

Programarea calculatoarelor

FMI

Secția Calculatoare și tehnologia informației, anul I

Cursul 5 / 30.10.2023

Programa cursului

□ Introducere

- Algoritmi
- Limbaje de programare.

□ Fundamentele limbajului C

- Introducere în limbajul C. Structura unui program C.
- Tipuri de date fundamentale. Variabile. Constante. Operatori. Expresii. Conversii.
- Tipuri derivate de date: pointeri, tablouri, șiruri de caractere, structuri, uniuni, câmpuri de biți, enumerări
- Instrucțiuni de control
- **Directive de preprocesare. Macrodefiniții.**
- **Funcții de citire/scriere.**
- Etapele realizării unui program C.

□ Fișiere text

- Funcții specifice de manipulare.

□ Funcții (1)

- Declarație și definire. Apel. Metode de transmitere a parametrelor. Pointeri la funcții.

□ Tablouri și pointeri

- Legătura dintre tablouri și pointeri
- Aritmetica pointerilor
- Alocarea dinamică a memoriei
- Clase de memorare

□ Șiruri de caractere

- Funcții specifice de manipulare.

Fișiere binare

- Funcții specifice de manipulare.

□ Structuri de date complexe și autoreferite

- Definire și utilizare

□ Funcții (2)

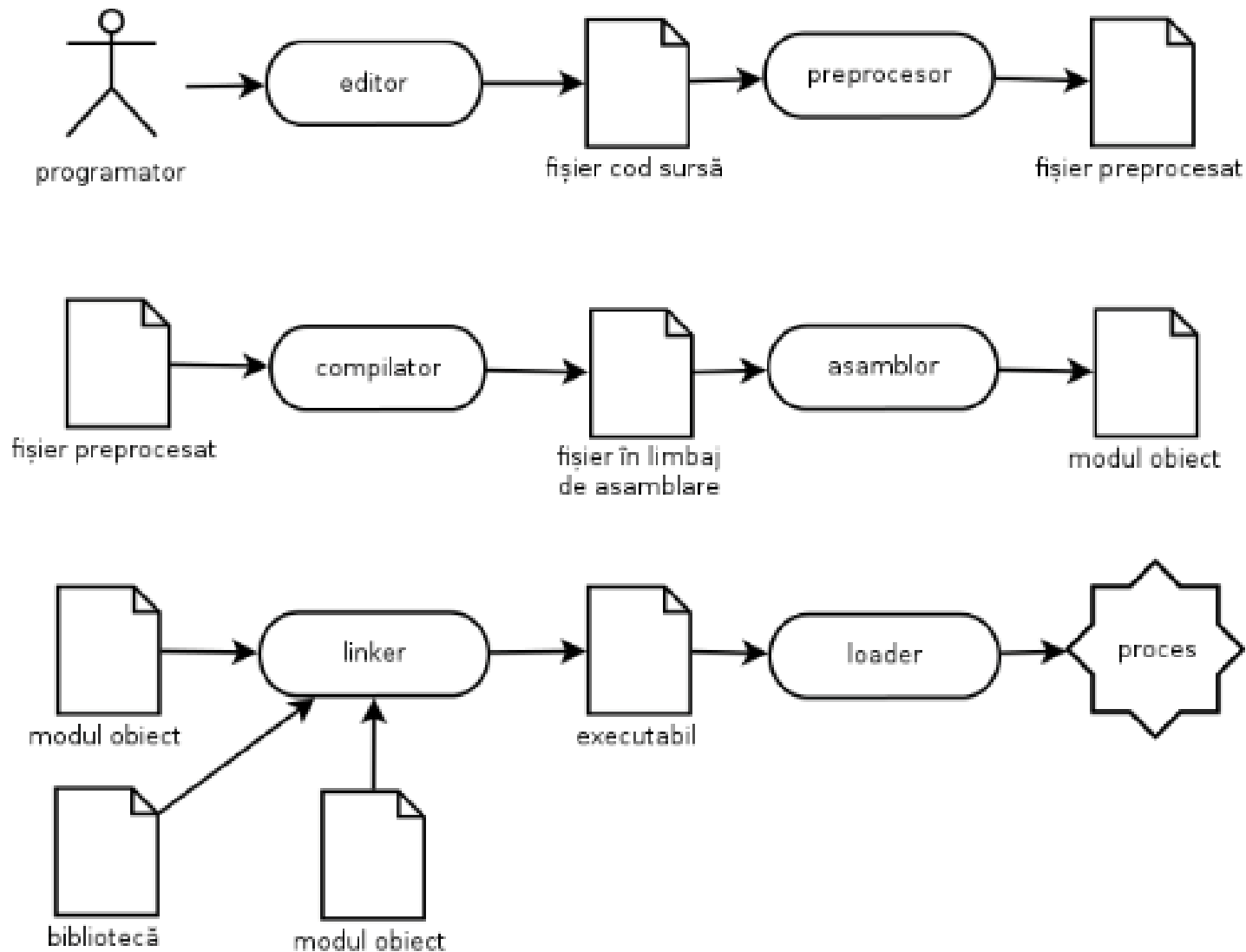
- Funcții cu număr variabil de argumente.
- Preluarea argumentelor funcției main din linia de comandă.



Cursul de azi

- 1. Etapele realizării unui program C**
2. Directive de preprocesare. Macrodefiniții
3. Funcții de citire scriere

Etapele realizării unui program în C



Etapele realizării unui program în C

❑ etape pentru obținerea unui cod executabil:

1. **editarea** codului sursă

- ❑ salvarea fișierului cu extensia `.c`

2. **preprocesarea**

- ❑ efectuarea directivelor de preprocesare (**#include**, **#define**)
- ❑ ca un editor – modifică și adaugă la codul sursă

3. **compilarea**

- ❑ verificarea sintaxei
- ❑ codul este tradus din cod de nivel înalt în limbaj de asamblare

4. **asamblarea**

- ❑ transformare în cod obiect (limbaj mașină) cu extensia `.o`, `.obj`
- ❑ nu este încă executabil !

5. **link-editarea** (editarea legăturilor)

- ❑ combinarea codului obiect cu alte coduri obiect (al bibliotecilor asociate fișierelor header)
- ❑ transformarea adreselor simbolice în adrese reale

Etapele realizării unui program în C

1. Preprocesarea

gcc -E program.c

```
# 5 "prog_aux.c"
int functie_prog_aux()
{
    printf("Acesta este prog_aux.c");
    return 0;
}
# 5 "program.c" 2
# 1 "prog_aux2.c" 1

int functie_prog_aux2()
{
    printf("Acesta este prog_aux2.c");
    return 0;
}
# 6 "program.c" 2

int main()
{
    int z = 0;
    functie_prog_aux();
    functie_prog_aux2();

    printf("n = %d \n",z);

    return 0;
}
```

Etapele realizării unui program în C

2. Compilarea gcc -S program.c (produce program.s)

```
.file "program.c"
.section .rdata,"dr"
LC0:
.ascii "Acesta este prog_aux.c\0"
.text
.globl _functie_prog_aux
.def _functie_prog_aux; .sc1 2; .type 32; .endif
_functie_prog_aux:
LFB14:
.cfi_startproc
pushl %ebp
.cfi_def_cfa_offset 8
.cfi_offset 5, -8
movl %esp, %ebp
.cfi_def_cfa_register 5
subl $24, %esp
movl $LC0, (%esp)
call _printf
movl $0, %eax
leave
.cfi_restore 5
.cfi_def_cfa 4, 4
ret
.cfi_endproc
LFE14:
.section .rdata,"dr"
LC1:
.ascii "Acesta este prog_aux2.c\0"
.text
.globl _functie_prog_aux2
.def _functie_prog_aux2; .sc1 2; .type 32; .endif
_functie_prog_aux2:
LFB15:
.cfi_startproc
pushl %ebp
.cfi_def_cfa_offset 8
.cfi_offset 5, -8
movl %esp, %ebp
.cfi_def_cfa_register 5
subl $24, %esp
movl $LC1, (%esp)
```

Etapele realizării unui program în C

3. Asamblarea

```

L -      .text      p      H      0
\data      @ 0A.bss
      € 0A.rdata      8      t      @ 0
@/4      $      @ 0
@/15      x      Ž      L      @ 0@U%âfiÇ
$      è      ÉÃU%âfiÇ$|      è      ÉÃU%âfaðfi      ÇD
$      è³ÿÿÿèÇÿÿÿ<D$ %D$Ç$/$      è      ÉÃ      Acesta este
prog_aux.c Acesta este prog_aux2.c n = %d
GCC: (MinGW.org GCC-6.3.0-1) 6.3.0      zR      ←
J J ^      J      AßB... B
| UÅ
J J      <      AßB... B
| UÅ
J J      \      6      ;      AßB... B
| wÅ
J J
-      !!      "
-      '      !!      <      ↑      ]
-      b      !!
-      @      -      \      -      .file      by      g program.c
      |      +      |
_ main      2      .text      L m
      .data      L      .bss
      .rdata      J      L 8
      |      L #      I      -      L x      L
      _ main      _ printf      S      .rdata
$zzz .eh_frame _functie_prog_aux _functie_prog_aux2 .rdata
$zzz .eh_frame

```


Etapele realizării unui program în C

- ❑ **preprocesare + compilare + asamblare + link-editare**

- ❑ `gcc program.c`

- ❑ produce `a.out` ca fisier executabil

- ❑ `gcc program.c -o alt_nume`

- ❑ produce `alt_nume` ca fisier executabil

- ❑ pentru proiecte mari (zeci de mii de linii de cod) daca schimbam o functie nu vrem sa recompilam intreg proiectul ci doar sa compilam fisierul cu functia schimbata.

- ❑ Link-editorul va produce codul obiect final.

Etapele realizării unui program în C

- ❑ Varianta modularizata:
- ❑ compilez fiecare fisier (modul) în parte
 - ❑ `gcc -c prog_aux.c` => produce **prog_aux.o**
 - ❑ `gcc -c prog_aux2.c` => produce **prog_aux2.o**
 - ❑ `gcc -c program.c` => produce **program.o**
- ❑ Link-editez codul obiect (am nevoie ca “program.c” să știe unde găsește funcțiile auxiliare)
 - ❑ `gcc prog_aux.o prog_aux2.o program.o -o prog_final`
 - ❑ produce fisierul executabil `prog_final`

Cursul de azi

1. Etapele realizării unui program C
- 2. Directive de preprocesare.
Macrodefiniții**
3. Funcții de citire scriere

Preprocesarea în limbajul C

- apare înaintea procesului de compilare a codului sursă (fișier text editat într-un editor și salvat cu extensia .c)
- preprocesarea codului sursă asigură:
 - includerea conținutului fișierelor (de obicei a fișierelor *header*)
 - definirea de macrouri (macrodefiniții)
 - compilarea condiționată
- constă în substituirea simbolurilor din codul sursă pe baza directivelor de preprocesare
- directivele de preprocesare sunt precedate de caracterul diez #

Directiva **#include**

- ❑ copiază conținutul fișierului specificat în textul sursă
- ❑ **#include <nume_fisier>**
 - ❑ caută nume_fisier în directorul unde se află fișierele din librăria standard instalată odată cu compilatorul
- ❑ **#include "nume_fisier"**
 - ❑ caută nume_fisier în directorul curent

Directiva #include. Exemple

```
// include fisierul stdio.h  
// din directoarele standard  
#include <stdio.h>
```

```
// include fisierul Liste.h;  
// cautarea se face intai in directorul  
// curent si dupa aceea in directoarele standard  
#include "Liste.h"
```

```
// include fisierul Masive.cpp din directorul  
// c:\Biblioteci; daca fisierul nu exista nu  
// mai este cautat in alta parte si se genereaza  
// o eroare de compilare  
#include "C:\Biblioteci\Masive.cpp"
```

Directiva **#define**

- ❑ folosită pentru definirea (înlocuirea) constantelor simbolice și a macrourilor
- ❑ definirea unei **constante simbolice** este un caz special al definirii unui macro

#define nume text

- ❑ în timpul preprocesării **nume** este înlocuit cu **text**
- ❑ **text** poate să fie mai lung decât o linie, continuarea se poate face prin caracterul \ pus la sfârșitul liniei
- ❑ **text** poate să lipsească, caz în care se definește o constantă vidă

Directivele #define si #include. Exemplu

Ce afișează programul?

main.c	ceva.txt
1 #include <stdio.h>	
2 #include "ceva.txt"	
3 #define MIN 0	
4	
5 int main()	
6 {	
7 int a,b;	
8 do	
9 {	
10 printf("a="); scanf("%d", &a);	
11 printf("b="); scanf("%d", &b);	
12 }while (a<=MIN b<=MIN);	
13 printf("cmmdc(%d,%d)= %d\n",a,b,divizor(a,b));	
14 return 0;	
15 }	

main.c	ceva.txt
1	int divizor(int a, int b)
2	{
3	int c=a%b;
4	while(c)
5	{ a=b; b=c; c=a%b;}
6	return b;
7	}

Directiva #define. Exemplu

main.c

```
1  #include <stdio.h>
2  // definire tip date
3  #define TIP int
4
5  // definire cod pe mai multe linii
6  #define DEBUG_PRINT printf( \
7      "File %s, line %d:\n"\
8      "x = %d, y = %d, z = %d ", \
9      __FILE__ , __LINE__ ,\
10     x,y,z)
11
12  int main()
13  {
14      TIP x=3,y=5,z;
15      x *=2;
16      y += x;
17      z = x * y;
18      DEBUG_PRINT;
19  return 0;
20 }
```

```
File main.c, line 18:
x = 6, y = 11, z = 66
```

Directiva #define. Exemplu

main.c

```
1  #include <stdio.h>
2  // definire tip date
3  #define TIP int
4
5  // definire cod pe mai multe linii
6  #define DEBUG_PRINT printf( \
7      "File %s, line %d:\n" \
8      "x = %d, y = %d, z = %d ", \
9      __FILE__ , __LINE__ , \
10     x,y,z)
11
12  int main()
13  {
14      TIP x=3,y=5,z;
15      x *=2; y += x; z = x * y;
16      DEBUG_PRINT;
17  return 0;
18  }
19
```

File main.c, line 16:
x = 6, y = 11, z = 66

Directiva #define. Exemplu

main.c

```
1  #include <stdio.h>
2
3  //constante simbolice
4  #define ALPHA 30
5  #define BETA ALPHA+10
6  #define GAMMA (ALPHA+10)
7
8  //macro-uri
9  #define MIN(a,b) ((a<b)? a:b)
10 #define ABS1(x) (x<0)?-x:x
11 #define ABS2(x) (((x)<0)?(-x):(x))
12 #define INTERSCHIMB(tip,a,b)\
13     {tip c; c=a;a=b;b=c;}
14
```

OBS. Cu paranteză
vs fără paranteză

```
70, 80
70, -30
-90, 30
-90, 30
-90, -90, 40
40, -90
```

main.c

```
14
15 int main()
16 {
17     int x=2*BETA, y=2*GAMMA;
18     printf("%d, %d\n",x,y);
19     int m=MIN(x,y);
20     printf("%d, %d\n",m,BETA-x);
21     int A=ABS1(BETA-x);
22     int A1=ABS1((BETA-x));
23     printf("%d, %d\n",A,A1);
24     int B=ABS2(BETA-x);
25     int B2=ABS2((BETA-x));
26     printf("%d, %d\n",B,B2);
27     int C=ABS2((x-ALPHA));
28     printf("%d, %d, %d\n",A,B,C);
29     INTERSCHIMB(int,A,C);
30     printf("%d, %d\n",A,C);
31
32     return 0;
33 }
```

Directiva #define. Exemplu

```
#include <stdio.h>
// definire parametru
#define DIM_VECTOR 20
// definire tip vector
#define TIP double
// definire mesaj
#define MESAJ "Calcul suma"
// definire cod pe mai multe linii
#define SEPARATOR printf( \
    "-----" \
    "\n");
TIP suma(TIP v[], int n){
    //validare lungime
    if (n > DIM_VECTOR)
        printf("dimensiunea este mai mare ca
        DIM_VECTOR\n");
    //calcul suma
    TIP suma = 0;
```

```
    for (int i = 0; i < n; i++)
        suma += v[i];
    return suma;}

int main()
{
    /*declarare vector folosind simbolurile
    specificate*/
    TIP v[DIM_VECTOR] = {1.1, 3.23, 6.62};
    int n = 3;
    //utilizare simboluri pentru mesaj
    MESAJ;
    //inserare cod prin preprocesor
    SEPARATOR;
    //apel functie
    printf("%f",suma(v, 3));
    return 0;
}
```

Directiva **#define**

- ❑ înlocuirea se continuă până în momentul în care **nume** nu mai este definit sau până la sfârșitul fișierului
- ❑ renunțarea la definirea unei constante simbolice se poate face cu directiva

#undef **nume**

Directiva **#define**

- definirea unui **macro**:

#define nume (lista-parametri) text

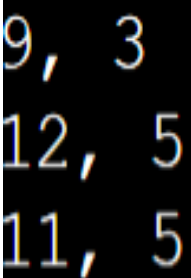
- numele macro-ului este **nume**
- lista de parametri este de ex.: **p1, p2, ..., pn**
- textul substituit este **text**
- parametrii formali sunt substituiți de cei actuali în **text**
- apelul macro-ului este similar apelului unei funcții
nume (p_actual1, p_actual2, ..., p_actualn)

Directiva #define

main.c

```
1  #include <stdio.h>
2  #define SQUARE(x) x*x
3
4  int main()
5  {
6      int a=3;
7      printf("%d, %d\n", SQUARE(a), a);
8      printf("%d, %d\n", SQUARE(a++), a);
9      printf("%d, %d\n", SQUARE(a+1), a);
10
11     return 0;
12 }
```

Ce afișează programul?



```
9, 3
12, 5
11, 5
```

Directiva #define

main.c

```
1 #include <stdio.h>
2 #define SQUARE(x) x*x
3 #define Square(x) (x)*(x)
4 int main()
5 {
6     int a=3;
7     printf("%d, %d\n", SQUARE(a), a);
8     printf("%d, %d\n", SQUARE(a++), a); //a++*(a++)
9     printf("%d, %d\n", SQUARE(a+1), a); //a+1*a+1
10    int b=3;
11    printf("%d, %d\n", Square(b), b);
12    printf("%d, %d\n", Square(b++), b); //b++*(b++)
13    printf("%d, %d\n", Square(b+1), b); //(b+1)*(b+1)
14
15    return 0;
16 }
```

Ce afișează programul?

```
9, 3
12, 5
11, 5
9, 3
12, 5
36, 5
```


Directiva #define

```
#include <stdio.h>
```

```
#define DOUBLE(x) x+x
```

```
#define Double(x) (x)+(x)
```

```
#define DoubleE(x) (x+x)
```

```
int main()
```

```
{
```

```
    int a=3;
```

```
    printf("%d, %d\n",DOUBLE(a),a);
```

```
    printf("%d, %d\n",10*DOUBLE(a),a);
```

```
    printf("%d, %d\n",10*Double(a),a);
```

```
    printf("%d, %d\n\n",10*DoubleE(a),a);
```

Ce afișează programul?

Directiva #define

```
#include <stdio.h>
```

```
#define DOUBLE(x) x+x
```

```
#define Double(x) (x)+(x)
```

```
#define DoubleE(x) (x+x)
```

```
int main()
```

```
{
```

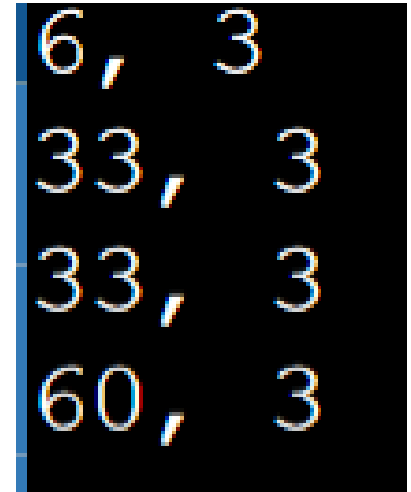
```
    int a=3;
```

```
    printf("%d, %d\n",DOUBLE(a),a); //a+a
```

```
    printf("%d, %d\n",10*DOUBLE(a),a); //10*a+a
```

```
    printf("%d, %d\n",10*Double(a),a); //10*a+a
```

```
    printf("%d, %d\n\n",10*DoubleE(a),a); //10*(a+a)
```



6,	3
33,	3
33,	3
60,	3

Directiva #define

```
#include <stdio.h>
```

Ce afișează programul?

```
#define DOUBLE(x) x+x
```

```
#define Double(x) (x)+(x)
```

```
#define DoubleE(x) (x+x)
```

```
int main()
```

```
{
```

```
    int a=3;
```

```
    printf("%d, %d\n", 10*DOUBLE(a+1), a);
```

```
    printf("%d, %d\n", 10*Double(a+1), a);
```

```
    printf("%d, %d\n\n", 10*DoubleE(a+1), a);
```

Directiva #define

```
#include <stdio.h>
```

```
#define DOUBLE(x) x+x
```

```
#define Double(x) (x)+(x)
```

```
#define DoubleE(x) (x+x)
```

```
int main()
```

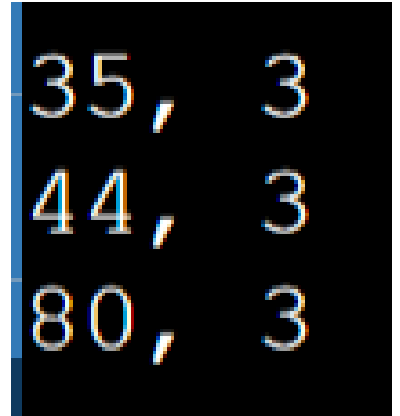
```
{
```

```
    int a=3;
```

```
    printf("%d, %d\n", 10*DOUBLE(a+1), a); //10*a+1+a+1
```

```
    printf("%d, %d\n", 10*Double(a+1), a); //10*(a+1)+(a+1)
```

```
    printf("%d, %d\n\n", 10*DoubleE(a+1), a); //10*(a+1+a+1)
```



35,	3
44,	3
80,	3

Directiva #define

```
#include <stdio.h>
```

Ce afișează programul?

```
#define DOUBLE(x) x+x
```

```
#define Double(x) (x)+(x)
```

```
#define DoubleE(x) (x+x)
```

```
int main()
```

```
{
```

```
    int a=3;
```

```
    printf("%d, %d\n",10*DOUBLE(a++),a);
```

```
    a=3; printf("%d, %d\n",10*Double(a++),a);
```

```
    a=3; printf("%d, %d\n\n",10*DoubleE(a++),a);
```

Directiva #define

```
#include <stdio.h>
```

```
#define DOUBLE(x) x+x
```

```
#define Double(x) (x)+(x)
```

```
#define DoubleE(x) (x+x)
```

```
int main()
```

```
{ int a=3;
```

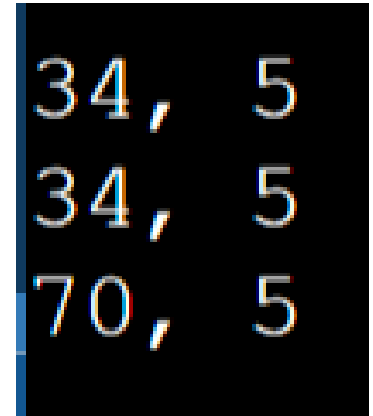
```
printf("%d, %d\n",10*DOUBLE(a++),a); //10*a++ +(a++)
```

```
a=3;
```

```
printf("%d, %d\n",10*Double(a++),a); //10*(a++)+(a++)
```

```
a=3;
```

```
printf("%d, %d\n\n",10*DoubleE(a++),a); //10*((a++)+(a++))
```



```
34, 5
34, 5
70, 5
```

Directiva #define

```
#define DOUBLE(x)  ( x ) + ( x )
```

```
a = 5;
```

```
printf(“%d \n”, 10 * DOUBLE( a ) );
```

//Echivalent cu:

```
a = 5;
```

```
printf(“%d \n”, 10 * ( a ) + ( a ) ); // => 55
```

Corectie:

```
#define DOUBLE(x)  ( ( x ) + ( x ) )
```

OBS. Toate macro-urile care evalueaza expresii numerice necesita parantezare in aceasta maniera pentru a evita interactiuni nedorite cu alti operatori

Directiva **#define**

- ❑ invocarea unui **macro** presupune înlocuirea apelului cu textul macro-ului respectiv
 - ❑ se generează astfel instrucțiuni la fiecare invocare și care sunt ulterior compilate
 - ❑ se recomandă utilizarea doar pentru calcule simple
 - ❑ parametrul formal este înlocuit cu textul corespunzător parametrului actual, corespondența fiind pur pozițională
- ❑ timpul de procesare este mai scurt când se utilizează macro-uri (apelul funcției necesită timp suplimentar)

Example

main.c

```
1  #include <stdio.h>
2
3  #define medie(a,b,c) {\
4      float m = 0;\
5      m = 0.4*(a + b) + 0.2*c;\
6  }
7
8  int main()
9  {
10     int x=2, y=3, z=4;
11     medie(x,y,z);
12     printf("medie = %f",m);
13
14     return 0;
15 }
```

input

Compilation failed due to following error(s).

```
main.c: In function 'main':
main.c:12:25: error: 'm' undeclared (first use in this function)
12 |     printf("medie = %f",m);
```

Example

main.c

```
1  #include <stdio.h>
2
3  #define medie(a,b,c) {\
4      float m = 0;\
5      m = 0.4*(a + b) + 0.2*c;\
6      return m;\
7  }
8
9  int main()
10 {
11     int x=2, y=3, z=4, m1;
12     m1=medie(x,y,z);
13     printf("medie = %f",m1);
14     return 0;
```

input

Compilation failed due to following error(s).

```
main.c: In function 'main':
main.c:3:22: error: expected expression before '{' token
3 | #define medie(a,b,c) {\
  | ^~
```

Example

main.c

```
1  #include <stdio.h>
2
3  #define medie(a,b,c) \
4      float m = 0;\
5      m = 0.4*(a + b) + 0.2*c;\
6
7  int main()
8  {
9      int x=2, y=3, z=4;
10     medie(x,y,z);
11     printf("medie = %f",m);
12
13     return 0;
14 }
15
```

medie = 2.800000

...Program finished with exit code 0
Press ENTER to exit console.

Example

main.c

```
1  #include <stdio.h>
2
3  #define medie(a,b,c) 0.4*(a + b) + 0.2*c
4
5
6  int main()
7  {
8      int x=2, y=3, z=4;
9
10     printf("medie = %f",medie(x,y,z));
11     return 0;
12 }
13 |
```

<



input

medie = 2.800000

...Program finished with exit code 0

Press ENTER to exit console.

Macro versus functii

Avantaj macro-uri:

Exista operatii pe care functiile nu le pot indeplini.

De exemplu:

```
#define MALLOC( n, type ) \
    ( (type *) malloc ( ( n ) * sizeof ( type ) ) )
```

//apel:

```
pi = MALLOC( 25, int );
```

Macro versus functii

Dezavantaj macro-uri: Efecte secundare

```
#define MAX( a , b )  ( ( a ) > ( b ) ) ? ( a ) : ( b )
```

...

```
x = 5; y = 8; z = MAX( x++ , y++ ) ;
```

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

//sau

```
z=((x++) > (y++))?(x++):(y++);
```

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

Afisare: x = 6, y = 10, z = 9

Macro versus functii

Proprietate	Macro	Funcție
Dimensiune cod	Codul macro-ului este introdus în program la fiecare apel (program în creștere)	Codul funcției apare o singură dată
Viteza execuție	Foarte rapid	Timp adițional dat de apel/return
Evaluare argumente	Argumente evaluate cu fiecare folosire în cadrul macro-ului; pot apărea efecte secundare	Argumente evaluate o singură dată (înainte de apel); nu apar efecte secundare datorate evaluărilor multiple
Tip argumente	Macro-urile nu au tip; funcționează cu orice tip de argument compatibile cu operațiile efectuate	Argumentele au tipuri: sunt necesare funcții diferite pentru tipuri diferite de argumente, chiar dacă funcțiile execută același task

Compilarea condiționată

```
1  #include <stdio.h>
2
3  #define VERSION 2
4
5  int main()
6  {
7
8      #if VERSION == 1
9      {
10         printf ("versiunea 1 \n");
11         printf ("Adaugam modulele pentru versiunea 1 ... \n");
12         // continua cu includerea diverselor module pentru versiunea 1
13     }
14
15     #elif VERSION == 2
16     {
17         printf ("versiunea 2 \n");
18         printf ("Adaugam modulele pentru versiunea 2 ... \n");
19         // continua cu includerea diverselor module pentru versiunea 2
20     }
21     #elif VERSION == 3
22     {
23         printf ("versiunea 3 \n");
24         printf ("Adaugam modulele pentru versiunea 3 ... \n");
25         // continua cu includerea diverselor module pentru versiunea 3
26     }
27
28     #endif
29     return 0;
30
31 }
```


Compilarea condiționată

- ❑ facilitează dezvoltarea dar în special testarea codului
- ❑ directivele care pot fi utilizate: **#if**, **#ifdef**, **#ifndef**

- ❑ directiva **#if**:

```
#if expr
    text
#endif
```

```
#if expr
    text1
#else  (#elif)
    text2
#endif
```

- ❑ unde **expr** este o expresie constantă care poate fi evaluată de către preprocesor,
- ❑ **text**, **text1**, **text2** sunt porțiuni de cod sursă
- ❑ dacă **expr** nu este zero atunci **text** respectiv **text1** sunt compilate, altfel numai **text2** este compilat și procesarea continuă după **#endif**

Compilarea condiționată

□ directiva **#ifdef**:

```
#ifdef nume
    text
#endif
```

```
#ifdef nume
    text1
#else
    text2
#endif
```

- unde **nume** este o constantă care este testată de către preprocesor dacă este definită,
- **text**, **text1**, **text2** sunt porțiuni de cod sursă
- dacă **nume** este definită atunci **text** respectiv **text1** sunt compilate, altfel numai **text2** este compilat și procesarea continuă după **#endif**

Compilarea condiționată

□ Exemplu:

```
#include <stdio.h>

#define medie(a,b,c) 0.4 * (a + b) + 0.2 * c

int main()
{
    int x = 2, y = 3, z = 4;

    #ifdef medie
    printf("medie = %f ", medie(x , y , z));
    #else
    printf("medie extra = %f ", 0.4 * (x + y) + 0.2 * z );
    #endif

    return 0;
}
```

Compilarea condiționată

□ directiva `#ifndef`:

<code>#ifndef nume</code>	<code>#ifndef nume</code>
<code>text</code>	<code>text1</code>
<code>#endif</code>	<code>#else</code>
	<code>text2</code>
	<code>#endif</code>

- unde `nume` este o constantă care este testată de către preprocesor dacă `NU este definită`,
- `text`, `text1`, `text2` sunt porțiuni de cod sursă
- dacă `nume` NU este definită atunci `text` respectiv `text1` sunt compilate, altfel numai `text2` este compilat și procesarea continuă după `#endif`

Compilarea condiționată

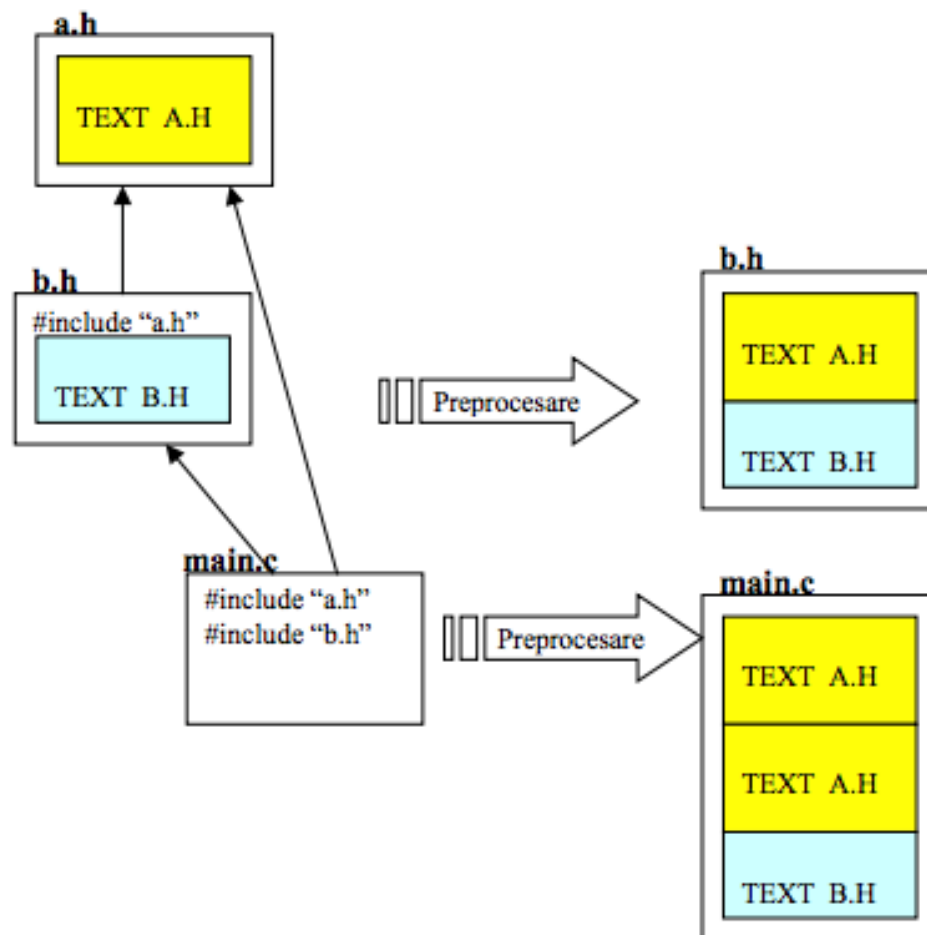
- ❑ directivele **#ifdef** și **#ifndef** sunt folosite de obicei pentru a evita incluziunea multiplă a modulelor în programarea modulară

- ❑ fișier antet “a.h”

- ❑ fișier antet “b.h”

- ❑ include pe “a.h”

- ❑ main include “a.h” și “b.h”



Compilarea condiționată

- ❑ directivele `#ifdef` și `#ifndef` sunt folosite de obicei pentru a evita incluziunea multiplă a modulelor în programarea modulară
- ❑ fișier antet “a.h”
- ❑ fișier antet “b.h”
- ❑ la începutul fiecărui fișier *header* se practică de obicei o astfel de secvență

```
#ifndef    _MODUL_A_  
#define    _MODUL_A_  
...  
#endif /* _MODUL_A_ */
```

Macro-uri predefinite

- există o serie de macro-uri predefinite care nu trebuie re/definite:

__DATE__

-data compilării

__CDECL__

-apelul funcției urmărește convențiile C

__STDC__

-definit dacă trebuie respectate strict regulile ANSI – C

__FILE__

-numele complet al fișierului curent compilat

__FUNCTION__

-numele funcției curente

__LINE__

-numărul liniei curente

Macro-uri predefinite

```
#include <stdio.h>
//constante simbolice
#define DEBUG
#define X -3
#define Y 5

int main()
{
#ifdef DEBUG
    printf("Suntem in functia %s\n", __FUNCTION__); //main
#endif
#if X+Y
    double a=3.1;
#else
    double a=5.7;
#endif
    a*=2;
#ifdef DEBUG
    printf("La linia %d valoarea lui a este %f\n", __LINE__, a); //18 6.2
#endif
    a+=10;
    printf("a este %f", a); //16.2
    return 0;
}
```


Cursul de azi

1. Etapele realizării unui program C
2. Directive de preprocesare.
Macrodefiniții
3. Funcții de citire / scriere

Funcții de citire și scriere

- ❑ operații de citire și scriere în C:
 - ❑ de la tastatură (stdin) și la ecran (stdout);
 - ❑ prin fișiere;
 - ❑ efectuate cu ajutorul funcțiilor de bibliotecă
- ❑ citirea de la tastatură și scrierea la ecran:
 - ❑ fără formatare:
getchar, putchar, getch, getche, putch, gets, puts
 - ❑ cu formatare:
scanf, printf
 - ❑ incluse în biblioteci:
 - ❑ **stdio.h** (getchar, putchar, gets, puts, scanf, printf)
 - ❑ **conio.h** (getch, getche, putch)

OBS: CODE::BLOCKS nu include biblioteca conio.h

Funcțiile getchar și putchar

- operații de citire și scriere a caracterelor:

- **int getchar(void)**

- citește un caracter de la tastatură

- așteaptă până este apasată o tastă și returnează valoarea sa → tasta apăsată are imediat ecou pe ecran.

- **int putchar(int c)**

- scrie un caracter pe ecran în poziția curentă a cursorului

- fișierul antet pentru aceste funcții: **stdio.h**

Funcțiile gets și puts

□ operații de citire și scriere a șirurilor de caractere:

□ `char *gets(char *s)`

- citește caractere din stdin și le depune în zona de date de la adresa `s`, până la apăsarea tastei Enter. În șir, tastei Enter îi va corespunde caracterul `'\0'`

- dacă operația de citire reușește, funcția întoarce adresa șirului, altfel valoarea `NULL (= 0)`

□ `int puts(const char *s)`

- scrie pe ecran șirul de la adresa `s` sau o constantă șir de caractere și apoi trece la linie nouă
- dacă operația de scriere reușește, funcția întoarce ultimul caracter, altfel valoarea `EOF (-1)`

□ fișierul antet pentru aceste funcții este `stdio.h`

Funcțiile gets și puts

```
main.c x
1  #include <stdio.h>
2  #include <stdlib.h>
3
4
5  int main(){
6
7      char sir[10];
8
9      printf("Ce zi e astazi: ");
10     gets(sir);
11     puts(sir);
12
13     puts("Alegeti DA sau NU. Optiunea dumneavoastra este: ");
14     gets(sir);
15     puts(sir);
16
17     Ce zi e astazi: joi
18     joi
19     Alegeti DA sau NU. Optiunea dumneavoastra este:
20     DA
    DA

    Process returned 0 (0x0)    execution time : 3.418 s
    Press ENTER to continue.
```

Funcția gets. Dezavantaje

- ❑ **char *gets(char *s)**
- ❑ primește ca input numai un buffer (s), nu stim dimensiunea lui
- ❑ problema de buffer overflow: citim în s mai mult decât dimensiunea lui, gets nu ne împiedică, scrie datele în alta parte
- ❑ Soluție: folosiți fgets: **char *fgets(char *s, int size, FILE *stream)**
 - ❑ fgets(buffer, sizeof(buffer), stdin);

OBS. În standardul C11 funcția gets este eliminată

Funcțiile printf și scanf

- ❑ funcții de citire (scanf) și scriere (printf) cu formatare;
- ❑ formatarea specifică conversia datelor de la reprezentarea externă în reprezentarea internă (scanf) și invers (printf);
- ❑ formatarea se realizează pe baza descriptorilor de format

`%[flags][width][.precision][length]specifier`

- ❑ detalii aici:

<http://www.cplusplus.com/reference/cstdio/printf/>

Funcțiile printf și scanf

- formatarea se realizează pe baza descriptorilor de format

`%[flags][width][.precision][length]specifier`

Specificator de format	Reprezentare
<code>%c</code>	caracter
<code>%s</code>	șir de caractere
<code>%d, %i</code>	întreg în zecimal
<code>%u</code>	întreg în zecimal fără semn
<code>%o</code>	întreg în octal
<code>%x</code>	întreg în hexazecimal fără semn (litere mici)
<code>%X</code>	întreg în hexazecimal fără semn (litere mari)
<code>%f</code>	număr real în virgulă mobilă
<code>%e, %E</code>	notație științifică - o cifră la parte întreagă
<code>%ld, %li, %lu, %lo, %lx</code>	cu semnificațiile de mai sus, pentru întregi lungi
<code>%p</code>	pointer

Funcția printf

□ prototipul funcției:

*int printf(const char *format, argument1, argument2, ...);*

unde:

- **format** este un șir de caractere ce definește textele și formatele datelor care se scriu pe ecran
- **argument1, argument2,...** sunt expresii. Valorile lor se scriu pe ecran conform specificatorilor de format prezenți în format
- funcția **printf** întoarce numărul de octeți transferați sau EOF (-1) în caz de eșec.

Funcția printf

```
main.c x
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(){
5
6      char sir[10] = "Azi e joi";
7      printf("Primul caracter din sirul \"%s\" este %c \n",sir,sir[0]);
8      int x = 1234;
9      printf("Reprezentare lui x in baza 10: x=%d\n",x);
10     printf("Reprezentare lui x in baza 8: x=%o\n",x);
11     printf("Reprezentare lui x in baza 16 (litere mici): x=%x\n",x);
12     printf("Reprezentare lui x in baza 16 (litere mari): x=%X\n",x);
13
14     float y = 12.34;
15     printf("Reprezentare lui y ca numar real: y=%f\n",y);
16     printf("Reprezentare lui y in notatie stiintifica: y=%e\n",y);
17     printf("Reprezentare lui y in notatie stiintifica: y=%E\n",y);
18
19     return 0;
20 }
21
```

Primul caracter din sirul "Azi e joi" este A
Reprezentare lui x in baza 10: x=1234
Reprezentare lui x in baza 8: x=2322
Reprezentare lui x in baza 16 (litere mici): x=4d2
Reprezentare lui x in baza 16 (litere mari): x=4D2
Reprezentare lui y ca numar real: y=12.340000
Reprezentare lui y in notatie stiintifica: y=1.234000e+01
Reprezentare lui y in notatie stiintifica: y=1.234000E+01

Process returned 0 (0x0) execution time : 0.004 s
Press ENTER to continue.

Funcția printf

```
main.c x
1  #include <stdio.h>
2  #include <stdlib.h>
3
4
5  int main(){
6
7      int a = printf("Ce zi e astazi: ");
8      printf("\n a = %d \n",a);
9
10     a = printf("Alegeti DA sau NU. Optiunea dumneavoastra este: ");
11     printf("\n a = %d \n",a);
12
13     return 0;
14 }
15
```

Ce zi e astazi:

a = 16

Alegeti DA sau NU. Optiunea dumneavoastra este:

a = 48

Process returned 0 (0x0) execution time : 0.005 s

Press ENTER to continue.



Modelatori de format

- ❑ mulți specificatori de format pot accepta modelatori care modifică ușor semnificația lor:
 - ❑ alinierea la stânga
 - ❑ minim de mărime a câmpului
 - ❑ numărul de cifre zecimale
- ❑ modelatorul de format se află între semnul procent și codul pentru format:
 - ❑ caracterul ‘—’ specifică aliniere la stânga;
 - ❑ șir de cifre zecimale specifică dimensiunea câmpului pentru afișare
 - ❑ caracterul ‘.’ urmat de cifre specifică precizia reprezentării

Modelatorul flags

- formatarea se realizează pe baza descriptorilor de format
 - %[flags][width][.precision][length]specifier

flags	description
-	Left-justify within the given field width; Right justification is the default (see <i>width</i> sub-specifier).
+	Forces to precede the result with a plus or minus sign (+ or -) even for positive numbers. By default, only negative numbers are preceded with a - sign.
(space)	If no sign is going to be written, a blank space is inserted before the value.
#	Used with o, x or X specifiers the value is preceeded with 0, 0x or 0X respectively for values different than zero. Used with a, A, e, E, f, F, g or G it forces the written output to contain a decimal point even if no more digits follow. By default, if no digits follow, no decimal point is written.
0	Left-pads the number with zeroes (0) instead of spaces when padding is specified (see <i>width</i> sub-specifier).

Modelatorul width

- formatarea se realizează pe baza descriptorilor de format
 - %[flags][width][.precision][length]specifier

width	description
(number)	Minimum number of characters to be printed. If the value to be printed is shorter than this number, the result is padded with blank spaces. The value is not truncated even if the result is larger.
*	The <i>width</i> is not specified in the <i>format</i> string, but as an additional integer value argument preceding the argument that has to be formatted.

Modelatorul precision

- formatarea se realizează pe baza descriptorilor de format
 - %[flags][width][.precision][length]specifier

.precision	description
.number	<p>For integer specifiers (d, i, o, u, x, X): <i>precision</i> specifies the minimum number of digits to be written. If the value to be written is shorter than this number, the result is padded with leading zeros. The value is not truncated even if the result is longer. A <i>precision</i> of 0 means that no character is written for the value 0.</p> <p>For a, A, e, E, f and F specifiers: this is the number of digits to be printed after the decimal point (by default, this is 6).</p> <p>For g and G specifiers: This is the maximum number of significant digits to be printed.</p> <p>For s: this is the maximum number of characters to be printed. By default all characters are printed until the ending null character is encountered.</p> <p>If the period is specified without an explicit value for <i>precision</i>, 0 is assumed.</p>
.*	The <i>precision</i> is not specified in the <i>format</i> string, but as an additional integer value argument preceding the argument that has to be formatted.

Modelatorul length

- formatarea se realizează pe baza descriptorilor de format
 - %[flags][width][.precision][length]specifier

	specifiers						
length	d i	u o x X	f F e E g G a A	c	s	p	n
(none)	int	unsigned int	double	int	char*	void*	int*
hh	signed char	unsigned char					signed char*
h	short int	unsigned short int					short int*
l	long int	unsigned long int		wint_t	wchar_t*		long int*
ll	long long int	unsigned long long int					long long int*
j	intmax_t	uintmax_t					intmax_t*
z	size_t	size_t					size_t*
t	ptrdiff_t	ptrdiff_t					ptrdiff_t*
L			long double				

Modelatori de format pentru printf

```
main.c x
1  #include <stdio.h>
2  #include <stdlib.h>
3
4
5  int main(){
6
7      double numar;
8      numar = 10.1234;
9
10     printf("numar=%f\n", numar);
11     printf("numar=%10f\n", numar);
12     printf("numar=%012f\n", numar);
13
14     printf("%.4f\n", 123.1234567);
15     printf("%3.8d\n", 1000);
16     printf("%10d\n", 1000);
17     printf("%-10d\n", 1000);
18     printf("%10.15s\n", "Acesta este un test simplu");
19
20     return 0;
21 }
22
```

numar=10.123400
numar= 10.123400
numar=00010.123400
123.1235
00001000
1000
1000
Acesta este un

Funcția printf

□ exemplu:

```
printf("valoarea lui x este: %-4.2f\n",3.14);  
printf("x=%i, y=%f, x=%o, x=%#x\n",15,3.14,15,15);  
printf("c= %c, c=%d\n",'%','%');  
printf("sir de caractere: %s\n", "ana are mere");  
printf("\\ \\ \" \' \n");
```

```
valoarea lui x este: 3.14  
x=15, y=3.140000, x=17, x=0xf  
c= %, c=37  
sir de caractere: ana are mere  
\\ \" '
```

Funcția scanf

□ prototipul funcției:

int scanf(const char * format ,adresa1, adresa2, ...);

unde:

- **format** este un șir de caractere ce definește textele și formatele datelor care se citesc de la tastatură
- **adresa1, adresa2,...** sunt adresele zonelor din memorie în care se păstrează datele citite după ce au fost convertite din reprezentarea lor externă în reprezentare internă.
- funcția **scanf** întoarce numărul de câmpuri citite și depuse la adresele din listă. Dacă nu s-a stocat nici o valoare, funcția întoarce 0.

Funcția scanf

- ❑ șirul de formatare (format) poate include următoarele elemente:
 - ❑ **spațiu alb**: funcția citește și ignoră spațiile albe (spațiu, tab, linie nouă) înaintea următorului caracter diferit de 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

Funcția scanf

main.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      int n,a;float m;
7      scanf("%d %f",&n,&m);
8      printf("n=%d\n m=%f\n",n,m);
9
10     printf("n=");
11     scanf("%d",&n);
12
13     printf("m=");
14     a=scanf("%f",&m);
15     printf("a=%d",a);
16
17     return 0;
18 }
```

```
n=25 m=3.2
n=25
m=3.200000
n=100
m=i37
a = 0
```

Funcția scanf

main.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      int a,b;char c;
7      for(;;)
8      {
9          printf("Introduceti 2 numere intregi\n");
10         if(scanf("%d %d",&a,&b)==2)
11             break;
12         else
13             while(c=getchar()!='\n'&&c!=EOF);
14     }
15
16     printf("am citit 2 numere: %d si %d\n",a,b);
17
18     return 0;
19 }
```

```
Introduceti 2 numere intregi
3 a
Introduceti 2 numere intregi
4 2 3
am citit 2 numere: 4 si 2
```

Cursul 5

- 1. Etapele realizării unui program C**
- 2. Directive de preprocesare. Macrodefiniții**
- 3. Funcții de citire/scriere**

Cursul 6

1. Fișiere: noțiuni generale
2. Fișiere text: funcții specifice de manipulare
3. Funcții