PROGRAMAREA CALCULATOARELOR

Andrei Patrascu

andrei.patrascu@fmi.unibuc.ro

Secția Calculatoare si Tehnologia Informatiei, anul I, 2018-2019

Cursul 5

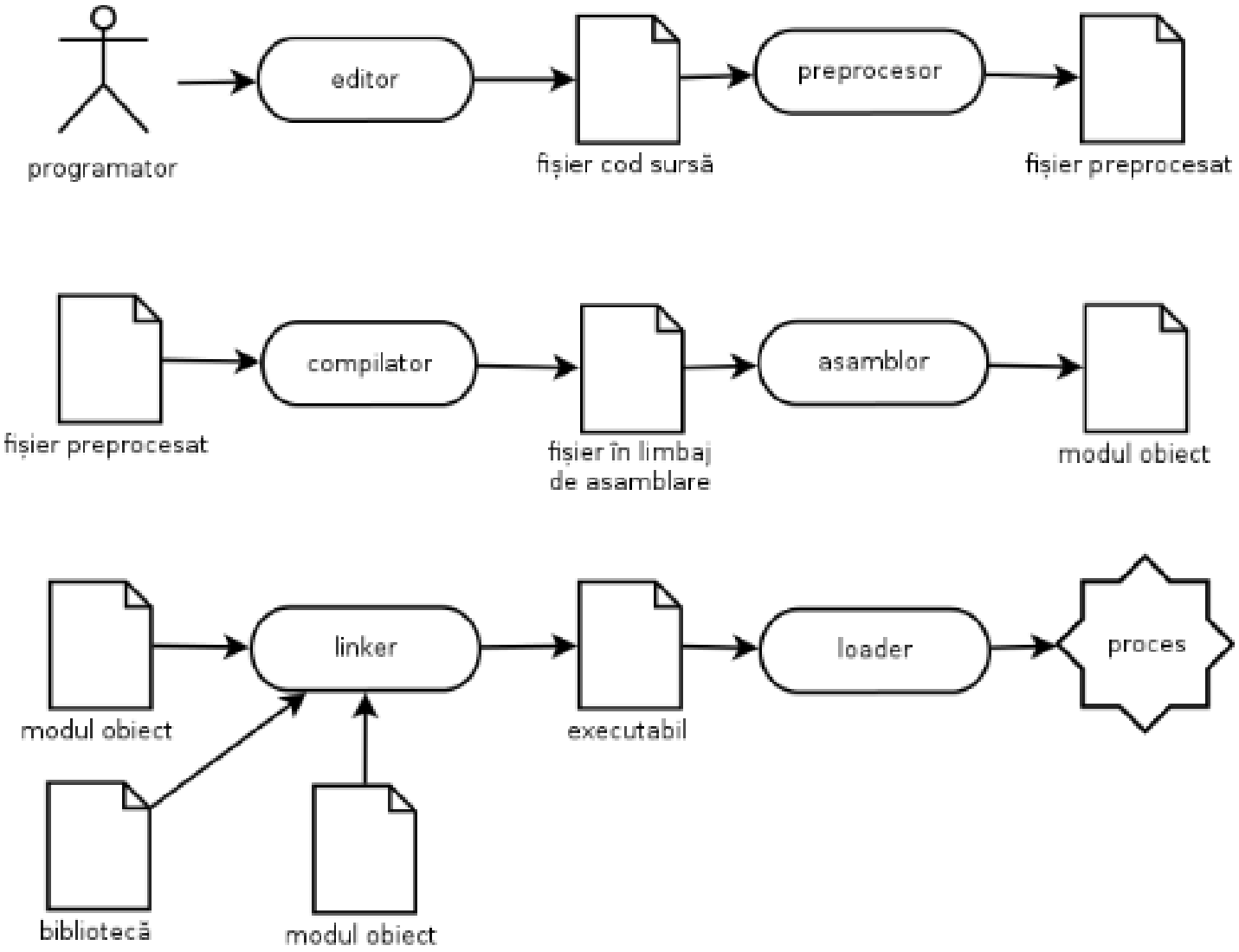
PROGRAMA CURSULUI

- **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: tablouri, șiruri de caractere, structuri, uniuni, câmpuri de biți, enumerări, pointeri
  - Instrucțiuni de control
  - Directive de preprocesare. Macrodefiniții.
  - Funcții de citire/scriere.
  - Etapele realizării unui program C.
- **Fișiere text**
  - Funcții specifice de manipulare.
- **Funcții (1)**
  - Declarare și definire. Apel. Metode de trasmitere a paramerilor. 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.

CUPRINSUL CURSULUI 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

- se parcurg următoarele etape pentru obținerea unui cod executabil:
  - **editarea** codului sursă
    - salvarea fișierului cu extensia *.c*
  - **preprocesarea**
    - efectuarea directivelor de preprocesare (**#include**, **#define**)
    - ca un editor – modifică și adaugă la codul sursă
  - **compilarea**
    - verificarea sintaxei
    - codul este tradus din cod de nivel înalt în limbaj de asamblare
  - **asamblarea**
    - transformare în cod obiect (limbaj mașină) cu extensia *.o, .obj*
      - nu este încă executabil !
  - l**ink-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

- **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

- **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;      .scl    2;      .type   32;     .endef
_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;     .scl    2;      .type   32;     .endef
_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

- **Asamblarea** gcc –c program.c (produce program.o)

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

CUPRINSUL CURSULUI DE AZI

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

PREPROCESARE ÎN LIMBAJUL C

- preprocesarea 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 #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

DIRECTIVA #DEFINE

- Exemplu:

- #define DEBUG_PRINT printf(" File %s line %d: \
" x = %d, y = %d, z = %d ", \
__FILE__ , __LINE__ ,\
x,y,z)

```
…..
x *=2;
y += x;
z = x * y;
DEBUG_PRINT;
```

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

#define SQUARE(x) x * x

Daca apelam: SQUARE(5), atunci in program preprocesorul substituie:

5 * 5

Ce afiseaza programul?

```
a = 5;
printf("%d \n", SQUARE( a + 1 ) );
```

DIRECTIVA #DEFINE

```
#define SQUARE(x)     x * x
```

Daca apelam: SQUARE(5), atunci in program preprocesorul substituie:

```
5 * 5
```

Ce afiseaza programul?

```
 a = 5;
printf("%d \n", SQUARE( a + 1 ) );    //echivalent cu a+1*a+1
                                      //  deci => 11
```

DIRECTIVA #DEFINE

Corectie:

#define SQUARE(x) (x) * (x)

Cand apelam: printf(“%d \n”, SQUARE(a + 1))

Preprocesorul inlocuieste: printf(“%d \n”, (a + 1)*(a + 1))

DIRECTIVA #DEFINE

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

Ce afiseaza programul?

```
  a = 5;
 printf(“%d \n”, 10 * DOUBLE( a ) );
```

DIRECTIVA #DEFINE

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

Ce afiseaza programul?

```
 a = 5;
printf("%d \n", 10 * DOUBLE( a ) );

//Echivalent cu:
 a = 5;
 printf("%d \n", 10 * ( a ) + ( a ) ); // => 55
```

DIRECTIVA #DEFINE

Corectie:

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

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 astfel 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)

EXEMPLE

- Ce afiseaza programul?

```c
#include <stdio.h>

#define medie(a,b,c) {\
float m = 0; \
m = 0.4 * (a + b) + 0.2 *c \
}

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

    medie(x,y,z);

    printf("medie = %f ",m);

    return 0;
}
```

EXEMPLE

- Ce afiseaza programul?

```
#include <stdio.h>

#define medie(a,b,c) {\
float m = 0; \
m = 0.4 * (a + b) + 0.2 *c \
}

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

    medie(x,y,z);

    printf("medie = %f ",m);
```

```
note: in expansion of macro 'medie'
error: 'm' undeclared (first use in this function)
note: each undeclared identifier is reported only once for each function it app
=== Build failed: 2 error(s), 1 warning(s) (0 minute(s), 0 second(s)) ===
```

EXEMPLE

```c
#include <stdio.h>

#define medie(a,b,c) \
float m = 0; \
m = 0.4 * (a + b) + 0.2 *c \


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

    medie(x,y,z);

    printf("medie = %f ",m);

    return 0;
}
```

EXEMPLE

```c
#include <stdio.h>

#define medie(a,b,c) \
float m = 0; \
m = 0.4 * (a + b) + 0.2 *c \


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

    medie(x,y,z);
```

```
medie = 2.800000
Process returned 0 (0x0)   execution time : 3.041 s
Press any key to continue.
```

EXEMPLE

```
#define medie(a,b,c) {\
float m = 0; \
m = 0.4 * (a + b) + 0.2 *c \
return m;\
}

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

    ml = medie(x,y,z);

    printf("medie = %f ",ml);

    return 0;
}
```

EXEMPLE

```c
#include <stdio.h>

#define medie(a,b,c) {\
float m = 0; \
m = 0.4 * (a + b) + 0.2 *c \
return m;\
}

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

    ml = medie(x,y,z);
```

```
In function 'main':
error: expected expression before '{' token
note: in expansion of macro 'medie'
warning: format '%f' expects argument of type 'double', but argument 2 has ty
warning: unused variable 'z' [-Wunused-variable]
```

EXEMPLE

```c
#include <stdio.h>

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

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

    printf("medie = %f ",medie(x,y,z));

    return 0;
}
```

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 );
```

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

MACRO VERSUS FUNCTII

Proprietate	Macro	Functie
Dimensiune cod	Codul macro-ului este introdus in program la fiecare apel (program in crestere)	Codul functiei apare o singura data
Viteza executie	Foarte rapid	Timp aditional dat de apel/return
Evaluare argumente	Argumente evaluate cu fiecare folosire in cadrul macro-ului; pot aparea efecte secundare	Argumente evaluate o singura data (inainte de apel); nu apar efecte secundare ale datorate evaluarilor multiple
Tip argumente	Macro-urile nu au tip; functioneaza cu orice tip de argument compatibile cu operatiile efectuate	Argumentele au tipuri: sunt necesare functii diferite pentru tipuri diferite de argumente, chiar daca functiile executa acelasi task

COMPILAREA CONDIȚIONATĂ

```c
#include <stdio.h>

#define VERSION 2

int main()
{

    #if VERSION == 1
        {
        printf ("versiunea 1 \n");
        printf ("Adaugam modulele pentru versiunea 1 ... \n");
        // continua cu includerea diverselor module pentru versiunea 1
        }

    #elif VERSION == 2
        {
        printf ("versiunea 2 \n");
        printf ("Adaugam modulele pentru versiunea 2 ... \n");
        // continua cu includerea diverselor module pentru versiunea 2
        }
    #elif VERSION == 3
        {
        printf ("versiunea 3 \n");
        printf ("Adaugam modulele pentru versiunea 3 ... \n");
        // continua cu includerea diverselor module pentru versiunea 3
        }

    #endif
    return 0;

}
```

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 dupa **`#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 dupa **`#endif`**

COMPILAREA CONDIȚIONATĂ

- Exemplu lab:

```c
#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`**:

```
#ifndef nume
     text
#endif
```

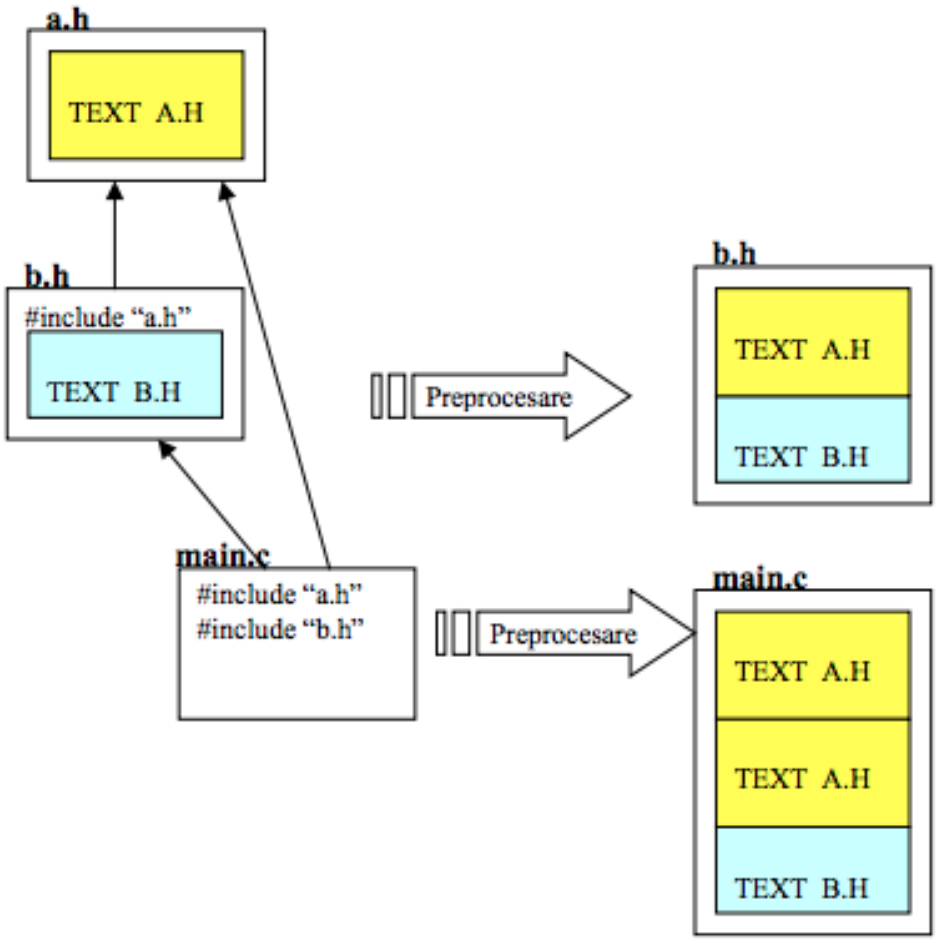
```
#ifndef nume
     text1
#else
     text2
#endif
```

  - 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 dupa **`#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

```
#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;
}
```

CUPRINSUL CURSULUI 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

- funcții pentru **citirea de la tastatură** și **scrierea la ecran**
  - fără formatare: getchar, putchar, getch, getche, putch, gets, puts
  - cu formatare: scanf, printf
  - incluse în bibliotecile stdio.h (getchar, putchar, gets, puts, scanf, printf) sau conio.h (getch, getche, putch)
    - 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 este **stdio.h**.

FUNCȚIILE GETCHAR ȘI PUTCHAR

- exemplu:

main.c

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


int main(){

    int optiune;

    printf("Alegeti DA sau NU. Optiunea dumneavoastra este : ");
    optiune = getchar();
    putchar(optiune);

    return 0;
}
```

```
Alegeti DA sau NU. Optiunea dumneavoastra este : DA
D
Process returned 0 (0x0)   execution time : 1.321 s
Press ENTER to continue.
```

FUNCȚIILE GETS ȘI PUTS

- operații de **citire** și **scriere** a șirurilor de caractere:
  - **char *gets(char *s)** – citește caractere din **stdin** si 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

- exemplu:

main.c

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


int main(){

    char sir[10];

    printf("Ce zi e astazi: ");
    gets(sir);
    puts(sir);

    puts("Alegeti DA sau NU. Optiunea dumneavoastra este: ");
    gets(sir);
    puts(sir);

    return 0;
}
```

```
Ce zi e astazi: joi
joi
Alegeti DA sau NU. Optiunea dumneavoastra este:
DA
DA

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

DE CE SĂ NU FOLOSIȚI FUNCȚIA GETS

- **char *gets(char *s)**
- primeste ca input numai un buffer (s), nu stim dimensiunea lui
- problema de buffer overflow: citim in s mai mult decat dimensiunea lui, gets nu ne impiedica, scrie datele in alta parte
- folositi fgets: **char *fgets(char *s, int size, FILE *stream)**
  - fgets(buffer, sizeof(buffer), stdin);
- in standardul C11 functia gets este eliminata

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

- functia **printf** întoarce numărul de octeți transferați sau EOF (-1) în caz de eșec.

FUNCȚIA PRINTF

main.c

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

int main(){

    char sir[10] = "Azi e joi";
    printf("Primul caracter din sirul \"%s\" este %c \n",sir,sir[0]);
    int x = 1234;
    printf("Reprezentare lui x in baza 10: x=%d\n",x);
    printf("Reprezentare lui x in baza 8: x=%o\n",x);
    printf("Reprezentare lui x in baza 16 (litere mici): x=%x\n",x);
    printf("Reprezentare lui x in baza 16 (litere mari): x=%X\n",x);

    float y = 12.34;
    printf("Reprezentare lui y ca numar real: y=%f\n",y);
    printf("Reprezentare lui y in notatie stiintifica: y=%e\n",y);
    printf("Reprezentare lui y in notatie stiintifica: y=%E\n",y);

    return 0;
}
```

```
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

- **exemplu:**

main.c

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


int main(){

    int a = printf("Ce zi e astazi: ");
    printf("\n a = %d \n",a);

    a = printf("Alegeti DA sau NU. Optiunea dumneavoastra este: ");
    printf("\n a = %d \n",a);

    return 0;
}
```

```
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 *width* sub-specifier).
+	Forces to preceed 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 *width* 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 *width* is not specified in the *format* 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	For integer specifiers (`d`, `i`, `o`, `u`, `x`, `X`): *precision* 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 *precision* of `0` means that no character is written for the value `0`. 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). For `g` and `G` specifiers: This is the maximum number of significant digits to be printed. 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. If the period is specified without an explicit value for *precision*, `0` is assumed.
.*	The *precision* is not specified in the *format* 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

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


int main(){

    double numar;
    numar = 10.1234;

    printf("numar=%f\n",numar);
    printf("numar=%10f\n",numar);
    printf("numar=%012f\n",numar);

    printf("%.4f\n",123.1234567);
    printf("%3.8d\n",1000);
    printf("%10d\n",1000);
    printf("%-10d\n",1000);
    printf("%10.15s\n","Acesta este un test simplu");

    return 0;
}
```

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

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

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 si 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

- exemplu:

main.c

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


int main(){

   int n,a;
   float m;

   scanf("n=%d m=%f",&n,&m);
   printf("n=%d\nm=%f\n",n,m);

   printf("n=");
   scanf("%d",&n);

   printf("m=");
   a = scanf("%f",&m);
   printf("a = %d \n",a);

   return 0;
}
```

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

Process returned 0 (0x0)   execution time : 14.176 s
Press ENTER to continue.
```

FUNCȚIA SCANF

- exemplu:

```
int main()
{
        int a, b;
        char c;
        for(;;)
        {
                printf("Introduceti 2 numere intregi\n");
                if(scanf("%d%d",&a,&b)==2)
                        break;
                else
                        while(c=getchar()!='\n' && c!=EOF);


        }
        printf("am citit 2 numere: %d si %d\n",a,b);
        return 0;
}
```

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