Programarea Calculatoarelor

Cursul 1: Concepte introductive. Tipuri de date. Funcții de intrare/ieșire

Ion Giosan

Universitatea Tehnică din Cluj-Napoca

Departamentul Calculatoare

Cadre didactice

- Curs
  - Ion Giosan
    - E-mail: Ion.Giosan@cs.utcluj.ro
    - Web page: http://users.utcluj.ro/~igiosan/teaching_pc.html
- Laborator
  - Eric Toader (grupa 1)
  - Amalia Nemes (grupa 2)
  - Ciprian Morosanu (grupa 3)
  - Rareș Popa (grupa 4)
  - Radu Drăgan (grupa 5)
  - Ion Giosan (master CSC)
- Seminar
  - Ion Giosan (grupele 1, 2, 3, 5)
  - Ciprian Morosanu (grupa 4)
  - Radu Drăgan (master CSC)

Conținutul cursului (1)

- Concepte introductive. Scheme logice și pseudocod. Primul program C. Tipuri de date. Funcții de intrare/ieșire
- Expresii și operatori
- Instrucțiuni
- Funcții
- Preprocesarea în C. Programarea modulară. Tipuri de variabile
- Pointeri (I). Declarare. Pointeri constanți. Pointeri și tablouri. Operații cu pointeri. Pointeri ca argument și valoare returnată
- Pointeri (II). Pointeri la pointeri. Alocarea dinamică a memoriei. Tablouri alocate dinamic. Pointeri la funcții

Conținutul cursului (2)

- Recursivitate
- Șiruri de caractere
- Tipurile de date structură, uniune și enumerare. Nume simbolice pentru tipuri de date
- Fișiere text. Fișiere binare. Funcții de prelucrare a fișierelor. Argumente la execuția programelor
- Recapitulare
- Pregătire pentru examen

Evaluare

- Nota finală la disciplina Programarea Calculatoarelor
  - 50% nota examen scris în sesiune
    - Obligatoriu >=5
  - 40% nota la laborator
    - Obligatoriu >=5
  - 10% teste date în timpul cursurilor
  - Bonusuri
    - Activitate seminar
    - Prezență sporită la cursuri

Etapele rezolvării unei probleme

- Definirea și analiza problemei
  - Enunțul clar, precis al problemei de rezolvat
  - Specificarea cerințelor și a datelor de intrare și ieșire
- Proiectarea algoritmului
  - Stabilirea metodei de rezolvare pas cu pas
- Implementarea programului
  - Codificarea într-un limbaj de programare
  - Depanarea programului
- Testarea și validarea programului
  - Rulează și se comportă conform așteptărilor
  - Oferă rezultate conform cerințelor stabilite
- Întreținerea programului
- Întocmirea documentației

Algoritmul

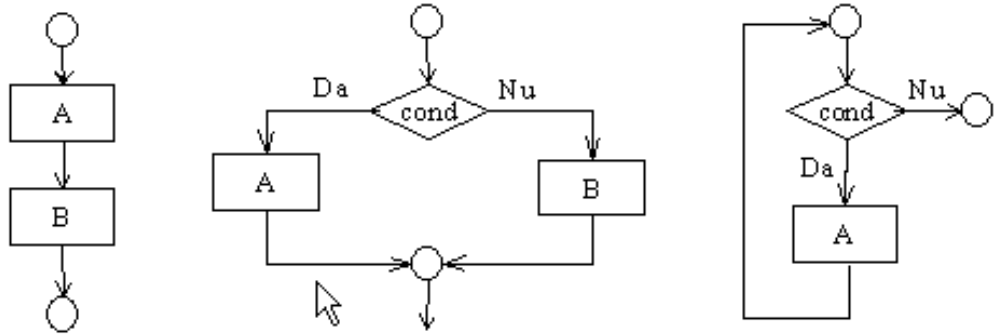
- Este un concept folosit pentru a desemna o mulțime finită de operații, complet ordonată în timp, care pornind de la date de intrare produce într-un timp finit date de ieșire
- Redă metoda de rezolvare a unei probleme într-un număr finit de paşi
- Proprietăți ale datelor
  - Domeniu finit al valorilor de intrare
  - Interval finit al valorilor de ieșire
- Cerințe generale ale algoritmului
  - Se termină după un număr finit de pași
  - Fiecare pas este clar definit
  - Timpul de rulare și dimensiunea memoriei folosite trebuie să fie cât mai mici
  - Trebuie să fie cât mai universal

Algoritmul - exemplu

- Enunțul problemei:
  Calculați și afișați suma cifrelor unui număr natural dat.
  Dacă numărul dat este negativ se va afișa 0.
- Pașii algoritmului
  1. Citește un număr natural ***n***
  2. Asignează sumei ***s*** valoarea inițială zero
  3. Dacă ***n*** este mai mic sau egal cu zero mergi la pasul 8
  4. Determină valoarea ultimei cifre ***c*** a lui ***n*** prin calculul restului împărțirii lui ***n*** la 10
  5. Adună sumei ***s*** valoarea lui ***c***
  6. Elimină ultima cifră a lui ***n*** prin împărțirea lui la 10 și păstrarea doar a câtului
  7. Mergi la pasul 3
  8. Scrie valoarea sumei ***s***

Descrierea algoritmilor (1)

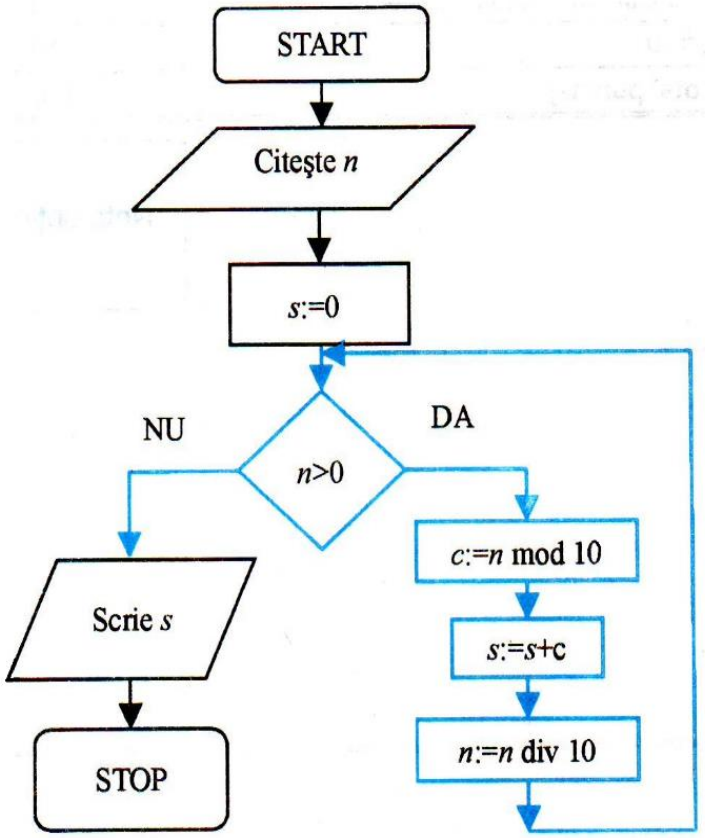
- Scheme logice
  - Reprezentare grafică
  - Regulile de calcul ale algoritmului sunt descrise prin blocuri (figuri geometrice) reprezentând operațiile (pașii) algoritmului
  - Ordinea lor de aplicare (succesiunea operațiilor) este indicată prin săgeți
  - Orice algoritm poate fi descris într-o schemă logică folosind una din următoarele trei structuri de control



(a) structura secventiala | (b) structura alternativa | (b) structura repetitiva

Descrierea algoritmilor (2)

- Scheme logice – exemplul anterior de algoritm



Operatorii
“**mod**” - *modulo* și
“**div**” - *division* calculează **restul** respectiv **câtul** împărțirii unui număr întreg la alt număr întreg

Regula de calcul:
a mod n = a – n * (a div n)

Exemple:
19 div 6 = 3
19 mod 6 = 1
-26 div 6 = -4
-26 mod 6 = -2

Descrierea algoritmilor (3)

- Limbajul Pseudocod
  - Este format din propoziții asemănătoare propozițiilor limbajului natural, care corespund structurilor de calcul folosite în construirea algoritmilor
  - Execuția unui algoritm descris în Pseudocod
    - Efectuarea operațiilor precizate de propozițiile algoritmului, în ordinea citirii lor
  - Propozițiile standard ale limbajului Pseudocod
    - Propozițiile simple sunt: CITEŞTE, SCRIE, atribuire și apelul de subprogram
    - Propozițiile compuse corespund structurilor alternative și repetitive
  - Structurile de control
    - Structura secvențială este redată prin concatenarea propozițiilor, simple sau compuse, ale limbajului Pseudocod, care vor fi executate în ordinea întâlnirii lor în text
    - Structura alternativă este redată în Pseudocod prin propoziția DACĂ
    - Structura repetitivă este redată în Pseudocod prin propoziția CÂTTIMP

Descrierea algoritmilor (4)

- Limbajul Pseudocod – exemplul anterior de algoritm

```
START
    CITEȘTE n
    s:=0
    CÂTTIMP n>0  execută
            c:=n mod 10
            s:=s+c
            n:=n div 10
    SFCÂT
    SCRIE s
STOP
```

Programarea, programul și limbajul de programare

- **Programarea** este activitatea de elaborare a unui produs program și presupune
  - Descrierea algoritmilor
  - Codificarea algoritmilor într-un anumit limbaj de programare
- **Programul** este reprezentarea unui algoritm într-un limbaj de programare și presupune
  - Descrierea datelor
  - Instrucțiuni de procesare
- **Limbajul de programare** este o notație utilizată pentru scrierea programelor și presupune
  - O sintaxă specifică
  - Utilizarea de cuvinte rezervate care au o semantică bine definită

Limbaje de programare

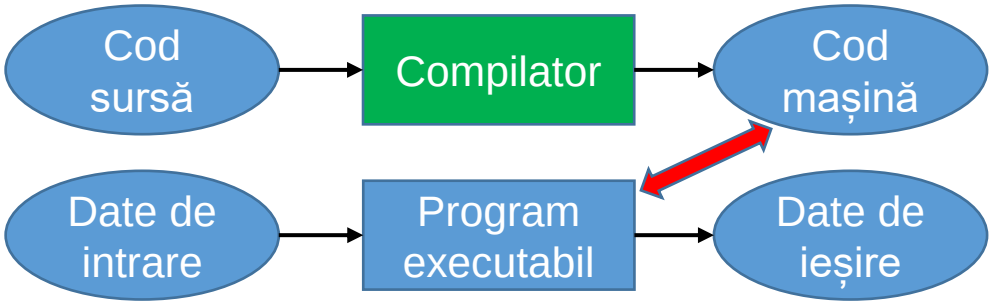
- De nivel scăzut
  - Oferă o abstractizare foarte redusă sau chiar deloc față de arhitectura setului de instrucțiuni a procesorului din sistemul de calcul
  - Exemplu: ASM – limbajul de asamblare
- De nivel înalt
  - Oferă o abstractizare ridicată prin utilizarea de elemente provenite din limbajul natural
  - Face ca scrierea unui program să fie mult mai ușoară și mai clară
  - Exemple: C, C++, JAVA, etc.

Paradigme de programare

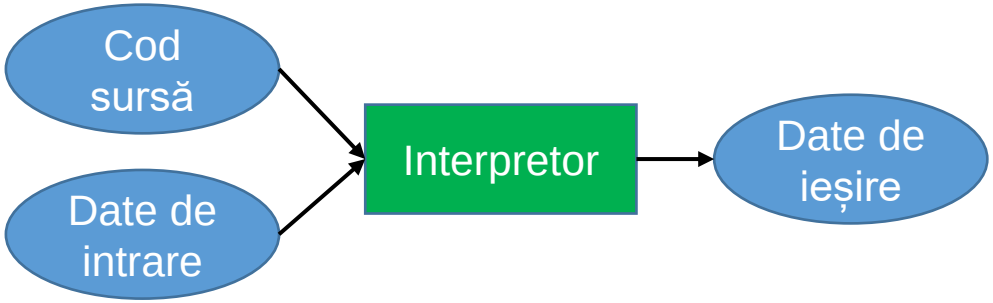
- Programarea imperativă – descrie exact secvența de comenzi care urmează a fi executată
  - Programare structurată, programare procedurală
    - FORTRAN, PASCAL, C
  - Programare orientată pe obiecte
    - C++, C#, JAVA
- Programarea declarativă – programul descrie doar ce trebuie să facă nu și cum
  - Programare funcțională
    - LISP, ML, Haskell
  - Programare logică
    - Prolog

Compilatoare vs. Interpretoare

- **Compilatorul** analizează programul și îl translatează în cod mașină
- Programul executabil poate fi executat independent de compilator de câte ori se dorește (execuție cu viteză ridicată!)



- **Interpretorul** analizează și execută instrucțiunile în același timp (execuția este mai lentă dar permite depanarea cu ușurință!)



Limbajul de programare C

- Dezvoltat de Dennis Ritchie la Bell Labs între anii 1969-1973
  - Dezvoltarea limbajului de programare C a fost asociată cu dezvoltarea sistemului de operare Unix
- Utilizat cel mai adesea pentru scrierea programelor eficiente și portabile
  - Sisteme de operare, compilatoare, interpretoare și alte produse software unde viteza de execuție este foarte importantă
- Limbaj popular, rapid și independent de platformă (portabil)
- Limbaj imperativ, structurat, compilat și scurt (număr redus de cuvinte cheie)
- Limbaj permisiv, dificil de înțeles și de modificat
  - Permite introducerea de erori greu de depistat în favoarea vitezei de execuție
  - Stilul de programare utilizat și documentarea codului scris sunt foarte importante

Standarde oficiale ale limbajului C

- C89/C90
  - Aprobat în 1989 de ANSI (American National Standards Institute) și în 1990 de către ISO (International Organization for Standardization)
  - Cunoscut sub numele de ANSI C
- C99
  - Aprobat în 1999
  - Include corecturile aduse C89/C90 dar și o serie de caracteristici proprii (ex. tipul de date long long, comentarii pe o singură linie, posibilitatea de a mixa declarațiile cu instrucțiunile de cod, etc.)
- C11
  - Aprobat în 2011
  - Rezolvă problemele C99 și introduce noi elemente (suport pentru Unicode, API pentru *multi-threading*, tipuri de date atomice etc.)
- C18
  - Aprobat în 2018
  - Rezolvă problemele C11 fără a introduce noi elemente

Cuvinte cheie în C

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Structura programelor C

- Un program C este compus din una sau mai multe funcții
  - Una singură este obligatorie: funcția **main**
  - Celelalte funcții sunt funcții definite opțional de către programator

- Structura de bază a unui program C conține:
  - Directive de preprocesare
  - Definiții de tipuri
  - Prototipuri de funcții
  - Variabile
  - Funcții
    - Obligatoriu funcția **main** !

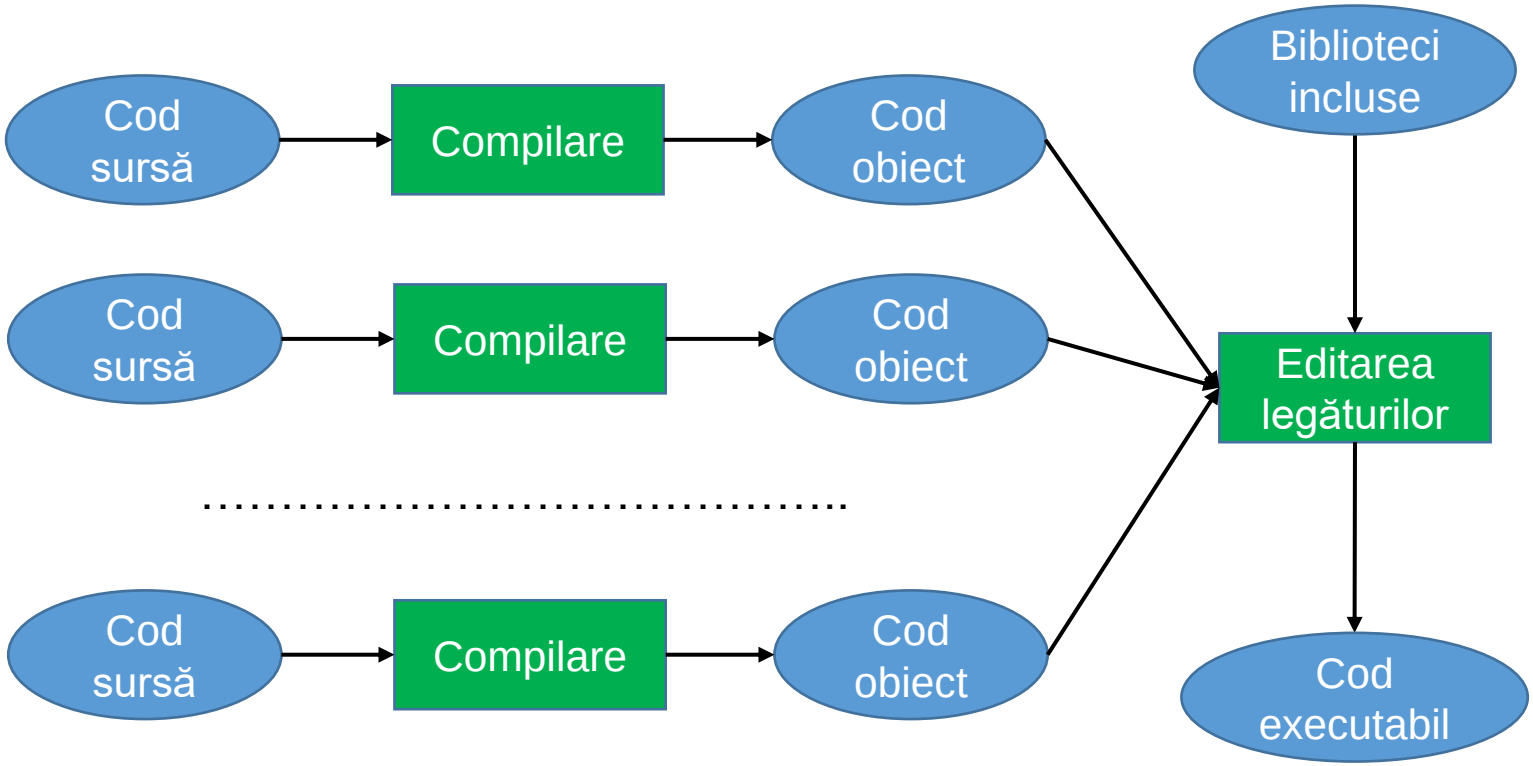
Formatul programelor C

- Instrucțiunile sunt terminate cu caracterul punct și virgulă
- Mai multe instrucțiuni pot fi scrise pe aceași linie
- Indentarea codului scris este recomandată pentru organizarea și citirea ușoară a acestuia
- Spațiile albe și indentarea sunt ignorate de către compilator
- Limbajul C este *case sensitive*
  - Se face diferența între litere mari și litere mici
- Toate cuvintele cheie se scriu cu litere mici

Compilarea și rularea programelor C (1)

- Editarea codului sursă
  - Salvarea fișierului scris cu extensia **.c**
- Preprocesarea
  - Efectuarea directivelor de preprocesare
  - Includerea fișierelor header (cu extensia **.h**) corespunzătoare bibliotecilor folosite
  - Ca un editor – modifică și adaugă la codul sursă
- Compilarea
  - Verificarea sintaxei
  - Transformare în cod obiect (limbaj mașină) (fișier cu extensia **.o**)
- Editarea legăturilor (*link-editarea*)
  - Combinarea codului obiect cu alte coduri obiect (al bibliotecilor asociate fișierelor header)
  - Transformarea adreselor simbolice în adrese reale
  - Se obține fișierul executabil cu extensia **.exe**

Compilarea și rularea programelor C (2)



Primul program în C (1)

```
1  // primul program in C
2  #include <stdio.h>
3
4  int main()
5  {
6    printf("Hello World!");
7    return 0;
8  }
```

Afișează la ieșirea standard:

Hello World!

- Linia 1: Secvența de caractere **//** introduce un comentariu inserat de programator. Acesta este valabil pe întreaga linie. Textul respectiv nu va fi compilat. Dacă se dorește comentarea mai multor linii acestea se încadrează între secvențele de caractere **/*** și ***/**

- Linia 2: Liniile care încep cu caracterul # sunt directive citite și interpretate de către preprocesor. În acest caz, directiva **#include <stdio.h>** include biblioteca **stdio.h** care va permite operații cu funcții de intrare/ieșire

Primul program în C (2)

```
1   // primul program in C
2   #include <stdio.h>
3
4   int main()
5   {
6     printf("Hello World!");
7     return 0;
8   }
```

Afișează la ieșirea standard:

Hello World!

- Linia 3: Liniile libere nu au niciun efect asupra programului, fiind ignorate de către compilator

- Linia 4: Antetul funcției **main.** Funcția **main** este o funcție specială în toate programele C, fiindcă reprezintă funcția care este chemată atunci când pornește execuția programului. Execuția tuturor programelor în C încep de la funcția **main**

Primul program în C (3)

```
1   // primul program in C
2   #include <stdio.h>
3
4   int main()
5   {
6     printf("Hello World!");
7     return 0;
8   }
```

Afișează la ieșirea standard:

Hello World!

- Liniile 5 și 8: Caracterele **{** și **}** grupează mai multe instrucțiuni, formând un bloc compus de instrucțiuni. În acest caz, între acestea sunt scrise instrucțiunile din corpul funcției **main**
- Linia 6: Apelul funcției **printf** folosită pentru afișarea textului **Hello world!**. După fiecare instrucțiune se inserează semnul punct-virgulă **;** cu rol de separare a instrucțiunilor
- Linia 7: Funcția **main** returnează un cod de eroare. În acest caz valoarea zero reprezintă terminare cu succes.

Identificatori și simboluri (1)

- Un program sursă este compus din identificatori și simboluri separate prin spații albe (spațiu **' '**, TAB **'\t'**, linie nouă **'\n'**)
- Identificatori
  - Nume de constante, tipuri, variabile, funcții
  - Numele
    - Secvență de litere, cifre și simboluri *underscore*
    - Primul caracter trebuie să fie literă sau *underscore*
    - Limitat la 31 de caractere (ANSI C)
  - Exemple
    - Corecte:

      A a _a alfa a10 a_10
    - Greșite:

      A! 10alfa a*

Identificatori și simboluri (2)

- Simboluri
  - Grupuri de caractere care nu sunt identificatori
  - Exemple
    - Operatori: **+ ++ && < <= != >** etc.
    - Constante numerice: **10.8 90 0x4f**
    - Caractere: **'A' 'b' '8'**
    - Șiruri de caractere: **"Programare in limbajul C"**

Tipuri de date în C (1)

- Cinci categorii de tipuri de date fundamentale
  - **int**
    - Tipul întreg
    - Poate reține valori întregi, ex. 1, 0, -532, etc.;
  - **char**
    - Tipul caracter – poate reține un singur caracter sub forma codului elementelor din setul de caractere specific
    - Codul ASCII (American Standard Code for Information Interchange) reprezentat pe 7 biți (poate reprezenta 128 de caractere) – set care este frecvent extins la codul Latin-1, pe 8 biți care poate reprezenta 256 de caractere
  - **float**
    - Tipul real (numere în virgulă mobilă) – simplă precizie – pot reține valori mai mari decât tipul întreg și care conțin parte fracționară
    - Ex. 4971.185, -0.72561, etc.
  - **double**
    - Tipul real (numere în virgulă mobilă) în dublă precizie – pot reține valori reale în virgulă mobilă cu o precizie mai mare decât tipul float
    - Ex. 4971.16548749848
  - **void**
    - Indică lipsa unui tip anume

Tipuri de date în C (2)

- Modificatori de tipuri de date
  - **signed**
    - Modificatorul implicit pentru toate tipurile de date
    - Bitul cel mai semnificativ din reprezentarea valorii este semnul
  - **unsigned**
    - Restricționează valorile numerice memorate la valori pozitive
    - Domeniul de valori este mai mare deoarece bitul de semn este liber și participă în reprezentarea valorilor
  - **short**
    - Reduce dimensiunea tipului de date întreg la jumătate
    - Se aplică doar pe întregi
  - **long**
    - Permite memorarea valorilor care depășesc limita de stocare specifică tipului de date
    - Se aplică doar pe *int* sau *double*
  - **long long**
    - Introdus în C99 pentru a facilita stocarea unor valori întregi de dimensiuni foarte mari

Tipuri de date în C (3)

Tip	Octeți mem.	Limita inferioară	Limita superioară	Valori
char	1	-128(-2^7)	127(2^7-1)	Întregi
unsigned char	1	0	255(2^8-1)	Întregi
short int	2	-32,768(-2^{15})	32,767(2^{15}-1)	Întregi
unsigned short int	2	0	65,535(2^{16}-1)	Întregi
int	4	-2,147,483,648(-2^{31})	2,147,483,647(2^{31}-1)	Întregi
unsigned int	4	0	4,294,967,295	Întregi
long int	4	-2,147,483,648(-2^{31})	2,147,483,647(2^{31}-1)	Întregi
unsigned long int	4	0	4,294,967,295	Întregi
long long int	8	-2^{63}	2^{63}-1	Întregi*
unsigned long long int	8	0	2^{64}-1	Întregi*

* Introdus doar din standardul C99

Tipuri de date în C (4)

Tip	Octeți mem.	Limita inferioară	Limita superioară	Valori	Precizie
float	4	$\pm 1.2*10^{-38}$	$\pm 3.4*10^{38}$	Reale	6 zecimale
double	8	$\pm 2.3*10^{-308}$	$\pm 1.7*10^{308}$	Reale	15 zecimale
long double	10	$\pm 3.4*10^{-4932}$	$\pm 1.1*10^{4932}$	Reale	19 zecimale

Tabelul ASCII – cu indicii în hexazecimal (baza 16)

*	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	TAB	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	

- Exemple:
  - La codul ASCII $41_{(16)} = 65_{(10)}$ corespunde caracterul **‘A’**
  - La codul ASCII $61_{(16)} = 97_{(10)}$ corespunde caracterul **‘a’**
  - La codul ASCII $20_{(16)} = 32_{(10)}$ corespunde caracterul **‘ ’** (spațiu)
  - La codul ASCII $30_{(16)} = 48_{(10)}$ corespunde caracterul **‘0’**

Constante (1)

- O constantă are un tip și o valoare care nu poate fi schimbată de-a lungul execuției programului
- Constante întregi
  - Zecimale
    - Șir de cifre zecimale, precedate opțional de semn
    - Pentru a indica lungimea și tipul cu semn sau fără semn
      - L, l: long
      - U, u: unsigned
      - UL, ul, LU, lu: unsigned long
    - Exemple: **100**, **100L**, **100U**, **100ul**
  - Octale (numere în baza 8)
    - Încep cu 0 și conțin doar cifre în baza 8 și pot fi doar de tip fără semn
    - Exemple: **0144** (=100 în baza 10), **0176** (=126 în baza 10)
  - Hexazecimale (numere în baza 16)
    - Încep cu 0x sau 0X și conțin doar cifre în baza 16 și pot fi doar de tip fără semn
    - Exemple: **0xab1** (=2737 în baza 10), **0X2F** (=47 în baza 10)

Constante (2)

- Constante caractere
  - Încep și se termină cu caracterul apostrof **'**
  - Au de fapt tip întreg
  - Au codul ASCII al caracterului ca și valoare
  - Caractere tipăribile
    - Între codurile ASCII 32-126
    - Exemple: **'A'**, **'y'**
  - Secvențe *Escape*
    - Exemple: **'\n'**, **'\t'**

Secvența *Escape*	**Reprezintă**
\a	*Alert (ANSI C)*
\b	*Backspace*
\f	*Form feed*
\n	*Newline*
\r	*Carriage return*
\t	*Horizontal TAB*
\v	*Vertical TAB*
****	*Backslash (\)*
\'	*Single quote (')*
\"	*Double quote (")*
\?	*Question mark (?)*
\0oo	Valoare octală. (**o** reprezintă o cifră în baza 8)
\xhh	Valoare hexazecimală. (**h** reprezintă o cifră în baza 16)

Constante (3)

- Constante șiruri de caractere
  - Secvență de caractere care începe și se termină cu caracterele ghilimele "
  - Exemple:
    - **"sir de caractere"**
    - **"apostrof ' reprezentat clasic"**
    - **"compilator \"C\""**
  - O constantă șir de caractere se poate scrie pe mai multe rânduri; în aceste cazuri la sfârșitul rândurilor care au continuare trebuie să se insereze caracterul ****
    - Exemplu: **"constanta pe mai **
      multe randuri"
  - Reprezentarea în memorie

0	1	2	...	n-1	n
Cod ASCII	Cod ASCII	Cod ASCII	...	Cod ASCII	'\0'

Declararea variabilelor

- Pentru variabile simple

```
tip identifcator {, identificator };
```

  - Exemple:

```
int i, j, k;
char c;
double x, y;
```

- Pentru variabile tablou

```
tip_baza identificator[lim] {[lim] }
        {, identificator[lim] {[lim] }};
```

  - Indicii sunt de la `0` la `lim-1` inclusiv
  - Limitele sunt expresii constante, evaluate în timpul compilării
  - Numele unui tablou reprezintă adresa primului său element
  - Exemple de declarări de tablouri:

```
int vector[100];          //tablou unidimensional
double matrice[10][15]; //tablou bidimensional
```

Inițializarea variabilelor (1)

- Pentru variabile simple

```
tip identifcator = expresie;
```

  - Exemple:

```
char c = 'B';
double x = 50 + 25.84;  //se evaluează expresia
```

- Pentru variabile tablou
  - Nu este necesară inițializarea tuturor elementelor acestora
  - Elementele tablourilor declarate și neinițializate
    - Sunt setate automat zero dacă tabloul este variabilă globală sau statică
    - Au valori nedefinite în cazul variabilelor locale (automate)
  - Elementele tablourilor declarate și inițializate parțial sunt setate automat la zero în toate situațiile
  - Tablou unidimensional declarat și inițializat

```
tip_baza identificator[lim] = {v0, v1, … , vn};
```

    - Exemplu

```
int a[10] = {20, 15, -352};
                                  // a[3],…,a[9] sunt toate 0
```

Inițializarea variabilelor (2)

- Pentru variabile tablou
  - Tablou bidimensional declarat și inițializat

```
tip_baza identificator[lim1][lim2] =
  {
     {v00, v01,..., v0n},
     {v10, v11 ,..., v1m},
     ...
     {vi0, vi1 ,..., vik}
  };
```

  - Exemplu

```
int mat[5][5] =
     {
        {1, 2, 3},
        {5, 6, 7, 8},
        {9, 10, 11},
     };                          // mat[0][3] este 0
                                 // mat[4][4] este 0
```

Funcții de intrare/ieșire

- Terminalul standard este terminalul folosit pentru execuția unui program. Există trei fișiere standard atașate acestuia:
  - Intrarea standard (**stdin**)
    - Implicit de la tastatură
  - Ieșirea standard (**stdout**)
    - Implicit pe ecran
  - Ieșirea standard pentru erori (**stderr**)
    - Implicit pe ecran
- Limbajul C nu are instrucțiuni pentru citire/scriere
- Operațiile de citire/scriere se realizează prin intermediul funcțiilor care au prototipurile declarate în biblioteca **stdio.h**
  - Biblioteca **conio.h** este o extensie, nefăcând parte din standard

Funcții de intrare/ieșire pentru caractere (1)

- **getch**

```
int getch(void);
```

  - Citește fără a afișa un caracter de la intrarea standard (**stdin**)
  - Citirea se face imediat, fără apăsarea tastei ENTER
  - Funcția returnează codul ASCII al caracterului citit

- **getche**

```
int getche(void);
```

  - Citește un caracter de la intrarea standard (**stdin**) și îl scrie la ieșirea standard (**stdout**)
  - Citirea se face imediat, fără apăsarea tastei ENTER
  - Funcția returnează codul ASCII al caracterului citit

- **putch**

```
int putch(int ch);
```

  - Scrie un caracter primit ca și parametru la ieșirea standard (**stdout**)
  - Funcția returnează codul ASCII al caracterului scris sau EOF în caz de eșec (EOF este în general valoarea -1)

Funcții de intrare/ieșire pentru caractere (2)

- **Exemplu**

```
#include <conio.h>

int main(void)
{
   putch('A'); // afiseaza caracterul 'A'
   putch(66); // 'B' este pe pozitia 66 in tabelul ASCII
   putch(97); // 'a' este pe pozitia 97 in tabelul ASCII
   char ch=getch(); //citeste un caracter fara a-l afisa
   putch(ch); //afiseaza caracterul citit
   ch=getche(); //citeste si afiseaza un caracater
   putch(ch); //afiseaza inca o data caracterul citit
   return 0;
}
```

Funcții de intrare/ieșire pentru șiruri de caractere (1)

- **gets**

```
char *gets(char *s);
```

  - Citește un șir de caractere de la intrarea standard (**stdin**) până la întâlnirea caracterului *newline* și îl memorează în șirul de caractere primit ca și argument
  - Caracterul *newline* este automat exclus din șirul memorat
  - Funcția returnează argumentul citit în caz de succes, iar în caz de eroare funcția returnează un pointer NULL
  - Funcția trebuie apelată cu grijă întrucât nu are nicio protecție împotriva citirii unui șir de caractere de dimensiune mai mare decât cea alocată
    - Se recomandă în locul acesteia utilizarea funcțiiei **fgets** care să citească din fișierul de intrare standard **stdin**

Funcții de intrare/ieșire pentru șiruri de caractere (2)

- **puts**

```
int puts(const char *s);
```

  - Scrie șirul de caractere dat ca și argument la ieșirea standard (**stdout**) urmat de caracterul *newline*
  - Caracterul terminal NULL al șirului de caractere nu este scris
  - Funcția este utilă pentru tipărirea mesajelor simple
  - Funcția returnează o valoare nenegativă în caz de succes sau EOF (-1) în caz de eșec

Funcții de intrare/ieșire pentru șiruri de caractere (3)

- **Exemplu**

```
#include <stdio.h>
int main()
{
   char s[201];
   puts("Introduceti un sir de maximum 200 de
   caractere si apoi apasati Enter:");
   gets(s); /* sirul de caractere este stocat in s */
   puts("Sirul de caractere introdus este:");
   puts(s); /* scrie la iesirea standard sirul s */
   return 0;
}
```

Funcții de intrare/ieșire pentru citire/scriere cu format (1)

- **scanf**

```
int scanf(const char *format, {adresa});
```

  - Citește cu format caractere de la intrarea standard (**stdin**) într-un mod controlat de șirul de caractere (formatul) trimis ca și prim argument
  - Șirul de caractere trimis ca și prim argument poate conține:
    - **Caracter spațiu**: funcția citește și ignoră niciunul, unul sau mai multe spații albe (spațiu, tab, linie nouă) înaintea următorului caracter diferit de spațiu din textul de la intrare
    - **Caracter diferit de spațiu**: funcția citește următorul caracter de la intrare și îl compară cu caracterul specificat în șirul de formatare; dacă se potrivește se trece la citirea următorului caracter, dacă nu funcția eșuează și următoarele caractere nu sunt citite
    - **Specificatori de format** - forma generală a unui specificator de format este

```
% indicator dimensiune modificator_tip conversie
```

  - Următoarele argumente sunt opționale și sunt adresele de memorie unde vor fi stocate valorile citite (variabilele de intrare)
  - Funcția returnează numărul de valori citite corect (numărul de argumente care s-au potrivit cu specificatorii de format) sau EOF (-1) în cazul în care apare o eroare de citire (din orice cauză)

Funcții de intrare/ieșire pentru citire/scriere cu format (2)

- **scanf**
  - Specificatori de format - încep obligatoriu cu caracterul **%** urmat de
    - Opțional un **indicator** reprezentat de caracterul ***** care determină ignorarea (se citește dar nu se stochează niciunde) textului introdus pentru specificatorul respectiv
    - Opțional un întreg zecimal reprezentând **dimensiunea maximă** (în număr de caractere) a textului care poate fi citit cu specificatorul respectiv
      - Citirea caracterelor se oprește fie atunci când dimensiunea maximă este atinsă fie când se întâlnește un caracter care nu se potrivește cu specificatorul respectiv
      - Majoritatea conversiilor ignoră caracterele spații albe (spațiu, TAB, linie nouă, etc.) aflate la începutul textului, acestea nefiind contabilizate în calculul dimensiunii maxime
      - Conversiile la tipul șir de caractere memorează la sfârșitul textului caracterul NULL ('\0'), acesta nefiind socotit în calculul dimensiunii maxime
    - Opțional unul sau două caractere cu rol de **modificator de tip**
    - Un caracter care specifică **conversia** care urmează a fi aplicată (care convertește textul citit și îl stochează sub forma unei valori într-o variabilă de un anumit tip)

Funcții de intrare/ieșire pentru citire/scriere cu format (3)

- **scanf**
  - Conversii posibile prin specificatori de format:

%d	Potrivește cu un întreg zecimal
%i	Potrivește cu un întreg în oricare format pe care limbajul C îl definește pentru o constantă întreagă
%o	Potrivește cu un întreg fără semn scris în octal (baza 8)
%u	Potrivește cu un întreg zecimal fără semn
%x, %X	Potrivește cu un întreg fără semn scris în hexazecimal (baza 16)
%f, %e, %g, %E, %G, %a	Potrivește cu un număr real cu semn în simplă precizie (**float**)
%p	Potrivește cu o valoare a unui pointer

Funcții de intrare/ieșire pentru citire/scriere cu format (4)

- **scanf**
  - Conversii posibile prin specificatori de format:

%s	Potrivește cu un șir de caractere care nu conține caractere spații albe (spațiu, TAB, linie nouă, etc.)
%[ ]	Potrivește cu un șir de caractere cu caractere aparținând unui anumit set de caractere specificat între parantezele drepte
%[^]	Potrivește cu un șir de caractere cu caractere care nu aparțin unui anumit set de caractere specificat între parantezele drepte după **^**
%c	Potrivește cu unul sau mai multe caractere; numărul de caractere este controlat de câmpul **dimensiune** din specificatorul de format (dacă acesta lipsește se va potrivi cu exact 1 caracter)
%n	Nu citește nici un carcater; numărul de caractere citite de **scanf** până în momentul respectiv sunt scrise la adresa corespunzătoare lui
%%	Potrivește doar cu caracterul **%**

Funcții de intrare/ieșire pentru citire/scriere cu format (5)

- **scanf**
  - Modificatori de tip utilizați în conversii posibile:

%lf	Potrivește cu o valoare de tip **double**
%Lf	Potrivește cu o valoare de tip **long double**
%hd	Potrivește cu un număr întreg zecimal de tip **short int**
%ld	Potrivește cu un număr întreg zecimal de tip **long int**
%lld	Potrivește cu un număr întreg zecimal de tip **long long int** (introdus doar în C99)
%hu	Potrivește cu un număr întreg zecimal fără semn de tip **unsigned short int**
%lu	Potrivește cu un număr întreg zecimal fără semn de tip **unsigned long int**
%llu	Potrivește cu un număr întreg zecimal fără semn de tip **unsigned long long int** (introdus doar în C99)

Funcții de intrare/ieșire pentru citire/scriere cu format (6)

- **Exemple**
  - Citirea unui caracter

```
char ch;
scanf("%c", &ch);
```

  - Citirea unui șir de caractere

```
char s[40];
scanf("%s", s);
```

  - Citirea a trei întregi cu valori în zecimal, octal și hexazecimal

```
int a, b, c;
scanf("%d %o %x", &a, &b, &c);
```

  - Citirea a trei numere reale de tip **float**, **double** și **long double**

```
float x;
double y;
long double z;
scanf("%f %lf %Lf", &x, &y, &z);
```

Funcții de intrare/ieșire pentru citire/scriere cu format (7)

- **printf**

```
int printf(const char *format,{expresii});
```

  - Scrie toate argumentele din lista de expresii la ieșirea standard (**stdout**) într-un mod controlat de șirul de caractere (formatul) trimis ca și prim argument

  - Șirul de caractere trimis ca și prim argument conține specificatori de format
    - Forma generală a unui specificator de format este

```
% indicator dimensiune precizie modificator_tip conversie
```

  - Funcția returnează numărul de caractere tipărite în caz de succes sau o valoare negativă în cazul apariției unei erori de scriere

Funcții de intrare/ieșire pentru citire/scriere cu format (8)

- **printf**
  - Specificatori de format - încep obligatoriu cu caracterul **%** urmat de
    - Opțional **indicatori** (niciunul sau mai mulți) care modifică comportamentul normal al specificației conversiei
    - Opțional un întreg zecimal reprezentând **dimensiunea minimă** (în număr de caractere) a câmpului în care se va scrie valoarea cu specificatorul respectiv
      - Este o valoare minimă. Dacă sunt necesare mai multe caractere pentru scriere atunci câmpul nu va fi trunchiat. Scrierea se face aliniată la dreapta în cadrul câmpului respectiv
      - Se poate specifica o dimensiune *****. În acest caz argumentul precedent din lista de argumente este utilizat ca și dimensiune a câmpului curent
    - Opțional unul sau mai multe caractere cu rol de **precizie** care specifică numărul de zecimale la scrierea valorilor numerice
      - Constă în caracterul punct **.** urmat opțional de un întreg zecimal
      - Se poate specifica o precizie *****. În acest caz argumentul precedent din lista de argumente este utilizat ca și precizie pentru câmpul curent
      - Se poate specifica ***** atât pentru **dimensiune** cât și pentru **precizie**. Ordinea argumentelor precedente care definesc pe acestea sunt în ordine: primul pentru dimensiune iar cel de-al doilea pentru precizie

Funcții de intrare/ieșire pentru citire/scriere cu format (9)

- **printf**
  - Specificatori de format – continuare
    - Opțional unul sau două caractere cu rol de **modificator de tip**
      - Asemănători celor de la **scanf**
    - Un caracter care specifică **conversia** care urmează a fi aplicată (care convertește valoarea stocată pe care o scrie la ieșirea standard într-un anumit format)
      - Asemănători celor de la **scanf**

Funcții de intrare/ieșire pentru citire/scriere cu format (10)

- **printf**
  - Indicatori pentru numere (întregi și reale)

-	Scrie valoarea în cadrul câmpului aliniată la stânga (în locul alinierii implicite la dreapta)
+	Scrie semul plus dacă valoarea este pozitivă
#	Scrie **0** ca și prim caracter pentru o valoare reprezentată în octal și respectiv **0x** pentru o valoare reprezentată în hexazecimal. Pentru valori reale indică faptul că punctul zecimal va fi prezent chiar dacă nu urmează nicio cifra zecimală după acesta
0	Completează câmpul cu caractere zero în locul spațiilor. Indicatorul este ignorat dacă este specificat și indicatorul **-**

Funcții de intrare/ieșire pentru citire/scriere cu format (11)

- **printf**
  - Conversii posibile prin specificatori de format:

%d %i	Scrie un întreg ca și un număr zecimal cu semn
%u	Scrie un întreg ca și un număr zecimal fără semn
%o	Scrie un întreg ca și un număr octal (în baza 8) fără semn
%x %X	Scrie un întreg ca și un număr hexazecimal (în baza 16) fără semn. **%x** utilizează litere mici, iar **%X** litere mari
%f	Scrie un număr real în notația normală (obișnuită). Nu contează dacă este de tip **float**, **double** sau **long double**
%e %E	Scrie un număr real în notația cu bază și exponent (puterile lui 10). **%e** utilizează litere mici, iar **%E** litere mari
%g	Scrie un număr real în notația scurtă

Funcții de intrare/ieșire pentru citire/scriere cu format (12)

- **printf**
  - Conversii posibile prin specificatori de format:

%a %A	Scrie un număr real în notația hexazecimală fracțională. **%a** utilizează litere mici iar **%A** litere mari
%c	Scrie un singur caracter
%s	Scrie un șir de caractere
%p	Scrie valoarea unui pointer
%n	Nu scrie nimic. Memorează numărul de caractere afișate până în acest moment în variabila corespunzătoare specificatorului
%%	Scrie doar caracterul **%**

Funcții de intrare/ieșire pentru citire/scriere cu format (13)

- **printf**
  - Modificatori de tip la specificatori de format

%Lf	Scrie o valoare de tip **long double**
%hd	Scrie un număr întreg zecimal de tip **short int**
%ld	Scrie un număr întreg zecimal de tip **long int**
%lld	Scrie un număr întreg zecimal de tip **long long int** (introdus doar în C99)
%hu	Scrie un număr întreg zecimal fără semn de tip **unsigned short int**
%lu	Scrie un număr întreg zecimal fără semn de tip **unsigned long int**
%llu	Scrie un număr întreg zecimal fără semn de tip **unsigned long long int** (introdus doar în C99)

Funcții de intrare/ieșire pentru citire/scriere cu format (14)

- **Exemplu – scriere numere întregi**
- Valorile tipărite pe rând cu diferite formate sunt: 0, 1, -1, 100000
- Specificator de format

```
"|%5d|%-5d|%+5d|%+-5d|% 5d|%05d|%5.0d|%5.2d|%d|\n"
```

- Rezultat tipărit

```
|    0|0    |   +0|+0   |    0|00000|     |   00|0|
|    1|1    |   +1|+1   |    1|00001|    1|   01|1|
|   -1|-1   |   -1|-1   |   -1|-0001|   -1|  -01|-1|
|100000|100000|+100000|+100000| 100000|100000|100000|100000|100000|
```

- Specificator de format

```
"|%5u|%5o|%5x|%5X|%#5o|%#5x|%#5X|%#10.8x|\n"
```

- Rezultat tipărit

```
|    0|    0|    0|    0|    0|    0|    0|  00000000|
|    1|    1|    1|    1|   01|  0x1|  0X1|0x00000001|
|100000|303240|186a0|186A0|0303240|0x186a0|0X186A0|0x000186a0|
```

Funcții de intrare/ieșire pentru citire/scriere cu format (15)

- **Exemplu – scriere numere reale**
- Valorile tipărite pe rând cu diferite formate sunt: 0, 0.5, 1, -1, 100, 1000, 10000, 12345, 100000, 123456
- Specificator de format

```
"|%13.4a|%13.4f|%13.4e|%13.4g|\n"
```

- Rezultat tipărit

```
|   0x0.0000p+0|        0.0000|    0.0000e+00|             0|
|   0x1.0000p-1|        0.5000|    5.0000e-01|           0.5|
|   0x1.0000p+0|        1.0000|    1.0000e+00|             1|
|  -0x1.0000p+0|       -1.0000|   -1.0000e+00|            -1|
|   0x1.9000p+6|      100.0000|    1.0000e+02|           100|
|   0x1.f400p+9|     1000.0000|    1.0000e+03|          1000|
|  0x1.3880p+13|    10000.0000|    1.0000e+04|          1e+04|
|  0x1.81c8p+13|    12345.0000|    1.2345e+04|     1.234e+04|
|  0x1.86a0p+16|   100000.0000|    1.0000e+05|          1e+05|
|  0x1.e240p+16|   123456.0000|    1.2346e+05|     1.235e+05|
```

Exemplu de program care utilizează *scanf* și *printf* (1)

```
#include <stdio.h>
int main()
{
    char a='A';
    int b=30;
    float c=74.588f;
    double d=-457.4578;
    char e[50]="primul curs de PC";

    printf("a este caracterul %c si are codul ASCII\
           %d\n",a,a);
    printf("b are valoarea zecimala %d; octala %o;\
           hexazecimala %x\n",b,b,b);
    printf("c are valoarea %f; in format scurt %g\n",c,c);
    printf("d are valoarea %f; in format scurt %g; rotunjit\
           cu doua zecimale %.2f\n",d,d,d);
    printf("e este sirul de caractere: %s\n",e);
```

Exemplu de program care utilizează *scanf* și *printf* (2)

```
    printf("Introduceti nume, nota la BAC, nota la admitere,\
            seria si grupa separate prin spatii\n");
    int n_arg=scanf("%s %f %lf %c %d",e,&c,&d,&a,&b);
    int n_car=printf("S-au citit corect %d\
                      argumente!\n",n_arg);

    printf("S-au afisat cu functia printf anterioara %d\
            caractere!\n",n_car);
    printf("Valorile variabilelor a,b,c,d,e sunt: %c %d %f\
            %f %s",a,b,c,d,e);

    return 0;
}
```

Exemplu de program care utilizează *scanf* și *printf* (3)

Rezultate afișate – cazul de test nr. 1:

```
a este caracterul A si are codul ASCII 65
b are valoarea zecimala 30; octala 36; hexazecimala 1e
c are valoarea 74.587997; in format scurt 74.588
d are valoarea -457.457800; in format scurt -457.458;
rotunjit cu doua zecimale -457.46
e este sirul de caractere: primul curs de PC
Introduceti nume, nota la BAC, nota la admitere, seria si
grupa separate prin spatii
alex 9.45 9.27 B 30219
S-au citit corect 5 argumente!
S-au afisat cu functia printf anterioara 31 caractere!
Valorile variabilelor a,b,c,d,e sunt: B 30219 9.450000
9.270000 alex
```

Exemplu de program care utilizează *scanf* și *printf* (4)

Rezultate afișate – cazul de test nr. 2:

```
a este caracterul A si are codul ASCII 65
b are valoarea zecimala 30; octala 36; hexazecimala 1e
c are valoarea 74.587997; in format scurt 74.588
d are valoarea -457.457800; in format scurt -457.458;
rotunjit cu doua zecimale -457.46
e este sirul de caractere: primul curs de PC
Introduceti nume, nota la BAC, nota la admitere, seria si
grupa separate prin spatii
alina 9,47 10 B 30217
S-au citit corect 2 argumente!
S-au afisat cu functia printf anterioara 31 caractere!
Valorile variabilelor a,b,c,d,e sunt: A 30 9.000000
-457.457800 alina
```

Exemplu de program care utilizează *scanf* și *printf* (5)

Rezultate afișate – cazul de test nr. 3:

```
a este caracterul A si are codul ASCII 65
b are valoarea zecimala 30; octala 36; hexazecimala 1e
c are valoarea 74.587997; in format scurt 74.588
d are valoarea -457.457800; in format scurt -457.458;
rotunjit cu doua zecimale -457.46
e este sirul de caractere: primul curs de PC
Introduceti nume, nota la BAC, nota la admitere, seria si
grupa separate prin spatii
ionut 6.85 7.42 1 noua
S-au citit corect 4 argumente!
S-au afisat cu functia printf anterioara 31 caractere!
Valorile variabilelor a,b,c,d,e sunt: 1 30 6.850000 7.420000
ionut
```

Funcții de intrare/ieșire pentru citire/scriere cu format din/în șiruri de caractere

- **sscanf**

```
int sscanf(const char *s,
           const char *format, {adresa});
```

  - Citește cu format din șirul de caractere (**s**) trimis ca și prim argument într-un mod controlat de șirul de caractere (**format**) trimis ca și al doilea argument
  - Este similară funcției **scanf**, deosebirea constă doar în sursa de unde are loc citirea: **s** în loc de **stdin**

- **sprintf**

```
int sprintf(char * str,
            const char *format,{expresii});
```

  - Scrie toate argumentele din lista de expresii în șirul de caractere (**str**) trimis ca și prim argument într-un mod controlat de șirul de caractere (**format**) trimis ca și al doilea argument
  - Este similară funcției **printf**, deosebirea constă doar în destinația unde are loc scrierea: **str** în loc de **stdout**

Exemplu de program care utilizează *sscanf* și *sprintf*

```
#include <stdio.h>
int main()
{
    char a[50]="Andrei 24 Cluj-Napoca 9.5";
    char nume[30], oras[20];
    int varsta;
    float nota;
    sscanf(a,"%s %d %s %f", nume,&varsta,oras,&nota);
    char text[100];
    sprintf(text,"%s are %d ani, este din %s si a obtinut \
                  nota %.2f",nume,varsta,oras,nota);
    puts(text);
    return 0;
}
```

Rezultate afișate:

```
Andrei are 24 ani, este din Cluj-Napoca si a obtinut nota 9.50
```