



# **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](mailto:Ion.Giosan@cs.utcluj.ro)
    - Web page: [http://users.utcluj.ro/~igiosan/teaching\\_pc.html](http://users.utcluj.ro/~igiosan/teaching_pc.html)

- Laborator

- Ciprian Moroşanu (grupa 1)
  - Amalia Nemes (grupa 2)
  - Rareş Popa (grupa 3)
  - Radu Drăgan (grupa 4)
  - Horaţiu Florea (grupa 5)
  - Ion Giosan (master CSC)

- Seminar

- Ion Giosan (grupele 1, 2, 3, 4)
  - Raluca Brehar (grupa 5 şi master CSC)





- 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  $\geq 5$
  - 40% nota la laborator
    - Obligatoriu  $\geq 5$
  - 10% teste date în timpul cursurilor
  - Bonusuri
    - Activitate seminar
    - Prezență sporită la cursuri



- 6



- 7



# 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  $s$





- 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





- 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: CITEȘTE, 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





# 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 notatie 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.



- 15



- 
- ```

graph LR
    A([Cod sursă]) --> B[Compiler]
    B --> C([Cod mașină])
    D([Date de intrare]) --> E[Program executabil]
    E --> F([Date de ieșire])
    E <--> C
  
```

- 
- ```
graph LR; A([Cod sursă]) --> C[Interpreter]; B([Date de intrare]) --> C; C --> D([Date de ieșire])
```
- The diagram illustrates the flow of the interpretation process. It consists of three blue oval nodes and one green rectangular node. On the left, two ovals labeled 'Cod sursă' (Source code) and 'Date de intrare' (Input data) have arrows pointing to a central green rectangle labeled 'Interpreter'. An arrow then points from the 'Interpreter' rectangle to a final blue oval on the right labeled 'Date de ieșire' (Output data).





- 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



- 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

- 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.)

- Aprobabil în 2011
- Rezolvă problemele C99 și introduce noi elemente (suport pentru Unicode, API pentru *multi-threading*, tipuri de date atomice etc.)

- Aprobă în 2018
- Rezolvă problemele C11 fără a introduce noi elemente



# Cuvinte cheie în C

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



- 20



# Formatul programelor C

---

- Instrucțiunile sunt terminate cu caracterul punct și virgulă
- Mai multe instrucțiuni pot fi scrise pe aceeași 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**





**Afișează la ieșirea standard:**

- 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





# Hello World!

- 25



# Hello World!

- Linii 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.



- 27



- Grupuri de caractere care nu sunt identificatori

- Operatori: `+` `++` `&&` `<` `<=` `!=` `>` etc.

- Costante numerice: **10.8 90 0x4f**

- Carattere: 'A' 'b' '8'

- Şiruri de caractere: **"Programare in limbajul C"**





- signed

- unsigned

- short

- long

- long long

- 30



# Tipuri de date în C (3)

Tip	Octeți mem.	Limita inferioară	Limita superioară	Valori
<b>char</b>	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
<b>int</b>	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

32





# 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'



- 34





- Secvență de caractere care începe și se termină cu caracterele ghilimele "
- Exemple:
  - "sir de caractere"
  - "apostrof ' reprezentat clasic"
  - "compiler \"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 \

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



## Declararea variabilelor

- ```
tip identficator {, identficator };
```

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

- ```
tip_baza identificador[lim] { [lim] }
        { , identificador[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 declarații de tablouri:

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





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

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



- # studio.h





# 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)
```

```
{
```

```
    putchar('A'); // afiseaza caracterul 'A'
```

```
    putchar(66); // 'B' este pe pozitia 66 in tabelul ASCII
```

```
    putchar(97); // 'a' este pe pozitia 97 in tabelul ASCII
```

```
    char ch=getch(); //citeste un caracter fara a-l afisa
```

```
    putchar(ch); //afiseaza caracterul citit
```

```
    ch=getche(); //citeste si afiseaza un caracater
```

```
    putchar(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 funcției **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



```
#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 modifier_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)



- Conversii posibile prin specificatori de format:

48





- Conversii posibile prin specificatori de format:

49



- Modificatori de tip utilizați în conversii posibile:

50



- Citirea unui caracter

- Citirea unui șir de caractere

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

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

51



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

- ```
% indicator dimensione precisione modificatore_tip conversione
```



# 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**



- Indicatori pentru numere (întregi și reale)

29 septembrie 2022



- Conversii posibile prin specificatori de format:

56





- Conversii posibile prin specificatori de format:

57



- Modificatori de tip la specificatori de format

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



- Specificator de format

- Rezultat tipărit

- Specificator de format

- Rezultat tipărit

59



# 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



```
#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\nd\n", a, a);
```

```
printf("b are valoarea zecimala %d; octala %o;\n  
      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\ncu doua zecimale %.2f\n", d, d, d);
```

```
printf("e este sirul de caractere: %s\n",e);
```



}



```
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
```



```
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
```





```
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
```



- **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**



**Rezultate afişate:**  
Andrei are 24 ani, este din Cluj-Napoca si a obtinut nota 9.50