5.4 a

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
int main(){
    int n,i,x,y,z;
    scanf("%d",&n);
    for(x=1;x<n;x++){
        for(i=2;i<=sqrt(x);i++)
                if(x%i==0) break;
```

Rezolvare 5.4 a

```
    if(i>sqrt(x)) { //este prim
            y=0;
            z=x;
            while(z){
                    y=y*10+z%10;
                    z/=10;
            }
            if(y==x)  printf("%d ",x);
        }
    }
    system("pause");
    return 0;
}
```

Rezolvare 5.4 b

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
int inv (int x){
    int y=0;
    while(x){
            y=y*10+x%10;
            x/=10;
            }
    return y;
}
```

Rezolvare 5.4 b

```
int main(){
    int n,i,x;
    scanf("%d",&n);
    for(x=1;x<n;x++){
        for(i=2;i<=sqrt(x);i++)
                if(x%i==0) break;

        if(i>sqrt(x))
            if(inv(x)==x)  printf("%d ",x);

    }
    system("pause");
    return 0;
}
```