3. Tipuri şi variabile.

3.1. Introducere

Mulţimile din matematică (**N**, **Z**, **R**, etc) sunt mulţimi infinite, cărora le sunt specifice anumite *operaţii*. De exemplu, cu elemente din **Z** se pot efectua operaţii precum: adunarea, scăderea, înmulţirea, împărţirea întreagă şi restul împărţirii întregi. În cazul împărţirii există *restricţia* ca împărţitorul să fie diferit de 0.

Tipurile de date ale limbajelor de programare se referă la mulţimi finite, cu operaţii şi restricţii specifice.

Un *tip de date* este precizat prin:
- *o mulţime finită de* ***valori*** corespunzătoare tipului (constantele tipului)
- *o mulţime de* ***operatori*** prin care se prelucrează valorile tipului
- *o multime de* ***restricţii*** de utilizare a operatorilor.

De exemplu tipul întreg (`int`) este definit prin:
- *mulţimea valorilor* reprezentând numere întregi (între **`-32768`** şi **`32767`**)
- *mulţimea operatorilor* **`: +, -, *, /, %`**
- *mulţimea restricţiilor*: pentru operatorul / împărţitorul nu poate fi 0, etc.

Tipurile pot fi *tipuri fundamentale* şi *tipuri derivate*.

3.2. Tipuri fundamentale.

Calculatoarele pot lucra în mod direct cu caractere, întregi şi reali. Acestea sunt *tipuri fundamentale* (*predefinite*).
Tipurile fundamentale (*predefinite* sau *de bază* sunt:

- tipul caracter (**`char`**)
- tipurile întregi (**`int, short, long`**)
- tipurile reale (**`float, double, long double`**)
- tipul vid (**`void`**)
- tipurile enumerate (**`enum`**)

Tipurile caracter, întreg, real şi tipurile enumerate sunt *tipuri aritmetice*, deoarece valorile lor pot fi interpretate ca numere.

Tipurile caracter, întreg şi enumerările sunt *tipuri întregi*.

În cele ce urmează, vom înţelege prin *obiect*, o zonă de memorie.

Declararea unui obiect specifică numai *proprietăţile obiectului, fără a aloca memorie* pentru acesta.

Definirea unui obiect specifică *proprietăţile obiectului şi alocă memorie* pentru obiect.
Declararea obiectelor ne permite referirea la obiecte care vor fi definite ulterior în acelaşi fişier sau în alte fişiere care conţin părţi ale programului.

3.2.1. Caracterele (tipul `char`)

Valorile asociate tipului caracter (**`char`**) sunt elemente din mulţimea caracterelor – setul de caractere ASCII. Drept consecinţă, caracterele pot fi tratate ca întregi, şi invers.

Afişarea sau citirea unei variabile de tip **`char`** la terminal se face cu descriptorul de format **`%c`**.

Fiecărei constante caracter i se asociază o valoare întreagă, valoarea caracterului în setul de caractere ASCII. Pentru reprezentarea de caractere în alt set de caractere (de exemplu Unicode) se foloseşte tipul **wchar_t.**

Un literal caracter se reprezintă prin caracterul respectiv inclus între apostrofi (dacă este tipăribil).

Caracterele netipăribile se reprezintă prin mai multe caractere speciale numite *secvenţe escape*.
Acestea sunt:

\n sfârşit de linie (LF)
\t tabulare orizontală (HT)
\v tabulare verticală (VT)
\b revenire la caracterul precedent (BS)
\r revenire la început de linie (CR)
\f avans la pagină nouă (FF)
\a alarmă (BEL)
**** caracterul ****
\? caracterul **?**
\' caracterul **'**
\" caracterul **"**
\ooo caracterul cu codul octal **ooo**
\xhh caracterul cu codul hexazecimal **hh**

3.2.2. Întregii (tipul `int`).

Întregii pot avea 3 dimensiuni: **int, short int** (sau **short**) şi **long int** (sau **long**).
Constantele intregi pot fi date in bazele:

```
10:   257,  -65,  +4928
 8:   0125,  0177
16:   0x1ac, 0XBF3
```

Afişarea unei variabile de tip **int** în baza 10 la terminal se face cu descriptorul de format **%d** sau **%i**, în baza 8 cu **%o**, iar în baza 16 cu **%x**.

Calificatorii `long, short` şi `unsigned`

Calificatorul **long** situat înaintea tipului **int** extinde domeniul tipului întreg de la

$(-2^{15},2^{15}-1)$ la **$(-2^{31}, 2^{31}-1)$.**

Constantele intregi lungi se scriu cu sufixul **L:** **125436L**.
Afişarea sau citirea unei variabile de tip **long int** la terminal se face cu descriptorul de format **%ld** sau **%li** în baza 10, **%lo** în baza 8 şi **%lx** în baza 16.

Calificatorul **short** situat înaintea tipului **int** restrânge domeniul întregilor.
Afişarea sau citirea unei variabile de tip **short int** la terminal se face cu descriptorul de format **%hd** sau **%hi** în baza 10, **%ho** în baza 8 şi **%hx** în baza 16.

Calificatorul **unsigned** înainte de **int** deplasează domeniul întregilor

$(-2^{15},2^{15}-1)$ la **$(0,2^{16}-1)$.**

Afişarea sau citirea unei variabile de tip **unsigned int** la terminal se face cu descriptorul de format **%ud** sau **%ui** în baza 10, **%uo** în baza 8 şi **%ux** în baza 16.
Constantele fără semn se specifică cu sufixul **U** sau **u**.

Există 6 tipuri întregi, formate folosind calificatorii:

{ [signed | unsigned] } { [short | long] } int

Avem următoarele echivalenţe:

int = signed int
short = short int = signed short int
long = long int
unsigned = unsigned int
unsigned short = unsigned short int
unsigned long = unsigned long int

Literali întregi fi scrişi în:
- *zecimal* - un şir de cifre zecimale, dintre care prima nu este 0.
- *octal* - un şir de cifre octale care începe cu cifra 0
- *hexazecimal* - un şir de cifre hexa care începe prin 0x.

Un număr negativ este precedat de semnul -. Există şi semnul + unar.

Exemple: **125 01473 0x2AFC +645 -8359**

Literalii întregi se pot termina prin **u** sau **U** (fără semn) sau **l** sau **L** (de tip lung).
Dacă constanta nu are sufix, atunci ea va aparţine primului tip din succesiunea: **int, long int**, **unsigned long int** care permite reprezentarea valorii.

3.2.3. Realii (tipurile `float` şi `double`).

Partea întreagă sau cea fracţionară din constanta reală poate lipsi:

întreg.fractie sau **întreg.** sau **.fractie**

Exemple: **2.25, 1., -.5, +234.5**

Constantele reale pot fi exprimate cu *mantisă* şi *exponent* (notaţia ştiinţifică):

mantisaEexponent = mantisa $10^{exponent}$

Exemple: **1.5e-3, 0.5E6**.

Tipul **float** asigură o precizie de 7 cifre zecimale semnificative şi exponentul maxim 38. Reprezentarea se face pe 4 octeţi.
Afişarea unei variabile de tip **float** la terminal se face cu descriptorii de format **%f** sau **%e**.

Tipul **double** este foarte asemănător tipului **float**, cu deosebirea că se asigură o precizie de 16 cifre zecimale semnificative şi exponentul maxim 306. Reprezentarea se face pe 8 octeţi.
Afişarea sau citirea unei variabile de tip **double** la terminal se face cu descriptorul de format **%lf**, iar a unei variabile de tip **long double** cu **%Lf**. Reprezentarea internă pentru **long double** se face pe 10 octeţi.

Numerele reale conţin punct zecimal şi/sau exponent, având forma:

[<parte întreaga>][.<parte fracţionara>][E<exponent>]

Partea întreagă si partea fracţionară pot lipsi, dar nu simultan. Punctul zecimal şi exponentul sunt opţionale, dar nu simultan.

Constanta poate avea un sufix:
- **f** sau **F** precizează o constantă de tip **float**

- **l** sau **L** precizează o constantă de tip **long double**.

Exemple: **.25 -7.628 15E-3**

3.2.4. Definiri de tip cu typedef.

Un tip de date poate avea şi un alt nume, prin folosirea declaraţiei **typedef**. De exemplu:

```
typedef int  intreg
```

Acelaşi efect se obţine cu directiva **define**:

```
#define  intreg int
```

3.2.5. Tipuri enumerate.

Folosirea tipului enumerare poate creşte claritatea programului, întrucât numele sunt mai semnificative decât valorile care se ascund în spatele lor.
Astfel este mai naturală folosirea valorilor **ROSU, ALB, NEGRU, VERDE** pentru a desemna nişte culori, decât a valorilor **0, 1, 2, 3**.
Constantele simbolice pot fi introduse cu macrodefiniţii **#define**.

```
#define FALSE 0
#define TRUE  1
```

Un *tip enumerat* foloseşte în locul valorilor tipului **0,1,2**,… nume simbolice:

```
enum CULORI {ROSU, VERDE, GALBEN, ALBASTRU, NEGRU};
enum boolean {FALSE, TRUE};
```

Este posibil sa forţăm pentru numele simbolice alte valori întregi decât 0,1,2,…

```
enum ZILE {LUNI=1, MARTI, MIERC, JOI, VIN, SAMB, DUM};
enum escapes
        {BELL='\a', BACKSPACE='\b',TAB='\t',
         NEWLINE='\n', VTAB='\v',RETURN='r'};
```

Folosind **typedef** putem defini tipuri enumerative:

```
typedef enum {ROSU, GALBEN, ALBASTRU} culori;
typedef enum { lu, ma, mi, jo, vi, si, du } zile;
```

3.2.6. Tipul vid (void).

Tipul **void** precizează o mulţime vidă de valori. Acesta se foloseşte pentru a specifica tipul unei funcţii care nu întoarce nici un rezultat, sau a unui pointer generic.

3.3. Tipuri derivate.

Tipurile derivate sunt construite pornind de la tipurile fundamentale. Tipurile derivate sunt:

1. tablourile
2. funcţiile
3. pointerii
4. structurile (sau înregistrările)
5. uniunile (înregistrările cu variante)

Pe baza tipurilor fundamentale se pot construi tipuri derivate ca:
- tablouri de obiecte de un anumit tip

- funcţii care întorc obiecte de un anumit tip
- pointeri care conţin adrese ale unor obiecte de un anumit tip
- structuri care conţin obiecte de tipuri diferite
- uniuni care conţin obiecte de tipuri diferite

3.4. Declararea variabilelor.

O variabilă constă din două componente: obiectul şi numele obiectului. Numele pot fi identificatori sau expresii. Definirea sau declararea unei variabile are forma:

```
clasă-memorie tip  declaratori;
```

Clasa de memorie poate fi omisă. Declaratorii sunt identificatori.

Fiecare declarator poate fi urmat de un *iniţializator*, care specifică valoarea iniţială asociată identificatorului declarator.
Asupra claselor de memorie vom reveni mai târziu, aşa că în exemplele curente le vom omite:

```
int i, j=0.
char c;
float x=1.5, y;
enum CULORI s1, s2=ROSU;
enum BOOLEAN p=TRUE;
culori c1, c2;
zile z, d;
```

3.5. Echivalenţa tipurilor.

Două tipuri se consideră echivalente în următoarele situaţii:
- echivalenţă structurală a tipurilor
- echivalenţa numelui tipului

Două obiecte sunt de tipuri structural echivalente, dacă au acelaşi tip de componente.
Două obiecte sunt de tipuri echivalente după nume, dacă au fost definite folosind acelaşi nume de tip.
Exemple:

```
typedef int integer;
int x, y; /* x şi y echivalente dupa numele tipului*/
integer z;/* x şi z de tipuri structural echivalente */
```