6. Funcţii.(1)

6.1. Apelarea funcţiilor.

In C noţiunea de funcţie este esenţială, deoarece asigură *un mecanism de abstractizare a controlului*: rezolvarea unei părţi a problemei poate fi încredinţată unei funcţii, moment în care suntem preocupaţi de *ce face funcţia*, fără a intra în detalii privind *cum face funcţia* anumite operaţii. Însăşi programul principal este o funcţie cu numele **`main()`**, iar programul C este reprezentat de o mulţime de definiri de variabile şi de funcţii.

Funcţiile pot fi clasificate în:

- funcţii care întorc un rezultat
- funcţii care nu întorc nici un rezultat (similare procedurilor din Pascal).

Apelul (referirea sau utilizarea) unei funcţii se face prin:

```
nume_funcţie (listă_parametri_efectivi)
```

Acesta poate apare ca o instrucţiune:

```
nume_funcţie (listă_parametri_efectivi);
```

De exemplu:

```
printf("x=%5.2lf\n",x);
mesaj();
```

Pentru funcţiile care întorc un rezultat apelul de funcţie poate apare ca operand într-o expresie. De exemplu:

```
y=sin(x);
nr_zile=bisect(an)+365;
```

Se remarcă faptul că lista de argumente (sau de parametri efectivi) poate fi vidă.
Funcţiile comunică între ele prin *lista de argumente* şi prin *valorile întoarse de funcţii*. Comunicarea poate fi realizată şi prin *variabilele externe*, definite în afara tuturor funcţiilor.

Exemplul 11: *O fracţie este cunoscută prin numărătorul* **`x`** *şi numitorul* **`y`***, valori întregi fără semn. Să se simplifice această fracţie.*

Simplificarea se va face prin cel mai mare divizor comun al numerelor **`x`** şi **`y`**. Vom utiliza o funcţie având ca parametri cele două numere, care întoarce ca rezultat , cel mai mare divizor comun a lor. Funcţia **`main()`** apelează funcţia **`cmmdc()`** transmiţându-i ca argumente pe **`x`** şi **`y`**. Funcţia **`cmmdc()`** întoarce ca rezultat funcţiei **`main()`**, valoarea celui mai mare divizor comun. Programul, în care vom ignora deocamdată definirea funcţiei **`cmmdc()`**, este:

```
#include <stdio.h>
int main()
{ unsigned long x, y, z;
  scanf("%lu%lu", &x, &y);
  printf("%lu / %lu =", x, y);
  z=cmmdc(x,y);
  x/=z;
  y/=z;
  printf("%lu / %lu\n", x, y);
```

```
  return 0;
}
```

6.2. Definiţii de funcţii.

În utilizarea curentă, o funcţie trebuie să fie *definită* înainte de a fi *apelată*. Aceasta impune o definire a funcţiilor programului în ordinea sortării topologice a acestora: astfel mai întâi se vor defini funcţiile care nu apelează alte funcţii, apoi funcţiile care apelează funcţii deja definite. Este posibil să eliminăm această restricţie, lucru pe care îl vom trata ulterior.
O funcţie se defineşte prin *antetul* şi *corpul* funcţiei.
Funcţiile nu pot fi incluse unele în altele; toate funcţiile se declară pe acelaşi nivel cu funcţia `main()`.
In versiunile mai vechi ale limbajului, în antetul funcţiei parametrii sunt numai enumeraţi, urmând a fi declaraţi ulterior. Lista de parametri, în acest caz, este o enumerare de identificatori separaţi prin virgule.

```
tip_rezultat nume_functie( lista_de_parametri_formali )
{ declarare_parametri_formali;
  alte_declaratii;
  instructiuni;
}
```

In mod uzual parametrii funcţiei se declară în antetul acesteia. Declararea parametrilor se face printr-o listă de declaraţii de parametri, cu elementele separate prin virgule.

```
tip_rezultat nume_functie( tip nume, tip nume,... )
{ declaratii;
  instructiuni;
}
```

O funcţie este vizibilă din locul în care a fost declarată spre sfârşitul fişierului sursă (adică definiţia funcţiei precede apelul).
Definirea funcţiei **`cmmdc()`** se face folosind algoritmul lui Euclid.

```
unsigned long cmmdc(unsigned long u, unsigned long v)
{ unsigned long r;
  do {r=u%v;
      u=v;
      v=r;
  } while (r);
  return u;
}
```

In cazul în care apelul funcţiei precede definiţia, trebuie dat, la începutul textului sursă, un *prototip al funcţiei*, care să anunţe că definiţia funcţiei va urma şi să furnizeze tipul rezultatului returnat de funcţie şi tipul parametrilor, pentru a permite compilatorului să facă verificările necesare.
Prototipul unei funcţii are un format asemănător antetului funcţiei şi serveşte pentru a informa compilatorul asupra:

- tipului valorii furnizate de funcţie;
- existenţa şi tipurile parametrilor funcţiei

Spre deosebire de un antet de funcţie, un prototip se termină prin **;**

```
tip nume( lista tipurilor parametrilor formali);
```

Din prototip interesează numai tipurile parametrilor, nu şi numele acestora, motiv pentru care aceste nume pot fi omise.

```
void f(); /*functie fara parametri care nu intoarce nici un
rezultat*/
int g(int x, long y[], double z);
int g(int, long[], double);   /*aici s-au omis numele
parametrilor*/
```

Dintre toate funcţiile prezente într-un program C prima funcţie lansată în execuţie este **main()**, independent de poziţia pe care o ocupă în program.
Apelul unei funcţii **g()**, lansat din altă funcţie **f()** reprezintă un transfer al controlului din funcţia **f()**, din punctul în care a fost lansat apelul, în funcţia **g()**. După terminarea funcţiei **g()** sau la întâlnirea instrucţiunii **return** se revine în funcţia **f()** în punctul care urmează apelului **g().** Pentru continuarea calculelor în **f(),** la revenirea din **g()** este necesară salvarea stării variabilelor (contextului) din **f()** în momentul transferului controlului. La revenire în **f(),** contextul memorat a lui **f()** va fi refăcut.
O funcţie apelată poate, la rândul ei, să apeleze altă funcţie; nu există nici o limitare privind numărul de apeluri înlănţuite.

6.3. Comunicarea între funcţii prin variabile externe. Efecte laterale ale funcţiilor.

Comunicarea între funcţii se poate face prin variabile externe tuturor funcţiilor; acestea îşi pot prelua date şi pot depune rezultate în variabile externe. În exemplul cu simplificarea fracţiei vom folosi variabilele externe **a, b** şi **c** . Funcţia **cmmdc()** calculează divizorul comun maxim dintre **a** şi **b** şi depune rezultatul în **c**. Întrucât nu transmite date prin lista de parametri şi nu întoarce vreun rezultat, funcţia va avea prototipul **void cmmdc()**:

```
#include <stdio.h>
unsigned long a, b, c; // variabile externe
// definirea functiei cmmdc()
int main()
{ scanf("%lu%lu", &a, &b);
  printf("%lu / %lu = ", a, b);
  cmmdc();
  a/=c;
  b/=c;
  printf("%lu / %lu\n", a, b);
  return 0;

}
```

Definiţia funcţiei **cmmdc()** din Exemplul 11, este:

```
void cmmdc()
{ unsigned long r;
  do { r = a%b;
       a = b;
       b = r;
  } while (r);
```

```
  c = a;
}
```

Dacă se execută acest program, se constată un rezultat ciudat, şi anume, orice fracţie, prin simplificare ar fi adusă la forma **1/0** ! Explicaţia constă în faptul că funcţia **cmmdc()** prezintă *efecte laterale*, şi anume modifică valorile variabilelor externe `a, b` şi `c`; la ieşirea din funcţie **a==c** şi **b==0**, ceeace explică rezultatul.

Aşadar, un *efect lateral* (sau secundar),reprezintă modificarea de către funcţie a unor variabile externe.
În multe situaţii aceste efecte sunt nedorite, duc la apariţia unor erori greu de localizat, făcând programele neclare, greu de urmărit, cu rezultate dependente de ordinea în care se aplică funcţiile care prezintă efecte secundare. Astfel într-o expresie în care termenii sunt apeluri de funcţii, comutarea a doi termeni ar putea conduce la rezultate diferite!

Vom corecta rezultatul, limitând efectele laterale prin interzicerea modificării variabilelor externe **a** şi **b**, ceea ce presupune modificarea unor copii ale lor în funcţia **cmmdc()** sau din programul de apelare

```
void cmmdc()
{ unsigned long r, ca, cb;
  ca = a;
  cb = b;
  do{r = ca%cb;
     ca = cb;
     cb = r;
  } while (r);
  c = ca;
}
```

Singurul efect lateral permis în acest caz – modificarea lui **c** asigură transmiterea rezultatului către funcţia apelantă.

Soluţia mai naturală şi mai puţin expusă erorilor se obţine realizând *comunicaţia* între funcţia **cmmdc()** şi funcţia **main()** nu prin variabile externe, *ci prin parametri*, folosind o funcţie care întoarce ca rezultat cmmdc.

Transmiterea parametrilor prin valoare, mecanism specific limbajului C, asigură păstrarea intactă a parametrilor actuali **x** şi **y**, deşi parametrii formali corespunzători: **u** şi **v** se modifică! Parametrii actuali **x** şi **y** sunt copiaţi în variabilele **u** şi **v**, astfel încât se modifică copiile lor nu şi **x** şi **y**.
In fişierul sursă funcţiile pot fi definite în orice ordine.Mai mult, programul se poate întinde în mai multe fişiere sursă. Definirea unei funcţii nu poate fi totuşi partajată în mai multe fişiere.

O funcţie poate fi apelată într-un punct al fişierului sursă, dacă în prealabil a fost definită în acelaşi fişier sursă, sau *a fost anunţată*.

Exemplul 12:

```
#include <stdio.h>
unsigned long fact(unsigned char); // prototipul anunta functia
int main()
{ printf("5!=%ld\n", fact(5));     // apel functie
  printf("10!=%ld\n", fact(10));
```

```
  getch();
}
long fact(unsigned char n)          // antet functie
{ long f=1;                         // corp functie
  short i;
  for (i=2; i<=n; i++)
    f*=i;
  return(f);
}
```

Tipurile funcţiilor pot fi:

- tipuri predefinite
- tipuri pointer
- tipul structură (înregistrare)

6.4. Funcţii care apelează alte funcţii.

Programul principal (funcţia `main()`) apelează alte funcţii, care la rândul lor pot apela alte funcţii. Ordinea definiţiilor funcţiilor poate fi arbitrară, dacă se declară la începutul programului prototipurile funcţiilor. In caz contrar definiţiile se dau într-o ordine în care nu sunt precedate de apeluri ale funcţiilor.

Exemplul 13: *Pentru o valoare întreagă şi pozitivă* **n** *dată, să se genereze triunghiul lui Pascal, adică combinările*:

C_0^0

C_1^0 C_1^1

. . .

C_n^0 C_n^1 . . . C_n^n

Combinările vor fi calculate utilizând formula cu factoriale: $C_n^p = n!/p!/(n-p)!$

```
#include <stdio.h>
unsigned long fact(int);      /*prototip functie factorial*/
unsigned long comb(int,int);  /*prototip functie combinari*/
int main()                /*antet functie main*/
{ int k,j,n;
  scanf("%d",&n);
  for (k=0;k<=n;k++)
    { for (j=0;j<=k;j++)
        printf("%6lu  ", comb(k,j));
      printf("\n");
    }
}
unsigned long comb(int n, int p) /*antet functie comb*/
{ return (fact(n)/fact(p)/fact(n-p));
}
/* functia fact a mai fost definita */
```

6.5. Programe cu mai multe fişiere sursă.

Pentru programele mari este mai comod ca acestea să fie constituite din mai multe fişiere sursă, întrucât programul este conceput de mai mulţi programatori (echipă) şi fiecare funcţie poate constitui un fişier sursă, uşurându-se în acest mod testarea. Reamintim că o funcţie nu poate fi împărţită între mai multe fişiere.
Un exemplu de program constituit din 2 fişiere sursă este:

```
/* primul fisier Fis1.c */
void F();  /* prototipul functiei definite in fisierul 2*/
#include <stdio.h>
void main(){
  F();   /* apelul functiei F */
  ...
}

/* al doilea fisier Fis2.c */
#include <stdio.h>
void F(){
  ...
}
```

6.6. Fişiere antet.

In C se pot utiliza o serie de funcţii aflate în bibliotecile standard. Apelul unei funcţii de bibliotecă impune prezenţa prototipului funcţiei în textul sursă. Pentru a simplifica inserarea în textul sursă a prototipurilor funcţiilor de bibliotecă, s-au construit *fişiere de prototipuri*. Acestea au extensia .h (*header* sau *antet*).

De exemplu fişierul **`stdio.`**h conţine prototipuri pentru funcţiile de bibliotecă utilizate în operaţiile de intrare / ieşire; fişierul **`string.h`** conţine prototipuri pentru funcţiile utilizate în prelucrarea şirurilor de caractere.
Includerea în textul sursă a unui fişier antet se face folosind directiva `#include`

Tabel 6.1. Funcţii declarate în fişierele antet

Fişier antet	Funcţii conţinute
`stdio.h`	**`printf, scanf, gets, puts,...`**
`conio.h`	**`putch, getch, getche, ...`**
`math.h`	**`sqrt, sin, cos, ...`**

6.7. Funcţii matematice uzuale.

Fişierul antet **`<math.h>`** conţine semnăturile (prototipurile) unor funcţii matematice des folosite. Dintre acestea amintim:

Tabel 6.2. Semnăturile funcţiilor din biblioteca matematică

Notaţie	Semnătură (prototip)	Operaţie realizată
sin(x)	**`double sin(double);`**	funcţii trigonometrice directe
cos(x)	**`double cos(double);`**	
tg(x)	**`double tan(double);`**	
arcsin(x)	**`double asin(double);`**	funcţii trigonometrice inverse
arcos(x)	**`double acos(double);`**	

arctg(x)	`double atan(double);`	
arctg(y/x)	`double atan2(double, double);`	
sinh(x)	`double sinh(double);`	funcţii hiperbolice
cosh(x)	`double cosh(double);`	
th(x)	`double tanh(double);`	
exp(x)	`double exp(double);`	exponenţială naturală
10^n	`double pow10(int);`	exponenţială zecimală
a^b	`double pow(double, double);`	exponenţială generală
ln(x)	`double log(double);`	logaritm natural
lg(x)	`double log10(double);`	logaritm zecimal
| x |	`double fabs(double);`	valoare absolută
	`int abs(int);`	
	`long labs(long);`	
$\sqrt{x}$	`double sqrt(double);`	rădăcină pătrată
	`long double sqrtl(long double);`	
$\lceil x \rceil$	`double ceil(double);`	întregul minim >= x
$\lfloor x \rfloor$	`double floor(double);`	întregul maxim <= x
conversii	`double atof(const char *);`	conversie şir de caractere în float
	`long double atold (const char *);`	conversie şir de caractere în long double

6.8. Probleme rezolvate.

1. Numerele naturale pot fi clasificate în: *deficiente, perfecte* sau *abundente*, după cum suma divizorilor este mai mică, egală sau mai mare decât valoarea numărului. Astfel: **n=12** este abundent deoarece are suma divizorilor: **sd = 1+2+3+4+6 = 16 > 12, n=6** este perfect: **sd = 1+2+3 = 6**, iar **n=14** este deficient deoarece **sd = 1+2+7 < 14**.

- Definiţi o funcţie având ca parametru un număr întreg `n`, funcţie care întoarce ca rezultat **-1, 0** sau **1** după cum numărul este deficient, perfect sau abundent.
- Scrieţi o funcţie `main()` care citeşte două valori întregi **x** şi **y** şi clasifică toate numerele naturale cuprinse între **x** şi **y** afişând după fiecare număr tipul acestuia, adică deficient, perfect sau abundent.

```
#include <stdio.h>
#include <stdlib.h>
int tip(int n){
    int sd, d;
    sd = 1;
    for(d=2; d<=n/2; d++)
        if(n%d==0)
            sd += d;
    if(sd < n) return -1;
    if(sd == n) return 0;
    return 1;
}

int main(){
```

```
      int n, x, y;
      scanf("%d%d", &x, &y);
      for(n=x; n<=y; n++){
         printf("%5d ", n);
         switch(tip(n)){
         case -1: printf("deficient\n" ); break;
         case  0: printf("perfect\n"); break;
         case  1: printf("abundent\n");
         }
      }
      system("PAUSE");
      return 0;
}
```

2. Verificaţi „conjectura lui Goldbach”, potrivit căreia orice număr par poate fi scris ca cel puţin o sumă a două numere prime. Programul va genera şi afişa descompunerile tuturor numerelor pare până la o limită dată `L`.

Vom considera şi pe 1 ca număr prim. Exceptând primele două descompuneri: 2=1+1 şi 4=1+3=2+2, celelalte descompuneri vor avea termenii numere prime impare. Pentru un număr par `p`, o eventuală descompunere, având termenii `a` şi `p-a` impune ca acestea să fie prime şi impare.

```
#include <stdio.h>
#include <stdlib.h>
int prim(int n){
    int d;
    for(d=2; d*d <= n; ){
       if(n%d==0) return 0;
       if(d==2)
          d = 3;
       else
          d+=2;
    }
    return 1;
}

int main(){
    int L, p, a, nd;
    scanf("%d", &L);
    /* scrierea primrlor doua descompuneri */
    printf("   2=1+1\n   4=1+3=2+2\n");
    for(p=6; p<=L; p+=2){
        printf("%4d ", p);
        nd = 0;
        for(a=1; a<=p/2; a+=2)
            if(prim(a) && prim(p-a)){
                nd++;
                printf("=%4d+%4d", a, p-a);
            }
```

```
        if(nd==0)
            printf("nu verifica conjectura\n");
        else
            printf("\n");
    }
    system("PAUSE");
    return 0;
}
```

6.9. Probleme propuse.

1. O pereche de numere naturale a şi b se numesc *numere prietene*, dacă suma divizorilor unuia dintre numere este egală cu celălalt număr. De exemplu 220 şi 284 sunt numere prietene deoarece:

sd(220) = 1+2+4+5+10+11+20+22+44+55+110 = 284
sd(284) = 1+2+4+71+142 = 220

a) Scrieţi o funcţie având ca parametri un număr natural, care întoarce suma divizorilor numărului.
b) Scrieţi o funcţie având ca parametri două numere naturale, care întoarce 1 sau 0, după cum cele două numere sunt sau nu prietene.
c) Scrieţi o funcţie **main(),** care în intervalul **x, y** dat găseşte toate perechile de numere prietene şi le afişează.

3. Să se rezolve ecuaţia **f(x)=0**, prin metoda tangentei, pornind cu un $x^{(0)}$ dat, şi calculând $x^{(k+1)}=x^{(k)} - f(x^{(k)})/f'(x^{(k)})$ cu o precizie dată **eps**, care se atinge când $|x^{(k+1)} - x^{(k)}| < eps$. Funcţiile **f(x)** şi **f'(x)** sunt date de programator.

4. Se consideră funcţia $f(x) = \ln (1 + x^2)$. Să se scrie un program care tabelează funcţia pe **n** intervale, fiecare interval fiind precizat prin capetele **a** şi **b** şi pasul de afişare **h**.

5. Să se scrie toate descompunerile unui număr par ca o sumă de două numere prime. Se va utiliza o funcţie care stabileşte daca un număr este sau nu prim.

6.Să se scrie în C:
a) Un subprogram funcţie pentru calculul valorii unei funcţii **f(x)** pentru o valoare dată **x**, definită astfel:

$$f(x) = \begin{cases} 1 - x\sqrt{-x} & \text{pentru } x < -1 \\ \sqrt{1 - x^2} & \text{pentru } -1 \le x \le 1 \\ 1 + x\sqrt{x} & \text{pentru } x > 1 \end{cases}$$

b) O funcţie pentru calculul integralei definite:

$$I = \int_a^b f(x)dx$$

prin metoda trapezelor, cu **n** paşi egali, pe intervalul **[a,b],** după formula

$$\mathbf{I} \cong \frac{\mathbf{h}}{2}\left[\mathbf{f(a)} + \mathbf{f(b)} + \sum_{\mathbf{i}=1}^{\mathbf{n}-1} \mathbf{f(a + ih)}\right]$$

unde **h=(b-a)/n**
c) Un program care calculează integrala unei funcţii definite la punctul a), folosind funcţia b), pe un interval dat **[p,q],** cu precizia **epsilon** (se repetă calculul integralei pentru **n=10, 20,...** paşi, până când diferenţa dintre două integrale succesive devine mai mică decât **epsilon**

7.Pentru un număr dat **N** să se afişeze toţi factorii primi ai acestuia şi ordinul lor de multiplicitate.Se va utiliza o funcţie care extrage dintr-un număr un factor prim.

8. Utilizând o funcţie pentru calculul celui mai mare divizor comun a două numere, să se calculeze c.m.m.d.c. a n elemente întregi ale unei liste date.

9. Pentru fiecare element al unei liste de numere întregi date, să se afişeze numărul prim cel mai apropiat de el ca valoare. Dacă două numere prime sunt la distantă egală de un element din listă se vor afişa ambele numere prime.