Laboratorul 3. OPERAȚII DE INTRARE/IEŞIRE PENTRU CONSOLĂ

Introducere

În C/C++, spre deosebire de alte limbaje, operațiile de intrare/ieşire (I/O – Input/Output) nu se efectuează pe baza unui tip de date dedicat şi a unor funcții încorporate în limbaj. Sistemul de intrare/ieşire nu este parte a limbajului, ci se adaugă printr-un set de funcții din biblioteca standard, ceea ce oferă programatorului un sistem I/O flexibil şi puternic.

Transferurile de date cu dispozitivele fizice de intrare/ieşire (consolă, disc, imprimantă, etc.) se fac prin intermediul sistemului de operare şi al unor dispozitive logice identice numite **stream** (flux).

Operațiile de intrare/ieşire se efectuează prin funcții din biblioteca C declarate (majoritatea) în fişierul **stdio.h**. C++ dispune suplimentar de un sistem propriu, realizat pe principiile POO (Programării Orientate pe Obiecte), dar acest aspect nu face obiectul acestei prezentări.

Se disting două tipuri de streamuri:

▪ **streamuri de tip text** – care transferă secvențe de caractere organizate în linii. În C, separarea liniilor se face prin caracterul linie nouă (LF – 0AH).

▪ **streamuri binare** - transferă o secvență de octeți fără nici o structură, deci fără alte prelucrări.

Distingerea dintre cele două categorii este convenabilă, dar nu obligatorie.

Există 5 dispozitive logice (constante de tip **FILE * -** definit în fişierul stdio.h) predefinite care se creează automat la lansarea în execuție a programului şi sunt închise automat la terminarea execuției:

▪ **stdin** – stream de tip text - reprezintă intrarea standard, care este asociată cu tastatura; poate fi redirectat de către sistemul de operare.

▪ **stdout** – stream de tip text – reprezintă ieşirea standard care e asociată cu monitorul; poate fi redirectat de către sistemul de operare.

▪ **stderr** - stream de tip text – reprezintă ieşirea standard pentru erori şi e asociată, de asemenea, cu monitorul.

▪ **strprn** – stream de tip binar – reprezintă ieşirea standard la imprimantă, asociată dispozitivului DOS PRN.

▪ **stdaux** – stream de tip binar – reprezintă intrare/ieşire auxiliară, asociată dispozitivului DOS AUX.

Unele funcții de intrare/ieşire sunt destinate unui anumit tip de date, cum ar fi char sau int, altele pot efectua operații cu oricare tip de date fundamental prin precizarea formatului specific fiecărui tip.

3.1. Operații la nivel de caracter

▪ **Funcții pentru operații de citire a unui caracter:**

Pentru citirea unui caracter din stdin pot fi folosite următoarele funcții:

```
int getchar(void);      // definită în stdio.h
```

Funcția întoarce primul caracter din stdin, corespunzător primei taste apăsate, dar după ce s-

a apăsat tasta Enter (CR). Caracterul este transformat în întreg fără semn. În caz de eroare, întoarce constanta EOF care are valoarea -1.

int getche(void); // definită în conio.h;

Funcția aşteaptă apăsarea unei taste, întoarce caracterul corespunzător şi îl afişează pe ecran (nu aşteaptă apăsarea tastei Enter).

int getch(void); // definită în conio.h;

Funcția aşteaptă apăsarea unei taste, întoarce caracterul corespunzător dar nu îl afişează pe ecran (nu aşteaptă apăsarea tastei Enter).

Exemple de folosire a funcțiilor de citire a unui caracter:

```
char ch;          // se declară variabila ch care poate primi valoarea citită prin una
                  // din funcțiile de citire a unui caracter
ch=getchar();     // funcția getchar() întoarce valoarea caracterului tastat, aceasta
                  // e preluată de variabila ch, dar numai după apăsarea tastei
                  // Enter; caracterul este afişat pe ecran
ch=getche();      // funcția getche() întoarce valoarea caracterului tastat, aceasta
                  // e preluată de variabila ch, fără a aştepta apăsarea tastei
                  // Enter; caracterul este afişat pe ecran
ch=getch();       // funcția getch() întoarce valoarea caracterului tastat, aceasta
                  // e preluată de variabila ch, fără a aştepta apăsarea tastei
                  // Enter; caracterul nu este afişat pe ecran
getch();          // funcția getch() întoarce valoarea caracterului tastat fără ca această
                  // valoare să fie utilizată; această instrucțiune se foloseşte de regulă
                  // pentru a întrerupe temporar evoluția programului până la apăsarea
                  // unei taste; caracterul nu este afişat pe ecran
```

▪ **Funcții pentru operații de afişare a unui caracter:**

Pentru scrierea unui caracter la stdout se folosesc funcțiile:

int putchar(int c); // definită în stdio.h

int putch(int c); // definită în conio.h

Funcțiile transferă un caracter către stdout (îl afişează pe ecran) şi întoarce ca rezultat caracterul transmis în caz de succes, sau EOF în caz de eşec.

Exemple de folosire a funcțiilor de afişare a unui caracter:

```
char ch='a';      // se declară variabila ch care poate primi valoarea citită prin una
                  // din funcțiile de citire a unui caracter
putch('x');       // se afişează caracterul constant 'x'
putchar('y');     // se afişează caracterul constant 'y'
putch(ch);        // se afişează conținutul variabilei ch, caracterul 'a'
putch('\n');      // caracterul constant '\n' (newline) nu este afişabil, dar mută
                  // cursorul pe linia următoare
putchar('\r');    // caracterul constant '\r' (CR-carriage return) nu este afişabil, dar
                  // mută cursorul la începutul rândului
putchar(99);      // se afişează caracterul constant cu valoarea 99 în codul ASCII, caracterul 'c'
putchar(0x63);    // se afişează caracterul constant cu valoarea 0x63 în codul
                  // ASCII, caracterul 'c'
```

3.2. Funcții pentru operații I/O pentru şiruri de caractere

▪ **Operații de citire a şirurilor de caractere:**

char * gets(char * s); // definită în stdio.h

Funcția citeşte caractere din stdin şi le depune în tabloul de la adresa s până întâlneşte caracterul '\n' (apăsarea tastei Enter). În şir '\n' este înlocuit cu terminatorul de şir '\0' . Dacă operația se încheie cu succes, funcția întoarce adresa şirului (egală cu valoarea parametrului), altfel întoarce valoarea NULL.

▪ **Operații de afişare a şirurilor de caractere:**

int puts (const char * s); // definită în stdio.h

Funcția copiază la stdout (pe ecran) şirul de la adresa s şi trece la linie nouă. În caz de succes întoarce ultimul caracter, altfel întoarce valoarea EOF.

Observații

- În C/C++ şirurile de caractere nu sunt definite printr-un tip de date predefinit, ci sunt construite ca tablouri cu elemente de tip char. Sunt formate din succesiuni de caractere încheiate cu caracterul '\0' (valoarea zero în codul ASCII).
- Un şir de caractere se declară cu sintaxa :

  char nume_sir[numar_caractere];

  Declarația se poate face şi cu inițializare.
- Numele şirului de caractere reprezintă adresa de memorie la care a fost alocat şirul, tipul datei fiind char * (pointer de char).

Exemple de folosire a funcțiilor I/O pentru şiruri de caractere:

```
char sir1[30]="primul sir";                 // declararea şirului sir1 cu inițializare
char sir2[30]="al doilea sir";              // declararea şirului sir2 cu inițializare
puts("sir1:");                              // afişarea şirului constant "sir1:"
puts(sir1);                                 // afişarea conținutului şirului sir1, se afişează
                                            // conținutul inițial
puts("sir2:");                              // afişarea şirului constant "sir2:"
puts(sir2);                                 // afişarea conținutului şirului sir2, se afişează
                                            // conținutul inițial
puts("\nIntrodu un sir de caractere:");
gets(sir1);                                 // se citeşte de la tastatură conținutul şirului sir1
puts("\nIntrodu un sir de caractere:");
gets(sir2);                                 // se citeşte de la tastatură conținutul şirului sir2
puts("sir1:");
puts(sir1);                                 // afişarea conținutului şirului sir1, se afişează
                                            // conținutul introdus de la tastatură
puts("sir2:");
puts(sir2);                                 // afişarea conținutului şirului sir2, se afişează
                                            // conținutul introdus de la tastatură
```

3.3. Operații de citire/scriere cu formatare

În cazul unei operații de citire, formatarea constă în conversia datelor de la reprezentarea lor externă (la dispozitivul de intrare) la reprezentarea internă, binară. Pentru o operație de ieşire (scrie) se efectuează conversia inversă.

Datele sunt introduse de la tastatură sub forma unor şiruri de caractere ASCII, corespunzătoare tastelor apăsate şi sunt separate de spațiere (spațiu, tab, linie nouă, etc.). O operație inversă este necesară pentru afişarea valorii pe ecran, deoarece dispozitivul de ieşire consolă aşteaptă un şir de coduri ASCII pe baza cărora creează imaginea de pe ecran.

Formatarea permite controlul afişării pe ecran (numărul de spații alocate, numărul de zecimale, etc.) şi al introducerii datelor (numărul de caractere alocate).

▪ **Citirea datelor cu formatare**

scanf(); // definită în stdio.h

Prototipul funcției este:

int scanf(const char* sir_de_formatare, [lista_adrese_variabile]);

unde : - **sir_de_formatare** – conține caractere ASCII şi specificatori de format.

- **valoarea returnată** de funcție reprezintă numărul câmpurilor a căror citire s-a realizat corect.

- **lista_adrese_variabile** – precizează adresele de memorie la care sunt memorate valorile introduse de la tastatură.

Pentru ca operația de intrare să se realizeze corect, trebuie ca numărul de formate precizate în şir_de_formatare şi numărul adreselor în lista_adrese_variabile să fie acelaşi şi tipurile precizate prin formate să corespundă tipurilor obiectelor ce se găsesc la adresele precizate.

▪ **Afişarea datelor cu formatare**

printf(); // definită în stdio.h

Prototipul funcției este:

int printf(const char* sir_de_formatare, [lista_valori]);

unde: - **sir_de_formatare** – conține caractere ASCII şi specificatori de format.

- **valoarea returnată** de funcție reprezintă numărul caracterelor afişate.

- **lista_valori** – reprezintă o înşiruire de expresii care returnează valori, cum ar fi nume de variabile, apeluri de funcții, expresii aritmetice, etc.

Pentru ca operația de afişare să se realizeze corect, trebuie ca numărul de formate precizate în şir_de_formatare şi numărul expresiilor din lista_valori să fie acelaşi şi tipurile precizate prin formate să corespundă tipurilor expresiilor.

Dacă lista de adrese sau de valori conțin mai multe expresii decât formatele din şirul de formatare, adresele sau valorile cărora nu le corespund formate vor fi ignorate. Dacă însă în şirul de formatare există formate care nu au corespondent adresă, respectiv valoare, fie se vor afişa valori aleatoare, fi se poate produce o eroare fatală care blochează execuția programului.

Regulile de alcătuire a şirurilor de formatare şi sintaxa celor două funcții, scanf() şi printf(), sunt similare.

Specificatorii de format încep cu caracterul % şi sunt definiți pentru toate tipurile predefinite de date. În tabelul nr. 3.1. sunt prezentate formatele pentru fiecare tip de date. Aceste secvențe pot fi completate cu modelatori de format.

Tabel nr. 3.1. Specificatori de format

Cod	Semnificație
%c	caracter, tipul char
%d	număr întreg zecimal
%i	număr întreg zecimal
%u	număr întreg fără semn
%o	număr întreg reprezentat în octal
%x	număr întreg reprezentat în hexazecimal, se folosesc cifrele a,b,c,d,e,f
%X	număr întreg reprezentat în hexazecimal, se folosesc cifrele A,B,C,D,E,F
%f	număr în virgulă mobilă
%e	număr în virgulă mobilă, format ştiințific, e pentru exponent
%E	număr în virgulă mobilă, format ştiințific, E pentru exponent
%g	număr în virgulă mobilă, format ştiințific, e pentru exponent
%G	număr în virgulă mobilă, format ştiințific, E pentru exponent
%s	şir de caractere
%p	adresă de memorie reprezentată în hexazecimal
%[ ]	Caută un set de caractere
%n	Primeşte o valoare egală cu numărul de caractere citite până atunci

Specificatorii de format pot fi însoțiți de modelatori de format, cum ar fi:

▪ **specificator pentru mărimea câmpului** – un întreg care determină numărul de caractere ale câmpului

▪ **specificator de precizie** – urmează după specificatorul pentru mărimea câmpului şi constă în punct urmat de un întreg

▪ **modelatorul + -** forțează afişarea semnului plus (+) dacă se afişează valoare pozitivă (afişarea semnului minus se face în mod implicit pentru valori negative)

▪ **modelatorul -** - forțează alinierea la stânga a câmpului afişat

▪ **modelatorul *** - se foloseşte pentru rezervarea unui loc pentru un argument

▪ **modelatorul #** - se afişează punctul zecimal pentru valori reprezentate în virgulă mobilă care nu au parte zecimală

- se afişează cifra 0 pe prima poziție a unui întreg octal
- se afişează 0x la începutul unui întreg hexazecimal

Descrierea sintaxei folosită pentru apelul funcției scanf() este făcută în Fig. 3.1, iar cea a apelului funcției printf() în Fig. 3.2.

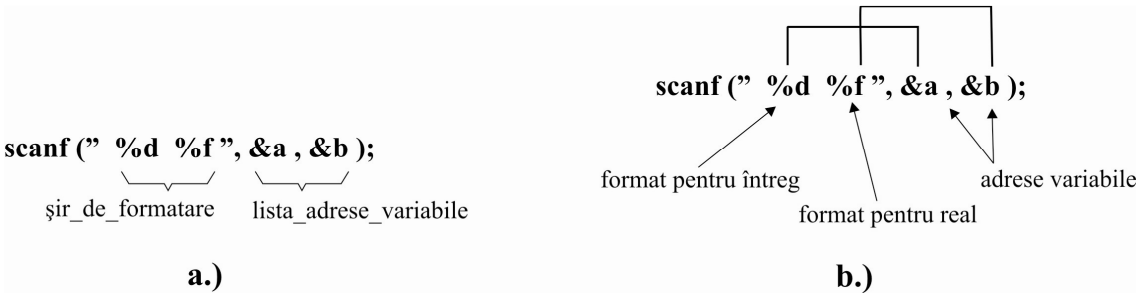

Fig. 3.1. Sintaxa apelului funcției scanf()
a.) descrierea parametrilor efectivi ; b.) corespondența dintre formatele precizate şi adresele variabilelor

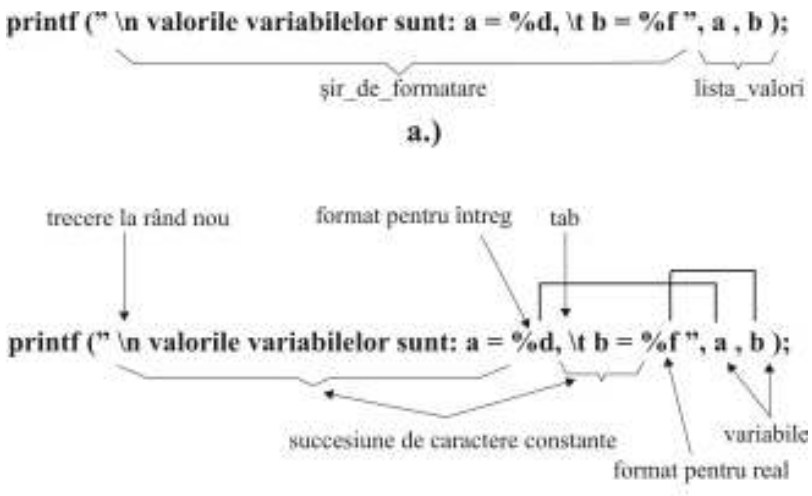

Fig. 3.2. Sintaxa apelului funcției printf()
a.) descrierea parametrilor efectivi ; b.) alcătuirea şirului de formatare şi corespondența dintre formatele precizate şi valori

Se consideră exemplul următor în care se declară două variabile, una int, cealaltă float. Se citesc valori de la tastatură pentru ambele variabile folosind funcția scanf() şi apoi se afişează valorile preluate de cele două variabile folosind funcția printf().

```
int a;
float b;
scanf("%d, %f", &a, &b);
printf("\na = %d, \t b = %f", a, b);
```

Observații

- Şirul de formatare al funcției scanf() este recomandat să fie alcătuit numai din formatele specifice pentru datele care urmează a fi citite. Orice alt caracter, în afara acestor formate, va trebui tastat la introducerea datelor. Acest lucru poate genera erori la citirea datelor. De exemplu, pentru instrucțiunea:

  scanf("a=%d", &a);

  la execuție, pentru ca citirea să se facă corect, trebuie tastat caracterul 'a',

caracterul ‘=’ şi apoi valoarea pentru variabila a.

- Dacă şirul de formatare al funcției printf() conține alte caractere în afara formatelor, acele caractere vor fi afişate pe ecran.
- Dacă şirul de formatare al funcției printf() nu conține nici un format, se va afişa şirul constant specificat, efectul fiind similar cu cel al funcției gets().

Ca orice şir constant şi şirul de formatare poate conține secvențe de escape. Dacă se doreşte afişarea caracterelor \, ‘ sau “, care au o semnificație anume în sintaxa specifică limbajului, acestea trebuie precedate de caracterul \. De exemplu, funcția

printf(“ \” \\ \” “);

va afişa pe ecran succesiunea de caractere “\”

- În cazul ambelor funcții, scanf() şi printf(), lista de adrese, respectiv lista de variabile pot lipsi, dar **şirul de formatare este obligatoriu să existe.**

Este important să se respecte corespondența între formatul datei precizat şi tipul variabilei căreia i se atribuie valoarea citită de la tastatură. În caz contrar, valorile preluate de variabile sunt aleatoare sau se pot genera erori fatale, execuția programului fiind întreruptă. Este situația prezentată în exemplul următor:

```
int a;
float b;
scanf(“%d, %f”, &b, &a);          // eroare, formatul %d corespunde unei variabile float,
                                  // iar formatul %f corespunde unei date int,
printf(“\na=%d, \tb=%f”, a, b);
```

În continuare sunt prezentate mai multe exemple prin care se explică modul de folosire a tuturor formatelor şi a modelatorilor de format.

```
int a;
printf("\nIntrodu un intreg:");          // se afişează şirul constant "\nIntrodu un intreg:"
scanf("%x", &a);                        // se citeşte variabila a în format hexazecimal
printf("\na=%d, a=%o, a=%x, a=%u, a=%c", a, a, a, a, a);   // se afişează valoarea variabilei
                                                           // a în diferite formate: zecimal, octal,
                                                           // hexazecimal, unsigned, caracter
float r;
printf("\nIntrodu un real:");
scanf("%f", &r);                        // se citeşte o valoare reală
printf("\nr=%f, r=%e, r=%g", r, r, r);  // se afişează valoarea citită în format clasic şi
                                        // format ştiințific
printf("\nIntrodu un real(in format stiintific):");
scanf("%e", &r);
printf("\nr=%f, r=%e, r=%g", r, r, r);
```

Observații

- Formatul %f este formatul specific valorilor reale:

parte_întreagă . parte_zecimală

- În mod implicit se afişează 6 zecimale. De exemplu, valoarea 12.34 va fi afişată: 12.340000.
- Formatul %e sau %E reprezintă formatul ştiinţific de scriere a realilor:

  x . xxxxxx E +/-xx

  De exemplu, valoarea 12.34 va fi afişată: 1.234000E+01.
- Formatul %g sau %G alege între formatul clasic de scriere sau formatul ştiinţific, astfel încât numărul de caractere utilizat să fie minim.

```
printf (”\n*%f*”, 12.3456);          // implicit se afişează 6 zecimale
printf (“\n*%10.2*”,12.3456);        // se precizează numărul de caractere al câmpului
                                     // (10-numărul total de caractere) şi precizia de
                                     // afişare (2 zecimale)
printf (“\n*%+10.2*”, 12.3456);      // se afişează semnul +
printf (“\n*%+010.2*”, 12.3456);     // caracterele neutilizate ale câmpului vor fi
                                     // completate cu 0
```

```
char ch;
scanf(“%c”, &ch);                    // se citeşte un caracter
printf(“\nc = %c, c = %d”, ch, ch);  // se afişează caracterul tastat şi valoarea
                                     // lui din codul ASCII
```

```
char sir[20];
printf("\nIntrodu un sir de caractere:");
scanf("%s", &sir);                   // se citeşte un şir de caractere
printf("\nsirul este=%s ", sir);     // se afişează şirul de caractere
```

Observaţii

- La citirea unui şir de caractere nu este necesar să se folosească operatorul adresă deoarece numele şirului reprezintă adresa la care este alocat în memorie. Deci următoarele două instrucţiuni sunt echivalente:

```
scanf("%s", &sir);
scanf("%s", sir);
```

```
char a[20], b[20];
printf("\nIntrodu un sir [abc]:");
scanf("%[abc]", a);                  // se citeşte un şir de caractere format numai din caracterele
                                     // ‘a’, ‘b’ sau ‘c’ ; la tastarea primului caracter diferit
                                     // de acestea, se va întrerupe operaţia de citire
printf("\nIntrodu alt sir [d-m]:");
scanf("%[d-m]", b);                  // se citeşte un şir de caractere format numai din
                                     // caracterele cuprinse în intervalul ‘d’ - ‘m’ ;
printf("\nSirurile sunt:");
printf("\na=%12s", a);                // se afişează conţinutul şirului a pe minim 12
                                     // caractere, completându-se cu spaţii dacă este
                                     // cazul; alinierea se face la dreapta
printf("\nb=%-10s", b);               // se afişează conţinutul şirului b pe minim 10
                                     // caractere, completându-se cu spaţii dacă este
                                     // cazul; alinierea se face la stânga
char a[20]=”abcdefgh”;
printf(“\n*%s*”, a);
```

```
printf("\n*%10s*", a);
printf("\n*%-10s*", a);
printf("\n*%5s*", a);                    // dacă mărimea câmpului e mai mică decât lungimea
                                         // şirului, acesta nu va fi trunchiat, ci se afişează întregul şir
```

Pentru secvența anterioară se afişează:

```
*abcdefgh*
*   abcdefgh*
*abcdefgh    *
*abcdefgh*
```

```
int a=10;
printf("adresa variabilei: %p\n", &a);// se afişează adresa variabilei a în format
                                      // hexazecimal
printf("adresa variabilei: %u\n", &a);// se afişează adresa variabilei a ca întreg
                                      // fără semn în reprezentare zecimală
```

```
printf("Exemplu de folosire a modelatorului # \n");
printf("%o \t %#o", 124);      // se afişează valoarea 124 transformată în format octal
printf("%x \t %#x", 124);      // se afişează valoarea 124 transformată în reprezentare
                               // hexazecimală
```

Pentru secvența anterioară se va afişa:

```
174             0174
7C              0x7C
```

Aşa cum s-a precizat, transferul informației în operațiile I/O se face prin zone de memorie tampon asociate cu stream-urile folosite în care se depune informația temporar, până la preluarea şi prelucrarea de funcțiile corespunzătoare.

Există posibilitatea ca la execuția unei funcții I/O în zona tampon să se găsească date reziduale rămase de la operațiile anterioare din diferite cauze, cum ar fi depăşirea dimensiunii câmpurilor, utilizarea de caractere ilegale, etc. Pentru a evita preluarea de date reziduale, este necesar ca înainte de aceste operații să se realizeze golirea zonelor de memorie tampon. Acest operație se poate face folosind funcția fflush(), definită în fişierul stdio.h care are prototipul:

int fflush(FILE* stream);

Funcția goleşte streamul specificat ca parametru şi returnează valoarea 0 dacă operația s-a efectuat cu succes şi EOF în caz contrar.

```
int a, b;
scanf("%2d", &a);
scanf("%2d", &b);
```

Dacă la execuția secvenței anterioare pentru variabila a se tastează 123456, variabila a preia două caractere, deci a=12, în zona tampon rămânând 3456. Ca urmare b va prelua valoarea 34 şi în zona tampon rămâne secvența 56. Aceasta va fi preluată de următoarea funcție de citire de la tastatură. Pentru a evita astfel de situații, se va folosi funcția fflush() :

```
fflush(stdin);
scanf("%2d", &a);
fflush(stdin);
scanf("%2d", &b);
```

Exemple

```
/*************************************************************************
```

Exemplul 3.1. - Se scrie un program în care se exemplifică folosirea funcţiilor de intrare - ieşire pentru caractere în care:

- se declară 3 variabile de tip char sau int;
- se citesc valori pentru cele trei variabile cu funcţiile getch(), getche(), respectiv getchar();
- se afişează conţinutul variabilelor folosind funcţiile putch() sau putchar()

Observaţii: Se afişează mesaje care să ajute la înţelegerea desfăşurării programului.

```
*************************************************************************/
#include<conio.h>
#include <stdio.h>

void main(void)
{ char c1, c2, c3;

  puts("Introdu un caracter : ");
  c1 = getch();
  puts("\ncaracterul tastat este : ");
  putch(c1);
  puts("\nIntrodu un caracter : ");
  c2 = getche();
  puts("\ncaracterul tastat este : ");
  putch(c2);
  puts("\nIntrodu un caracter : ");
  c3=getchar();
  puts("\ncaracterul tastat este : ");
  putch(c3);
  putch('\n');
 }


/*************************************************************************
```

Exemplul 3.2. - Se scrie un program în care se exemplifică folosirea funcţiilor de intrare - ieşire pentru şiruri de caractere în care:

- se declară 2 şiruri de caractere cu iniţializare;
- se afişează conţinutul şirurilor cu funcţia puts();
- se citesc valori pentru cele două şiruri cu funcţia gets();
- se afişează din nou conţinutul şirurilor folosind funcţia puts().

Observaţie: Se vor afisa mesaje care sa ajute la înţelegerea desfăşurării programului.

```
*************************************************************************/
#include<stdio.h>
#include<conio.h>

void main(void)
{
  char sir1[30]="primul sir";
  char sir2[30]="al doilea sir";

  puts("sir1:");
  puts(sir1);
  puts("sir2:");
  puts(sir2);
```

```
 puts("\nIntrodu un sir de caractere:");
 gets(sir1);
 puts("\nIntrodu un sir de caractere:");
 gets(sir2);
 puts("sir1:");
 puts(sir1);
 puts("sir2:");
 puts(sir2);
 getch();
}
```

/**

Exemplul 3.3. - Se scrie un program în care se exemplifică folosirea funcțiilor de intrare - ieşire pentru diverse tipuri de date.

**/

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

void main()
{       char  caracter = 'A';
        int   nr_intreg = 20;
        float nr_real = 123.45;
        double nr_double = 1122.33E3;
        char sir[30] = "Exemplu";

        printf("caracter = %c\n", caracter );
        printf("intreg = %d\n", nr_intreg);
        printf("real = %f\n", nr_real );
        printf("real dublu = %e\n", nr_double);
        printf("sir de caractere = %s\n", sir);
        getch();
}
```

/**

Exemplul 3.4. - Se scrie un program în care se exemplifică folosirea funcțiilor de intrare - ieşire pentru tipul de date int. Se folosesc toate diverse formate compatibile cu tipul int.

**/

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

void main()
{  int a, b;

  printf("\nIntrodu un intreg:");
  scanf("%d", &a);
  printf("\na=%d, a=%o, a=%x, a=%u, a=%c", a, a, a, a, a);
  getch();
```

Format pentru reprezentarea în baza 10 (zecimal).

Format pentru reprezentarea în baza 8 (octal).

Format pentru reprezentarea în baza 16 (hexazecimal).

Format pentru reprezentarea ca întreg fără semn.

Format pentru reprezentarea ca şi caracter.

```
 printf("\nIntrodu 2 intregi:");
 scanf("%d%d", &a, &b);
 printf("\na=%d, a=%o, a=%x, a=%u, a=%c", a, a, a, a, a);
 printf("\nb=%d, b=%o, b=%x, b=%u, b=%c", b, b, b, b, b);
 getch();
 printf("\nIntrodu 2 intregi:");
 scanf("%2d%2d",&a,&b);
 printf("\na=%d, a=%o, a=%x, a=%u, a=%c", a, a, a, a, a);
 printf("\nb=%d, b=%o, b=%x, b=%u, b=%c", b, b, b, b, b);
 getch();
 printf("\n*%+6d*",1);
 printf("\n*%+6d*",12);
 printf("\n*%+6d*",123);
 printf("\n*%+6d*",1234);
 printf("\n*%+6d*",12345);
 getch();
}
```

Se precizează numărul de caractere preluate de variabilă

/**

Exemplul 3.5. - Se scrie un program în care se exemplifică folosirea funcțiilor de intrare - ieşire pentru tipul de date float. Se folosesc toate diverse formate compatibile cu tipul float.

**/

```
#include <stdio.h>
#include <conio.h>
void main()
{
 float a, b;
 printf("\nIntrodu un real:");
 scanf("%f", &a);
 printf("\na=%f, a=%e, a=%g", a, a, a);

 printf("\n*%12f*", a);
 printf("\n*%.2f*", a);
 printf("\n*%12.2f*", a);
 printf("\n*%+12.2f*", a);
 getch();
}
```

Format de afişare a valorilor reale prin parte întreagă şi parte fracționară.

Format de afişare a valorilor reale în format ştiințific.

Se precizează numărul de caractere folosite în afişare.

Se face afişare cu 2 zecimale.

Se cere afişarea semnului.

Întrebări. Exerciții. Probleme.

1. Să se scrie un program în care se exemplifică folosirea funcțiilor de intrare - ieşire cu formatare pentru şiruri de caractere. Se folosesc diverse formate compatibile cu şiruri de caractere, similar exemplelor 3.3, 3.4.

2. Să se scrie un program în care se exemplifică folosirea funcțiilor de intrare - ieşire cu formatare pentru diverse tipuri de date (char, int, float, double). Se vor folosi şi modelatorii cunoscuți. Se are în vedere asocierea corectă a formatelor cu tipurile datelor asociate.