

# **Programare orientată obiect**

## **Obiective**

- **Cunoașterea și înțelegerea conceptelor specifice programării orientate obiect**
- **Abilități de programare în limbajele de programare C și C++**

## **Obiectivele specifice:**

- Scrierea de programe de scară mică/mijlocie cu interfețe grafice utilizator folosind C++ și QT.
- Proiectarea orientată obiect pentru programe de scară mică/mijlocie
- Explicarea/Înțelegerea structurilor de tip clasă ca fiind componente fundamentale în construirea aplicațiilor.
- Înțelegerea rolului moștenirii, polimorfismului, legării dinamice și a structurilor generice în realizarea codului reutilizabil.
- Utilizarea claselor/modulelor scrise de alți programatori în dezvoltarea sistemelor proprii

## Bibliography

1. A.V. Aho, J.E. Hopcroft, J.D. Ullman, Data Structures and Algorithms, Addison-Wesley Publ., Massachusetts, 1983.
2. R. Andonie, I. Garbacea, Algoritmi fundamentali. O perspectiva C++, Editura Libris,
3. Alexandrescu, Programarea moderna in C++. Programare generica si modele de proiectare aplicative, Editura Teora, 2002
4. M. Frentiu, B. Parv, Elaborarea programelor. Metode si tehnici moderne, Ed. Promedia, Cluj-Napoca, 1994.
5. E. Horowitz, S. Sahni, D. Mehta, Fundamentals of Data Structures in C++, Computer Science Press, Oxford, 1995.
6. K.A. Lambert, D.W. Nance, T.L. Naps, Introduction to Computer Science with C++, West Publishing Co., New-York, 1996.
7. L. Negrescu, Limbajul C++, Ed. Albastra, Cluj-Napoca 1996.
8. Dan Roman, Ingineria programarii obiectuale, Editura Albastra, Cluj\_Napoca, 1996.
9. B. Stroustup, The C++ Programming Language, Addison Wesley, 1998.
10. Bruce Eckel, Thinking in C++, [www.bruceeckel.com](http://www.bruceeckel.com)

## Activități:

Curs: 2h/sapt

Seminar: 1h/sapt

Lab:2h/sapt

## Notare:

- Nota Laborator (30%)
- Examen Practic (40%)
- Examen Scris (30%)

Cerinte: Minim 5 la fiecare (lab, examen scris, examen practic)

Minim 12 Prezente la laborator

Minim 5 Prezente seminar

## Contact:

Mail: [istvanc@cs.ubbcluj.ro](mailto:istvanc@cs.ubbcluj.ro)

Adresa web: [www.cs.ubbcluj.ro/~istvanc/oop](http://www.cs.ubbcluj.ro/~istvanc/oop)

# Curs 1 – Programare orientată obiect

- Introducere - POO
- Limbajul C
  - sintaxă
  - instrucțiuni
  - tipuri de date
  - funcții

## **Evoluția limbajelor de programare**

### Cod mașină

- programul în format binar, executat direct de procesor

### Limbaj de asamblare

- instrucțiuni în format binar înlocuit cu mnemonice, etichete simbolice pentru acces la memorie

### Procedural

- descompune programul în proceduri/funcții

### Modular

- descompune programul în module

### Orientat obiect

- descompune programul într-o mulțime de obiecte care interacționează

# Paradigma de programare orientată-obiect

Este metodă de proiectare și dezvoltare a programelor:

- Oferă o abstractizare puternică și flexibilă
- Programatorul poate exprima soluția în mod mai natural (se concentrează pe structura soluției nu pe structura calculatorului)
- Descompune programul într-un set de obiecte, obiectele sunt elementele de bază
- Obiectele interacționează pentru a rezolva problema, există relații între clase
- Tipuri noi de date modelează elemente din spațiul problemei, fiecare obiect este o instanță a unui tip de data (clasă)

Un obiect este o entitate care:

- are o stare
- poate executa anumite operații (comportament)

Poate fi privit ca și o combinație de:

- date (atribute)
- metode

## Concepte:

- obiect
- clasă
- metodă (mesaj)

## Proprietăți:

- abstractizare
- încapsulare
- moștenire
- polimorfism

## Limbajul de programare C/C++

### De ce C/C++:

- Folosit la scară largă atât în mediul academic cât și în industrie
- limbaj hibrid , implementează toate conceptele necesare pentru programare orientat obiect (C++)
- multe limbaje de programare au evoluat din C/C++ (Java, C#). Este mult mai ușor să înveți aceste limbaje dacă știi C/C++

### Avantaje:

- **Concis, cod lizibil**
- **Performanță:** programele scrise în c/c++ în general sunt mai rapide decât cele scrise în alte limbaje de programare
- **Întreținere:** codul fiind concis programatorul are de întreținut un volum mai mic de cod
- **Portabil:** se pot scrie aplicații pentru orice fel de procesor, sistem de operare
- **Productivitate:** C++ a fost creat cu scopul principal de a mari productivitate (față de C).

Limbajul C - 1972 creat de Dennis M. Ritchie la Bell Telephone Laboratories pentru a dezvolta sistemul de operare UNIX

## Limbaje compilate. Procesul de compilare

Programul C/C++ trebuie compilat pentru a putea fi executat.

Programul scris în fișiere text ( fișiere sursă ) trebuie transformat în cod binar ce poate fi executat de procesor:

**Fișiere sursa** - fișiere text, conțin programul scris într-un limbaj de programare

| - compilatorul, analizează fișierele și crează fișiere obiect

**Fișiere Obiect** - fișiere intermediare, conține bucați incomplete din programul final

| - linker, combină mai multe fișiere obiect și crează programul care poate fi executat de calculator

**Executabil**

| - sistemul de operare, încarcă fișierul executabil în memorie și execută programul

|  
**Program în memorie**

În C/C++, pașii sunt executate înaintea rulării programului .

În unele limbaje transformarea se execută în timpul rulării. Acesta este una dintre motivele pentru care C/C++ în general are o performanță mai bună decât alte limbaje mai recente.

**Python (Interpretat) vs C/C++ (compilat)**

## **Mediu de dezvoltare pentru C/C++ (IDE - Integrated Development Environment)**

### **Compiler**

**MinGW** - - Minimalist GNU for Windows, implementare nativă Windows pentru compilatorul GNU Compiler Collection (GCC)

**MinSYS** – colecție de utilitare GNU (bash, make, gawk, grep)

### **Eclipse IDE**

Eclipse CDC – mediu de dezvoltare integrat pentru C/C++ bazat pe platforma Eclipse

Colecție de pluginuri care oferă instrumentele necesare pentru a dezvolta aplicații în C/C++

Visual Studio C++

Mediu de dezvoltare integrat pentru C/C++ (alte limbaje: C#, python, etc)

Folosește compilatorul C/C++ de la Microsoft

### **Project C - Hello Word**



## Elemente de bază

C/C++ este case senzitive (a <> A)

### Identificator:

- Secvență de litere si cifre , începe cu o literă sau “\_”(underline).
- Nume pentru elemente din program (nume variabile, funcții, tipuri, etc)
- Ex. i, myFunction, rez,

### Cuvinte rezervate (Keywords):

- Identificatori cu semantica specială pentru compilator
- int, if, for, etc.

### Literals:

- Constante specificate direct în codul sursă
- Ex. “Hello”, 72, 4.6, ‘c’

### Operatori:

- aritmetici, pe biti, relaționali, etc
- +, -, <<

### Separatori:

- Semne de punctuație folosite pentru a defini structura programului
- ; { } , ( )

### Whitespace:

- Caractere ignorate de compilator
- space, tab, linie nouă

### Commentarii:

- // this is a single line comment
- /\* This is a  
\* multiline comment  
\*/
- sunt ignorate de compilator

## Tipuri de date

Un tip de date definește domeniul de valori și operațiile ce sunt definite pentru valorile din domeniu

### Tipuri de date (Built in):

Name	Description	Size	Range
char	Character or small integer.	1byte	signed: -128 to 127 unsigned: 0 to 255
short int (short)	Short Integer.	2bytes	signed: -32768 to 32767 unsigned: 0 to 65535
int	Integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
long int (long)	Long integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
float	Floating point number.	4bytes	+/- 3.4e +/- 38 (~7 digits)
double	Double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)
long double	Long double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)
wchar_t	Wide character.	2 or 4 bytes	1 wide character

- operații: +, -, \*, /, %
- relații: <, >, <=, >=, ==, !=
- operațiile se pot executa doar dacă operanzii sunt de tipuri compatibile
- precedența operatorilor dictează ordinea de execuție

## Vectori

Daca **T** e un tip de dată:

- $T[n]$  – este un vector cu  $n$  elemente de tip  $T$
- indicii sunt de la 0 la  $n-1$
- operatorul de indexare:  $[ ]$
- vector multidimensional :  $t[n][m]$

```
#include <stdio.h>

int main() {
    int a[5]; // create an array with 5 elements
    a[0] = 1; //index start from 0
    a[1] = 2;
    printf("a[0]=%d \n", a[0]);
    printf("a[1]=%d \n", a[1]);
    //!!! a[2] uninitialised
    printf("a[2]=%d \n", a[2]);

    int b[] = { 1, 2, 3, 5, 7 };
    printf("b[0]=%d \n", b[0]);
    b[0] = 10;
    printf("b[0]=%d \n", b[0]);
    return 0;
}
```

# C String

- Sunt reprezentate ca un vector de caractere **char**, ultimul caracter este **'\0'** (marchează sfârșitul șirului)
- se folosește ca și orice vector C
- C include un modul pentru a manipula C-string-uri (**<string.h>**)

**strlen** – Returnează numărul de caractere dintr-un C string

**strcpy** – Copiază caractere de la sursa în C stringul destinație.

-Obs. Nu putem folosi operatorul = pentru a le copia (nu se pot copia nici vectori normali)

**strcmp** – comparare de C stringuri returnează zero: a==b, negativ:a<b, pozitiv:a>b.

- Obs. Folosirea operatorilor ==,<,> pe C strings (sau orice array) compară adrese de memorie

**strcat** – Concatenează sursa cu destinație (adaugă la sfârșitul Cstringului destinație)

Obs. Nici o metoda nu alocă memorie sau verifică dacă există suficient loc pentru operație.

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

int main() {
    char name[100];

    strcpy(name, "Popescu");

    printf("name:%s l=%d", name, strlen(name));

    char name2[100];

    strcpy(name2, name);
    printf("name:%s l=%d", name2, strlen(name2));

    while (getchar() != 'e'){};

    return 0;
}
```

# Structuri (Record)

- este o colecție de elemente de tipuri diferite
- permite gruparea diferitelor tipuri de date simple într-o structură

<b>struct</b> name{ type1 field1; type2 field2 }	<b>struct</b> car{ int year; int nrKm; }	car c; c.year = 2010 c.nrKm = 30000;
---	---	--

```
#include <stdio.h>

//introduce a new struct called Car
typedef struct {
    int year;
    int km;
} Car;

int main() {
    Car car, car2;

    //initialise fields
    car.year = 2001;
    car.km = 20000;

    printf("Car 1 fabricated:%d Km:%d \n", car.year, car.km);

    //!!! car2 fields are uninitialised
    printf("Car 1 fabricated:%d Km:%d \n", car2.year, car2.km);
    return 0;
}
```

## Declarații de variabile

- Introduce un nume în program, asociază numele cu un tip
  - Se alocă memorie conform tipului variabilei
- Tipul variabilei este folosit de compilator pentru a decide care sunt valorile și operațiile posibile
  - Valoarea este nedefinită după declarare.
  - Este recomandat să combinăm declarația cu inițializarea
    - Trebuie ales nume sugestiv pentru orice variabilă din program
    - Variabila trebuie inițializată după declarație cu o valoare

<type> <identifier>

Ex. int i

long rez

int j=7

## Constante

- numerice: 1, 12, 23.5
- sir de caractere: "Hello World"
- caracter: 'c'

## Rules and recommendations (*Industrial Strength C++ Mats Henricson, Erik Nyquist*)

### Names (variables, constants, types, functions ...)

If names are not chosen, written and administrated with care, then you will end up with a program that is hard to understand, read and maintain.

- ☐ Use meaningful names.
- ☐ Use English names for identifiers.
- ☐ Be consistent when naming functions, types, variables and constants

### Comments

Specify every method using comments

Inside a function/method: Comments are unfortunately hard to maintain, so with a few exceptions they should only explain what is not obvious from reading the program itself.

- Each file should contain a copyright comment.
- Each file should contain a comment with a short description of the file content.
- Each method should have specifications included in a comment
- Every file should declare a local constant string that identifies the file.
- Use // for comments.
  
- All comments should be written in English.

## Referințe și pointeri

- Pointer este un tip de date special, folosit pentru a lucra cu adrese de memorie
- poate stoca adresa unei variabile, adresa unei locații de memorie

## Declarare

- ca și orice variabilă de alt tip doar ca se pune '\*' înaintea numelui variabilei.
- Ex: `int *a; long *a, char *a`

## Operatori

- *address of* '&': - returneaza adresa de memorie unde este stocat valoarea dintr-o variabilă
- *dereferencing* '\*' - returneaza valoarea stocata în locația de memorie specificată

```
#include <stdio.h>

int main() {
    int a = 7;
    int *pa;

    printf("Value of a:%d address of a:%p \n", a, &a);
    //assign the address of a to pa
    pa = &a;
    printf("Value of pa:%d address of pa:%p \n", *pa, pa);

    //a and pa refers to the same memory location
    a = 10;
    printf("Value of pa:%d address of pa:%p \n", *pa, pa);
    return 0;
}
```



## Instrucțiuni

Toate instrucțiunile se termina cu: ; (excepție instrucțiunea compusă)

Expresii: `a = b+c; i++; a==b`

Instrucțiune vidă: ;

Instrucțiunea compusă:

```
{  
//multiple statements here  
}
```

## Atribuire

- Operator de atribuire: =
- Atribuire o valoare la o variabilă (inițializează sau scimba valoare variabilei)

## if, if-else, else if

```
if (condition){  
    //statements executed only if the condition is true  
}
```

```
if (condition){  
    //statements executed if the condition is true  
} else {  
    //statements executed only if the condition is not true  
}
```

```
if (condition1){  
    //statements executed if the condition1 is true  
} else if (condition2){  
    //statements executed only if condition1 is not true and the condition2 is true  
}
```

- condition, condition1, condition2 - sunt expresii
- orice expresie are o valoare
- valoarea 0 înseamnă fals, orice altă valoare înseamnă adevărat

## switch-case

```
switch(expression)
{
    case constant1:
        statementA1
        statementA2
        ...
        break;
    case constant2:
        statementB1
        statementB2
        ...
        break;
    ...
    default:
        statementZ1
        statementZ2
        ...
}
```

Se evaluează expresia, dacă valoarea este egal cu constant1 atunci tot ce e pe ramura **case** constant1: se execută până la primul **break**;

Dacă nu e egal cu constant1 se verifică cu constant2, 3...

Dacă nici o constantă nu este egală cu rezultatul expresiei atunci se execută ramura **default**

## Instrucțiuni repetitive

Execută repetat un set de instrucțiuni

### while , do-while

```
while(condition)
{
    statement1
    statement2
    ...
}
```

Cât timp condiția este adevărată ( $\neq 0$ ) se execută corpul de instrucțiuni

```
do
{
    statement1
    statement2
    ...
}
while(condition);
```

## for

```
for(initialization; condition; incrementation)  
{  
    //body  
}
```

*initialization* – inițializează una sau mai multe variabile

*incrementation* – se execută la fiecare iterație

*condition* – se verifică la fiecare iterație, cât timp e adevărat corpul de instrucțiuni se execută

<pre><b>for</b>(<i>initialization</i>; <i>condition</i>; <i>incrementation</i>) {     statement1     statement2     ... }</pre>	<pre><i>initialization</i> <b>while</b>(<i>condition</i>) {     statement1     statement2     ...     <i>incrementation</i> }</pre>
---	---

## Citire/Scriere

**printf()** - tipărește în consola (la ieșirea standard)

```
#include <stdio.h>

int main() {
    int nr = 5;
    float nrf = 3.14;
    char c = 's';
    char str[] = "abc";

    printf("%d %f %c %s", nr, nrf , c, str);
    return 0;
}
```

**scanf()** - citește de la tastatura

```
int main() {
    int nr;
    float f;
    printf("Enter a decimal number:");
    //read from the command line and store the value in nr
    scanf("%d", &nr);
    printf("The number is:%d \n", nr);

    printf("Enter a float:");
    if (scanf("%f", &f) == 0) {
        printf("Error: Not a float:");
    } else {
        printf("The number is:%f", f);
    }
    //wait until user enters 'e'
    while(getchar()!='e');
    return 0;
}
```

# Calculator numere raționale

## Problem statement

Profesorul are nevoie de un program care permite elevilor să învețe despre numere raționale. Programul ajută studenții să efectueze operații aritmetice cu numere raționale

```
#include <stdio.h>

int main() {
    int totalM = 0;
    int totalN = 1;
    int m;
    int n;
    while (1) {
        printf("Enter m, then n to add\n");
        scanf("%d", &m);
        scanf("%d", &n);
        totalM = totalM * n + m * totalN;
        totalN = totalN * n;
        printf("Total: %d/%d\n", totalM, totalN);
    }
    return 0;
}
```

## Funcții

*Funcția main este executat când lansam în execuție un program C/C++*

### Declarare

<result type> name ( <parameter list>);

<result-type> - tipul rezultatului, poate fi orice tip sau *void* *daca funcția nu returnează nimic*

<name> - numele funcției

<parameter-list> - parametrii formali

Corpul funcției nu face parte din declarare

### Definiție

```
<result type> name(<parameter list>){  
//statements - the body of the function  
}
```

- **return** <exp> rezultatul expresiei se returnează, execuția funcției se termina
- o funcție care nu este void trebuie neapărat să returneze o valoare prin expresia ce urmează după **return**
- declararea trebuie să corespundă cu definiția (numele parametrilor poate fi diferit)



## Specificații

- Nume sugestiv
- O scurtă descriere a funcției (ce face)
- Semnificația parametrilor
- condiții asupra parametrilor (precondiții)
- ce se returnează
- relația dintre parametri și rezultat (post condiții)

```
/*  
 * Verify if a number is prime  
 * nr - a number, nr>0  
 * return !=0 if the number is prime (1 and nr are the only dividers)  
 */  
int isPrime(int nr);
```

**precondiții** - sunt condiții care trebuie să fie satisfăcute de parametrii actuali înainte de a executa corpul funcției

**postcondiții** - condiții care sunt satisfăcute după execuția funcției

## Apelul de funcții

name (<parameter list>);

- Toate expresiile date ca parametru sunt evaluate înainte de execuția funcției
- Parametrii actuali trebuie să corespundă cu parametri formali (număr, poziție, tip)
- declarația trebuie să apară înainte de apel

## Supraîncărcare (Overloading)

- Pot exista mai multe funcții cu același nume (dar parametrii formali diferiți)
- La apel se va executa funcția care corespunde parametrilor actuali

### Vizibilitate (scope)

Locul unde declarăm variabila determină vizibilitate lui (unde este variabila accesibilă).

Variabile, funcții declarate în interiorul unei instrucțiuni compuse (`{ }`) sunt vizibile doar în interiorul instrucțiunii compuse

Funcțiile au domeniul lor de vizibilitate - variabilele definite în interiorul funcțiilor sunt accesibile doar în funcție, ele sunt distruse după apelul funcției. (**variabile locale**)

Variabilele definite în afara funcțiilor sunt accesibile în orice funcție (**variabile globale**).

## Transmiterea parametrilor : prin valoare sau prin referință

### Transmitere prin valoare:

La apelul funcției se face o copie a parametrilor.

Schimbările făcute în interiorul funcției nu afectează variabilele exterioare.

Este mecanismul implicit de transmitere a parametrilor în C

### Transmitere prin referință:

La apelul funcției se transmite adresa locației de memorie unde se află valoarea variabilei.

Modificările din interiorul funcției sunt vizibile și în afară.

```
void byValue(int a) {
    a = a + 1;
}

void byRef(int* a) {
    *a = *a + 1;
}

int main() {
    int a = 10;
    byValue(a);
    printf("Value remain unchanged a=%d \n", a);
    byRef(&a);
    printf("Value changed a=%d \n", a);
    return 0;
}
```

Vectorul este transmis prin referință

## Calculator

```
/*
 * Return the greatest common divisor of two natural numbers.
 * Pre: a, b >= 0, a*a + b*b != 0
 */
int gcd(int a, int b) {
    if (a == 0) {
        return b;
    } else if (b == 0) {
        return a;
    } else {
        while (a != b) {
            if (a > b) {
                a = a - b;
            } else {
                b = b - a;
            }
        }
        return a;
    }
}

/*
 * Add (m, n) to (toM, toN) - operation on rational numbers
 * Pre: toN != 0 and n != 0
 */
void add(int* toM, int* toN, int m, int n) {
    *toM = *toM * n + *toN * m;
    *toN = *toN * n;
    int gcdTo = gcd(abs(*toM), abs(*toN));
    *toM = *toM / gcdTo;
    *toN = *toN / gcdTo;
}

//global variables. Store the current total
int totalM = 0;
int totalN = 1;
int main() {
    int m,n;
    while (1) {
        printf("Enter m, then n to add\n");
        scanf("%d", &m);scanf("%d", &n);
        add(&totalM, &totalN, m, n);
        printf("Total: %d/%d\n", totalM, totalN);
    }
    return 0;
}
```

# Curs 1

- Introducere - POO
- Limbajul C
  - sintaxă
  - tipuri de date
  - instrucțiuni
  - funcții