



Programare procedurala

- suport de curs -

Dobrovat Anca - Madalina

**An universitar 2016 – 2017
Semestrul I**

Curs 5



Agenda cursului

- 1. Scurta recapitulare a cursului anterior:**
Expresii si operatori – evaluare, precedenta;
Operatorii pe biti;
- 2. Directive de preprocesare. Macrodefinitii si compilare conditionata**
- 3. Tipuri structurate de date (partea 1 – tablouri uni- si bi- dimensionale)**



Recapitulare curs 4 – Reprezentarea datelor in memorie. Operatii si expresii

Operatii pe biti

1. Negatia (complementul fata de 1) \sim : $\sim(-47) = 46$
2. Conjunctia $\&$: $17 \& 18 = 16$
3. Disjunctia $|$: $17 | 18 = 19$
4. XOR (“sau exclusiv”) \wedge : $17 \wedge 18 = 3$
5. shl (shift left) \ll : $100 \ll 4 = 1600$
6. shr (shift right) \gg : $100 \gg 4 = 6$



Recapitulare curs 4 – Reprezentarea datelor in memorie. Operatii si expresii

Aplicatii

3. Analizati programul urmator. Daca este corect spuneti ce se va afisa, in caz contrar explicati de unde provine eroarea .

```
int main()
{
    short int a = -65000, b = 10;
    printf("\n %d \n", a);
    float f = (float)a/b;
    f = (int)(f*100);
    f = f/100;
    float c = 40000, d = 10;
    int g = (int)c/d;
    g = (float)(g*100);
    g = g/100;
    printf("%.2f  %d", f, g);
}
```

R.
536
53.6 4000



Recapitulare curs 4 – Reprezentarea datelor in memorie. Operatii si expresii

Expresii si operatori – evaluare, precedenta

Expresii aritmetice si operatori aritmetici

Cei mai prioritari	+	(plus unar)
	-	(minus unar)
	*	(înmulțire)
	/	(împărțire)
	%	(restul împărțirii)
Cei mai puțin prioritari	+	(adunare)
	-	(scădere)

Asociativitatea operatorilor aritmetici

1. **Unari (asociativi la dreapta)** – Gruparea de la dreapta la stanga
2. **Binari (asociativi la stanga)** – Gruparea de la stanga la dreapta



Recapitulare curs 4 – Reprezentarea datelor in memorie. Operatii si expresii

Expresii si operatori – evaluare, precedenta

Operatori de atribuire

Asociativi la dreapta

$$A = B = C = 7 \quad \Leftrightarrow \quad A = (B = (C = 7))$$

Operatori de atribuire compusi

$+=$, $-=$, $*=$, $/=$, $\%=$, etc. (combinati cu operatorii pe biti)

- **In evaluarea expresiilor conteaza ordinea de precedenta**

$$A *= B + C \quad \Leftrightarrow \quad A = A * (B + C)$$



Recapitulare curs 4 – Reprezentarea datelor in memorie. Operatii si expresii

Expresii si operatori – evaluare, precedenta

Operatori de incrementare / decrementare

Preincrementarea / Predecrementarea au aceeasi prioritate ca operatorii unari si sunt asociativi la dreapta.

Postincrementarea / Postdecrementarea au prioritate crescuta fata de operatorii unari si sunt asociativi la stanga.

Operatori relationali

Asociativi la stanga si au **prioritate mai mica** decat operatorii aritmetici.

Operatori de egalitate ==, !=

Asociativi la stanga si au **prioritate mai mica** decat operatorii relationali.



Recapitulare curs 4 – Reprezentarea datelor in memorie. Operatii si expresii

Expresii si operatori – evaluare, precedenta

Operatori logici: !, &&, ||

! (NOT) are prioritate egala cu operatorii unari.

&& (AND) si || (OR) au prioritate mai mica decat operatorii relationali si de egalitate.

Operatori pe biti

Ordinea de precedenta: ~, <<, >>, &, ^, |



Recapitulare curs 4 – Reprezentarea datelor in memorie. Operatii si expresii

Expresii si operatori – evaluare, precedenta

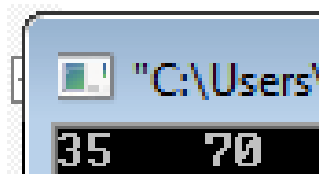
Operatorul virgula

Cea mai mica prioritate!

Expl

```
int main()
{
    int x = 12, y = 23, s, p;

    s = x+y, p = s * 2, printf("%d %d", s, p);
}
```

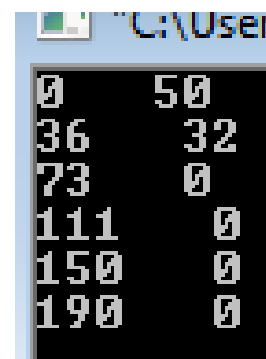


"C:\Users\
35 70

Expl

```
int main()
{
    int x, y, s, p, i;

    for(x = 12, y = 23, s = 0, i = x + y; i <= 40; i++, s += i, p &= s)
        printf("%d %d\n", s, p);
}
```



"C:\Users\
0 50
36 32
73 0
111 0
150 0
190 0



Recapitulare curs 4 – Reprezentarea datelor in memorie. Operatii si expresii

Ordinea de precedență și asociativitate

Operator	Description	Associativity
()	Parentheses (function call) (see Note 1)	left-to-right
[]	Brackets (array subscript)	
.	Member selection via object name	
->	Member selection via pointer	
++ --	Postfix increment/decrement (see Note 2)	
++ --	Prefix increment/decrement	right-to-left
+ -	Unary plus/minus	
! ~	Logical negation/bitwise complement	
(type)	Cast (convert value to temporary value of type)	
*	Dereference	
&	Address (of operand)	
sizeof	Determine size in bytes on this implementation	
* / %	Multiplication/division/modulus	left-to-right
+ -	Addition/subtraction	left-to-right
<< >>	Bitwise shift left, Bitwise shift right	left-to-right
< <=	Relational less than/less than or equal to	left-to-right
> >=	Relational greater than/greater than or equal to	
== !=	Relational is equal to/is not equal to	left-to-right
&	Bitwise AND	left-to-right
^	Bitwise exclusive OR	left-to-right
	Bitwise inclusive OR	left-to-right
&&	Logical AND	left-to-right
	Logical OR	left-to-right
? :	Ternary conditional	right-to-left
=	Assignment	right-to-left
+= -=	Addition/subtraction assignment	
*= /=	Multiplication/division assignment	
%= &=	Modulus/bitwise AND assignment	
^= =	Bitwise exclusive/inclusive OR assignment	
<<= >>=	Bitwise shift left/right assignment	
,	Comma (separate expressions)	left-to-right



Directive de preprocesare

Preprocesarea → prelucrarea textului sursa al programului inaintea etapei de compilare.

- directive de preprocesare (recunoscute de compilator prin prezenta caracterului "#").

Includere fisiere - #include

Constante simbolice - #define

Compilare conditionata - #if, #elif, #else, #endif, #ifdef (echivalent cu #if defined), #ifndef (echivalent cu #if !defined)

Macrodefinitii

Alte directive - #line, #error, #pragma



Directive de preprocesare

Includere fisiere – Directiva #include

#include <fisier_sursa>
#include "fisier_sursa"

- includerea totala a fisierului sursa in fisierul care contine directiva

#include <fisier_sursa> → fisierul este cautat in directoarele standard

#include "fisier_sursa" → fisierul este cautat intai in directorul curent, iar dupa aceea, daca nu este gasit, in directoarele standard.

Obs. Forma cu "" permite si specificarea caii complete catre fisierul inclus → nu se mai face cautarea si in directoarele standard.

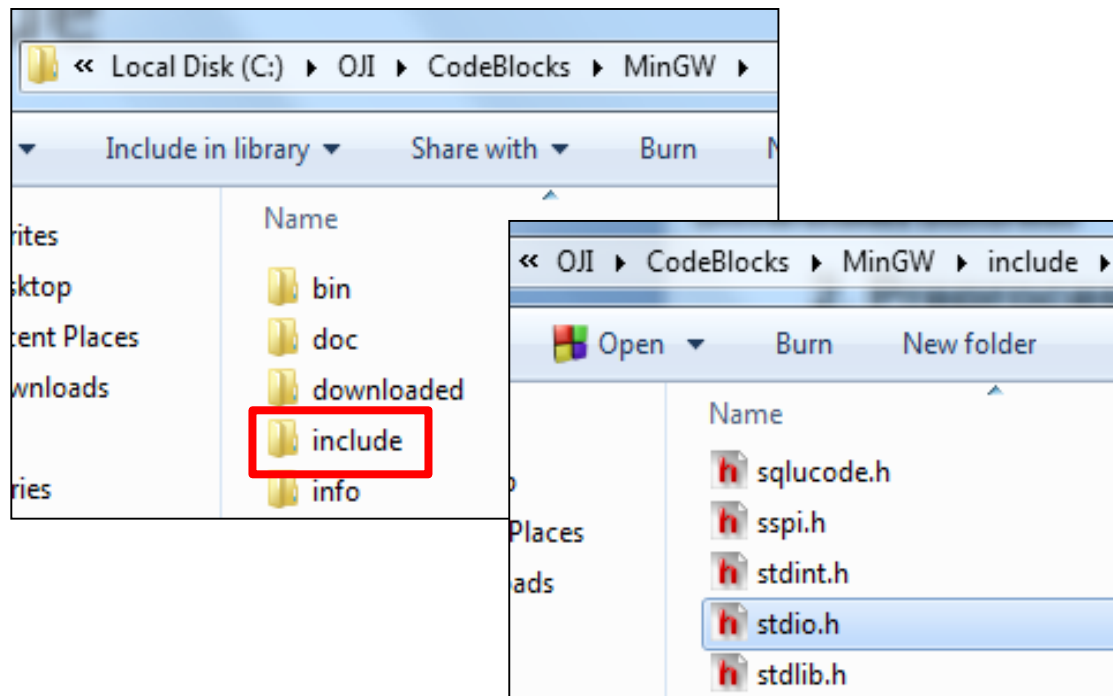


Directive de preprocesare

Includere fisiere – Directiva `#include`.

Expl.

`#include<stdio.h>` - include fisierul din directoarele standard (INCLUDE)



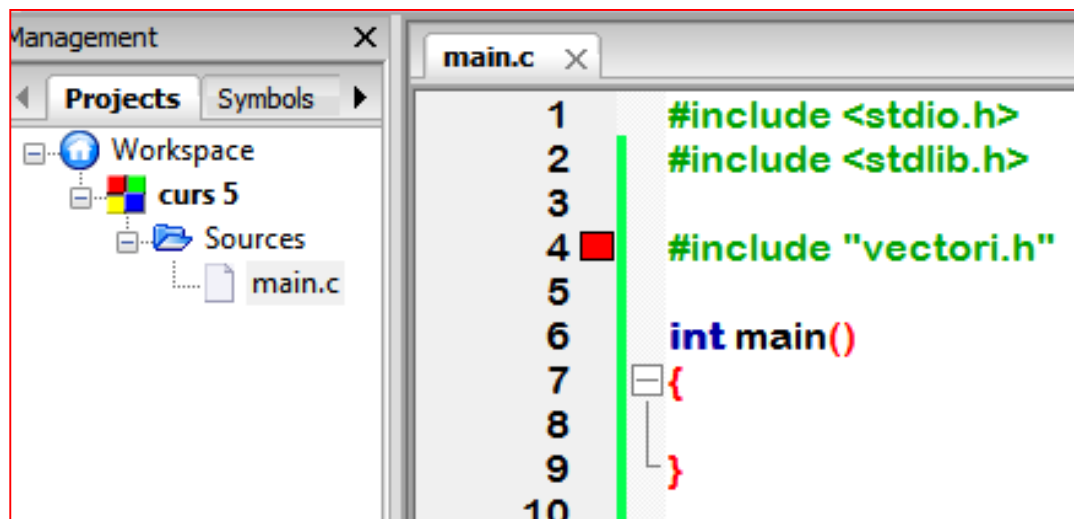


Directive de preprocesare

Includere fisiere – Directiva #include.

Expl.

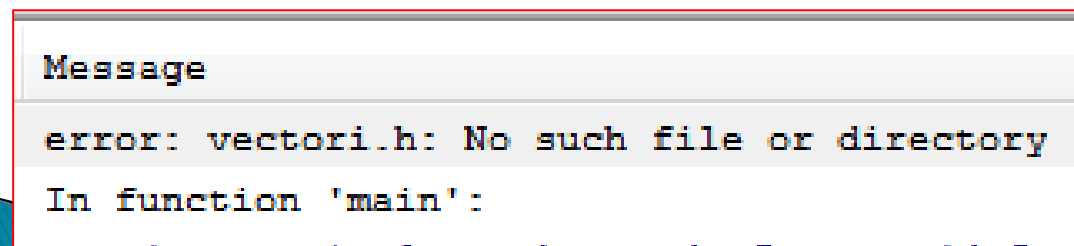
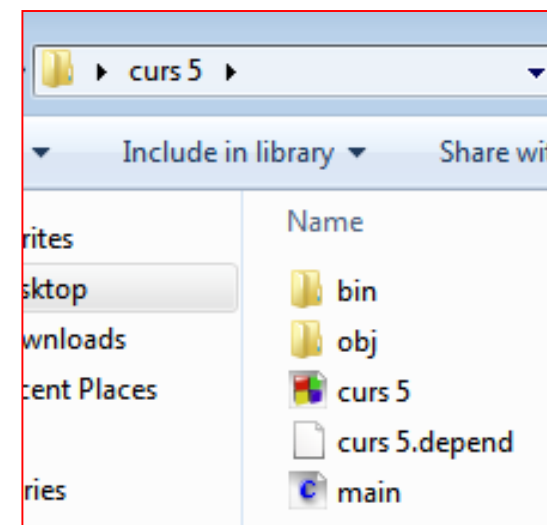
#include "Vectori.h" – cautarea initial in folderul curent si apoi in INCLUDE



The screenshot shows a code editor window titled 'main.c'. The code is as follows:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #include "vectori.h"
5
6  int main()
7  {
8
9  }
10
```

The left sidebar shows a project structure with 'Workspace' containing 'curs 5', which contains 'Sources' and 'main.c'.



The message box displays the following error:

```
Message
error: vectori.h: No such file or directory
In function 'main':
```



Directive de preprocesare

Includere fisiere – Directiva #include.

Expl.

#include "C: \Desktop\Matrice.cpp" – cautarea in folderul DESKTOP. Daca nu exista, atunci se genereaza eroare de compilare

```
main.c X
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "C:\Desktop\Matrice.cpp"
4
5  int main()
6  {
7
8      return 0;
9  }
```

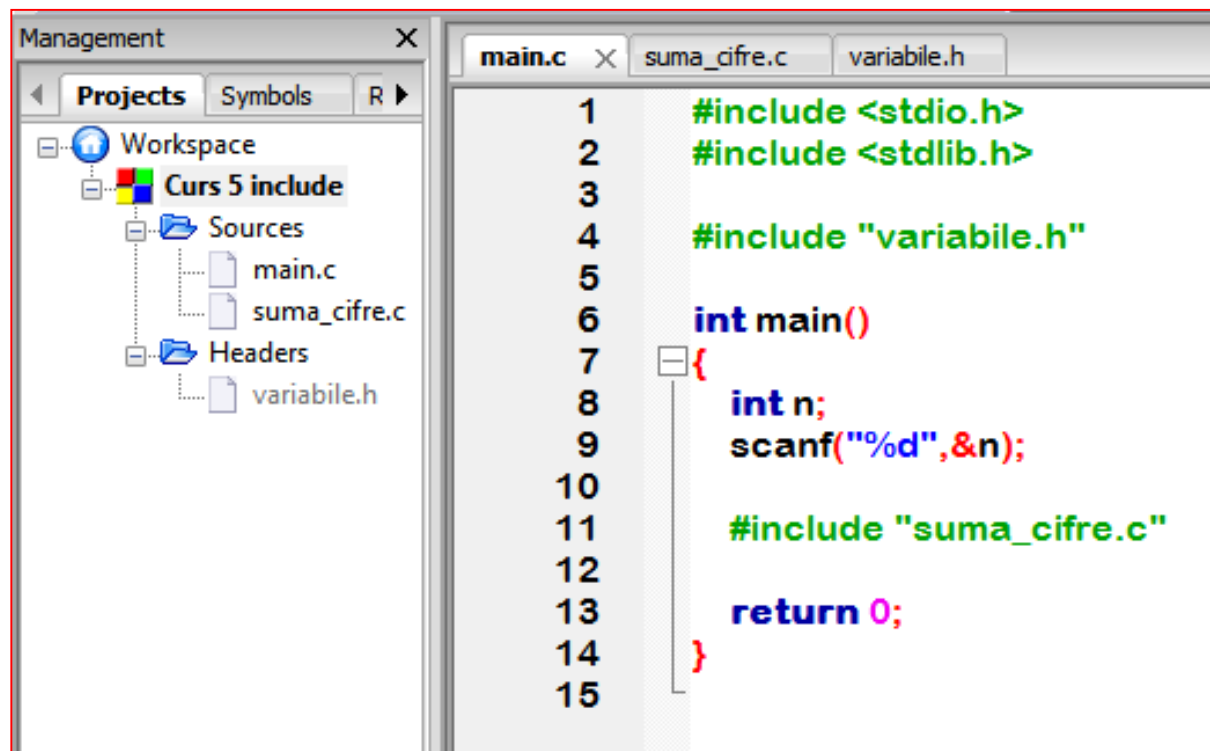
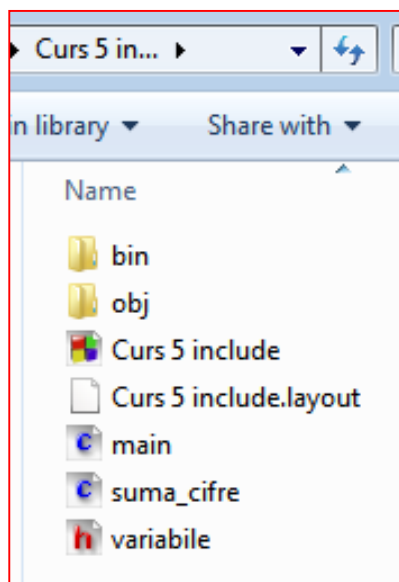
Message
=== test3, Debug ===
error: C:\Desktop\Matrice.cpp: No such file or directory
=== Build finished: 1 errors, 0 warnings ===



Directive de preprocesare

Expl.

- Includere fisiere antet (.h) si fisier sursa (.c)





Directive de preprocesare

Expl.

- Includere fisiere antet (.h) si fisier sursa (.c)

```
3
4  #include "variabile.h"
5
6  int main()
7  {
8      int n;
9      scanf("%d",&n);
10
11     #include "suma_cifre.c"
12
13     return 0;
14 }
```

```
suma_cifre.c  variabile.h  X
1  #ifndef VARIABLE_H_INCLUDED
2  #define VARIABLE_H_INCLUDED
3
4  int s = 0;
5
6  #endif // VARIABLE_H_INCLUDED
7
```

```
suma_cifre.c  X  variabile.h
1  /** rezolvarea cerintei in fisier separat */
2  int n1 = n;
3  while (n!=0)
4  {
5      s += n%10;
6      n /= 10;
7  }
8  printf("Suma cifrelor lui %d este %d.",n1,s);
9
```



Directive de preprocesare

Constante simbolice – Directiva #define

#define simbol valoare

Efect → inlocuirea tuturor aparitiilor lui **simbol** in codul sursa (cu exceptia aparitiilor in cadrul unor constante de tip sir, in comentarii sau in componenta unui alt identificator) cu **valoare** inaintea compilarii textului sursa.

Definirea unei valori pe mai multe linii → caracterul \ la sfarsitul fiecarei linii.

Redefinire in cadrul celuiasi fisier → ***#define simbol alta_valoare***.

Valabilitatea unei definiri se incheie in urmatoarele cazuri:

- la sfarsitul fisierului sursa
- la invalidarea simbolului prin intermediul directivei ***#undef simbol***

Sursa: http://ase.softmentor.ro/LimbajeEvaluate/03_Preprocesare.htm



Directive de preprocesare

Constante simbolice – Directiva #define

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

#define DIM 100
#define TIP double
#define MESAJ "Recapitulare! \n"

void verificare(void)
{
    printf (MESAJ);
}

int main ()
{
    int n;
    TIP a[DIM];
    // Apel de functie
    verificare();
    // Testare dimensiune
    scanf ("%d",&n);
    if (n >= DIM)
        printf ("Depasire limita vector \n\n");
    else
        printf ("Limita normala \n\n");
    return 0;
}
```

```
/*
#define DIM 100
#define TIP double
#define MESAJ "Recapitulare! \n"
*/

void verificare(void)
{
    printf ("Recapitulare! \n");
}

int main ()
{
    int n;
    double a[100];
    // Apel de functie
    verificare();
    // Testare dimensiune
    scanf ("%d",&n);
    if (n >= 100)
        printf ("Depasire limita vector \n\n");
    else
        printf ("Limita normala \n\n");
    return 0;
}
```



Directive de preprocesare

Constante simbolice – Directiva #define

Definirea unei valori pe mai multe linii → caracterul \ la sfarsitul fiecarei linii.

Redefinire in cadrul celuiasi fisier → *#define simbol alta_valoare*.

```
#define MESAJ "Reca" \
            "pitulare \n"

#define MAX 100

int main()
{
    int a;
    printf(MESAJ);

    printf("Const max = %d\n",MAX);

    #define MAX 200
    printf("Const max = %d\n",MAX);

    // #undef MAX
    // printf("Const max = %d\n",MAX);
}
```

```
"C:\Users\Ank\Desktop"
Recapitulare
Const max = 100
Const max = 200
```



Directive de preprocesare

Constante simbolice – Directiva #define

- invalidarea simbolului prin intermediul directivei *#undef simbol*

```
13  
14     printf("Const max = %d\n",MAX);  
15  
16     #define MAX 200  
17     printf("Const max = %d\n",MAX);  
18  
19     #undef MAX  
20     printf("Const max = %d\n",MAX);  
21
```

Line	Message
	In function 'main':
20	error: 'MAX' undeclared (first use in this function)
20	error: (Each undeclared identifier is reported only once
20	error: for each function it appears in.)



Directive de preprocesare

Compilare conditionata – Directivele `#if`, `#elif`, `#else`, `#endif`

```
#if expresie_1  
sectiune_1  
#elif expresie_2  
sectiune_2  
...  
#else  
sectiune_n  
#endif
```

- **expresie_i** este o expresie formata din constante sau simboluri definite cu `#define`; expresia este considerata adevarata daca este diferita de 0;
- **partile `#elif` si `#else` sunt optionale;**
- **sectiune_n** reprezinta o secventa de cod de inclus in codul sursa al programului in functie de valoarea expresiei;



Directive de preprocesare

Compilare conditionata – Directivele #if, #elif, #else, #endif

```
#define A 10
#define B 0
#define C -10

int main()
{
    printf("\n #if ... #endif \n");
    #if A
    printf(" Constanta A e diferita de 0 \n");
    #endif

    printf(" \n #if ... #else ... #endif \n");
    #if B
    printf(" Constanta B e diferita de 0 \n");
    #else
    printf(" Constanta B = 0 \n");
    #endif

    printf(" \n #if ... #elif ... #else .... #endif \n");
    #if A-C
    printf(" Diferenta A - C e diferita de 0 \n");
    #elif B
    printf(" B != 0 \n");
    #elif A
    printf(" Constanta A e diferita de 0 \n");
    #else
    printf("Final.");
    #endif
}
```

```
"C:\Users\Ank\Desktop\curs 5\bin\Debug\curs 5.e"

 #if ... #endif
Constanta A e diferita de 0

 #if ... #else ... #endif
Constanta B = 0

 #if ... #elif ... #else .... #endif
Diferenta A - C e diferita de 0
```




Directive de preprocesare

Compilare conditionata – Directiva `#ifdef` (echivalent cu `#if defined`)

```
#ifdef expresie  
Sectiune  
#endif
```

```
#ifdef expresie  
Sectiune_1  
#else  
Sectiune_2  
#endif
```

- **expresie** este o expresie formata din constante sau simboluri definite cu `#define`;
- expresia este considerata adevarata daca este diferita de 0;



Directive de preprocesare

Compilare conditionata – Directiva #ifdef (echivalent cu #if defined)

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



Directive de preprocesare

Compilare conditionata – Directiva `#ifdef` (echivalent cu *`#if defined`*)

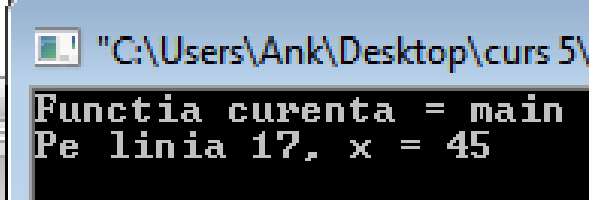
```
#define TEST

int main()
{
    int x = 70;

    #ifdef TEST
    printf("Functia curenta = %s \n", __FUNCTION__);
    #endif

    x = 45;

    #ifdef TEST
    printf("Pe linia %d, x = %d\n", __LINE__, x);
    #endif
}
```



```
"C:\Users\Ank\Desktop\curs 5\"
Functia curenta = main
Pe linia 17, x = 45
```



Directive de preprocesare

Compilare conditionata – Directiva `#ifndef` (echivalent cu `#if !defined`)

```
#ifndef expresie  
Sectiune  
#endif
```

```
#ifndef expresie  
Sectiune_1  
#else  
Sectiune_2  
#endif
```

- **expresie** este o expresie formata din constante sau simboluri definite cu `#define`;
- Daca expresia **NU** este definita, se aplica Sectiune.

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



Directive de preprocesare

Compilare conditionata – Exemple

Evitarea dublei incluziuni a fisierelor header:

La începutul fiecărui fișier *header* se *practică de obicei o astfel de secvență*

```
#ifndef Teste_h
#define Teste_h

// definitii obiecte
// ...

#endif //Teste_h
```



Directive de preprocesare

Compilare conditionata – Exemple

```
#define DIM 100
#define TIP double

#if !DIM
#define DIM 77
#else
#define DIM 88
#endif

#if TIP
#define TIP char
#elif !MESAJ
#define MESAJ "\nMesaj pentru compilare conditionata!\n"
#endif

int main ()
{
    int n=30;
    TIP a[DIM];

    printf("DIM = %d \n", DIM);
    printf("TIP = %d", sizeof(a[3]));
    printf(MESAJ);
    return 0;
}
```

```
"C:\Users\Ank\Desktop\Curs 12\bin\Debug\Curs 12
DIM = 88
TIP = 8
Mesaj pentru compilare conditionata!
```



Directive de preprocesare

Macrodefinitii

Secvente de cod parametrizate care pot fi inlocuite in textul sursa.

#define nume_macro(lista_param fara tip) corp

Apelul unui macro este similar cu apelul unei functii, **dar timpul de procesare este mai scurt.**

Etape:

- apelul macrodefinitiei din codul sursa este inlocuit cu corpul acesteia
- parametrii macrodefinitiei sunt inlocuiti cu valorile primite ca parametrii



Directive de preprocesare

Macrodefinitii

```
// definire macro
#define MAX(x,y)  x > y ? x : y
#define PATRAT(x) x*x
#define SCHIMB(tip,a,b) \
{ tip aux = a; a = b; b = aux;}

int main()
{
    // exemplu de utilizare
    int a = 7, b = 10, c, x = 2, y, z = 2, t;
    c = MAX(a,b);
    printf("c = %d\n", c);

    y = PATRAT(x+5);
    printf("y = %d\n", y);

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

    SCHIMB(int,y,t);
    printf("y = %d   t = %d\n", y,t);
    SCHIMB(int,y,t);
    printf("y = %d   t = %d\n", y,t);
}
```

```
"C:\Users\Ank\Desktop
c = 10
y = 17
z = 4      t = 16
y = 16     t = 17
y = 17     t = 16
```




Directive de preprocesare

Macrodefinitii

Exemplu preluat din ([2]) - Conversie din litera mica in litera mare

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

```
#define uppercase(x,n) for (i = 0; i < n; i++) x[i] = toupper(x[i])
```

```
int main ()
```

```
{
```

```
    int i;
```

```
    char a[6] = {'h', 'e', 'l', 'l', 'o', '\0'};
```

```
    printf ("a lowercase: %s\n", a);
```

```
    uppercase(a,6);
```

```
    printf ("a uppercase: %s\n",a);
```

```
    return 0;
```

```
}
```

"C:\Users\Ank\Desktop\Curs 12\b

```
a lowercase: hello
a uppercase: HELLO
```

[2] <http://webhost.uoradea.ro/cpopescu/progc/Cursul10.ppt>



Directive de preprocesare

Alte directive ([1])

#line, #error, #pragma

```
#include <stdio.h>
#define LINE200 200

int main(void){
    func_1(); func_2();
}

#line 100
func_1(){
    printf("Func_1 - the current line
number is %d\n", __LINE__);
}

#line LINE200
func_2(){
    printf("Func_2 - the current line
number is %d\n", __LINE__);
}
```

#line 5 “p1” – provoaca incrementarea contorului de linie cu 5 relativ la fisierul p1.

#error [text de afisat] – afisarea unui mesaj de catre compilator

#pragma lista_de_asociere – informatie dependenta de implementare:

#pragma COPYRIGHT “sir”

#pragma LINES n – nr. de linii pe pagina

[1] Albeanu G – Programare procedurala (note de curs 2013)



Tipuri structurate de date

1. Tablouri uni si bidimensionale

Sintaxa → tablouri unidimensionale

tip nume_variabila [dimensiune];

double tab [100] ;

0.3	-1.2	10	5.7	...	0.2	-1.5	1
0	1	2	3		97	98	99

tab [3] = 5.7;

int a[5];

3	-12	10	7	1
0	1	2	3	4

a [1] = 3;

char b1 [34];

A	&	*	+	...	c	M	#
0	1	2	3	...			

b1 [1] = '&';



Tipuri structurate de date

1. Tablouri uni si bidimensionale

Cantitatea de memorie necesara pentru inregistrarea unui tablou este direct proportionala cu tipul si marimea sa.

tip nume [dimensiune] → **sizeof(nume) = sizeof (tip) * dimensiune ;**

```
double tab [100] ;

int a[5];

char b1 [34];

printf("Stocarea unui tablou de elemente double = %d octeti\n",sizeof(tab));
printf("Stocarea unui tablou de elemente int = %d octeti\n",sizeof(a));
printf("Stocarea unui tablou de elemente char = %d octeti\n",sizeof(b1));
```

C:\Users\Ank\Desktop\testSizeof\bin\Debug\testSizeof.exe

```
Stocarea unui tablou de elemente double = 800 octeti
Stocarea unui tablou de elemente int = 20 octeti
Stocarea unui tablou de elemente char = 34 octeti
```



Tipuri structurate de date

1. Tablouri uni si bidimensionale

Tablouri unidimensionale → liste de informatii de acelasi tip, stocate in locatii de memorie contigue in ordinea indicilor.

```
11  
12 int a[5];  
13  
14 for(i=0;i<5;i++)  
15 printf("Adresa elem %d din tablou = %d \n",i,&a[i]);  
16  
17  
18 C:\Users\Ank\Desktop\testSizeof\bin\Debug\testSizeof.exe  
19 Adresa elem 0 din tablou = 2686728  
20 Adresa elem 1 din tablou = 2686732  
21 Adresa elem 2 din tablou = 2686736  
22 Adresa elem 3 din tablou = 2686740  
Adresa elem 4 din tablou = 2686744
```

```
char b1 [34];  
  
for(i=0;i<5;i++)  
printf("Adresa elem %d din tablou = %d \n",i,&b1[i]);  
  
C:\Users\Ank\Desktop\testSizeof\bin\Debug\testSizeof.exe  
Adresa elem 0 din tablou = 2686694  
Adresa elem 1 din tablou = 2686695  
Adresa elem 2 din tablou = 2686696  
Adresa elem 3 din tablou = 2686697  
Adresa elem 4 din tablou = 2686698
```



Tipuri structurate de date

1. Tablouri uni si bidimensionale

Aplicatie Citirea si afisarea elementelor unui tablou unidimensional

```
int main()
{
    int a[100]; // tabloul a cu maxim 100 de elemente
    int i; // indicele cu care parcurg tabloul
    int n; // lungimea efectiva a tabloului

    printf("Cate elem introduceti? ");
    scanf("%d", &n);

    printf("\nDati elem. sirului: \n");
    for(i=0;i<n;i++)
        scanf("%d",&a[i]);

    printf("\nAfisarea sirului: ");
    for(i=0;i<n;i++)
        printf("%d ",a[i]);
}
```

```
C:\Users\Ank\Desktop\testSizeof\bin\Debug\t...
Cate elem introduceti? 4

Dati elem. sirului:
12
45
-78
3

Afisarea sirului: 12 45 -78 3
```



Tipuri structurate de date

1. Tablouri uni si bidimensionale

Sintaxa → tablouri bidimensionale

tip nume_variabila [dimensiune1][dimensiune2];

int a[3][5];

0	3	-12	10	7	1
1	10	2	0	-7	41
2	-3	-2	0	0	2
	0	1	2	3	4

a [1][4] = 41;



Tipuri structurate de date

1. Tablouri uni si bidimensionale

Cantitatea de memorie necesara pentru inregistrarea unui tablou este direct proportionala cu tipul si marimea sa.

tip nume [dimensiune1][dimensiune2] →

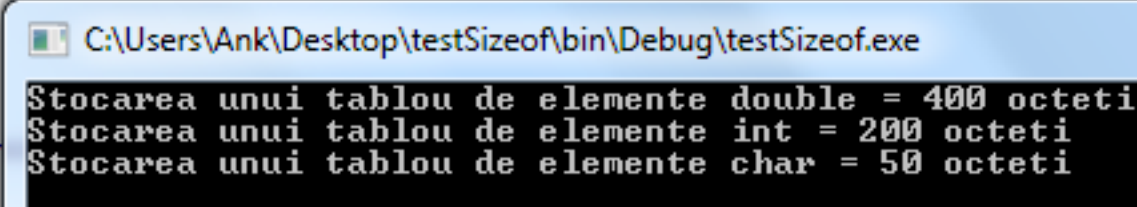
sizeof(nume) = sizeof (tip) * dimensiune1 * dimensiune2 ;

```
double tab [5][10] ;

int a[5][10];

char b1[5][10];

printf("Stocarea unui tablou de elemente double = %d octeti\n",sizeof(tab));
printf("Stocarea unui tablou de elemente int = %d octeti\n",sizeof(a));
printf("Stocarea unui tablou de elemente char = %d octeti\n",sizeof(b1));
```





Tipuri structurate de date

1. Tablouri uni si bidimensionale

Aplicatie Citirea si afisarea elementelor unui tablou bidimensional

```
int main()
{
    int a[5][10]; // tabloul a cu maxim 5 linii si 10 coloane
    int i,j; // i = inice pt linii, j = indice pt coloane
    int n; // numarul de linii
    int m; // numarul de coloane

    printf("Dati nr de linii si de coloane: ");
    scanf("%d%d", &n, &m);

    printf("\nDati elem. matricei: \n");
    for(i=0;i<n;i++)
        for(j=0;j<m;j++)
            scanf("%d",&a[i][j]);

    printf("\nAfisarea matricei: \n");
    for(i=0;i<n;i++)
    {
        for(j=0;j<m;j++)
            printf("%d ",a[i][j]);
        printf("\n");
    }
}
```



Tipuri structurate de date

1. Tablouri uni si bidimensionale

Tablouri bidimensionale → stocarea in locatii de memorie

```
int main()
{
    int a[5][10]; // tabloul a cu maxim 5 linii si 10 coloane
    int i,j; // i = inice pt linii, j = indice pt coloane
    int n; // numarul de linii
    int m; // numarul de coloane

    printf("Dati nr de linii si de coloane: ");
    scanf("%d%d", &n, &m);

    /*
    printf("\nDati elem. matricei: \n");
    for(i=0;i<n;i++)
        for(j=0;j<m;j++)
            scanf("%d", &a[i][j]);
    */
    printf("\nAfisarea adreselor elementelor matricei: \n");
    for(i=0;i<n;i++)
    {
        for(j=0;j<m;j++)
            printf("%d ", &a[i][j]);
        printf("\n");
    }
}
```

```
C:\Users\Ank\Desktop\testSizeof\bin\Debug\testSizeof.exe
Dati nr de linii si de coloane: 3
4
Afisarea adreselor elementelor matricei:
2686520 2686524 2686528 2686532
2686560 2686564 2686568 2686572
2686600 2686604 2686608 2686612
Process returned 0 (0x0)   execution time
Press any key to continue.
```

Linia 0: adrese → 2686520 ... 2696556

Linia 1: adrese → 2686560 ... 2696596

Linia 2: adrese → 2686600 ... 2696636



Concluzii

- 1. S-au recapitulat notiunile de Limbaj C predate in cursul 4:
Precedenta si asociativitatea operatorilor, expresii;**
- 2. S-au prezentat directivele de preprocesare si macrodefinitiiile;**
- 3. S-au prezentat notiunile introductive referitoare la tipuri
structurate de date.**



Perspective

Cursul 6:

1. Tipuri derivate de date (partea a 2-a)

- Structuri, uniuni, câmpuri de biți, enumerări.
- Pointeri.