Cursul 2
FUNDAMENTELE LIMBAJULUI C (I)

OBIECTIVE

- constante;
- identificatori;
- declaratii;
- tipuri fundamentale;
- masive;
- enumerari.

PREZENTARE GENERALA

CONSTANTE<home>

In majoritatea limbajelor, scopul unui program este acela de a procesa anumite date. Aceste date pot aparea sub forma constantelor sau ca locatii de memorie, acestea fiind identificate cu ajutorul variabilelor. O constanta poate fi de forma unui numar intreg real (sau in virgula flotanta), sub forma unui caracter sau sir de caractere.

Constantele intregi sunt reprezentate ca intregi cu semn si ocupa cel putin 2 octeti. O constanta poate fi considerata fara semn (pozitiva deci) cand se utilizeaza sufixul **u** sau **U**. Prin aceasta se modifica numai domeniul de valori, nu si dimensiunea spatiului necesar memorarii sale. Pentru a modifica dimensiunea spatiului alocat unei constante intregi sau reale se va utiliza sufixul **l** sau **L**.

Constantele reale pot ocupa 4 octeti, daca sunt declarate cu sufixul **f** sau **F**, sau pot ocupa 8 octeti daca nu au nici un sufix.

Constantele de tip caracter sunt alcatuite dintr-un singur caracter cuprins intre apostrofuri.

Constantele caracter ocupa un singur octet, insa li se poate mari spatiul rezervat cu ajutorul literei **L** folosita ca un prefix al unui grup de 2 caractere cuprinse intre apostrofuri.

Sirurile de caractere sunt delimitate de ghilimele. Aceste constante sunt memorate in locatii succesive de memorie, numarul lor fiind egal cu numarul caracterelor sirului plus unu, cel din urma octet reprezentand spatiul rezervat terminatorului de sir, caracterul nul , '**\0**'.

IDENTIFICATORI<home>

Identificatorii sunt formati cu ajutorul caracterelor alfanumerice si liniuta de subliniere. Primul caracter al unui identificator nu poate fi o cifra.

In cadrul multimii identificatorilor posibili, o clasa aparte o reprezinta cuvintele cheie. Identificatorii inceputi cu liniuta de subliniere atrag atentia programatorului asupra faptului ca aceste cuvinte cheie sunt posibile variabile interne, **asm**, **_asm**, **case**, **_ds**.

O alta categorie de identificatori, apropiata de cea a cuvintelor cheie, este cea a identificatorilor utilizati in cadrul bibliotecilor incluse in programele C. Este buna evitarea utilizarii lor in alte contexte decit cele stabilite in cadrul bibliotecilor.

DECLARATII<home>

Toate variabilele folosite trebuie declarate inainte, cu toate ca anumite declaratii pot fi facute in functie de context. O declaratie specifica un tip si este urmata de o lista de una sau mai multe variabile de acelasi tip.

```
int i,n; int i;
```
sau `int n;`
```
char c,linie[80]; char c;
char linie[80];
```

Variabilele pot fi initializate chiar in momentul declaratiei lor, cu toate ca exista anumite restrictii.

```
char backslash='\\';
int i=0;
float eps=1.0e-5;
```

Daca variabila in discutie este externa sau statica, initializarea are loc o singura data, inainte ca programul sa-si inceapa executia. Variabilele automate initializate explicit sunt initializate la fiecare apel al functiei in care sunt continute.

TIPURI FUNDAMENTALE<home>

Tipurile de date se pot imparti in 2 categorii : tipuri fundamentale si tipuri derivate. Tipurile fundamentale ale limbajului C sunt:

a). **char** – reprezinta tipul caracter pe un octet;
b). **int** – reprezinta tipul intreg pe 2 octeti;
c). **long** - reprezinta tipul intreg pe 4 octeti;
d). **float** – reprezinta numar real pe 4 octeti;
e). **double** - reprezinta numar real pe 8 octeti;

Aceste tipuri fundamentale admit diferite variante numite tipuri de baza de date. Aceste tipuri de date sunt:

Tip	*Reprezentare (in biti)*	*Domeniu*
char	**8**	**-128…127**
unsigned char	**8**	**0…255**
signed char	**8**	**-128…127**
int	**16**	**-32768…32767**
unsigned int	**16**	**0…65535**
signed int	**16**	**-32768…32767**
short int	**16**	**-32768…32767**
unsigned short int	**16**	**0…65535**
signed short int	**16**	**-32768…32767**
long int	**32**	**-2147483648…2147483746**
signed long int	**32**	**-2147483648…2147483746**
unsigned long int	**32**	**0…4294967295**
float	**32**	**3.4E-38…3.4E+38**
double	**64**	**1.7E-308…1.7E+308**

long double	**80**	**3.4E-4932…1.1E+4932**

MASIVE<home>

Masivele de date sau tablourile, din randul carora provin vectorii si matricile, sunt tipuri de date foarte apropiate pointerilor si referintelor. Tablourile sunt definite prin intermediul perechilor de paranteze "**[**" si "**]**".

```
char linie[80];
```

Acest exemplu defineste "**linie**" ca fiind un sir de 80 de caractere si in acelasi timp "**linie**" va constitui un pointer la caracter. Daca **pc** este un pointer la un caracter, declarat prin **char pc** atunci atribuirea **pc=&linie[0]**; face ca **pc** sa refere primul element al tabloului "**linie**" (de indice zero). Aceasta inseamna ca **pc** contine adresa lui **linie[0]**. Acum, atribuirea **c=*pc** va copia continutul lui **linie[0]** in **c**.

Trebuie tinuta seama de diferenta ce exista intre numele unui tablou si un pointer. Un pointer este o variabila astfel ca **pc=linie** si **pc++** sunt operatii permise. In schimb un nume de tablou este o constanta si nu o variabila, iar constructiile de tipul **linie=pc** sau **linie++** sunt interzise. Singurele operatii permise a fi efectuate asupra numelor masivelor, in afara celor de indexare, sunt cele care pot actiona asupra constantelor. Este posibila definirea masivelor multidimensionale cu ajutorul tablourilor de tablouri.

```
char ecran[25][80];
```

Tablourile sunt memorate pe linii. Prima dimensiune a unui masiv se foloseste numai pentru a determina spatiul ocupat de acesta, ea nefiind luata in consideratie decat la determinarea unui element de indici dati. Este permisa omiterea primei dimensiuni a unui tablou, daca tabloul este extern, alocarea facindu-se in cadrul altui modul, sau cand se efectueaza initializarea tabloului in declaratie, in acest ultim caz fiind determinata dimensiunea din numarul de elemente initializate. Initializarea masivelor poate avea loc chiar in cadrul declararii acestora.

```
int point[2]={10,19};
char mesaj[6]="Salut";
```

ENUMERARI<home>

Tipurile enumerate sunt introduse prin sintaxa :

```
enum nume {membru1,membru2,…} var1,var2,…;
enum culori {ROSU,VERDE,ALBASTRU};
```

```
culoare_punct,culoare_linie;
enum culori culoare_cerc,culoare_fond;
```

Acest exemplu defineste tipul de data **culori** si declara variabilele **culoare punct** si **culoare linie**, urmate de declararile a inca doua variabile, **culoare cerc** si **culoare fond** . In limbajul C , numele ce urmeaza cuvantului "**enum**" este chiar numele tipului de data si nu o eticheta de tip. Daca nu exista riscul aparitiei confuziilor, este permisa declararea variabilei si prin suntaxa urmatoare:

```
culori cerneala;
```

Membrii unui tip enumerat sunt numai de tip intreg. Valoarea fiecaruia este obtinuta prin incrementarea cu 1 a valorii membrului anterior; primul membru avand implicit valoarea 0. Este permisa initializarea unui membru cu o valoare oarecare, avandu-se in vedere ca doi membri ai aceluiasi tip nu pot avea aceeasi valoare.Valorile membrilor urmatori se vor stabili conform regulilor mentionate .

```
enum ANOTIMP {iarna=1,primavara,vara,toamna};
enum BOOLEAN{fals,adevarat} conditie;
enum DIRECTIE{up,down,right,left,none=0}; // ilegal
```

Putem defini tipuri enumerate fara a specifica numele acestora. Procedand astfel putem grupa un set de constante fara a denumi acea multime.

```
enum {bine,foarte_bine,cel_mai_bine};
```

In ceea ce priveste utilizarea variabilelor de tip enumerat, limbajul C permite atribuiri de tipul **conditie=0**, dar acest tip de atribuiri genereaza avertismente din partea compilatorului. Este bine ca astfel de atribuiri sa fie insotite de conversia de tip corespunzatoare.

```
conditie=fals;
conditie=(enum BOOLEAN)0;
```

Datele de tip enumerat definite in interiorul structurilor C nu sunt vizibile in afara acestora. Structurile sunt cele introduse prin cuvintele cheie **struct**, **union.**