



Programare procedurala

- suport de curs -

Dobrovat Anca - Madalina

**An universitar 2016 – 2017
Semestrul I**

Curs 3



Agenda cursului

1. Fundamentele limbajului C

- Structura unui program in C
- Tipuri de date fundamentale
- Variabile
- Constante
- Operatori
- Expresii
- Conversii
- Instructiuni
- Functii de citire / scriere

2. Complexitatea algoritmilor – notiuni introductive



Fundamentele limbajului C

Programele C sunt propozitii formate cu simboluri ale alfabetului C: atomi lexicali (tokens) si separatori.

Atomii lexicali - identificatori, constante, operatori, semne de punctuatie.

Cuvinte cheie (32)

C89 = ANSI C : 32 de cuvinte cheie

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

C99: ANSI C + alte 5 cuvinte cheie

_Bool _Complex _Imaginary inline restrict



Fundamentele limbajului C

Structura generala a unui program in C

- modul principal (functia main, clasa aplicatie cu metoda main)
- zero, unul sau mai multe module (functii/proceduri, metode ale clasei aplicatie) care comunica intre ele si/sau cu modulul principal prin intermediul parametrilor si/sau a unor variabile globale

Unitatea de program cea mai mica si care contine cod este **functia / procedura** si contine:

- partea de declaratii/definitii
- partea imperativa (comenzile care se vor executa)

Avantaje: compilare separata, reutilizarea codului, lucrul in echipa, lucrul la distanta, testarea/verificarea codului – Unit testing



Structura unui program generalizat

Comentarii

directive de preprocesare

declarații de variabile globale

subprograme

int main(void)

{

declarații de variabile locale instrucțiuni

}



Fundamentele limbajului C

Structura unui program in C simplu

Directive de preprocesare

- ❑ directive de definiție: `#define N 10`
- ❑ directive de includere a bibliotecilor: `#include <stdio.h>`
- ❑ directive de compilare condiționată: `#if, #ifdef, ...`

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;
 - ❑ `main` este obligatoriu;
- ❑ antetul și corpul funcției.



Structura unui program in C simplu

Instrucțiuni

- formează corpul funcțiilor
 - exprimate sub formă de comenzi
- 5 tipuri de instrucțiuni:
 - instrucțiunea declarație;
 - instrucțiunea atribuire;
 - instrucțiunea apel de funcție;
 - instrucțiuni de control;
 - instrucțiunea vidă;
- toate instrucțiunile (cu excepția celor compuse) se termină cu caracterul ";"
 - caracterul ; nu are doar rol de separator de instrucțiuni ci instrucțiunile încorporează caracterul ; ca ultim caracter
 - omiterea caracterului ; reprezintă eroare de sintaxă



Fundamentele limbajului C

Structura unui program generalizat

Citirea unui numar natural si afisarea inversului

```
main.c x
1 //Citirea unui numar natural si afisarea inversului
2
3 #include <stdio.h>
4
5 void citire_natural(int *a)
6 {
12
13 int invers(int n)
14 {
23
24 int inv;
25
26 int main()
27 {
28     int n;
29     printf("Citirea unui numar natural\n");
30     citire_natural(&n);
31     printf("Numarul citit este %d\n", n);
32     inv = invers(n);
33     printf("Inversul numarului citit este %d\n", inv);
34     return 0;
35 }
```

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

```
Citirea unui numar natural
1234
Numarul citit este 1234
Inversul numarului citit este 4321
```




Fundamentele limbajului C

Structura unui program generalizat

Citirea unui numar natural si afisarea inversului

```
5 void citire_natural(int *a)
6 {
7     do
8     {
9         scanf("%d", &*a);
10    } while(*a < 0);
11 }
12
13 int invers(int n)
14 {
15     int og = 0;
16     while(n != 0)
17     {
18         og = og * 10 + n % 10;
19         n = n / 10;
20     }
21     return og;
22 }
```

```
C:\Users\Ank\Desktop\teste\bin\Debug\teste.exe
Citirea unui numar natural
1234
Numarul citit este 1234
Inversul numarului citit este 4321
```



Fundamentele limbajului C

Tipuri de date fundamentale

- Tipul de dată specifică
 - natura datelor care pot fi stocate în variabilele de acel tip
 - necesarul de memorie și
 - operațiile permise asupra acestor variabile

5 tipuri de date de baza:

-char, int, float, double, void

- C99 a introdus tipul `_Bool` (true, false)
 - De fapt valori întregi (0 fals, orice altceva adevărat)



Fundamentele limbajului C

Tipuri de date fundamentale

- tipul întreg – **int**: 1, 0, -77, etc.;
- tipul **caracter** – **char**: poate reține un singur caracter sub forma codului elementelor din setul de caractere specific (codul **ASCII**)
 - poate reprezenta 128 de caractere
 - codul **Latin- 1**, extinde codul ASCII pe 8 biți - poate reprezenta 256 de caractere

char	1 octet (8 biți) în domeniul -128 până la 127
int	16-biți OS : 2 octeți în domeniul -32768 până la 32767 32-biți OS : 4 octeți în domeniul -2,147,483,648 până la 2,147,483,647



Fundamentele limbajului C

Tipuri de date fundamentale

- 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

float	4 octeți în domeniul 10^{-38} până la 10^{38} cu precizie de 7 zecimale
double	8 octeți în domeniul 10^{-308} până la 10^{308} cu precizie de 15 zecimale



Fundamentele limbajului C

Tipuri de date fundamentale

- Programtorul 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
- Limitele specifice unui sistem de calcul pot fi aflate din fișierele header `limits.h` și `float.h`
 - exemplu: `CHAR_MAX`, `INT_MAX`, `INT_MIN`, `UINT_MAX`



Fundamentele limbajului C

Tipuri de date fundamentale

```
printf ("Nr octeti pentru char = %d\n", sizeof(char));  
printf("Limita minima pentru char = %d\n", CHAR_MIN);  
printf("Limita maxima pentru char = %d\n\n", CHAR_MAX);
```

```
Nr octeti pentru char = 1  
Limita minima pentru char = -128  
Limita maxima pentru char = 127
```

```
printf ("Nr octeti pentru int = %d\n", sizeof(int));  
printf("Limita minima pentru int = %d\n", INT_MIN);  
printf("Limita maxima pentru int = %d\n\n", INT_MAX);
```

```
Nr octeti pentru int = 4  
Limita minima pentru int = -2147483648  
Limita maxima pentru int = 2147483647
```



Fundamentele limbajului C

Tipuri de date fundamentale

```
printf ("Nr octeti pentru float = %d\n", sizeof(float));  
printf("Limita minima pentru float = %e\n", FLT_MIN);  
printf("Limita maxima pentru float = %e\n", FLT_MAX);  
printf("Numarul de zecimale pentru float = %d\n\n", FLT_DIG);
```

```
Nr octeti pentru float = 4  
Limita minima pentru float = 1.175494e-038  
Limita maxima pentru float = 3.402823e+038  
Numarul de zecimale pentru float = 6
```

```
printf ("Nr octeti pentru double = %d\n", sizeof(double));  
printf("Limita minima pentru double = %e\n", DBL_MIN);  
printf("Limita maxima pentru double = %e\n", DBL_MAX);  
printf("Numarul de zecimale pentru double = %d\n\n", DBL_DIG);
```

```
Nr octeti pentru double = 8  
Limita minima pentru double = 2.225074e-308  
Limita maxima pentru double = 1.797693e+308  
Numarul de zecimale pentru double = 15
```



Fundamentele limbajului C

Tipuri de date fundamentale

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 în 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)

long long - Introdus in C99 pentru a facilita stocarea unor valori întregi de dimensiuni foarte mari

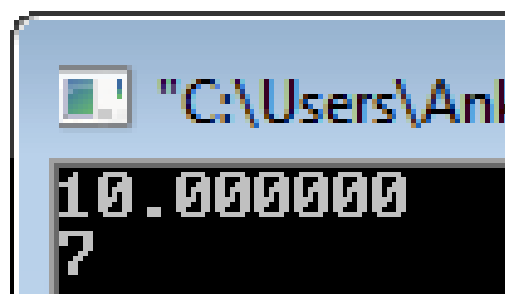
- cel puțin 8 octeți.



Fundamentele limbajului C

Tipuri de date fundamentale

```
int a = 10, a1;  
float b = 7.7, b1;  
char t = 'A', c1;  
  
b1 = a;  
printf("%f\n", b1);  
a1 = b;  
printf("%d\n", a1);
```



```
"C:\Users\Ank  
10.000000  
7
```

Conversii de tip / Operatorul cast

```
int a = 120, a1;  
char t = 'A', t1;  
  
t1 = a;  
printf("%c\n", t1);  
a1 = t;  
printf("%d\n", a1);
```



```
"C:\Users\Ank  
x  
65
```

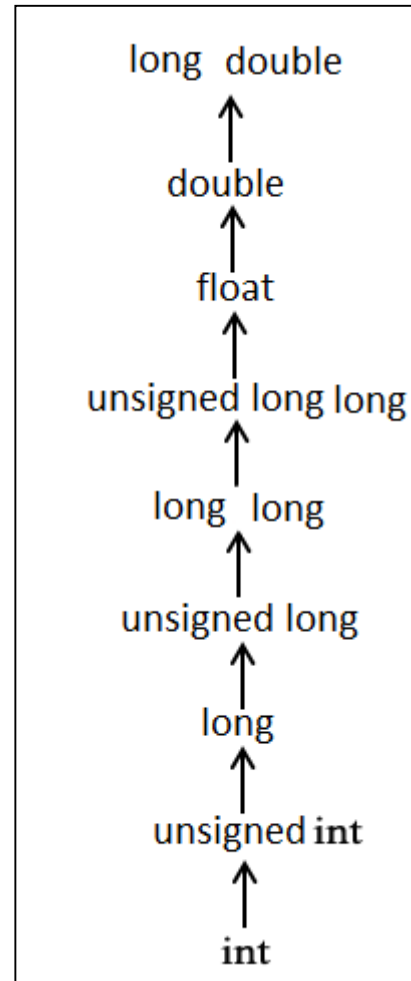


Fundamentele limbajului C

Tipuri de date fundamentale

Conversii de tip / Operatorul cast

Conversia aritmetica uzuala



Sursa: https://www.tutorialspoint.com/cprogramming/c_type_casting.htm



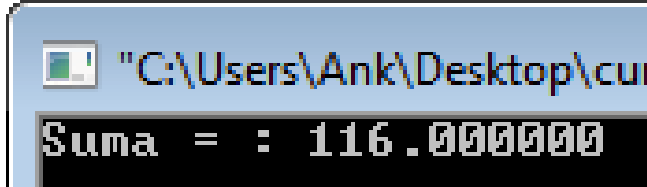
Fundamentele limbajului C

Tipuri de date fundamentale

Conversii de tip / Operatorul cast

Exemplu

```
int i = 17;  
char c = 'c'; /* ascii = 99 */  
float sum;  
  
sum = i + c;  
printf("Suma = : %f\n", sum );
```



"C:\Users\Ank\Desktop\cu
Suma = : 116.000000



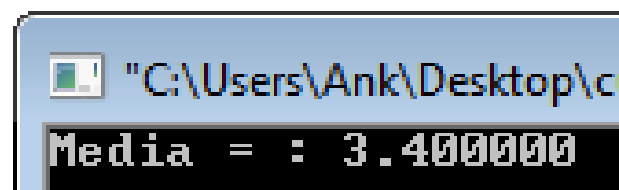
Fundamentele limbajului C

Tipuri de date fundamentale

Conversii de tip / Operatorul cast

Exemplu

```
int sum = 17, contor = 5;  
double m;  
  
m = (double) sum / contor;  
printf("Media = : %f\n", m );
```



"C:\Users\Ank\Desktop\cu
Media = : 3.400000



Fundamentele limbajului C

Comentarii

Complet ignorate de compilator

Tipuri de comentariu

C89 oferă un singur fel de comentariu

```
/* linie de cod comentata1  
linie de cod comentata 2  
-----  
linie de cod comentata n */
```

Obs: nu imbricate

C99 oferă și comentariul pe o singură linie

```
// linie de cod comentata
```



Fundamentele limbajului C

Expresii

Variabile si Constante

- stochează datele necesare programului
- referirea la aceste date se face prin numele lor simbolice, adică prin **identificatori**
- **Variabilele** stochează date care pot fi modificate în timpul execuției
- **Constantele** păstrează aceeași valoare (cea cu care au fost inițializate) până la terminarea programului



Fundamentele limbajului C

Expresii

Variabile si Constante

Sintaxa: **tip nume_variabila = valoare_constanta;**

Nume de identificatori (variabile, constante, functii, etichete etc.)

- unul sau mai multe caractere
- primul este “_” sau o litera
- urmatoarele: cifre, litere sau “_”

Corect	Inc corect
numarator	1numarator
test23	salut!
bilant_mare	bilant...mare

Case sensitive – nume, NUME, Nume – identificatori diferiti

Un identificator nu poate fi un cuvânt cheie sau numele unei functii din biblioteca C



Fundamentele limbajului C

Expresii

Variable

- numele variabilei nu are legatura cu tipul ei

Tipuri de variabile

- **locale** (definite in interiorul unei functii)
- **parametri formali** (declarare dupa numele functiei)
- **globale** (cunoscute de intreg programul)



Fundamentele limbajului C

Expresii - Variabile locale

```
void functie(int a, float c)
{
    int b;
    b = a * 3;
    printf("variabila locala b = %d\n",b);

    a = a - 5;
    printf("parametrul formal a in functie = %d\n",a);

    c = c + 7 ;
    printf("parametrul formal c in functie = %f\n",c);
}
```

```
"C:\Users\Ank\Desktop\curs 3\bin\Debug\curs 3.exe"
10
4.5
variabila locala b = 30
parametrul formal a in functie = 5
parametrul formal c in functie = 11.500000
parametrul efectiv a in main = 10
parametrul efectiv c in main = 4.500000
```



Fundamentele limbajului C

Expresii - Parametri formali

Daca o functie urmeaza sa foloseasca argumente, ea trebuie sa declare Variabilele pe care le accepta ca valori ale argumentelor (i.e. **parametri formali**).

Se comporta ca o variabila locala a functiei.

```
void functie(int a, float c)
```



```
{
```

```
int main()
```



```
{
```

```
    int a;
```

```
    float c;
```

```
    scanf("%d%f", &a, &c);
```

```
    functie(a, c);
```

```
    printf("parametrul efectiv a in main = %d\n", a);
```

```
    printf("parametrul efectiv c in main = %f\n", c);
```

```
    return 0;
```

```
}
```

"C:\Users\Ank\Desktop\curs 3\bin\Debug\curs 3.exe"

```
10
```

```
4.5
```

```
variabila locala b = 30
```

```
parametrul formal a in functie = 5
```

```
parametrul formal c in functie = 11.500000
```

```
parametrul efectiv a in main = 10
```

```
parametrul efectiv c in main = 4.500000
```



Fundamentele limbajului C

Expresii - Variabile locale si parametri

```
main.c x
1  #include <stdio.h>
2
3  void f1()
4  {
5      int a;
6      a = 10;
7      printf("Valoare variabilei locale din f1= %d \n",a);
8  }
9
10 void f2()
11 {
12     int a;
13     a = 33;
14     printf("Valoare variabilei locale din f2= %d \n",a);
15 }
16
17 void f3(int x)
18 {
19     x = 56;
20     printf("Valoare parametrului din f3= %d \n",x);
21 }
22
23 int main()
24 {
25     int b;
26     f1();
27     f2();
28     f3(b);
29     return 0;
30 }
```

```
C:\Users\Ank\Desktop\teste\bin\Debug\teste.exe
Valoare variabilei locale din f1= 10
Valoare variabilei locale din f2= 33
Valoare parametrului din f3= 56

Process returned 0 (0x0)   execution time : 0.011 s
Press any key to continue.
-
```



Expresii - Variabile globale

Se declara in afara oricarei functii si sunt cunoscute in intreg programul

Pot fi utilizate de catre orice zona a codului

Isi pastreaza valoarea pe parcursul intregii executii a programului.

Orice expresie are acces la ele, indiferent de tipul blocului de cod in care se afla expresia.



Fundamentele limbajului C

Expresii - Variabile globale

```
#include <stdio.h>

int a = 10;

void f1()
{
    a = 30;
    printf("Valoare variabilei globale in f1: %d \n",a);
}

int main()
{
    printf("Valoarea initiala a variabilei globale: %d\n", a);
    a = 20;
    printf("Valoarea variabilei globale in main: %d\n", a);
    f1();
    return 0;
}
```

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

```
Valoarea initiala a variabilei globale: 10
Valoarea variabilei globale in main: 20
Valoare variabilei globale in f1: 30
```



Fundamentele limbajului C

Constante

Expresii

Const

Variabilele de tip **const** nu pot fi modificate de program (dar pot primi valori initiale).

```
const int a = 10;
```

Modelatorul **const** poate fi folosit pentru a proteja obiectele indicate de argumentele unei functii pentru a nu fi modificate de acea functie.

(Exemple in laborator).



Fundamentele limbajului C

Constante


Expresii

```
const int a = 10;  
const float b = 7.7;  
const char t = 'A';
```

```
int main()
```

```
{
```

```
    printf("a = %d, b = %f, t = %c\n", a, b, t);
```



"C:\Users\Ank\Desktop\curs 3\bin\Debug\cur
a = 10, b = 7.700000, t = A

(Exemple in laborator).



Fundamentele limbajului C

Constante

Expresii

```
5  const int a = 10;
6  const float b = 7.7;
7  const char t = 'A';
8
9  int main()
10 {
11
12     printf("a = %d, b = %f, t = %c\n",a,b,t);
13
14     a = 29;
15     printf("a = %d, b = %f, t = %c\n",a,b,t);
16
```

Line	Message
	In function 'main':
14	error: assignment of read-only variable 'a'
	=== Build finished: 1 errors, 0 warnings ===



Operatori

pot fi unari, binari sau ternari, fiecare având o precedență și o asociativitate bine definite

Precedență	Operator	Descriere	Asociativitate
1	<code>[]</code>	Indexare	stanga-dreapta
	<code>. și -></code>	Selecție membru (prin structură, respectiv pointer)	stânga-dreapta
	<code>++ și --</code>	Postincrementare/postdecrementare	stânga-dreapta

Sursa: <http://ocw.cs.pub.ro/courses/programare/laboratoare/lab02>



Fundamentele limbajului C

Operatori

Precedență	Operator	Descriere	Asociativitate
2	!	Negare logică	dreapta-stânga
	~	Complement față de 1 pe biți	dreapta-stânga
	++ și --	Preincrementare/predecrementare	dreapta-stânga
	+ și -	+ și - unari	dreapta-stânga
	*	Dereferențiere	dreapta-stânga
	&	Operator <i>adresă</i>	dreapta-stânga
	(tip)	Conversie de tip	dreapta-stânga
	sizeof()	Mărimea în octeți	dreapta-stânga

Sursa: <http://ocw.cs.pub.ro/courses/programare/laboratoare/lab02>



Fundamentele limbajului C

Operatori

Precedență	Operator	Descriere	Asociativitate
3	*	Înmulțire	stânga-dreapta
	\/	Împărțire	stânga-dreapta
	%	Restul împărțirii	stânga-dreapta
4	+ și -	Adunare/scădere	stânga-dreapta
5	<< și >>	Deplasare stânga/dreapta a biților	stânga-dreapta
6	<	Mai mic	stânga-dreapta
	<=	Mai mic sau egal	stânga-dreapta
	>	Mai mare	stânga-dreapta
	>=	Mai mare sau egal	stânga-dreapta

Sursa: <http://ocw.cs.pub.ro/courses/programare/laboratoare/lab02>



Fundamentele limbajului C

Operatori

Precedență	Operator	Descriere	Asociativitate
7	==	Egal	dreapta
	!=	Diferit	stânga-dreapta
8	&	ȘI pe biți	stânga-dreapta
9	^	SAU-EXCLUSIV pe biți	stânga-dreapta
10		SAU pe biți	stânga-dreapta
11	&&	ȘI logic	stânga-dreapta
12		SAU logic	stânga-dreapta
13	?:	Operator condițional	dreapta-stânga

Sursa: <http://ocw.cs.pub.ro/courses/programare/laboratoare/lab02>



Fundamentele limbajului C

Operatori

Precedență	Operator	Descriere	Asociativitate
14	=	Atribuire	dreapta-stânga
	+= și -=	Atribuire cu adunare/scădere	dreapta-stânga
	*= și /=	Atribuire cu multiplicare/împărțire	dreapta-stânga
	%=	Atribuire cu modulo	dreapta-stânga
	&= si =	Atribuire cu ȘI/SAU	dreapta-stânga
	^=	Atribuire cu SAU-EXCLUSIV	dreapta-stânga
	<<= și >>=	Atribuire cu deplasare de biți	dreapta-stânga
15	,	Operator secvența	stânga-dreapta

Sursa: <http://ocw.cs.pub.ro/courses/programare/laboratoare/lab02>



Exemple

Operatori

Prioritatea pre / postfixarii

```
int a = 10, b, c;  
b = a++;  
c = ++a;  
printf("a = %d    b = %d    c = %d\n", a, b, c);
```

```
"C:\Users\Ank\Desktop\curs 3\bin\  
a = 12    b = 10    c = 12
```

Prioritatea operatorilor matematici

Care este rezultatul operatiilor $10/3*5 + 7 - 12/(2*3)$



Fundamentele limbajului C

Exemple

Operatori

Prioritatea operatorilor logici

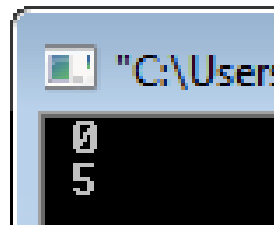
Ce va afisa?

```
int a = 1, b = 1, c = 0;
```

```
printf(" %d \n", a && !b
```

Prioritatea operatorilor din
grupuri diferite

```
int a = 5, b = 1, c;  
c = a + b && 0;  
  
printf(" %d \n", c);  
  
c = a + (b && 0);  
  
printf(" %d \n", c);
```





Fundamentele limbajului C

Instructiuni

- **instrucțiune = proces de prelucrare al datelor într-un limbaj de programare**

Exemple

; - instrucțiunea vidă

scanf ("%d",&x); - instructiuni de citire a datelor

printf("A \n"); - instructiuni de afisare a datelor

B = B - A;

B = -A;

B = 2 * A; instructiuni de atribuire

B *= 2;



Fundamentele limbajului C

Structuri secventiale

Exemplu

Sintaxa:

Instrucțiune1;

Instrucțiune2;

Instrucțiune n;

```
int a,b,c;  
c = 17;  
printf (" a = ");  
scanf ("%d",&a);  
b = a / 2;  
c += 10;  
printf("a = %d, b = %d, c = %d \n\n\n",a,b,c);
```

```
"C:\Users\Ank\Desktop\curs 3\  
a = 10  
a = 10, b = 5, c = 27
```



Fundamentele limbajului C

Structuri decizionale

Instrucțiunea IF

Exemplu

Sintaxa:

if (expresie)
set de instructiuni;

```
if (a > b)
printf("\nMaximul = %d\n",a );
```

```
if (a != b && a != c && b!= c)
{
    printf ("\nvalori distincte\n");
    if ( (a+b)/2 == c)
        printf ("\nc este media aritmetica a lui a si b\n");
}
```

```
"C:\Users\Ank\Desktop\curs 3\bin\Debug\curs 3.exe"
dati a,b,c 10 4 7
Maximul = 10
valori distincte
c este media aritmetica a lui a si b
```



Fundamentele limbajului C

Structuri decizionale

Instrucțiunea **IF ELSE**

Sintaxa:

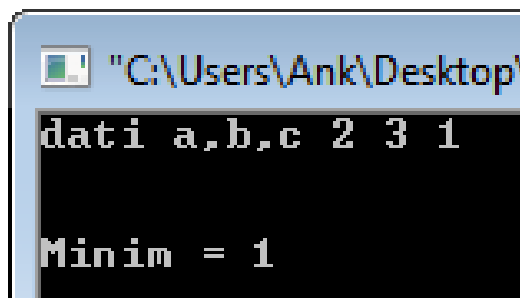
```
if (expresie)
    set de instructiuni;
else
    set de instructiuni
```

Exemplu

```
if (a < b)
    min = a;
else
    min = b;

if (c < min)
    min = c;

printf ("\n\nMinim = %d\n\n", min);
```



```
"C:\Users\Ank\Desktop\'
dati a,b,c 2 3 1

Minim = 1
```



Fundamentele limbajului C

Structuri decizionale

Operatorul ternar

Sintaxa:

(expresie) ? ({set_de_instructiuni1}) : ({set_de_instructiuni2});

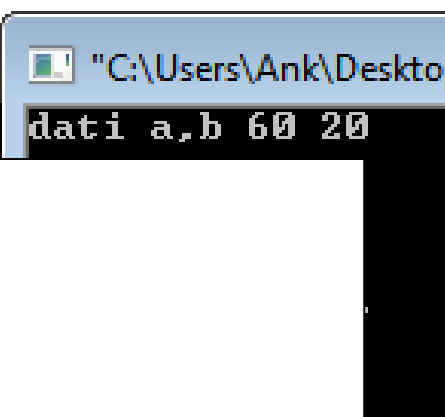
Exemple

```
min = (a < b) ? a : b;
```

```
printf ("\n\nMinim = %d\n\n", min);
```

```
(a % b == 0) ? ({printf("b este divizor"); a -= b;}) :  
              ({printf("b nu este divizor"); a += b;});
```

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





Fundamentele limbajului C

Structuri decizionale

Instrucțiunea **SWITCH**

Sintaxa:

```
switch (expresie){  
    case (expresie_1)  
        instructiune_1;  
    case (expresie_2):  
        instructiune_2;  
    .....  
    case (expresie_n):  
        instructiune_n;  
default:  
    instructiune_(n+1);  
}
```

Exemplu (fara break)

```
switch (a)  
{  
    case 1: a++;  
    case 2: a = a * 2;  
    default : a = a +5;  
}  
  
printf ("\n\n a = %d \n\n",a);
```

a = 2

a = 4

a = 9

```
"C:\Users\  
a = 1  
  
a = 9
```



Fundamentele limbajului C

Structuri decizionale

Instrucțiunea **SWITCH**

Sintaxa:

```
switch (expresie){  
    case (expresie_1)  
        instructiune_1; break;  
    case (expresie_2):  
        instructiune_2; break;  
    .....  
    case (expresie_n):  
        instructiune_n; break;  
default:  
    instructiune_(n+1);  
}
```

Exemplu (cu break)

```
switch (a)  
{  
    case 1: a++; break;  
    case 2: a = a * 2;  
    default : a = a +5;  
}  
  
printf ("\n\n a = %d \n\n",a);
```

```
"C:\Users\A  
a = 1  
a = 2
```

```
switch (a)  
{  
    case 1: a++;  
    case 2: a = a * 2; break;  
    default : a = a +5;  
}  
  
printf ("\n\n a = %d \n\n",a);
```

```
"C:\Users\  
a = 1  
a = 4
```



Fundamentele limbajului C

Structuri decizionale

Instrucțiunea **SWITCH**

Exemplu

```
char oper;  
scanf("%c", &oper);  
switch (oper)  
{  
    case ('+'):   
        printf("Operatorul de adunare!\n");  
        break;  
    case ('-'):   
        printf("Operatorul de scadere!\n");  
        break;  
    case ('*'):   
        printf("Operatorul de inmultire!\n");  
        break;  
    default:   
        printf("Operator ilegal!\n");  
}
```

"C:\Users\Ank\Desktop\curs 3\bin\
*
Operatorul de inmultire!

"C:\Users\Ank\Desktop\cu

Operator ilegal!



Fundamentele limbajului C

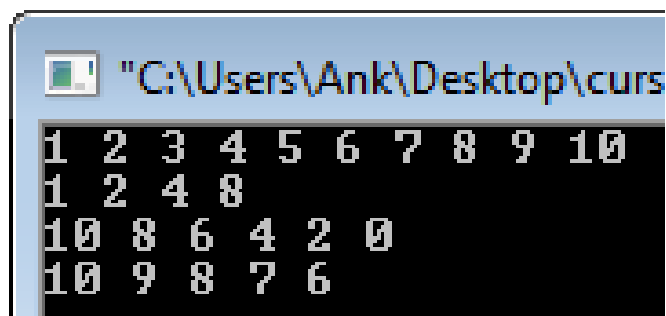
Structuri repetitive

Instrucțiunea **FOR**

Sintaxa:

```
for (expresie1; expresie2; expresie3)  
    set instructiuni;
```

Exemple (int)



```
"C:\Users\Ank\Desktop\curs  
1 2 3 4 5 6 7 8 9 10  
1 2 4 8  
10 8 6 4 2 0  
10 9 8 7 6
```

```
int x;  
char y;  
double z;  
  
for (x = 1; x <= 10; x++)  
    printf("%d ", x);  
  
printf("\n");  
  
for (x = 1; x <= 10; x = x * 2)  
    printf("%d ", x);  
  
printf("\n");  
  
for (x = 10; x >= 0; x = x - 2)  
    printf("%d ", x);  
  
printf("\n");  
  
for (x = 10; x > 5; x-- )  
    printf("%d ", x);
```



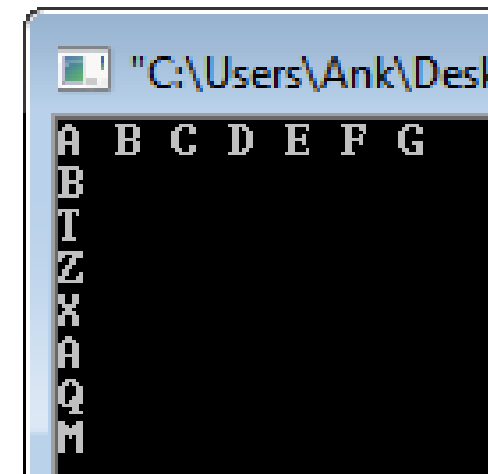

Fundamentele limbajului C

Structuri repetitive

Instrucțiunea **FOR**

Exemple (char)

```
int x;  
char y;  
double z;  
  
for (y = 'A'; y <= 'G'; y++)  
    printf("%c ", y);  
  
printf("\n");  
  
scanf("%c", &y);  
for( ; y != 'M'; )  
{  
    scanf("%c", &y);  
}
```





Fundamentele limbajului C

Structuri repetitive

Instrucțiunea **FOR**

Exemple (double)

```
int x;  
char y;  
double z;
```

"C:\Users\Ank\Desktop\curs 3\bin\Debug\curs 3.exe"

```
5.400000 5.200000 5.000000 4.800000 4.600000 4.400000 4.200000  
1.100000 2.100000 3.100000 4.100000 5.100000 6.100000
```

```
for (z = 5.4; z >= 4.1; z = z - 0.2)  
    printf("%1f ", z);
```

```
printf("\n");
```

```
z = 1.1;
```

```
for( ; ; )
```

```
{
```

```
    printf("%1f ", z);
```

```
    if (z > 5.1) break;
```

```
    z++;
```

```
}
```



Fundamentele limbajului C

Structuri repetitive

Instrucțiunea **WHILE**

Exemplu

Sintaxa:

while (expresie)
 set instructiuni;

```
x = 1546  
Suma cifrelor = 16
```

```
int x, s;  
  
do  
{  
    printf("x = ");  
    scanf("%d", &x);  
}while (x <= 0);
```

```
s = 0;  
while(x)  
{  
    s += x % 10;  
    x /= 10;  
}
```

```
printf("Suma cifrelor = %d \n\n", s);
```



Fundamentele limbajului C

Structuri repetitive

Instrucțiunea **DO WHILE**

Exemplu

Sintaxa:

```
do {  
    instructiuni;  
}while (expresie);
```

```
int x, s;
```

```
do  
{  
    printf("x = ");  
    scanf("%d", &x);  
}while (x <= 0);
```

```
s = 0;  
while(x)  
{  
    s += x % 10;  
    x /= 10;  
}  
  
printf("Suma cifrelor = %d \n\n", s);
```

```
"C:\Users\Ank\Desktop\c"  
x = -7  
x = -9  
x = -1  
x = 0  
x = 1546  
Suma cifrelor = 16
```



Fundamentele limbajului C

Structuri repetitive

Legatura dintre instructiunile repetitive

Suma si produsul primelor n nr naturale

int S=0, P=1, k;

```
for (k=1; k<=n; k++)  
{  
    S+=k; P*=k;  
}
```

```
k=1;  
while (k<=n)  
{  
    S+=k; P*=k;  
    k++;  
}
```

```
k=1;  
do  
{  
    S+=k; P*=k;  
    k++;  
} while (k<=n);
```

Ce valori vor avea variabilele S si P pentru n=0 ?



Fundamentele limbajului C

Structuri repetitive

Facilitati de intrerupere a unei secvente

Instructiunea Break

- asigura iesirea dintr-o bucla la nivelul imediat superior
- in cadrul instructiunii switch – pentru directionarea fluxului in afara instructiunii

Instructiunea Continue

- se utilizeaza pentru intreruperea executiei iteratiei curente



Fundamentele limbajului C

Structuri repetitive

Instrucțiunea **CONTINUE**

Exemplu

Sintaxa:

`continue;`

citește 10 numere
și află câte din ele
sunt pare

```
int  nrPare=0, N=10, k, nr;
for(k = 0; k < N; k++)
{
    scanf("%d", &nr);
    if ((nr % 2) != 0)
        continue;
    nrPare ++;
}
printf("nrPare = %d\n", nrPare);
```

4
7
1
3
5
5
8
4
9
9
5
nrPare = 3



Fundamentele limbajului C

Structuri repetitive

Sintaxa:

`break;`

la primul număr
impar citit se iese din
bucă (se citesc maxim
10 numere pare)

Exemplu

Instrucțiunea **BREAK**

```
int  nrPare=0, N=10, k, nr;
for(k = 0; k < N; k++)
{
    scanf("%d",&nr);
    if ((nr % 2) != 0)
        break;
    nrPare ++;
}
printf("nrPare = %d\n",nrPare);
```

```
"C:\Users\Ank
4
7
nrPare = 1
```




Fundamentele limbajului C

Funcții de citire / scriere 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 formate în concordanță cu specificatorii de format** (subșir care începe cu caracterul %) și inserate în locul și pe pozițiile acestora din cadrul șirului



Funcții de citire / scriere 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



Functii de citire / scriere cu format

scanf

- Șirul de formatare poate include următoarele elemente:
 - **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 nepreluare



Fundamentele limbajului C

Functii de citire / scriere cu format

– 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	Character
s	Șir de caractere
p	Adresa (pointer)



Agenda cursului

1. Fundamentele limbajului C
2. Complexitatea algoritmilor – notiuni introductive
 - complexitate : timp, spatiu de memorie, notatii asimptotice,
“time.h”



Complexitatea algoritmilor

Analiza complexității unui algoritm => determinarea resurselor de care acesta are nevoie pentru a produce datele de ieșire.

Resurse - timpul de executare
- spatiu de memorie etc.

Obs: Modelul masinii pe care va fi executat algoritmul nu presupune existenta operatiilor paralele (operatiile se executa secvential).

Notatie: $T(n)$ – timp de rulare al unui algoritm (in general masurat in nr. de comparatii sau de mutari)

Cazuri:

- cel mai favorabil
- cel mai nefavorabil
- mediu



Complexitatea algoritmilor

De ce se alege, in general, cazul cel mai defavorabil?

- este cel mai raspandit
- timpul mediu de executare este de multe ori apropiat de timpul de executare in cazul cel mai defavorabil
- ofera o limita superioara a timpului de executare (avem certitudinea ca executarea algoritmului nu va dura mai mult)



Complexitatea algoritmilor

**Intuitiv – Numărarea operațiilor elementare realizate de un algoritm
În cazul cel mai defavorabil**

Aflarea maximului unui vector de dimensiune n	# operații elementare realizate de algoritm în cazul cel mai defavorabil
<code>maxim = v[0];</code>	$\rightarrow 2$ (o indexare + o atribuire)
<code>for (i=1; i<n;i++)</code>	$\rightarrow 2n$ (o atribuire + n comparații + $(n-1)$ incrementări)
<code>if (maxim < v[i])</code>	$\rightarrow 2(n-1)$ ($n-1$ indexări + $n-1$ comparații)
<code>maxim = v[i];</code>	$\rightarrow 2(n-1)$ ($n-1$ indexări + $n-1$ atribuiri)
<code>return maxim</code>	$\rightarrow 1$ (o întoarcere a rezultatului)
$T(n) = 6n-1$ operații elementare	



Complexitatea algoritmilor

Notatii:

$T(n)$ – timp de rulare al unui algoritm (in general masurat in nr. de comparatii sau de mutari)

C_{\max} – numarul maxim de comparatii (obtinut in cazul cel mai defavorabil)

C_{\min} – numarul minim de comparatii (cazul cel mai favorabil)

M_{\max} – numarul maxim de mutari (operatii elementare) – caz defavorabil

M_{\min} – numarul minim de mutari – caz favorabil



Complexitatea algoritmilor

Exemple

1. Produsul a doua numere complexe ([1])

```
int main()
{
float a,b,c,d, p, q;
float t1, t2;
scanf("%f%f%f%f", &a, &b, &c, &d);
t1 = a*c; t2 = b*d; p = t1 - t2;
t1 = a*d; t2 = b*c; q = t1 + t2;
printf("Real = %f, Imaginar = %f", p, q);
}
```

- memorie pentru 8 variabile

Operatii elementare: 4 inmultiri, o adunare
si o scadere

```
int main()
{
float a,b,c,d, p, q;
float t1, t2, t3, t4;
scanf("%f%f%f%f", &a, &b, &c, &d);
t1 = a + b; t2 = t1 * c; t1 = d - c; t3 = a * t1;
q = t2 + t3; t1 = d + c; t4 = b * t1; p = t2 - t4;
printf("Real = %f, Imaginar = %f", p, q);
}
```

- memorie pentru 10 variabile

Operatii elementare: 3 inmultiri, 3 adunari si 2
scaderi

**Operatia de inmultire e mai costisitoare
decat adunarea / scaderea.**



Complexitatea algoritmilor

Exemple

2. Cmmdc a 2 numere

Euclid:

```
int a,b,r;  
scanf("%d%d", &a, &b);  
r = a % b;  
while (r!=0)  
{ a = b;  
  b = r;  
  r = a % b;  
}  
printf("Cmmdc = %d", b);
```

Scaderi repetate:

```
int a,b,r;  
scanf("%d%d", &a, &b);  
while (a != b)  
if (a > b )  
    a = a - b;  
else  
    b = b - a;  
printf("Cmmdc = %d", a);
```

Algoritm brut:

```
int a,b,c,i,min;  
scanf("%d%d", &a, &b);  
  
if (a < b) min = a;  
else min = b;  
  
for(i = 1 ; i <= min; i++)  
if ( %i==0 && b%i == 0)  
    c = i;  
printf("Cmmdc = %d", c);
```

Cate operatii se executa daca se citesc initial numerele 97 si 99?



Complexitatea algoritmilor

Exemple

3. Determinarea maximului dintr-un sir

```
int main()
{
    int a[100], n, i, max;
    // citire vector

    max = v[1];
    for(i = 2; i <= n; i++)
        if (max < v[i]) max = v[i];

    printf("Maximul = %",max);
    return 0;
}
```

Obs: $C_{\max} = C_{\min} = C_{\text{mediu}} = n - 1$

$M_{\min} = 0$ // maximul se afla pe prima pozitie

$M_{\max} = n - 1$ // maximul se afla pe ultima
pozitie in vectorul initial

$M_{\text{mediu}} = (n - 1) / 2$

Exemplu: $n = 6$

$v = (-32, 1, 56, 89, -20, 100)$

- max = 100, gasit dupa 5 comparatii

Tema:

1. Un program eficient pentru verificarea primalitatii unui numar.
2. Determinati simultan maximul si minimul dintr-un sir folosind $3n/2 + O(1)$ comparatii



Complexitatea algoritmilor

Exemple

4. Cautarea unei valori intr-un sir **ordonat** (Cautarea binara)

```
int main()
{
    int left = 0, right = n - 1;
    int mid = (left + right) / 2;
    while (left <= right && val != v[mid])
    {
        if (val < v[mid]) right = mid - 1;
        else left = mid + 1;
        mid = (left + right) / 2;
    }
    if (v[mid] == val) loc = mid;
    else loc = UNDEFINED;
}
```

- **$O(\log_2 n)$**

Exemplu: caut (fara succes) elementul 10 in
sirul $v = (20, 30, 40, 50, 60, 70, 80, 90, 100)$

- compar 10 cu 60 (elem din mijloc); $10 \neq 60$
- $10 < 60 \Rightarrow$ caut in $v = (20, 30, 40, 50)$

- compar 10 cu 30 (noul elem din mijloc)
- $10 < 30 \Rightarrow$ caut in $v = (20)$

- compar 10 cu 20 (unicul elem)
- cautare fara succes

$n = 9, \lceil \log_2 9 \rceil = 3$



Complexitatea algoritmilor

Exemple

5. Ordonarea unui sir folosind Interschimbarea directa

```
int main()
{
    int v[100], n, i, j, aux;
    // citire vector
    for (i = 1; i < n; i++)
        for(j = i+1; j <= n; j++)
            if (v[i] > v[j])
                { aux = v[i];
                  v[i] = v[j];
                  v[j] = aux;
                }
    // Afisare vector ordonat
}
```

Caz	Comparatii	Mutari
Cel mai favorabil	$n(n - 1) / 2$	0
Cel mai defavorabil	$n(n - 1) / 2$	$3n(n - 1) / 2$
mediu	$n(n - 1) / 2$	$3n(n - 1) / 4$

$O(n^2)$



Complexitatea algoritmilor

Exemple

6. Ordonarea unui sir folosind **Insertia directa**

```
int main()
{
  int v[100], n, i, j, aux;
  // citire vector
  for (i = 2; i<=n; i++)
  {
    x = v[i];
    j = i - 1;
    while (j>0 && x < v[j])
    {
      v[j+1] = v[j];
      j--;
    }
    v[j+1] = x;
  }
  // Afisare vector ordonat
}
```

Caz	Comparatii	Mutari
Cel mai favorabil	$n - 1$	$2(n - 1)$
Cel mai defavorabil	$\frac{1}{2} n^2 + \frac{1}{2} n - 1$	$\frac{1}{2} n^2 + \frac{3}{2} n - 2$
mediu	$(\frac{1}{2} n^2 + \frac{1}{2} n - 1)/2$	$C_{\text{mediu}} + 2(n - 1)$

$O(n^2)$

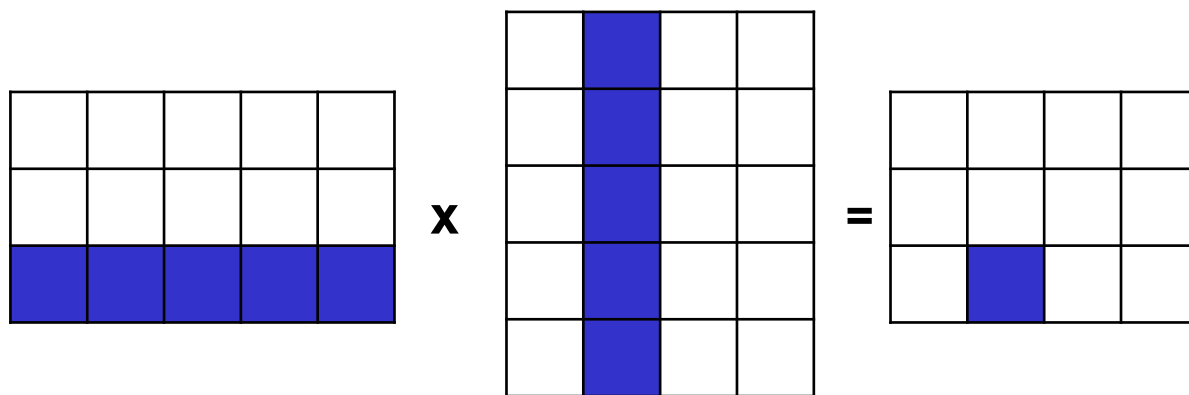


Complexitatea algoritmilor

Exemple

6. Inmultirea a doua matrice

```
int main()
{
  int a[10][20], b[20][30], c[10][30];
  int n, m, p, i, j, k;
  // citire matrice a si b
  for(i=1; i<=n; i++)
    for(k=1; k<=p; k++)
      { c[i][k] = 0;
        for(j=1; j<=m; j++)
          c[i][k] = c[i][k] + a[i][j] * b[j][k];
      }
  // Afisare matrice produs
}
```



$$A_{n,m} \times B_{m,p} = C_{n,p}$$

$O(m \cdot n \cdot p)$

Tema(*)

Inmultirea optima a unui sir de matrice =>
Programare dinamica => numar minim de
inmultiri

Programare Procedurala – Curs 3



Complexitatea algoritmilor

Notatia asimptotica

$T(n)$ – timp de rulare al unui algoritm (comparatii / mutari)

Obs: Exista un timp minim si un timp maxim de rulare

$$C_{\min} \leq T(n) \leq C_{\max}$$

Margini superioare (si inferioare) ([2])

Timp de rulare $T(n)$ - margine superioara

$$T(n) \leq \frac{1}{2}n^2 + \frac{3}{2}n - 2 \text{ (expl.)} \Rightarrow T(n) = O(n^2)$$

Timp de rulare $T(n)$ - margine inferioara

$$3(n-1) \leq T(n) \text{ (expl.)} \Rightarrow T(n) = \Omega(n)$$

Timp de rulare $T(n)$ in cazul cel mai nefavorabil

$$T(n) = a \cdot C_{\max} + b \text{ } (\exists \text{ const } a, b > 0) \Rightarrow T(n) = \Theta(n^2)$$



Complexitatea algoritmilor

Notatia asimptotica

comportarea lui $T(n)$ cind $n \rightarrow \infty$ ([2])

Formal

$f : \mathbb{N} \rightarrow \mathbb{R}_+$ (f asimptotic pozitiva)

$O(g) := \{f \mid \exists c > 0, \exists n_0 \text{ a.i. } 0 \leq f(n) \leq cg(n) \text{ oricare } n \geq n_0\}$

$\Omega(g) := \{f \mid \exists c > 0, \exists n_0 \text{ a.i. } 0 \leq cg(n) \leq f(n) \text{ oricare } n \geq n_0\}$

$\Theta(g) := \{f \mid \exists c_1, c_2 > 0, \exists n_0 \text{ a.i. } 0 \leq c_1g(n) \leq f(n) \leq c_2g(n) \text{ oricare } n \geq n_0\}$

[2] Ceterchi R. – Algoritmi si Structuri de Date (note de curs 2013)



Complexitatea algoritmilor

Fisierul antet time.h ([1])

Inclus pentru masurarea timpului

Contine prototipul functiei `clock()` => nr. de tacte de ceas de timp real scurs de la inceperea programului, pana in punctul in care se plaseaza aceasta functie

`clock()/CLK_TCK` => timpul in secunde (**Obs:** CLK_TCK este denumirea pentru CLOCKS_PER_SEC in versiunile anterioare Microsoft C.)

Plasand doua asemenea expresii, inaintea si dupa apelul subprogramului aflat sub masurare, diferenta lor da timpul consumat de algoritm.



Complexitatea algoritmilor

Fisierul antet time.h ([1])

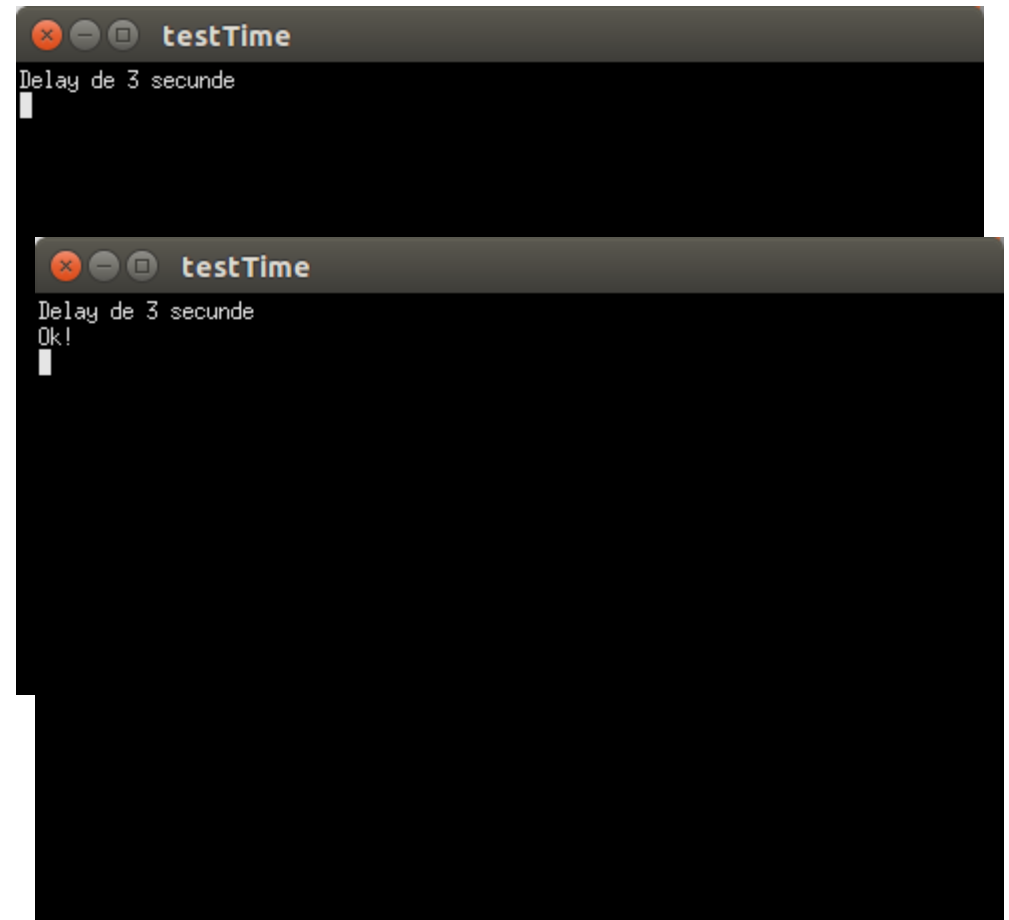
Exemplu

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

// pauza pentru un numar specificat de milisekunde
void sleep(clock_t f)
{
    clock_t g;
    g = f + clock();
    while (g > clock()) ;
}

int main()
{
    long i = 600000000L;
    clock_t start, finish;
    double duration;

    //Delay pentru un timp specificat
    printf("Delay de 3 secunde\n");
    sleep((clock_t)3*CLOCKS_PER_SEC);
    printf("Ok!\n");
}
```





Complexitatea algoritmilor

Fisierul antet time.h ([1])

Exemplu

```
//Masurarea duratei unui eveniment
printf("Timpul de executie a %ld bucle vide este ",i);
start = clock();
while(i--);
finish = clock();
duration = (double)(finish - start)/CLOCKS_PER_SEC;
printf("%.2lf secunde \n", duration);
```

```
testTime
Delay de 3 secunde
Ok!
Timpul de executie a 6000000000 bucle vide este 1.3 secunde

Process returned 0 (0x0)   execution time : 4.327 s
Press ENTER to continue.
```



Concluzii

- 1. S-au reamintit notiunile fundamentale ale limbajului C: tipuri de date, variabile, constante, expresii, instructiuni etc.**
- 2. S-au introdus cateva notiuni legate de complexitatea algoritmilor**



Perspective

Cursul 4:

1. Fundamentele limbajului C – directive de preprocesare
2. Tipuri derivate de date
 - Tablouri. Șiruri de caractere.
 - Structuri, uniuni, câmpuri de biți, enumerări.
 - Pointeri.