Modul 3 – Funcții de intrare/ieşire în limbajul C

Funcții de intrare/ieşire în C

În limbajul C, nu există instrucțiuni de intrare/ieşire (citire/scriere), tocmai pentru a mări portabilitatea limbajului. Pentru a realiza citiri şi scrieri se apelează funcții de intrare/ieşire din bibliotecile mediului de programare.
Pentru operații de citire a datelor inițiale şi de afişare a rezultatelor sunt definite funcții standard, declarate în fişierul antet *stdio.h. S*e vor prezenta şi funcțiile de intrare/ieşire nestandard cele mai uzuale, având prototipul in *conio.h*. Utilizarea acestor funcții într-un program, va presupune deci, includerea respectivelor fişiere antet.
Aici vom prezenta numai acele funcții folosite pentru citire de la tastatură şi pentru afişare pe ecran, deci pentru lucru cu fişierele standard numite *stdin* şi *stdout*. Fişierele standard *stdin* şi *stdout* sunt fişiere text. Un fişier text este un fişier care conține numai caractere ASCII grupate în linii (de lungimi diferite), fiecare linie terminată cu un terminator de linie format din unul sau două caractere. În sistemele Windows se folosesc două caractere ca terminator de linie: *n* şi *r*, adică *newline* (trecere la linie nouă) şi *return (*trecere la început de linie). În sistemele Unix/Linux se foloseşte numai caracterul *n* (*newline*) ca terminator de linie, caracter generat de tasta *Enter*.

Funcțiile I/O (Input/Output - intrare/ieşire) din C pot fi grupate în câteva familii:

- **Funcții de citire/scriere caractere individuale: *getchar, putchar, getch, putch*;**
- **Funcții de citire/scriere linii de text: *gets, puts*;**
- **Funcții de citire/scriere cu format (cu conversie): *scanf, printf*.**

Funcții de citire/scriere pentru caractere şi şiruri de caractere

Pentru operațiile I/O la nivel de caracter:

- **int getchar ();**
- **int putchar (int c);**
- **int getche ();**
- **int getch ();**
- **int putch (int c);**

Pentru operații I/O cu șiruri de caractere:

- **char * gets(char * s);**
- **int puts(const char * șir);**

Tabelul următor conține o descriere detaliată a funcțiilor pentru operațiile I/O la nivel de caracter:

Funcție	Exemple	Descriere	Rezultat
int getchar();	`char c; c=getchar();`	Citeşte un caracter	Returnează codul unui caracter citit de la tastatura sau valoarea *EOF* (**End Of File** - constantă simbolică definită în *stdio.h*, având valoarea -1) dacă s-a tastat Ctrl+Z.
int putchar(int c);	`char c; putchar(c);`	Tipăreşte pe ecran caracterul transmis ca parametru	Returnează codul caracterului sau *EOF* în cazul unei erori.
int getche();	`char c; c=getche();`	Citeşte un caracter (aşteaptă apăsarea unei taste, chiar dacă în buffer-ul de intrare mai sunt caractere neprelucrate) şi afişează caracterul pe ecran	Returnează *EOF* la tastarea lui Ctrl+Z, respectiv *CR* (*\r*, cu codul 13) la tastarea lui *Enter*.
int getch();	`char c; c=getch();`	Analog cu funcția de mai sus, dar caracterul nu se transmite în ecou (nu se afişează pe ecran).	
int putch(int c);	`char c; putch(c);`	Tipăreşte pe ecran caracterul transmis ca parametru	Returnează codul caracterului sau *EOF* în cazul unei erori.

Funcțiile *getch*, *putch*, *getche* nu sunt standard, de aceea este bine să se evite utilizarea lor!

Tabelul următor conține o descriere detaliată a funcțiilor pentru operațiile I/O cu șiruri de caractere:

Funcție	Exemple utilizare	Descriere	Rezultat
char * gets(char * s);	`char sir[10]; gets(sir);`	Citeşte caractere până la întâlnirea lui *Enter*; acesta nu se adaugă la şirul s. Plasează */0* la sfârşitul lui s. Obs: codul lui *Enter* e scos din buffer-ul de intrare.	Returnează adresa primului caracter din şir. Dacă se tastează CTRL+Z returnează *NULL*.
int puts(const char * s);	`char sir[10]; puts(sir);`	Tipăreşte şirul primit ca parametru, apoi *NewLine*, cursorul trecând la începutul rândului următor	Returnează codul ultimului caracter din şir sau *EOF* la insucces

Funcții de citire/scriere cu format

Funcțiile *scanf* şi *printf* permit citirea cu format şi respectiv scrierea cu format pentru orice tip de date. Antetul şi descrierea acestor funcții sunt:

int printf (format, arg1, arg2, ...);

- **afişează pe ecran valorile expresiilor din lista argumentelor, conform formatului specificat;**
- **argumentele pot fi constante, variabile, expresii**
  - **dacă nu apare nici un argument, pe ecran se tipăresc doar caracterele din şirul format.**
- **formatul este un şir de caractere care trebuie să includă câte un descriptor de format pentru fiecare din argumente.**
  - **caracterele din format care nu fac parte din descriptori se tipăresc pe ecran.**

Returnează numărul de valori tipărite sau EOF în cazul unei erori.

int scanf (format, adr1, adr2, ...);

- **Citeşte informațiile tastate pe care le interpretează conform specificatorilor din format, memorând valorile citite la adresele transmise ca parametri.**
- **Formatul este un şir de caractere care trebuie să includă câte un descriptor de format pentru fiecare dintre valorile citite.**
- **Adresele sunt pointeri sau adresele variabilelor ale căror valori se citesc;**
  - **Adresa unei variabile se obține folosind operatorul de adresare &, astfel: &nume_variabila**
  - **Valorile întregi sau reale consecutive introduse de la tastatură trebuie separate de cel puțin un spațiu alb (enter, spațiu, tab)**

Returnează numărul de valori citite sau EOF dacă s-a tastat Ctrl/Z.

Funcția *printf*

Descriptorii de format admişi în funcția *printf* sunt:

Specificatori format	Descriere
%d	întreg zecimal cu semn
%i	întreg zecimal, octal (0) sau hexazecimal (0x, 0X)
%o	întreg în octal, fără 0 la inceput
%u	întreg zecimal fără semn
%x, %X	întreg hexazecimal, fără 0x/0X; cu a-f pt. %x, A-F pt. %X
%c	caracter
%s	şir de caractere, până la '\0' sau nr. de caractere dat ca precizie

%f, %F	real fără exp.; precizie implicită 6 poz.; la precizie 0: fără punct real (posibil cu exponent)
%e, %E	numere reale cu mantisă şi exponent (al lui 10); precizie implicită 6 poz.; la precizie 0: fără punct
%g, %G	numere reale în format %f sau %e, funcție de valoare real, ca %e, %E dacă exp. < -4 sau precizia; altfel ca %f. Nu tipăreşte zerouri sau punct zecimal în mod inutil
%p	pointer, în formatul tipărit de printf
%ld, %li	numere întregi lungi
%lf, %le, %lg	numere reale în precizie dublă (*double*)
%Lf, %Le, %Lg	numere reale de tip *long double*
%%	caracterul procent

Exemple de utilizare *printf*:

```
printf ("\n");                          // trecere la o noua linie
printf ("\n Eroare \n");                // scrie „ Eroare ” pe o linie nouă

int a=3;
printf ("%d\n",a);
// scrie „3” ca întreg şi trece la linia următoare

int a=5, b=7;
printf ("a=%d  b=%d\n", a, b);
//scrie „a=5 b=7” şi trece la linia următoare

int g=30, m=5, s=2;
printf ("%2d grade %2d min %2d sec\n", g,m,s);
// scrie „30 grade  5 min   2 sec„ şi schimba linia

int anInt = 12345;
float aFloat = 55.6677;
double aDouble = 11.2233;
char aChar = 'a';
char aStr[] = "Hello";

printf("The int is %d.\n", anInt);       //The int is 12345.
printf("The float is %f.\n", aFloat);    //The float is 55.667702.
printf("The double is %f.\n", aDouble); //The double is 11.223300.
printf("The char is %c.\n", aChar);      //The char is a.
printf("The string is %s.\n", aStr);     //The string is Hello.

printf("The int (in hex) is %x.\n", anInt);   //The int (in hex) is 3039.
printf("The double (in scientic) is %e.\n", aDouble);
    //The double (in scientic) is 1.122330e+01.

printf("The float (in scientic) is %E.\n", aFloat);
   //The float (in scientic) is 5.566770E+01.
```

Programatorul trebuie să asigure concordanța dintre descriptorii de format şi tipul variabilelor sau expresiilor care urmează argumentului, deoarece funcțiile *scanf* şi *printf* nu fac nici o verificare şi nu semnalează neconcordanțe. Exemplele următoare trec de compilare dar afişează incorect valorile variabilelor:

```
int i=3;
float f=3.14;
printf ("%f \n", i);          // scrie 0.00 (în Dev-Cpp)
printf ("%i \n", f);          // scrie 1610612736  (Dev-Cpp)
```

Funcțiile *scanf* şi *printf* folosesc noțiunea de *câmp* (field): un câmp conține o valoare şi este separat de alte câmpuri prin spații albe. Fiecare descriptor de format poate conține mărimea câmpului, ca număr întreg. Această mărime se foloseşte mai ales la afişare, pentru afişare numere pe coloane, aliniate la dreapta. În lipsa acestei informații mărimea câmpului rezultă din valoarea afişată.
Exemple:

```
printf("%d %d",a,b);          // 2 campuri separate prin blanc
printf("%8d8%d",a,b);         // 2 câmpuri de cate 8 caractere

int number = 123456;
printf("number=%d.\n", number);
//number=123456.

printf("number=%8d.\n", number);
//number=  123456.
printf("number=%3d.\n", number);  // Dim camp prea mica.E ignorata.
//number=123456.

double value = 123.14159265;
printf("value=%f;\n", value);
//value=123.141593;
printf("value=%6.2f;\n", value);
//value=123.14;
printf("value=%9.4f;\n", value);
//value= 123.1416;
printf("value=%3.2f;\n", value);  // Dim camp prea mica.E ignorata.
//value=123.14;
```

Dacă nu se precizează mărimea câmpului şi numărul de cifre de la partea fracționară pentru numere, atunci funcția *printf* alege automat aceste valori:

- dimensiunea câmpului rezultă din numărul de caractere necesar pentru afişarea cifrelor, semnului şi altor caractere cerute de format;
- numărul de cifre de la partea fracționară este 6 indiferent dacă numerele sunt de tip *float* sau *double* sau *long double*, dacă nu este precizat explicit.

Se poate preciza numai mărimea câmpului sau numai numărul de cifre la partea fracționară.
Exemple:

```
float a=1.; double b=0.0002; long double c=7.5; float d=-12.34;
printf ("%.0f  %20lf  %20.10Lf  %f  \n", a, b, c, d);
```

Specificând dimensiunea câmpului în care se afişează o valoare, putem realiza scrierea mai multor valori în coloane. Dacă valoarea de afişat necesită mai puține caractere decât este mărimea câmpului, atunci această valoare este aliniată la dreapta în câmpul respectiv.
Secvența următoare va scrie trei linii, iar numerele afişate vor apare într-o coloană cu cifrele de aceeaşi pondere aliniate unele sub altele:

```
int a=203, b=5, c=16;
printf ("%10d \n %10d \n %10d \n",a,b,c);
```

Formatul cu exponent ("%e") este util pentru numere foarte mari, foarte mici sau despre ale căror valori nu se ştie nimic. Numărul este scris cu o mantisă fracționară (între 0 şi 10) şi un exponent al lui 10, după litera E (e).

La formatul "%g" *printf* alege între formatul "%f" sau "%e" în funcție de ordinul de mărime al numărului afişat: pentru numere foarte mari sau foarte mici formatul cu exponent, iar pentru celelalte formatul cu parte întreagă şi parte fracționară.

Între caracterul '%' şi literele care desemnează tipul valorilor scrise mai pot apare, în ordine:

a) **un caracter ce exprimă anumite opțiuni de scriere:**
   - (minus): aliniere la stânga în câmpul de lungime specificată
   + (plus): se afişează şi semnul '+' pentru numere pozitive
   0 : numerele se completează la stânga cu zerouri pe lungimea *w*
   # : formă alternativă de scriere pentru numere

b)
- **un număr întreg *w* ce arată lungimea câmpului pe care se scrie o valoare, sau**
- **caracterul * dacă lungimea câmpului se dă într-o variabilă de tip int care precede variabila a cărei valoare se scrie.**

c) **punct urmat de un întreg, care arată precizia (numărul de cifre de după punctul zecimal) cu care se scriu numerele neîntregi.**

d) **una din literele 'h', 'l' sau 'L' care modifică lungimea tipului numeric.**

```
/* Exemplu de utilizare a opțiunii '0' pentru a scrie întotdeauna două
cifre, chiar şi pentru numere de o singură cifră: */
int ora=9, min=7, sec=30;
printf ("%02d:%02d:%02d\n",ora, min, sec); // scrie  09:07:30

//Exemplu ce afiseaza 10 spatii
printf("afisez 10 spatii: %*c",10,' ');

// Exemplu de utilizare a opțiunii '-' pentru aliniere şiruri la stânga:
char  a[] = "unu", b[] ="cinci", c[]= "sapte" ;
printf (" %-10s \n %-10s \n %-10s \n", a, b, c);

int i1 = 12345, i2 = 678;
printf("Hello, first int is %d, second int is %5d.\n", i1, i2);
//Hello, first int is 12345, second int is   678.
printf("Hello, first int is %d, second int is %-5d.\n", i1, i2);
//Hello, first int is 12345, second int is 678  .
```

```
char msg[] = "Hello";
printf("xx%10sxx\n", msg);     //xx      Helloxx
printf("xx%-10sxx\n", msg);    //xxHello      xx
```

În general trebuie să existe o concordanță între numărul şi tipul variabilelor şi formatul de citire sau scriere din funcțiile *scanf* şi *printf*, dar această concordanță nu poate fi verificată de compilator şi nici nu este semnalată ca eroare la execuție, dar se manifestă prin falsificarea valorilor citite sau scrise. O excepție notabilă de la această regulă generală este posibilitatea de a citi sau scrie corect numere de tip *double* cu formatul "%f" (pentru tipul *float*), dar nu şi numere de tip *long double* (din cauza diferențelor de reprezentare internă a exponentului).

Funcția scanf

Descriptorii de format admişi în funcția *scanf* sunt:

Specificatori format	Descriere
%d	întreg zecimal cu semn
%i	întreg zecimal, octal (0) sau hexazecimal (0x, 0X)
%o	întreg în octal, precedat sau nu de 0
%u	întreg zecimal fără semn
%x, %X	întreg hexazecimal, precedat sau nu de 0x, 0X
%c	orice caracter; nu sare peste spații (doar " %c")
%s	şir de caractere, până la primul spațiu alb. Se adaugă '\0'.
%e, %E, %f, %F, %g, %G, %a, %A	real (posibil cu exponent)
%p	pointer, în formatul tipărit de printf
%ld, %li	numere întregi lungi
%lf	numere reale în precizie dublă (*double*)
%Lf	numere reale de tip *long double*
%[…]	şir de caractere din mulțimea indicată între paranteze
%[^…]	şir de caractere exceptând mulțimea indicată între paranteze
%%	caracterul procent

După cum spuneam mai sus, argumentele funcției *scanf* sunt de tip pointer şi conțin adresele unde se memorează valorile citite. Pentru variabile numerice aceste adrese se obțin cu operatorul de adresare (*&*) aplicat variabilei care primeşte valoarea citită.
Numerele citite cu *scanf* pot fi introduse pe linii separate sau în aceeaşi linie dar separate prin spații albe sau caractere *Tab*. Între numere succesive pot fi oricâte caractere separator ('\n','\t',' '). Un număr se termină la primul caracter care nu poate apare într-un număr.

Exemple:

```
int n, a,b;
scanf("%d", &n);        // citeşte un întreg în variabila n
scanf("%d%d", &a,&b);   // citeşte doi întregi in a şi b
float rad;
scanf("%f", &rad);      // citeşte un numar real in rad
char c;
scanf("%c", &c);        // citeşte un caracter in c

// program test scanf
#include <stdio.h>

int main() {
   int anInt;
   float aFloat;
   double aDouble;

   printf("Introduceti un int: "); // Mesaj afisat
   scanf("%d", &anInt);            // citeşte un întreg în variabila anInt
   printf("Valoarea introdusa este %d.\n", anInt);

   printf("Introduceti un float: ");// Mesaj afisat
   scanf("%f", &aFloat);            // citeşte un float în variabila aFloat
   printf("Valoarea introdusa este %f.\n", aFloat);

   printf("Introduceti un double: ");// Mesaj afisat
   scanf("%lf", &aDouble);        // citeşte un întreg în variabila aDouble
   printf("Valoarea introdusa este %lf.\n", aDouble);

   return 0;
}
```

La citirea de şiruri trebuie folosit un vector de caractere, iar în funcția *scanf* se foloseşte numele vectorului (echivalent cu un pointer). Exemplu:

```
char s[100];
scanf("%s", s);   //uneori funcționează dar este greşit:  scanf("%s, &s);
```

Chiar şi spațiile trebuie folosite cu atenție în şirul format din *scanf*. Exemplu de citire care poate crea probleme la execuție din cauza blancului din format:

```
scanf("%d  ",&a);  // corect este  scanf ("%d",&a);
```

Funcția *scanf* nu poate afişa nimic, iar pentru a precede introducerea de date de un mesaj trebuie folosită secvența *printf*, *scanf*. Exemplu:

```
printf ("n= ");
scanf ("%d", &n);   // NU: scanf("n=%d", &n);
```

De reținut diferența de utilizare a funcțiilor *scanf* şi *printf*!
Exemplu:

```
scanf("%d%d", &a, &b);  // citeşte două numere întregi în a şi b
printf("%d %d", a, b);  // scrie valorile din a şi b separate de un spațiu
```

Alte exemple de utilizare a funcțiilor *scanf* şi *printf*:

```
int i;
float f;
double d;
scanf("%d%f%lf",&i, &f, &d); //citeste un intreg, un real si un double

char c;
printf("Rezultatul este: %c\n",c); //afiseaza un mesaj si un character

float a;
double b;
scanf("%f", &a);        // citire variabila float
printf("%5.2f",a);      // sunt afisate minim 5 caractere, maxim 2 zecimale
scanf("%lf", &b);       // citire variabila double
printf("%-4.2lf",b);    // afisare cu aliniere stanga
printf("%+4.2lf",b);    // afisare cu adaugare semn (+,-)
```

Observații

Toate funcțiile de citire menționate (excepție *getch*, *getche*) folosesc o zonă tampon (*buffer*) în care se adună caracterele tastate până la apăsarea tastei *Enter*, moment în care conținutul zonei buffer este transmis programului. În acest fel este posibilă corectarea unor caractere introduse greşit înainte de apăsarea tastei *Enter*.

Caracterul *\n* este prezent în zona buffer numai la funcțiile *getchar* şi *scanf*, dar funcția *gets* înlocuieşte acest caracter cu un octet zero, ca terminator de şir în memorie (rezultatul funcției "gets" este un şir terminat cu zero).

Funcția *scanf* recunoaşte în zona buffer unul sau mai multe *câmpuri* (*fields*), separate şi terminate prin caractere spațiu alb (blanc, '\n', '\r','\f'); drept consecință, preluarea de caractere din buffer se opreşte la primul spațiu alb, care poate fi şi caracterul terminator de linie. Următorul apel al funcției *getchar* sau *scanf* se uită în zona buffer dacă mai sunt caractere, înainte de a aştepta introducerea de la taste şi va găsi caracterul terminator de linie rămas de la citirea anterioară.

Pentru ca un program să citească corect un singur caracter de la tastatură avem mai multe soluții:

- apelarea funcției fflush(stdin) înainte de oricare citire; această funcție goleşte zona buffer asociată tastaturii şi este singura posibilitate de acces la această zonă tampon.
- introducerea unei citiri false care să preia terminatorul de linie din buffer (cu *getchar*(), de exemplu).
- utilizarea funcției nestandard *getch* (declarată în *conio.h*), funcție care nu foloseşte o zonă buffer la citire
- citirea unui singur caracter ca un şir de lungime 1:

```
char c[2];        // memorie ptr un caracter şi pentru terminator de şir
scanf ("%1s",c);  // citeşte şir de lungime 1
```

Unele medii integrate închid automat fereastra în care se afişează rezultatele unui program. Pentru menținerea rezultatelor pe ecran vom folosi fie o citire falsă (pune programul în aşteptare) fie instrucțiunea: system(“pause”); din stdlib.h.

Fluxuri de intrare/ieşire in C++

În C++ s-a introdus o altă posibilitate de exprimare a operațiilor de citire-scriere, pe lângă funcțiile standard de intrare-ieşire din limbajul C. În acest scop se folosesc câteva clase predefinite pentru fluxuri de I/O (declarate în fişierele antet *iostream.h* şi *fstream.h*).
Un flux de date (*stream*) este un obiect care conține datele şi metodele necesare operațiilor cu acel flux. Pentru operații de I/O la consolă sunt definite variabile de tip flux, numite *cin* (console input) respectiv *cout* (console output).
Operațiile de citire sau scriere cu un flux pot fi exprimate prin metode ale claselor flux sau prin doi operatori cu rol de extractor din flux (>>) sau insertor în flux (<<). Atunci când primul operand este de un tip flux, interpretarea acestor operatori nu mai este cea de deplasare binară ci este extragerea de date din flux (>>) sau introducerea de date în flux (<<).
Operatorii << şi >> implică o conversie automată a datelor între forma internă (binară) şi forma externă (şir de caractere). Formatul de conversie poate fi controlat prin cuvinte cheie cu rol de "modificator".
Exemplu de scriere şi citire cu format implicit:

```
#include <iostream.h>
void main ( ) {
int n; char s[20];
cout << " n= ";  cin >> n;
cout << " un şir: "; cin >> s;  cout << s << "\n";
}
```

Într-o expresie ce conține operatorul << primul operand trebuie să fie *cout* (sau o altă variabilă de un tip *ostream*), iar al doilea operand poate să fie de orice tip aritmetic sau de tip *char** pentru afişarea şirurilor de caractere. Rezultatul expresiei fiind de tipul primului operand, este posibilă o expresie cu mai mulți operanzi (ca la atribuirea multiplă).
Exemplu:

```
cout << "x= " << x << "\n";
```

În mod similar, într-o expresie ce conține operatori >> primul operand trebuie să fie *cin* sau de un alt tip *istream*, iar ceilalți operanzi pot fi de orice tip aritmetic sau pointer la caractere.
Exemplu:

```
cin >> x >> y;
```

Este posibil şi un control al formatului de scriere prin utilizarea unor *modificatori*, însă nu vom detalia aici caest aspect, deoarece nu vom folosi aceste facilități care țin de C++, ele fiind date doar ca titlu informativ.