Introducere în limbajul C

CURS NR. 1/1

Programarea calculatoarelor

Programare

- Formă de comunicare între om și calculator
  - Pentru rezolvarea unor sarcini
- Cu ajutorul limbajelor de programare

Limbaj de programare

- Limbaj artificial
  - Cu sintaxa și semantica bine definite
- Permite specificarea instrucțiunilor de efectuat de către calculator
  - codifică o metodă de rezolvare a unei probleme – exprimă un algoritm

Programarea calculatoarelor

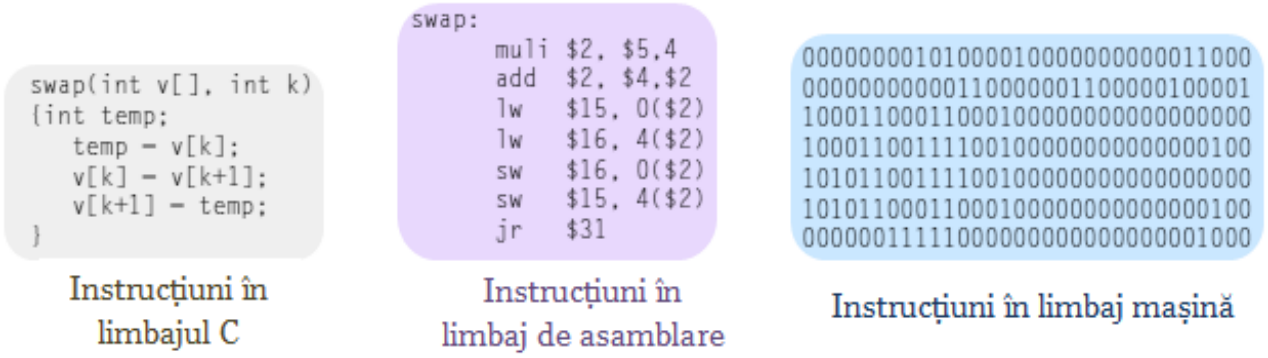
Limbajul nativ al calculatorului

- **Limbaj mașină** (cod obiect) - șabloane de numere binare
- Dependente de platforma (de arhitectura sistemului)

Alte limbaje trebuie convertite în cod mașină

- Folosind compilatoare - limbaj compilat, sau
- Folosind interpretoare – limbaj interpretat

- Limbaje low-level și high-level
  - Apropierea față de limbajul mașină



Reprezentările în limbajul C, limbaj de asamblare și limaj mașină a instrucțiunilor de interschimbare a două valori

Dezvoltarea produselor software

Programarea

- Activitatea de elaborare și dezvoltare a produselor program (software)
- Include
  - **Analiza** cerințelor
  - **Proiectarea** metodei de rezolvare
  - **Implementarea** (codificarea)
  - **Testarea**, verificarea și validarea
  - **Întreținerea** și mentenanța
  - **Documentarea**

Ne interesează în mod special

Algoritmi

- Metodă exprimată pas cu pas de soluționare a unei probleme

un algoritm este o succesiune finită, ordonată și bine definită (exprimată clar și precis) de operații executabile (instrucțiuni, pași) care constituie o metodă (procedură, tehnică) corectă de rezolvare a unei probleme pornind dintr-o stare inițială, folosind datele disponibile și ajungând în starea finală dorită.

Flux de control

- Ordinea de execuție a operațiilor din algoritm

Programare structurată

- Teorema Böhm-Jacopini
  - fluxul de control poate fi exprimat folosind doar trei structuri de control
    - **Structura secvențială**
    - **Structura alternativă (decizională)**
    - **Structura repetitivă (ciclică)**

Reprezentarea algoritmilor

Programul

- Setul de instrucțiuni și ordinea de execuție trebuie să reflecte pașii algoritmului

Alte metode de reprezentare

- Independente de limbaj
- Pseudocod
- Schema logica
- Limbaj natural

Important

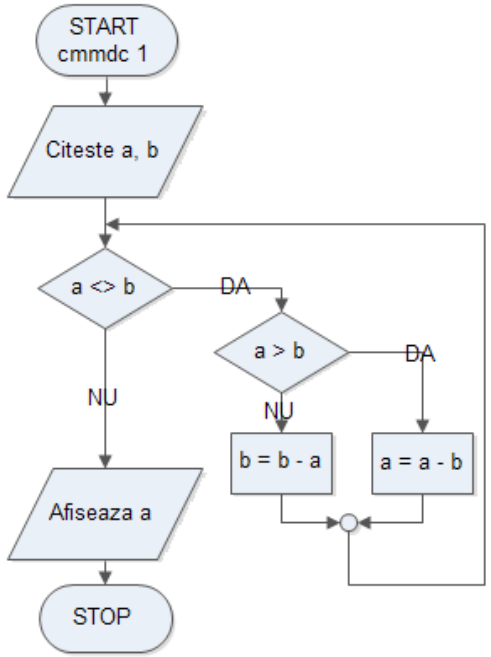
- Formularea clară și precisă a instrucțiunilor
- Specificarea riguroasă a ordinii de execuție
- Reprezentarea independentă de limbaj
- Pot exista mai multe metode de rezolvare
  - Alegerea metodei celei mai potrivite

Exemple – pseudocod și schemă logică

Pseudocod

```
Algoritm cmmdc1
        Întreg: a, b
1       Citește a, b
2       Cât timp a<>b execută
3           Dacă a > b atunci
4               a = a – b
            altfel
5               b = b – a
            sfârșitdacă
        sfârșitcâttimp
6       afișează a
sfârșit cmmdc
```

Schema logică



Paradigme de programare

Programare **imperativă**

- instrucțiunile specificate ca și comenzi
- ordinea de execuție a comenzilor este esențială

- *Limbaj **procedural***
  - unitate de program: procedura, funcția, subrutina și metoda

- *Limbaj **structurat***
  - impune o structură logică
    - Facilitează înțelegerea programului și timpul redus de dezvoltare
    - ușurința în introducerea modificărilor
  - Ex. Pascal, Basic, C

Programare **orientată-obiect**

- Ex. C++, Java, C#

Programare **declarativă**

- Descrie CE trebuie să facă, dar nu și CUM

- Programare **funcțională** (Ex. Haskell, Lisp)
  - Origine în matematica - teoria funcțiilor

- Programare **logică** (Ex. Prolog)
  - Extragerea cunoștințelor din fapte și relații de bază
  - Se bazează pe axiome și reguli de inferență

Limbajul C

Popular, rapid și independent de platformă

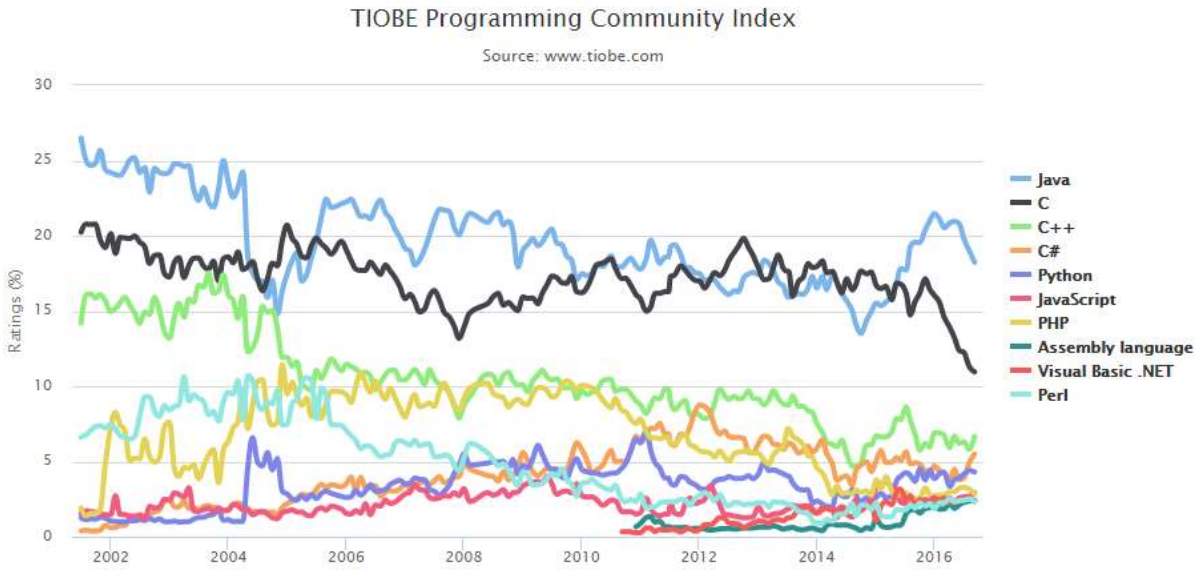
Este un limbaj utilizat cel mai adesea pentru scrierea programelor **eficiente** și **portabile**

- sisteme de operare, compilatoare, interpretoare și alte produse software
- Dezvoltat pentru a fi un limbaj practic și eficient
  - Integrare bună cu modul de gândire a programatorilor

Limbajul C a fost dezvoltat la începutul anilor 1970 în cadrul Bell Laboratories de către Dennis Ritchie

- În strânsă legătură cu sistemele de operare UNIX
- Pornind de la limbajul B – dezvoltat de Ken Thomson

- Cele mai multe limbaje de programare populare sunt
  - Descendenți direcți ai limbajului C sau
  - Prezintă o puternică influență din partea limbajului C



Preluat de pe http://www.tiobe.com/tiobe_index

Trei standarde oficiale active ale limbajului C

!

- **C89** (C90) – aprobat în 1989 de ANSI (American National Standards Institute) și în 1990 de către ISO (International Organization for Standardization)
- C89 a eliminat multe din incertitudinile legate de sintaxa și gramatica limbajului.
  - Cele mai multe compilatoare de C sunt compatibile cu acest standard, cunoscut și ca ANSI C

- **C99** – standard aprobat în 1999, care include corecturile aduse C89 dar și o serie de caracteristici proprii care în unele compilatoare apăreau ca extensii ale C89 până atunci
  - Majoritatea compilatoarelor oferă suport (chiar dacă nu complet) pentru acest standard
    - GCC începând cu versiunea 3.0
    - Visual Studio începând cu versiunea 2013, compatibilitate mai bună în VS 2015
  - Câteva dintre dintre elementele noi introduse în C99
    - Tablouri cu număr variabil de elemente
    - Noi tipuri de data: long long, _Bool, _Complex
    - Funcții *inline*
    - Pointeri *restrict*

- **C11** – standard aprobat în 2011 și care rezolvă erorile apărute în standardul C99 și introduce noi elemente, însă suportul oferit de compilatoare pentru C11 este mai limitat decât suportul pentru C99 (multe compilatoare sunt în curs de adaptare la acest standard)
    - GCC începând cu versiunea 4.7, compatibilitate mai bună în gcc 4.9
    - Visual Studio încă nu oferă suport pentru acest standard (dar multe funcții apar ca extensii proprii)

  - Câteva dintre elementele noi introduse în C11 vizează concurența, securitatea și ușurința în folosire
    - Suport pentru multithreading
    - Biblioteci standard mai sigure
    - Conformitate mai bună cu alte standarde din industrie

Caracteristici principale

C este un limbaj imperativ, structurat, compilat și scurt

Limbaj compilat

- compilatorul transformă instrucțiunile limbajului C în limbaj mașină

Paradigma de programare imperativă, procedurală și structurată

- Instrucțiuni specificate sub forma unor comenzi care sunt grupate într-o ierarhie de subprograme (denumite funcții) și care pot forma module

Limbaj de nivel de mijloc

- Hibrid: între nivelul coborât și nivelul înalt
- permite accesul la date și comenzi aflate aproape de nivelul fizic folosind o sintaxă specifică limbajelor de nivel înalt
- Potrivit pentru programarea sistemelor de operare

Limbaj scurt

- număr redus de cuvinte cheie
- multe funcționalități nu sunt incluse în limbajul de bază ci necesită includerea unor biblioteci standard

Caracteristici principale

C este un limbaj eficient, flexibil și portabil

Eficient

- programe de viteză mare
  - Destinat și aplicațiilor unde înainte se lucra în limbaj de asamblare
- impune puține constrângeri
  - control asupra performanței
- reutilizarea ulterioară a subprogramelor

Flexibilitate

- unul dintre cele mai larg răspândite limbaje de programare
  - domeniu larg de aplicabilitate

Portabilitate

- limbaj independent de hardware
  - aplicațiile pot rula pe platforme diferite cu mici (sau fără) modificări

Caracteristici principale

C este un limbaj permisiv și poate fi dificil de înțeles și modificat

Permisiv

- puține constrângeri
  - presupune că stim ce vrem să facem
- mărește puterea și ușurința în programare

- permite introducerea unor erori care sunt foarte greu de depistat
  - Libertatea are un preț

Poate fi dificil de înțeles

- Un stil de programare adecvat este foarte important
  - programe obfuscate (Obfuscated C Code Contest www.ioccc.org)

Poate fi dificil de modificat

- Documentarea codului este foarte importantă

Why use C then if it's so dangerous?
Because C gives you power over the false reality of abstraction and liberates you from stupidity.

Zed Shaw, Intro to *Learn C The Hard Way*

Cuvinte cheie

Standardul C89			
auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Adăugate în standardul C99		
_Bool	_Complex	inline
	_Imaginary	restrict

Adăugate în standardul C11		
_Alignas	_Generic	_Thread_local
_Alignof	_Noreturn	
_Atomic	_Static_assert	

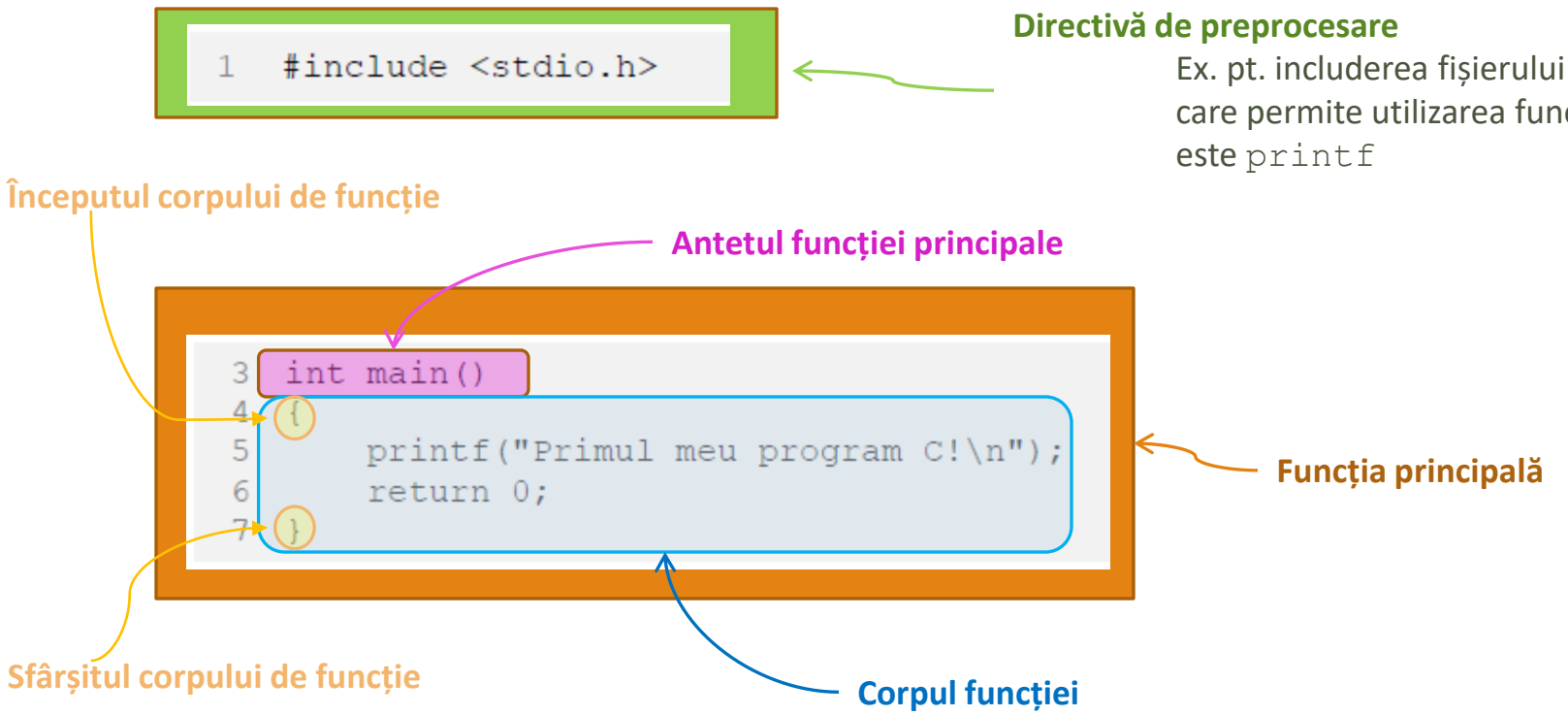
Primul program C

```c
#include <stdio.h>

int main()
{
   printf("Primul meu program C \n");
   return 0;
}
```

Efectul acestui program este afisarea mesajului: Primul meu program C

Primul program C - explicat



Primul program C

!

Observații

- ***main*** nu este cuvânt cheie al limbajului C, dar îl utilizăm doar pentru numirea funcției principale

C este case sensitive

- Face diferență între literele mari și litere mici

Toate cuvintele cheie se scriu cu litere mici

- Excepție cuvintele cheie adăugate în standardele C99 si C11 – acestea încep cu _Majusculă...
  - De cele mai multe ori se folosesc macrodefinițiile asociate în loc de cuvântul cheie
    - Ex. În loc de **_Bool** folosim **bool** din **stdbool.h**
      În loc de **_Complex** folosim **complex** din **complex.h**

Instrucțiunile se termină cu caracterul “;”

Mai multe instrucțiuni pot fi scrise pe aceeași linie

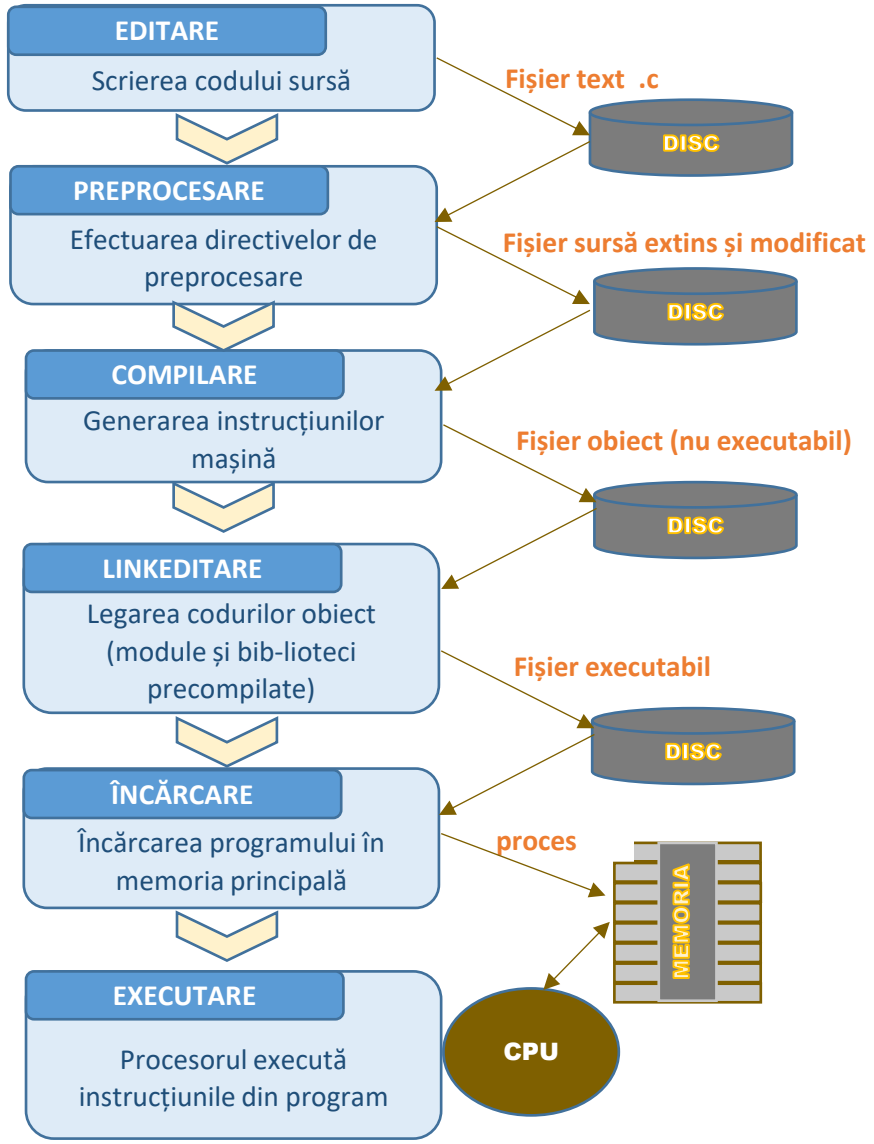
Spațiile ajută la organizarea codului

- Compilatorul ignoră spațiile albe (spațiu, tabulator, etc.)

De la cod sursă la program executabil

Etape

- Editarea codului sursă
  - Scrierea instrucțiunilor în limbajul de programare și
  - Salvarea fișierului cu extensia *.c*
- Preprocesarea
  - Efectuarea directivelor de preprocesare
  - Ca un editor – modifică și adaugă la codul sursă
- Compilarea
  - Verificarea sintaxei
  - Transformare în cod obiect (limbaj mașină) cu extensia *.obj*
  - Nu este încă executabil !
- Linkeditarea (editarea legăturilor)
  - Combinarea codului obiect cu alte coduri obiect
    - Al bibliotecilor asociate fișierelor header incluse
    - Al celorlalte fisiere sursă din program (modular)
  - Transformarea adreselor simbolice în adrese relocabile



Forma generală a unui program C simplu

!

directive de preprocesare

```
int main(void)
{
        instrucțiuni
}
```

Forma generală a unui program C simplu

Directive de preprocesare

- Directive de **definire**
  - #define N 10
- Directive de **includere** a fișierelor header
  - #include <stdio.h>
- Directive de **compilare condiționată**
  - #if, #ifdef, ...
- **Altele**
  - #pragma, #error, ...

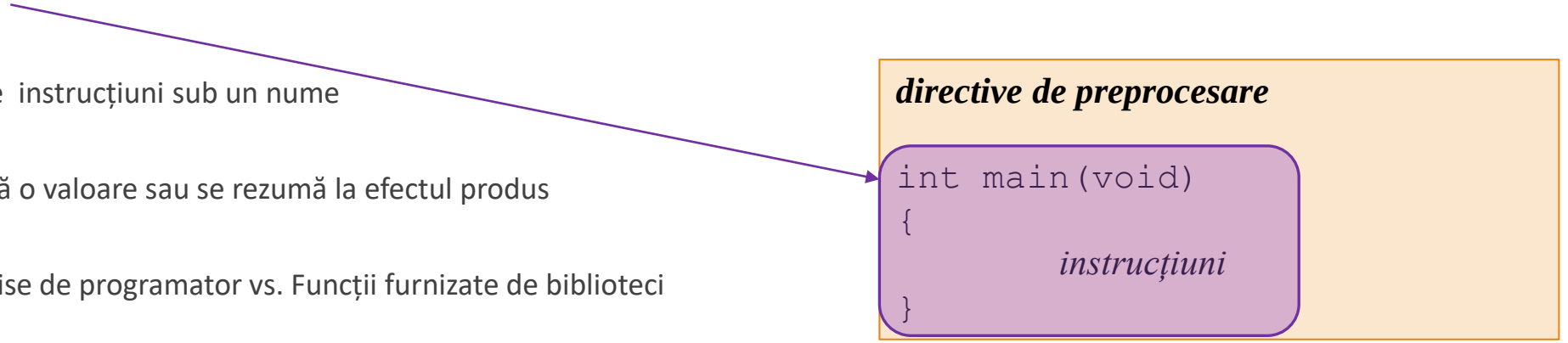
directive de preprocesare

```
int main(void)
{
        instrucţiuni
}
```

Forma generală a unui program C simplu

Funcții

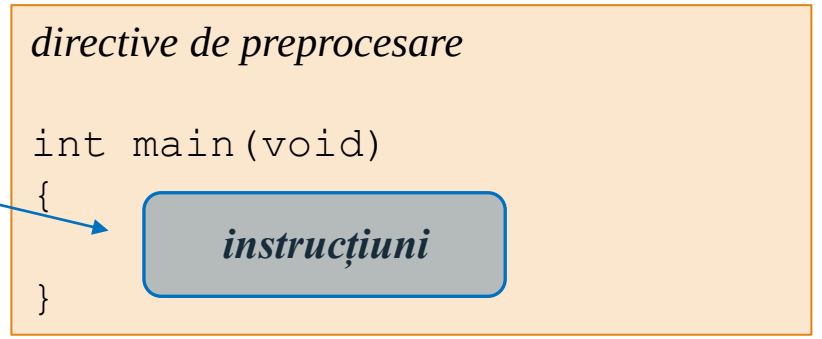
- Grupări de instrucțiuni sub un nume
- Returnează o valoare sau se rezumă la efectul produs
- Funcții scrise de programator vs. Funcții furnizate de biblioteci
- Programul poate conține mai multe funcții
  - Funcția `main` este obligatorie
- Antetul funcției
  - Tip returnat
  - Nume funcție
  - Lista parametrilor formali



Forma generală a unui program C simplu

Instrucțiuni

- Formează corpul funcțiilor
  - Exprimate sub formă de comenzi



- 5 tipuri
  - instrucțiunea **declarație**,
  - instrucțiunea **atribuire**,
  - instrucțiunea **apel de funcție**,
  - instrucțiuni de **control** și
  - instrucțiunea **vidă**.

- Toate instrucțiunile se termină cu caracterul ”;”
  - cu excepția instrucțiunilor compuse
  - ”;” nu are doar rol de separator de instrucțiuni ci instrucțiunile incorporează simbolul ”;” ca ultim caracter
    - omiterea acestuia introduce erori de sintaxă

Forma generală a unui program C mai complex


```
comentarii
directive de preprocesare
declarații de variabile globale

int main(void)
{
        declarații de variabile locale
        instrucțiuni
}
```

Forma generală - exemplu

```c
/*
*  suma.c
*  calculeaza si afiseaza suma a doua numere intregi
*/

/* acesta este un comentariu care
se extinde pe mai multe randuri */

// comentariu pe un singur rand - introdus in C99

#include <stdio.h>

int a = 10;   // variabila globala intreaga initializata

int main()
{
   int b = 5, sum;   // doua variabile locale intregi
   sum = a + b;
   printf("Rezultatul insumarii este: %d \n", sum);

   return 0;
}
```

Comentarii

Formă de documentare a acțiunilor programatorului

Complet ignorate de compilator

Foarte utile pentru organizarea și înțelegerea cu ușurință a programului

Tipuri de comentariu

- C89 oferă un singur fel de comentariu
  - Începe cu /* și se termină cu */
    - se pot extinde pe mai multe linii
      - nu pot fi imbricate
      - utile în inserarea unor explicații mai lungi

- C99 oferă și comentariul pe o singură linie
  - Începe cu // și se termină la finalul liniei
    - utile ca și comentarii inserate pe marginea codului

Comentarii

Recomandare

- La început comentariu cu datele de identificare
  - numele fișierului sursă, scopul programului, autorul și data scrierii respectiv a ultimei modificări

Comentariile pe mai multe linii pot urma stiluri diferite

```
/* Nume: suma.c
   Scop: calcul suma  */
```

Dezavantaj: simbolul de terminare este mai dificil de văzut.
- eroarea de omitere a simbolului de terminare este mai frecventă

```
/************************
*  Nume: suma.c         *
*  Scop: calcul suma    *
************************/
```

Cadrul ajută la structurarea mai clară a programului
– delimitare mai bună

```
/*  Nume: suma.c
*   Scop: calcul suma
*/
```

Eliminarea a trei dintre laturile cadrului din exemplul de mai sus
- mai puțin efort
- vizibilitate bună

```
//  Nume: suma.c
//  Scop: calcul suma
```

Simbolul de început de comentariu trebuie repetat pe fiecare linie.
Elimină problema omiterii simbolului de terminare

Variabile și constante

Stochează datele necesare programului

- Valorile stocate în memoria sistemului în mod transparent programatorului
  - referirea la aceste date se face prin numele lor simbolice, adică prin identificatori

Variabilele stochează date care pot fi modificate în timpul execuției

Constantele își păstrează valoarea cu care au fost inițializate

- Nu se poate schimba valoarea unei constante

Identificatori

Fiecare constantă şi variabilă trebuie să aibă un tip și un nume unic

Reguli și recomandări pentru **numire**

- Sunt permise doar literele alfabetului, cifrele şi liniuţa de legare _ (underscore)
  - Identificatorul nu poate începe cu o cifră
    - Ex. Nu putem declara variabile având numele: 2suma, etc.
- Literele mari sunt tratate diferit de literele mici (case sensitive)
    - Ex. Maxim, maxim, maXim şi MaxiM ar desemna variabile diferite
- Identificatorul nu poate fi cuvânt cheie al limbajului C
    - Ex. for, while, exit, etc.
- Se recomandă alegerea unor nume sugestive ca și identificatori
    - Ex. *suma* în loc de *w12* pentru o variabilă care reprezintă o sumă
- Mascare – atenție la problemele care pot apare
  - Ex. Variabila locală cu același nume cu variabila globală (variabila globală este mascată – nu poate fi accesată)

Tipuri de date

Tipul de dată indică

- **natura datelor** care pot fi stocate în variabilele de acel tip
- **necesarul de memorie** (numărul de octeți ocupați) și
- **operațiile permise** asupra acestor variabile

Limbajul C are 5 tipuri fundamentale de date

- 4 tipuri aritmetice
  - **char**, **int**, **float**, **double**
- Tipul **void**

Programatorul poate crea noi tipuri prin combinarea tipurilor de bază

Reprezentarea și spațiul ocupat de diferitele tipuri de date depind de

- Platformă, sistem de operare și compilator

Tipuri de date

!

Limbajul C are **cinci categorii fundamentale** de tipuri de date (Link)

- tipul **întreg** – **int**: poate reține valori întregi, ex. 1, 0, -532, etc.;
- tipul **caracter** – **char**: poate reține un singur caracter sub forma codului elementelor din setul de caractere specific
  - codul **ASCII** (American Standard Code for Information Interchange) reprezentat pe 7 biți (poate reprezenta 128 de caractere) – set care este frecvent extins la codul **Latin-1**, pe 8 biți care poate reprezenta 256 de caractere
- tipul **real** (numere în virgulă mobilă) – **simplă precizie** – **float**: pot reține valori mai mari decât int și care conțin parte fracționară
  - de ex. 4971.185, -0.72561, etc.
- tipul **real** (numere în virgulă mobilă) **în dublă precizie** – **double**: pot reține valori reale în virgulă mobilă cu o precizie mai mare decât tipul float
- tipul **void**: indică lipsa unui tip anume. Un tip incomplet
  - Putem avea doar pointeri la void și funcții care returnează void (similar cu procedurile din ale limbaje)

Modificatori de tip

!

signed

- modificatorul implicit pentru toate tipurile de date
  - bitul cel mai semnificativ din reprezentarea valorii este semnul

unsigned

- restricționează valorile numerice memorate la valori pozitive
  - domeniul de valori este mai mare deoarece bitul de semn este liber și participă la reprezentarea valorilor

short

- reduce dimensiunea tipului de date întreg la jumătate
  - se aplică doar pe **întregi**

long

- permite memorarea valorilor care depășesc limita de stocare specifică tipului de date
  - se aplică doar pe **int** sau **double**
    - la int dimensiunea tipului de bază se dublează
    - La double se mărește dimensiunea de regulă cu doi octeți (de la 8 la 10 octeți)

Tipuri de date incluse începând cu C99

C99

- **long long**
  - Facilitează stocarea unor valori întregi de dimensiuni foarte mari - reprezentare pe cel puțin 8 octeți
  - Modificator de tip care se aplică doar asupra tipului **int**

- **Tipuri întregi de dimensiune fixă** (Link)
  - Ex. **int8_t**, **int16_t** , **int32_t** , **int64_t**, **intmax_t**, etc.
  - Valori întregi pe număr specificat de biți (definite în fișierul header **stdint.h**)

- **Tipul bulean** (Link)
  - Tipul **_Bool** (De fapt valori întregi: 0 fals, orice altceva adevărat)
  - Macrodefiniții: bool, true, false (definite în fișierul header **stdbool.h**)

- **Tipuri complexe**
  - Tipurile **_Complex** si **_Imaginary**
  - Macrodefiniții: complex, imaginary (definite în fișierul header **complex.h**)
    - Ex. float complex, double complex, long double complex

Modificatori de tip

Abrevieri

- limbajul permite abrevierea tipului prin omiterea cuvântului cheie **int**

◦ short int	->	short
◦ long int	->	long
◦ unsigned short int	->	unsigned short
◦ long long int	->	long long

Alți modificatori (vor fi prezentate ulterior)

- const
- volatile
- restrict

Tipuri de date - clasificare

Tipul character

- Este de fapt un tip întreg – conține codul caracterului reprezentat

Tipuri întregi

- Întregi cu semn: **signed char**, **short**, **int**, **long**, **long long** (C99)
- Întregi fără semn: **_Bool** (C99), **unsigned char**, **unsigned short**, **unsigned int**, **unsigned long**, **unsigned long long** (C99)

Tipuri flotante

- Tipuri reale: **float**, **double**, **long double** (C99)
- Tipuri complexe: **float _Complex**, **double _Complex**, **long double _Complex**
- Tipuri imaginare: **float _Imaginary**, **double _Imaginary**, **long double _Imaginary**

Tipul void: void

Tipuri de date

Limitele specifice unui sistem de calcul pot fi aflate din fișierele header limits.h și float.h (Link)

- exemple: CHAR_MAX, INT_MAX, INT_MIN, UINT_MAX

Pentru determinarea numărului de octeți ocupați de un anumit tip de date se folosește operatorul *sizeof*

```c
/*
*  nume: sizeof.c
*  scop: numarul de octeti ocupati si domeniul de valori pentru int si
double
*
*/

#include <stdio.h>
#include <limits.h>
#include <float.h>

int main(void) {
   printf("Un int se aloca pe: %d octeti \n", sizeof(int));
   printf("Cel mai mic int este: %d\n", INT_MIN);
   printf("Cel mai mare int este: %d\n\n", INT_MAX);

   printf("Un double se aloca pe: %d octeti \n", sizeof(double));
   printf("Cel mai mic double este: %e\n", DBL_MIN);
   printf("Cel mai mare double este: %e\n", DBL_MAX);

   return 0;
}
```

Tipuri de date

Numărul de octeți ocupați și domeniile de valori uzuale ale tipurilor de bază

Tip	Specificator de tip	Număr octeți	Domeniu de valori	Aproximativ
Caracter	**char**	1	-128 ... 127	
	signed char			
	unsigned char		0 ... 255	
Intreg	**short** int	2	-32768 ... 32767	$\pm 3.27 \cdot 10^4$
	signed short int			
	unsigned short int		0 ... 65535	max. $6.55 \cdot 10^4$
	int	4	-2,147,483,647 ... 2,147,483,647	$\pm 2.14 \cdot 10^9$
	signed int			
	unsigned int		0 ... 4,294,967,295	max. $4.29 \cdot 10^9$
	long int	4 sau 8	Valorile pt. 4 octeți – vezi int Valorile pt. 8 octeți – vezi long long int	
	signed long int			
	unsigned long int			
	long long int	8	-9,223,372,036,854,775,807 ... 9,223,372,036,854,775,807	$\pm 9.22 \cdot 10^{18}$
	signed long long int			
	unsigned long long int		0 ... 18,446,744,073,709,551,615	max. $1.84 \cdot 10^{19}$
Real	float	4	Min. $\pm 1.175,494,3 \cdot 10^{-38}$ Max. $\pm 3.402,823,4 \cdot 10^{38}$	$\pm 3.4 \cdot 10^{\pm 38}$ Precizie: 6 zecimale
	double	8	Min. $\pm 2.225,073,858,507,201,4 \cdot 10^{-308}$ Max. $\pm 1.797,693,134,862,315,7 \cdot 10^{308}$	$\pm 1.7 \cdot 10^{\pm 308}$ Precizie: 15 zecimale
	long double	8 - **10** - 12- 16	cel puțin caracteristicile tipului double	Cel puțin $\pm 1.7 \cdot 10^{\pm 4932}$ Precizie: 18 - 34 zecimale

Valori constante

Reprezintă valori fixe (prestabilite) care nu pot fi modificate în decursul programului

Tipul acestor valori este dat de modul de scriere

Categorii de constante

- **Constante întregi**
  - pot fi precedate de un semn (-, +)
  - pot fi aplicate și modificatori ”U” – unsigned, ”l” – long, “ll” – long long
    - Ex. 10, -267, 472103U, 8492371L

  - Constantele întregi sunt interpretate implicit ca fiind valori în **baza 10**
  - Specificarea bazei de numerație pentru valorile întregi se face folosind un prefix special, și anume:

    - pentru exprimare în **octal** numărul se prefixează cu un 0 (zero)
      - ex. Valoarea 10 în zecimal este 012 în octal, valoarea 20 este 024;

    - pentru exprimare în **hexazecimal** numărul se prefixează cu 0x sau 0X
      - ex. Valoarea 10 în zecimal este 0xA în hexa, valoarea 20 este 0X14.

- **Constante în virgulă mobilă**: implicit sunt de tipul double, dar pot fi restricționate la float prin adăugarea caracterului ”f”, sau extinse la long double folosind “Lf”
  - Ex. 14.643, 123.4567f, -53864.123456e8

Valori constante

Categorii de constante (cont.)

- **Constante caracter**
  - se reprezintă printr-un caracter scris între apostrofuri sau prin codul ASCII al caracterului
    - Ex. ‘a’ este echivalent cu codul ASCII 97

  - Există și un set special de constante caracter, numite și caractere escape, care se scriu folosind caracterul ‘\’ ca și prefix.
    - Ex. \n – trecere la linie nouă
      \t – tabulator
      \0 – caracterul nul, etc.

- **Constante șir de caractere**
  - sunt reprezentate sub forma mai unui șir de caractere cuprinse între ghilimele
    - Ex. ”Hello World!”
      ”Acesta este un sir de caractere ”
      ”12345”

Instrucțiuni de declarare

- Necesită specificarea *tipului* și *numelui*
  - Exemple

```
int a;
float x;
char c;
```

- Fiecare declarare se termină cu caracterul ”;”

- Dacă mai multe variabile au același tip, declararea lor poate fi combinată

```
int a, b;
float x, y;
char c, ch;
```

- C89 impune ca declarațiile să preceadă instrucțiunile

- Standardul C99 renunță la această restricție și permite scrierea intercalată a declarațiilor și a instrucțiunilor
  - cu condiția ca întotdeauna **datele să fie declarate înainte să fie folosite**

Domeniul de vizibilitate

Domeniul de vizibilitate al unei variabile

- determinat de locul în care se realizează declararea
  - Variabilă ***globală*** este vizibilă și accesibilă în întreg programul
  - Variabilele declarate în cadrul funcțiilor se numesc ***locale*** și au vizibilitate doar în blocul de instrucțiuni unde au fost declarate

Instrucțiuni de atribuire

Folosind operatorul =

Înainte de atribuire variabila trebuie declarată

```
int a;
char c = 'A';
float b = 3.14, d = -24.16;
a = 5;
```

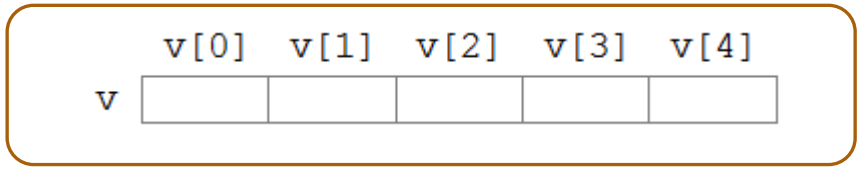
Tablouri unidimensionale - Vectori

Cel mai simplu tip de date agregat este *tabloul unidimensional*, numit și ***vector***

Este o structură de date înrudite compusă din **valori de același tip** grupate sub un nume unic

- Elementele tabloului
  - Acces prin index
    - Atenție: în C indexul începe de la 0

Declararea




```
int v[5];              // vector de intregi
float distante[25];    // vector de valoari reale
char sir[100];         // vector de caractere
```

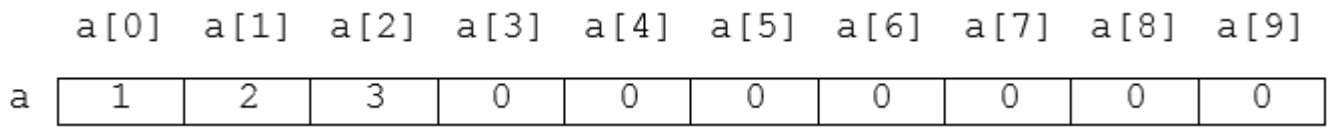
Tablouri unidimensionale - Vectori

Inițializare la declarare

```
int v[5] = { 10, 5, 8, 7, 4 };

//  elementele care nu sunt inițializate explicit
//  se inițializează implicit la 0
int a[10] = { 1, 2, 3 };
```

- Vectorul de întregi a după inițializare arată astfel:



- Inițializarea **după** declarare implică instrucțiuni repetitive pentru a parcurge elementele iterativ

- Accesul la elemente: prin procedeul de indexare

```
int sum;
int v[5] = { 10, 5, 8, 7, 4 };

v[2] = v[1] - 1;
sum = v[0] + v[1] + v[2] + v[3] + v[4];
```

Tablouri unidimensionale - Vectori

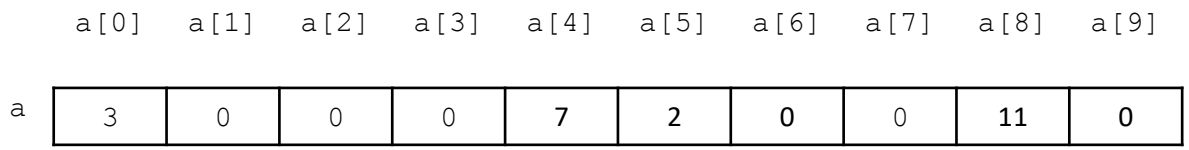
Inițializatori desemnați (C99)

- Practic dacă doar câteva elemente necesită inițializare explicită
  - Restul elementelor pentru care nu se specifică explicit o valoare, se inițializează implicit la 0 (zero)

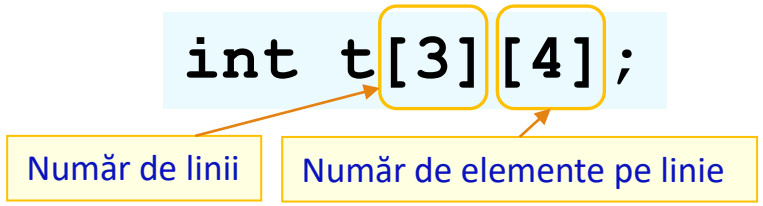
- Exemplu

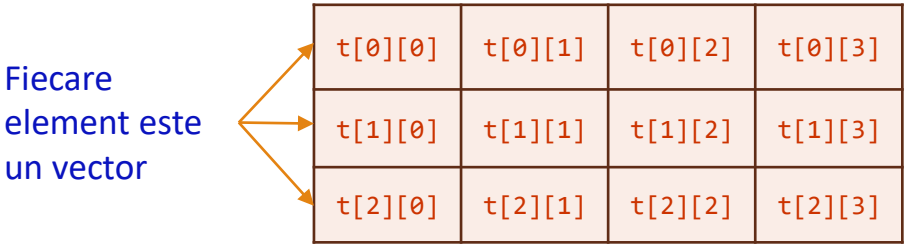
```
int a[10] = { 3, [4] = 7, 2, [8] = 11 };
```

  - În tabloul de 10 elemente
    - primul element este inițializat la 3
    - Elementul de la indexul 4 este inițializat la 7
    - Elementul următor (indexul 5) este inițializat la 2
    - Elementul de la indexul 8 este inițializat la 11
    - Elementele de pe pozițiile 1, 2, 3, 6, 7 și 9 se inițializează implicit la 0

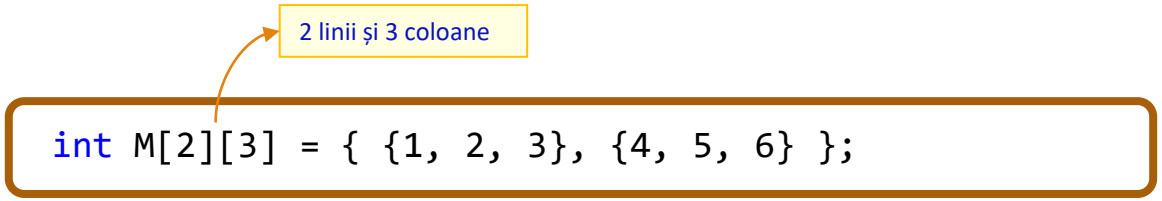


Tablouri bidimensionale – Matrice





Inițializare la declarare



- Inițializarea **după** declarare implică instrucțiuni repetitive pentru a parcurge elementele iterativ
- Accesul la elemente: prin procedeul de indexare

```
int sum;

M[1][2] = M[0][0] * 2;
sum = M[0][0] + M[0][1] + M[0][2] + M[1][0] + M[1][1] + M[1][2];
```

Funcții de intrare/ ieșire

CURS NR. 1/2

Funcții de intrare/ ieșire

!

Biblioteca standard de intrare/ieșire oferă mai multe categorii de funcții pentru efectuarea operațiilor de I/E

- disponibile prin includerea fișierului header **stdio.h**

- la nivel de **caracter**
  - Funcțiile **getchar** și **putchar**

- la nivel de **linie**
  - Funcțiile **gets** și **puts**

- cu **format**
  - Funcțiile **scanf** și **printf**
    - Specificatori de format

- Dispozitivul standard de intrare – implicit tastatura
- Dispozitivul standard de ieșire – implicit ecranul monitorului

Funcții de intrare/ ieșire

Funcții de citire/scriere a unui caracter

- Funcția **getchar** preia un caracter de la tastatură (de la dispozitivul standard de intrare) și returnează caracterul sub formă de întreg

- Funcția **putchar** afișează pe ecran (scrie în dispozitivul standard de ieșire) caracterul specificat ca și parametru

```
//  functii_IO_1.c
//  getchar si putchar
//  citirea și afișarea unui singur caracter

#include <stdio.h>

int main()
{
   int c;
   c = getchar();
   putchar(c);

   return 0;
}
```

Funcții de intrare/ ieșire

Funcții de citire/scriere a unui caracter

- Exemplu

```
//  functii_IO_2.c
//  getchar si putchar
//  citirea și afișarea caracterelor până la X

#include <stdio.h>

int main() {
   int c;
   printf("Introduceti un text. Caracterul X determină oprirea:  ");

   while ((c = getchar()) != 'X') {
      putchar(c);
   }

   return 0;
}
```

- Observație
  - Terminalul folosește o **zonă tampon** (buffer) la nivel de linie
  - Acesta este motivul pentru care putem introduce un rând întreg de text înainte ca `getchar` să proceseze caracterele

Funcții de intrare/ ieșire

Funcții de citire/scriere a unei linii de text

- Funcția **gets** preia o linie de text de la tastatură (sau dispozitivul standard de intrare) și o salvează într-un șir de caractere specificat ca parametru (C11 a eliminat gets)

- Funcția **puts** afișează șirul de caractere primit ca paramteru pe ecranul monitorului (pe dispozitivul standard de ieșire)

```c
//  functii_IO_4.c
//  gets si puts
//  citirea și afișarea unui sir de caractere

#include <stdio.h>

int main()
{
   char  sir[250];
   printf("Introduceti un sir de caractere: ");
   gets(sir);
   printf("Sirul de caractere introdus este: ");
   puts(sir);

   return 0;
}
```

Atenție:
C11 a exclus funcția gets() din standard
- folosim fgets() în schimb

```c
fgets(sir, 200, stdin);
```

Funcții de intrare/ ieșire

Funcțiile pentru citirea/scrierea cu format

- Funcțiile **printf** și **scanf** permit controlul formatului în care se scriu respectiv se citesc datele

- **printf**
  - afișează un șir de caractere la ieșirea standard – implicit pe ecranul monitorului
  - dacă șirul conține **specificatori de format**, atunci **argumentele adiționale** (care urmează după șir) sunt formatate în concordanță cu specificatorii de format (subșir care începe cu caracterul %) și inserate în locul și pe pozițiile acestora din cadrul șirului
  - Exemplu

Caracter special: trecere la linie nouă

```
int a = 4;

printf("Media numerelor %d si %f este %.2f \n", a, 7.0, (a+7.0)/2 );
```

  - Rezultat

```
Media numerelor 4 si 7.000000 este 5.50
```

Funcții de intrare/ ieșire

Funcțiile pentru citirea/scrierea cu format

- **printf** – exemplu

```
//  functii_IO_printf_1.c
//  printf
//  exemplificare simpla

#include <stdio.h>

int main()
{
   char c = 'a';
   int n = 10, m = 025;
   float x = 12.34f;

   printf("c = %c \n", c);
   printf("n = %d, m (octal) = %o, m (zecimal) = %d\n", n, m, m);
   printf("x = %f\n", x);

   return 0;
}
```

- Rezultat

```
c = a
n = 10, m (octal) = 25, m (zecimal) = 21
x = 12.340000
```

Funcții de intrare/ ieșire

!

Funcțiile pentru citirea/scrierea cu format

- **printf** – specificatorii de format încep cu % urmat de:

Specificator	Format
d	Întreg cu semn în baza 10
i	Întreg cu semn în baza 8, 10 sau 16
U	Întreg fără semn în baza 10
O	Întreg fără semn în baza 8
x	Întreg fără semn în baza 16
X	Întreg fără semn în baza 16 (cu majuscule)
f	Real în simplă precizie în baza 10
e	Real cu notație științifică (mantisă și exponent)
E	Real cu notație științifică (mantisă și exponent) cu majuscule
g	Real în reprezentarea mai scurtă (%e sau %f)
G	Real în reprezentarea mai scurtă (%e sau %f) cu majuscule
c	Caracter
s	Șir de caractere
p	Adresa (pointer)

Funcții de intrare/ ieșire

Funcțiile pentru citirea/scrierea cu format

printf – Specificatorul de format mai poate conține și subspecificatori între caracterul % și caracterele listate mai sus astfel încât șablonul general pentru orice specificator de format este:

%[indentare][lățime_rezervată][.precizie][lungime]specificator

-	Indentare la stânga în cadrul spațiului rezervat. Implicit se face indentare la dreapta
+	Impune afișarea semnului chiar și pentru valori pozitive. Implicit doar valorile negative sunt precedate de semn
0	Umplerea spațiului rezervat pe pozițiile unde a rămas gol cu 0 (zerouri) în loc de spații
#	Folosit pentru • afișarea valorilor întregi octale sau hexazecimale, deci împreună cu specificatorii o și x, pentru a preceda valoarea cu 0 (pentru octal) sau 0x (pentru hexa); • afișarea valorilor reale cu punct zecimal, chiar dacă zecimalele sunt doar zero-uri. Implicit nu se pune punctul zecimal dacă este urmat doar de zero-uri.

numărul minim de poziții rezervate pentru afișarea valorii.
- Dacă valoarea care trebuie afișată este mai scurtă, restul pozițiilor se vor umple cu spații.
- Dacă valoarea depășește spațiul rezervat nu se face trunchiere

h	indică `short int`
l	indică `long int`
ll	(C99) indică `long long int`
L	(C99) indică `long double`

- La întregi indică numărul minim de cifre care trebuie afișate
- La reale (f, e și E) indică numărul de cifre zecimale care trebuie afișate după punctul zecimal. Implicit se afișează șase zecimale.
- În cazul g și G, indică numărul maxim de cifre semnificative care trebuie afișate.
- În cazul s, indică numărul maxim de caractere care trebuie afișate. Implicit se afișează toate caracterele din șir până la întâlnirea caracterului nul.

Funcții de intrare/ ieșire

- **printf** – alt exemplu

```c
//  functii_IO_6.c
//  printf
//  afișarea unor valori sub diverse formate

#include <stdio.h>

int main()
{
    printf("Un sir: %30s \n", "Programare in C");
    printf("Un sir indentat la stg: %-30s \n", "Programare in C");
    printf("Un intreg pe 10 pozitii: %10d \n", 123);
    printf("Un intreg cu umplere cu 0: %010d \n", 123);
    printf("Un nr real in simpla precizie: %f \n", 1.2345);
    printf("Acelasi numar cu 2 zecimale: %.2f \n", 1.2345);
    printf("Acelasi numar pe 10 pozitii: %10.2f \n", 1.2345);
    printf("Acelasi numar indentat stanga: %-10.2f \n", 1.2345);
    printf("Un intreg in hexa: %x \n", 77);
    printf("Acelasi intreg in hexa: %#x \n", 77);
    printf("Un intreg in octal: %o \n", 77);
    printf("Acelasi intreg in octal: %#o \n", 77);

    return 0;
}
```

```
Un sir:                         Programare in C
Un sir indentat la stg: Programare in C
Un intreg pe 10 pozitii:        123
Un intreg cu umplere cu 0: 0000000123
Un nr real in simpla precizie: 1.234500
Acelasi numar cu 2 zecimale: 1.23
Acelasi numar pe 10 pozitii:       1.23
Acelasi numar indentat stanga: 1.23
Un intreg in hexa: 4d
Acelasi intreg in hexa: 0x4d
Un intreg in octal: 115
Acelasi intreg in octal: 0115
Press any key to continue . . .
```

Funcții de intrare/ ieșire

!

Funcțiile pentru citirea/scrierea cu format

- **scanf**
  - citește date de intrare (implicit de la tastatura) în formatul indicat de șirul de formatare și le salvează la adresele indicate de argumentele adiționale (variabilele de intrare)

  - Șirul de formatare poate include următoarele elemente:
    - Spațiu alb: funcția citește și ignoră spațiile albe (spațiu, tab, linie nouă) înaintea următorului caracter diferit de spațiu
      - un singur spațiu în șirul de formatare se suprapune asupra oricâtor spații din șirul introdus, inclusiv asupra nici unui spațiu

    - Caracter diferit de spațiu, cu excepția caracterului %: funcția citește următorul caracter de la intrare și îl compară cu caracterul specificat în șirul de formatare
      - Dacă se potrivește, funcția are succes și trece mai departe la citirea următorului caracter din intrare
      - Dacă nu se potrivește, funcția eșuează și lasă următoarele caractere din intrare nepreluate

    - Specificator de format: secvențe care încep cu caracterul %, care indică tipul și formatul datei preluate.

Funcții de intrare/ ieșire

!

Funcțiile pentru citirea/scrierea cu format

- **scanf**
  - Specificatorii de format sunt în principiu aceiași cu specificatorii descriși în cazul funcției printf, șablonul general este:

```
%[*][lățime][lungime]specificator
```

    - Subspecificatorul opțional * *(asterisc)* indică faptul că data cu formatul indicat este preluată de la intrare, dar nu este salvată la vreo adresă indicată de argumentele adiționale.
    - Subspecificatorul opțional *lățime* indică numărul maxim de caractere care vor fi preluate în operația curentă.
    - Subspecificatorul opțional *lungime* indică modificarea dimensiunii pe care se salvează tipul de dată considerat. Valorile cele mai frecvente sunt ca și în cazul funcției printf: h, l, ll și L.
    - Valoarea returnată de funcția scanf indică numărul de argumente adiționale preluate cu succes, sau valoarea EOF.

Funcții de intrare/ ieșire

Funcțiile pentru citirea/scrierea cu format

- **scanf**
  - funcție de potrivire a șabloanelor
    - Încearcă suprapunerea între șirul de formatare și șirul introdus
      - de la stânga la dreapta
      - La întâlnirea unui specificator de format se încearcă extragerea din șirul de intrare a unei valori de tipul și modificatorul indicat
        - Se oprește la un caracter care nu poate fi parte dintr-o valoare de tipul cerut sau prin care se depășește *lățimea*
    - În caz de succes funcția trece mai departe la procesarea următoarelor elemente din șirul de formatare și apoi returnează numărul de elemente preluate cu succes
    - În caz de eșec funcția revine și restul șirului de intrare care a rămas neprocesat va rămâne în zona tampon, și va forma prima parte a șirului de intrare la următoarea citire

- exemplu

```c
//  functii_IO10.c
//  scanf - mod de functionare

#include <stdio.h>

int main() {
   int a, b;
   float x, y;

   printf("Introduceti datele: ");
   scanf("%d%d%f%f", &a, &b, &x, &y);

   printf("\nDatele introduse sunt: \
          \na: %d \t b: %d \t x: %f \t y: %f \n",
           a, b, x, y);

   return 0;
}
```

```
Introduceti datele: 123-41.723.154
```

```
a: 123   b: -41          x: 0.723000     y: 0.154000
```