Programarea Calculatoarelor
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.**

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

Observație: Nu orice combinație este posibilă (exemplu: short long int).
Combinații posibile:

Tip de date	Memorie	Domeniu de valori	Observații
char	**≥ 1 bytes**		- poate fi **signed/unsigned** în funcție de compilator/sistem de operare/arhitectură - folosit când vrem să memorăm caractere
signed char		Cel puțin [-128,+127] ([2^7, 2^7-1])	- folosit când vrem să stocăm un întreg
unsigned char		Cel puțin [0, 255]	
short **short int** **signed short** **signed short int**	**≥ 2 bytes**	Cel puțin [-32768, +32767] *(mai rar [-32767, +32767])*	
unsigned short **unsigned short int**		Cel puțin [0, +65535]	- short int pozitiv
int **signed** **signed int**		Cel puțin [-32768, +32767]	- deseori echivalent cu **long (long int)**, implicit domeniul de valori este [-2147483648,+ 2147483647]
unsigned **unsigned int**		Cel puțin [0, +65535]	- int pozitiv - deseori echivalent cu **unsigned long (unsigned long int)**, implicit domeniul de valori este [0,+4294967295]
long **long int** **signed long** **signed long int**	**≥ 4 bytes**	Cel puțin [-2147483648, +2147483647]	

unsigned long **unsigned long int**		Cel puțin [0,+4294967295]	- long pozitiv
long long **long long int** **signed long long** **signed long long int**	**≥ 8 bytes**	Cel puțin [-2^{63}, $+2^{63}$-1]	
unsigned long long **unsigned long long int**		[0, $+2^{64}$-1]	- long long pozitiv
float	**≥ 4 bytes**	Aproximativ (pe pozitiv) [-1.2E+38, 3.4e+38]	- cel puțin 6 zecimale exacte
double	**≥ 8 bytes**	Aproximativ (pe pozitiv) [-2.3E+308, 1.7e+308]	- cel puțin 15 zecimale exacte
long double	**≥ 10 bytes**	Aproximativ (pe pozitiv) [-3.4E+4932, 1.1E+4932]	- cel puțin 19 zecimale exacte

Observații:

1. 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).
2. 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("%Le", DBL_MAX);
3. Constante

   Implicit, compilatorul C stabileşte 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));

2. Declararea unei variabile:

tip_data *nume_variabila*;

3. Declararea unei functii:

tip_data[1)] *nume_functie* (lista_parametri[2)]) {
corpul_functiei
return[3)] *rezultat*;
}

[1)]tipul de data pentru functii poate fi **void**, caz in care functia nu returneaza un rezultat;
[2)] lista_parametri poate fi vida sau poate fi de forma:
tip_data1 *nume_parametru1*, tip_data2 *nume_parametru2*, ...
[3)]**return** este cuvânt cheie

4. Citiri/scrieri de la tastatură:

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

4.1. Antetul funcției **scanf**:

int[4] scanf (const[1] char * format[2], lista_parametri[3]); - citeste date din stdin (standard input) și le memorează conform parametrului format în locațiile de memorie indicate în lista de parametri.

[1] cuvântul cheie **const** se folosește pentru a declara variabilele constante, cele care nu își schimbă valoarea;

[2] format reprezintă un șir de caractere constant de forma : **%[*][dimensiune][modificator]tip**, unde

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

[3] de forma **&a**, unde a este variabila in care se va stoca valoarea citita;

[4] în caz de succes, funcția returnează numărul de itemi citiți; 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);
```

4.2. Antetul functiei **printf**:

int[3] printf (const char * format[1], lista_parametri[2]); - scrie datele in stdout (standard output) conform formatului

[1] format reprezinta un sir de caractere constant de forma : **%[flag-uri][dimensiune][.precizie][lungime]tip**, unde

flag-uri	**-:** 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 aparea 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; **0:** adaugă zero-uri la stânga în locul spațiilor libere
dimensiune	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
.precizie	**.număr** : 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
lungime	vezi **modificator** din scanf
tip	vezi **tip** din scanf **Observații:** **f**: număr real în virgulă mobilă afișat în forma **[-]xxx.yyyyyy** (implicit 6 cifre) **e, E**: real afişat sub forma **[-]x.yyyyyye+zz** **g, G**: realizează conversia și afișarea ca descriptorii **f** sau **e** astfel încât să

	apară un număr minim de cifre afişate **[domeniu]** – scanare pentru set de caractere – seconsumă caracterele cât timp ele fac parte din domeniu

[2)] de forma **a, b**, unde a şi b sunt variabilele ce vor fi scrise;

[3)] în caz de succes, returnează numărul de itemi scrişi; 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

// *Explicaţie:* **%*c** se consumă următorul caracter

```
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

// *Explicaţie:* **%*[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

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

scanf("%[^.]s", sir); (unde **sir** este şir de caractere)

// *Explicaţie:* 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 şiruri de caractere)

// *Exemplu de test:* 123AZIEJOI829MaineEsteVineri1341

// *Explicaţie:* [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	**Semnificaţia**
\b	Backspace
\n	Rând nou

\r	Retur de car
\t	Spaţiu de tabulare orizontal
\”	Ghilimele
%%	%
\0	Nul
\\	Backslash (linie înclinată spre stânga)
\v	Spaţiu 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)

Exemplu: **printf("Litera este \x41 \n");** sau **printf("Litera este %c \n", '\x41');**
printf("Litera este \101 \n");
printf("5 \n 2 3 \r 6");

5. 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; if (a>=3 && (!(a>100) || a==200)) a--;
relationali	>, >=, <, <=, ==, !=	
logici la nivel de bit	&, |, ^, ~, >>, <<	int a = 24, b = 2, c; c = (a>>b) | 202;
atribuire combinaţi	+=, -=, *=, /=, %=, &=, |=, >>=, <<=	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);

Observaţie:
Alte aspecte importante ce pot genera rezultate greşite sunt precedenţa şi asociativitatea operatorilor.
(vezi http://en.cppreference.com/w/c/language/operator_precedence)
Exemple:
(Precedenţă)

```
int x=1, y=0;
        printf(“%d”, x == 1 << y == 0);
```

Operatorul << are precedenţă 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 si 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.