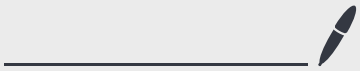
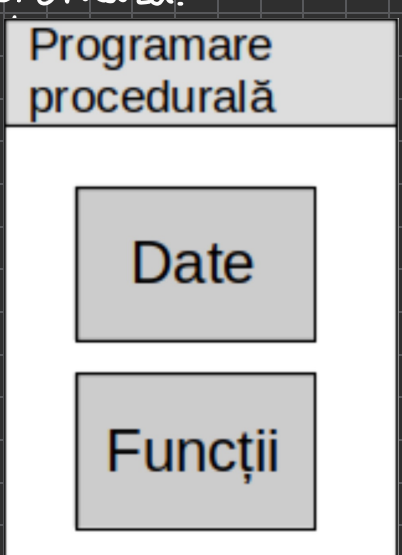
Clase si obiecte C++

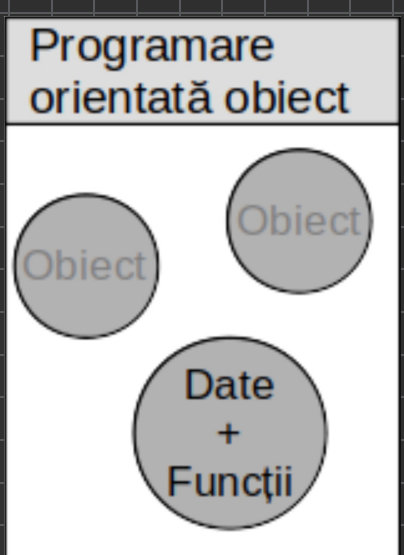


Generalitati

În programarea procedurală un program este alcătuit din două componente distincte: **date** și **funcții**, funcțiile prelucrând datele la care au acces prin intermediul parametrilor.

Programarea orientată pe obiect (OPP) este o modalitate de proiectare a programelor în care datele prelucrate și operațiile cu acestea sunt încapsulate în aceeași structură, numită obiect. Funcțiile care fac parte dintr-un obiect au acces la datele care caracterizează acel obiect, iar un program este alcătuit din mai multe obiecte, care interacționează.





Programarea orientată pe obiecte se bazează pe următoarele principii.

- încapsulare = mecanismul prin care atât datele, cât și funcțiile sunt plasate în aceeași structură, numită clasă și stabilirea nivelului de acces la conținutul acesteia;
- abstractizare = identificarea datelor și funcțiilor relevante pentru o anumită clasă;
- moștenire = proprietatea claselor de a prelua date și metode ale unor clase definite anterior. Clasa inițială se numește clasă de bază, iar cea nouă se numește clasă derivată;
- polimorfism = posibilitatea ca atât clasa de bază, cât și clasa derivată să conțină metode cu același nume, dar diferite ca funcționalitate.

Noțiunile de clasă și obiect

Clasa este un tip de date, similar cu struct. Câmpurile clasei sunt de tip dată și de tip funcție.

Câmpurile clasei se mai numesc **membri**, astfel avem, **date membru** și **funcții membre**.

Obiectul reprezintă o dată de tipul clasei – ex. o variabilă.

Pentru membrii unei clase se precizează anumite caracteristici, numite **modificatori de acces**, care stabilesc cum se face accesul la aceștia:

- **private** – interzic accesul la date și funcții în afara clasei;
- **public** – permit accesul la date și funcții în afara clasei;
- **protected** – interzic accesul din afara clasei, dar îl permit din clasele derivate

Exemplu

```C++
#include <iostream>

using namespace std;

class Fractie{
    private:
        int numarator, numitor;
    public:
        void afiseaza(){
            cout << numarator << "/" << numitor << " ";
        }
        void seteaza(int , int);
};

void Fractie::seteaza(int a , int b){
    numarator = a , numitor = b;
}

int main(){
    Fractie F;
    F.seteaza(3 , 4);
    F.afiseaza();
    /// F.numarator = 7; //eroare, data este privata

    return 0;
}
```

- Fractie este numele clasei;
- F este numele obiectului;
- clasa are două date membre: numărător și numitor. Ele sunt private - nu pot fi accesate din exteriorul clasei;
- clasa are două funcții membre: seteaza(), care dă valori datelor membre și afiseaza(), care afișează datele membre. Ele sunt publice.

Observații

- Clasele se definesc similar cu structurile

```
class Fractie{
    /// ...
};
```
C++

Diferența dintre clase și structuri este că accesul la câmpurile unei structuri este implicit public, iar accesul la membrii unei clase este implicit privat.

- Accesul la membrii clasei se face prin operatorul . sau în cazul pointerelor la clase prin ->.

```
Fractie X;
X.afiseaza();
```
C++

- Obiectele unei clase se declară la fel cum se declară variabilele de orice tip.

```C++
Fractie x, y; /// două obiecte
Fractie V[ 10 ]; /// un tablou cu 10 elemente de tip obiect
Fracte * p; /// pointer la obiect. Obiectul încă nu există!
```

- Între două obiecte ale aceleiași clase, se pot realiza atribuiri.

```C++
Fractie x , y;
x.seteaza(3 , 4);
y = x;
y.afiseaza();
```

- Putem declara pointeri la obiecte. Accesul la datele și funcțiile membru se face prin intermediul operatorului `->`.

```C++
Fractie x , * p;
x.seteaza(3 , 4);
p = & x;
p->afiseaza();
```

- Funcțiile unei clase pot fi declarate astfel:
  - plasăm în interiorul clasei definiția funcției, obținând o **metodă inline** — în exemplul de mai sus funcția `afiseaza()` a fost scrisă în acest mod.
  - plasăm în interiorul clasei numai antetul funcției, iar definiția o plasăm în exteriorul clasei. În exemplul de mai sus am scris în acest

mod. metoda (funcția) seteaza(). Pentru a preciza că este vorba despre o metodă a clasei și nu o funcție oarecare se folosește operatorul ::

```cpp
void Fractie::seteaza(int a , int b){
    numarator = a , numitor = b;
}
```

Constructori

Constructorul reprezintă un mecanism prin care datele membre primesc valori la crearea obiectului.

- constructorii sunt metode (funcții membre) și se apelează la declararea obiectului
- Constructorul:
  - este o funcție fără tip
  - are același nume cu clasa
- o clasă poate avea multipli constructori care diferă prin numărul și tipul parametrilor
- constructorii sunt funcții, iar blocul lor poate conține orice fel de instrucțiune

Exemplu

```C++
class Fractie{
    private:
        int numarator, numitor;
        void simplifica(){
            int a = numarator, b = numitor , r;
            while(b)
                r = a % b, a = b, b = r;
            numarator /= a, numitor /= a;
        }
    public:
        void afiseaza(){
            cout << numarator << "/" << numitor << endl;
        }
        void seteaza(int a , int b){
            numarator = a , numitor = b;
        }
        Fractie(){
            numarator = 0, numitor = 1;
        }
        Fractie(int a){
            numarator = a, numitor = 1;
        }
        Fractie(int a , int b){
            if(b == 0)
                b = 1;
            numarator = a, numitor = b;
            simplifica();
        }
};

int main(){
    Fractie x;
    x.afiseaza();
    Fractie y(3);
    y.afiseaza();
    Fractie z(3 , 2);
    z.afiseaza();
    return 0;
}
```

Observații

- dacă o clasă are doar constructori cu parametri, constructorul implicit nu mai există. În consecință, nu se pot declara obiecte fără parametrii, ceea ce este de multe ori necesar. De aceea, dacă o clasă are constructor, este necesar să aibă și constructor fără parametri.

Constructori de copiere

Constructorul de copiere primește ca parametru o referință la un obiect deja creat. Scopul acestui constructor este de a copia datele dintr-un obiect în alt obiect, din aceeași clasă, care există deja. Acest lucru este folositor atunci când există pointeri deoarece cu operatorul = nu se pot copia datele

Sintaxa este:

```C++
NumeClasa (const NumeClasa &);
```

Exemplu:

```C++
Fractie(const Fractie & F)
{
    numarator = F.numarator();
    numitor = F.numitor();
    simplifica();
}
```

Deconstructor

Deconstructorul este o metodă publică care se apelează la eliminarea din memorie a unui obiect. O clasă poate avea un singur destructor, iar numele lui este identic cu cel al clasei, dar precedat de caracterul ~. Destructorul este o funcție fără tip și fără parametri.

Exemplu

```C++
class Fractie{
    private:
        int numarator, numitor;
    public:
        ~Fractie(){
            cout << "Fractia " << numarator << "/" << numitor <<
        }
};

int main(){
    Fractie x;

    return 0;
}
```

Funcții prietene

Funcțiile prietene permit accesarea membrilor privați ai unui obiect în interiorul unei funcții care nu este metodă a obiectului. Funcțiile prietene sunt:

- declarate în interiorul clasei; antetul lor este precedat de cuvântul friend
- funcția prieten are un parametru de tipul clasei; în funcție se pot accesa membrii privați ai acesteia

Exemplu.

```C++
class Fractie{
    private:
        int numarator, numitor;
    public:
        friend void Afiseaza(Fractie F);
};

void Afiseaza(Fractie F)
{
    cout << F.numarator << "/" << F.numitor << endl;
}

int main(){
    Fractie x;
    Afiseaza(x);

    return 0;
}
```

Cuvântul cheie this

This reprezintă un pointer către obiectul curent

Este necesar:

- când parametrii unor metode au același nume cu datele membru. De exemplu o clasă conține variabila x și o funcție cu un parametru x, atunci variabila poate fi referită prin expresia this->x.
- când se dorește ca o funcție să returneze obiectul.

Exemplu:

```
#include<iostream>
using namespace std;

/* local variable is same as a member's
name */
class Test
{
private:
   int x;
public:
   void setX (int x)
   {
       // The 'this' pointer is used to
retrieve the object's x
       // hidden by the local variable 'x'
       this->x = x;
   }
   void print() { cout << "x = " << x <<
endl; }
};

int main()
{
   Test obj;
   int x = 20;
   obj.setX(x);
   obj.print();
   return 0;
}
```