

# Programare procedurală

## - Laborator 1 -

### 1. Tipuri de date fundamentale

- **char**: dimensiunea unui char este a celei mai mici unități adresabile de memorie, cel mai des **1 byte**;
- **\_Bool**: stochează doar 0/1; ocupă **1 byte** (sau macro-ul **bool** - cu includerea librăriei standard **<stdbool.h>** - vom reveni ulterior asupra macro-urilor);
- **int**: stochează un număr întreg. Lungimea sa depinde de compilator, sistemul de operare și arhitectura hardware: **2** sau **4 bytes**.
- **float**: stochează numere în virgule mobile; ocupă cel puțin **4 bytes**;
- **double**: stochează numere în virgule mobile; ocupă cel puțin **8 bytes**.

#### 1.1 Modificatori de tip

- **short** – numai pentru tipul de date **int**;
- **long** – pentru **int** și **double**;
- **signed / unsigned** – doar pentru tipurile de date întregi.

Nu orice combinație este posibilă (exemplu: short long int). Combinațiile posibile sunt redate în tabelul de mai jos.

Tip de date	Memorie	Domeniu de valori	Observații
<b>char</b>	$\geq 1$ bytes		- poate fi <b>signed/unsigned</b> în funcție de compilator/sistem de operare/arhitectură - folosit când vrem să memorăm caractere
<b>signed char</b>		Cel puțin [-128,+127] ( $[2^7, 2^7-1]$ )	- folosit când vrem să stocăm un întreg
<b>unsigned char</b>		Cel puțin [0, 255]	
<b>short</b> <b>short int</b> <b>signed short</b> <b>signed short int</b>	$\geq 2$ bytes	Cel puțin [-32768, +32767] (mai rar [-32767, +32767])	
<b>unsigned short</b> <b>unsigned short int</b>		Cel puțin [0, +65535]	- short int pozitiv
<b>int</b> <b>signed</b> <b>signed int</b>		Cel puțin [-32768, +32767]	- deseori echivalent cu <b>long (long int)</b> , implicit domeniul de valori este $[-2^{31}, 2^{31}-1]$
<b>unsigned</b> <b>unsigned int</b>		Cel puțin [0, +65535]	- int pozitiv - deseori echivalent cu <b>unsigned long (unsigned long int)</b> , implicit domeniul de valori este $[0, +2^{32}-1]$
<b>long</b> <b>long int</b> <b>signed long</b> <b>signed long int</b>	$\geq 4$ bytes	Cel puțin $[-2^{31}, 2^{31}-1] = [-2147483648, +2147483647]$	

<b>unsigned long</b> <b>unsigned long int</b>		Cel puțin [0, +2 <sup>32</sup> -1 ]	- long pozitiv
<b>long long</b> <b>long long int</b> <b>signed long long</b> <b>signed long long</b> <b>int</b>	≥ 8 bytes	Cel puțin [-2 <sup>63</sup> , +2 <sup>63</sup> -1]	
<b>unsigned long long</b> <b>unsigned long long int</b>		[0, +2 <sup>64</sup> -1]	- long long pozitiv
<b>float</b>	≥ 4 bytes	Aproximativ (pe pozitiv) [-1.2E+38, 3.4e+38]	- cel puțin 6 zecimale exacte
<b>double</b>	≥ 8 bytes	Aproximativ (pe pozitiv) [-2.3E+308, 1.7e+308]	- cel puțin 15 zecimale exacte
<b>long double</b>	≥ 10 bytes	Aproximativ (pe pozitiv) [-3.4E+4932, 1.1E+4932]	- cel puțin 19 zecimale exacte

### Observații:

- Există definite în header-ul **<stdint.h>** tipuri de date de dimensiune specificată.  
Exemple: **int8\_t** (int pe 8 biți), **int16\_t**, **int64\_t**, **uint32\_t**  
(vezi <http://en.cppreference.com/w/c/types/integer>).
- Se pot folosi constante macro pentru determinarea limitelor unui tip de date (header-ul **<limits.h>** pentru tipurile de date întregi și **<float.h>** pentru tipurile de date în virgule mobile (vezi <http://en.cppreference.com/w/cpp/types/climits>).  
Exemple: **printf("%d", INT\_MIN);**  
**printf("%f", DBL\_MAX);**
- Implicit, compilatorul C stabilește pentru o constantă numerică cel mai scurt tip de date compatibil care o poate păstra. Excepție fac constantele float care sunt asimilate tipului **double**. Pentru a schimba tipul implicit stabilit de compilator, pot fi adăugate sufixele F (pentru float), U (unsigned), L (long int sau long double).

Exemple: **printf("%d", sizeof(123.23));**  
**printf("%d", sizeof(123.23F));**  
**printf("%d", sizeof(123.23L));**

## 1.2 Declararea de variabile și funcții

`tip_data nume_variabila;`

```
tip_data1) nume_functie (lista_parametri2)) {  
    corpul_functiei  
    return3) rezultat;  
}
```

<sup>1)</sup>tipul de data pentru functii poate fi **void**, caz în care functia nu returneaza un rezultat;

<sup>2)</sup> lista\_parametri poate fi vida sau poate fi de forma:

tip\_data1 nume\_parametru1, tip\_data2 nume\_parametru2, ...

<sup>3)</sup>**return** este cuvânt cheie

## 2. Citiri și scrieri de la tastatură

Librăria <stdio.h> trebuie inclusă pentru a folosi funcțiile de citire, respectiv scriere de la tastatură: **scanf**, **printf**.

### 2.1. Antetul funcției scanf

**int<sup>4)</sup> scanf ( const<sup>1)</sup> char \* format<sup>2)</sup>, lista\_parametri<sup>3)</sup> )**

- citește date din stdin (standard input) și le memorează conform parametrului *format* în locațiile de memorie indicate în lista de parametri.

<sup>1)</sup> cuvântul cheie **const** se folosește pentru a declara variabilele constante, cele care nu își schimbă valoarea;

<sup>2)</sup> format reprezintă un șir de caractere constant de forma : %[\*][**dimensiune**][**modifier**]**tip**, unde

<b>dimensiune</b>	reprezintă numărul maxim de caractere ce pot fi citite în operația curentă de citire
<b>modifier</b>	<b>h</b> : short int (pentru <b>d</b> ) sau unsigned short int (pentru <b>o</b> , <b>u</b> și <b>x</b> ) <b>l</b> : long int (pentru <b>d</b> ), unsigned long int (pentru <b>o</b> , <b>u</b> și <b>x</b> ), long long int (pentru <b>d</b> ) sau double (pentru <b>e</b> , <b>f</b> și <b>g</b> ) <b>L</b> : long double (pentru <b>e</b> , <b>f</b> și <b>g</b> )
<b>tip</b>	<b>c</b> : caracter <b>d</b> : întreg zecimal <b>f</b> , <b>e</b> , <b>E</b> , <b>g</b> , <b>G</b> : număr în virgulă mobilă <b>o</b> : întreg în baza 8 <b>s</b> : șir de caractere <b>u</b> : întreg zecimal fără semn (unsigned) <b>x</b> , <b>X</b> : întreg în baza 16, cu cifrele A-F mici sau mari

<sup>3)</sup> de forma **&a**, unde a este variabila în care se va stoca valoarea citită;

<sup>4)</sup> în caz de succes, funcția returnează numărul de itemi citați; în caz contrar returnează EOF (End Of File – constantă definită).

### Exemple

```
scanf("%d %f", &n, &m);
```

```
scanf("%lld", &x);
```

```
scanf("%Lf %c %hd", &a, &b, &c);
```

## 2.2. Antetul funcției printf

**int<sup>3)</sup> printf (const char \* format<sup>1)</sup>, lista\_parametri<sup>2)</sup>)**

- scrie datele în stdout (standard output) conform formatului

<sup>1)</sup>format reprezintă un sir de caractere constant de forma : % [flag-uri] [dimensiune] [.precizie] [lungime] tip, unde

<b>flag-uri</b>	-: left-justify conform dimensiunii; +: numărul va fi precedat de semn (+ sau -); #: pentru o, x, X – valorile vor fi precedate de 0, 0x, respectiv 0X; pentru e, E, f – virgula va apărea în output, chiar dacă nu există zecimale după virgule; pentru g, G – ca pentru e, E, f, dar zero-urile de după virgulă vor apărea; <b>0</b> : adaugă zero-uri la stânga în locul spațiilor libere
<b>dimensiune</b>	numărul minim de caractere ce vor fi scrise; dacă numărul este mai mic decât dimensiunea, se adaugă spații; dacă este mai mare nu se trunchiază
<b>.precizie</b>	<b>.număr</b> : pentru întregi – numărul minim de scris (se adaugă zerouri înaintea numărului dacă acesta are mai puține cifre); pentru virgulă mobilă – numărul de cifre de scris după virgulă; pentru șiruri de caractere (c) – numărul maxim de caractere de scris
<b>lungime</b>	vezi <b>modificator</b> din scanf
<b>tip</b>	vezi <b>tip</b> din scanf <b>Observații:</b> <b>f</b> : număr real în virgulă mobilă afișat în forma [-]xxx.yyyyyy (implicit 6 cifre) <b>e, E</b> : real afișat sub forma [-]x.yyyyyye+zz <b>g, G</b> : realizează conversia și afișarea ca descriptorii <b>f</b> sau <b>e</b> astfel încât să apară un număr minim de cifre afișate <b>[domeniu]</b> – scanare pentru set de caractere – se consumă caracterele cât timp ele fac parte din domeniu

<sup>2)</sup> de forma **a, b**, unde a și b sunt variabilele ce vor fi scrise;

<sup>3)</sup> în caz de succes, returnează numărul de itemi scriși; în caz contrar, returnează un număr negativ.

### Exemple

```
printf("%d\n", n);
printf("%+08d\n", y);
printf("%+08.4d\n", x);
printf("%-+08.4d\n", x);
printf("%-#8x\n", v);
printf("%+.2lf\n", x);
printf("Litera este %c\n", 'A');
printf("Litera este %c\n", 65);
```

```
scanf("%d%*c%*c%*c%*c %*c %d%*c%d",&x,&y,&z);
```

// Exemplu de test: 507 \* 12345#4608

// Explicație: %\*c se consumă următorul caracter (nu se asignează niciunei variabile)

```
printf("%d %d %d",x,y,z);
```

```
scanf("%d%*[:]%d%*c%d",&x,&y,&z);
```

```
printf("%d %d %d",x,y,z);
```

// Exemplu de test: 12:07:32

// Explicație: %\*[domeniu] se consumă următorul caracter dacă face parte din domeniu

```
scanf("%d%*3[.,!? ;:-]%d%*c%d",&x,&y,&z);
```

```
printf("%d %d %d",x,y,z);
```

*// Exemple de teste: 12...35-47 sau 12?35\*47*

*// Explicație: %\*3[domeniu] se consumă maxim 3 caractere, dacă ele fac parte din domeniu*

```
scanf("%[^.]\s", sir); (unde sir este șir de caractere)
```

*// Explicație: citirea are loc până la întâlnirea caracterului .*

```
scanf("%d%[A-Z]%d%[A-Za-z]%d",&x,s1,&y,s2,&z); (unde s1, s2 sunt șiruri de caractere)
```

*// Exemplu de test: 123AZIEJOI829MaineEsteVineri1341*

*// Explicație: [domeniu] Se citesc caractere cât timp ele fac parte din domeniu*

```
printf("%d %d %d\n",x,y,z);
```

### Constante de tip backslash caracter

Codul	Semnificația
\b	Backspace
\n	Rând nou
\r	Retur de car
\t	Spațiu de tabulare orizontal
\'	Ghilimele
%%	%
\0	Nul
\\	Backslash (linie înclinată spre stânga)
\v	Spațiu de tabulare vertical
\a	Alertă
\N	Constantă în octal (unde N este constantă în octal)
\xN	Constantă în hexazecimal (unde N este o constantă în hexazecimal)

### Exemple

```
printf("Litera este \x41 \n");
```

```
printf("Litera este %c \n", '\x41');
```

```
printf("Litera este \101 \n");
```

```
printf("5 \n      2 3 \r 6");
```

### 3. Operatori

Clasificare	Operator	Exemplu
atribuire	=	int a,b=7,c; c=a+b+2;
aritmetici unari	+, -	int a = 5, b; b = -a; b = +b;
aritmetici binari	+, -, *, /, %	int a = 5, b = 3; float c; c = (a + b) / (a - b) * 2;
preincrementali, predecrementali	++, --	int a = 5, b = 3, c; c = ++a + b; (a va avea valoarea 6, b : 3, c : 9)
postincrementali, postdecrementali	++, --	int a = 5, b = 3, c; c = a++ + b; (a va avea valoarea 6, b: 3, c : 8)
logici	!, &&,	int a = 5;
relationali	>, >=, <, <=, ==, !=	if (a>=3 && (!(a>100)    a==200)) a--;
logici la nivel de bit	&,  , ^, ~, >>, <<	int a = 24, b = 2, c; c = (a>>b)   202;
atribuire combinați	+=, -=, *=, /=, %=, &=,  =, >>=, <<=	int a = 3; a+=5;
decizional	?:	int a,b; b = (a>5)?1:0;
de conversie	(tip)	int a,b; float c; c = (float)a/b;
sizeof	sizeof(tip)	printf("%d", sizeof(int))
virgula	,	int x,i=1,b; x=(b=3,i=i+2,3 ,7);

#### Observație:

Alte aspecte importante ce pot genera rezultate greșite sunt precedența și asociativitatea operatorilor.  
(vezi [http://en.cppreference.com/w/c/language/operator\\_precedence](http://en.cppreference.com/w/c/language/operator_precedence))

Exemple:

(Precedență)

**int x=1, y=0;**

**printf("%d", x == 1 << y == 0);**

Operatorul << are precedență mai mare decât operatorul ==.

**int x=1, y=0;**

**printf("%d", (x == 1) << (y == 0));**

(Asociativitate)

Asociativitate de la stânga la dreapta:

**int a = 1, b = 2, c = 3;**

**if (c > b > a)**

**printf("Adevarat");**

**else**

**printf("Fals");**

## Probleme

1. Se dă ecuația de gradul al II-lea:  $ax^2 + bx + c = 0$ . Să se calculeze rădăcinile ecuației folosind ori de câte ori se poate operatorii de atribuire combinați.
2. Se citesc trei numere întregi de la tastatură. Să se afișeze maximul dintre cele 3 numere folosind operatorul decizional.
3. Se citește un număr întreg  $n$  de la tastatură. Să se calculeze  $n*8$ ,  $n/4$  și  $n*10$  folosind operatorii logici de deplasare la nivel de bit.
4. Se citește un număr întreg de la tastatură. Să se determine dacă acesta este par sau impar folosind doar operatorii logici la nivel de biți.

**Adițional, puteți implementa problemele de la seminar.**