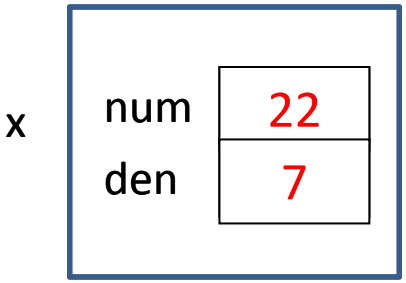
# Chapitre 1 :Programmation Orientée Objet :

Concepts de base de l’orienté objet.

# Déclaration de classes

```
#include <iostream>
using namespace std;
class Ratio
{    private :        int num, den;
  public: void assign(int n, int d) {num=n; den=d;}
             double convert() {return double(num)/den;}
             void invert() {int temp=num; num=den; den = temp;}
             void print() {cout << num << '/' << den;};
};
main()
{ Ratio x;
  x.assign(22,7);
  cout << "x = " ;  x.print();
  cout << " = " << x.convert() << endl;
  x.invert();  cout << "1/x = ";  x.print();
  cout << endl; }
```




x=22 / 7

=3,14285

1/x=7 / 22

## Visibilité des membres d'une classe

Les membres d'une classe peuvent être déclarés avec les qualificatifs suivants :

private : (valeur par défaut) : non accessibles en dehors de la classe

public : accessibles en dehors de la classe, ces membres constituent l'interface de la classe.

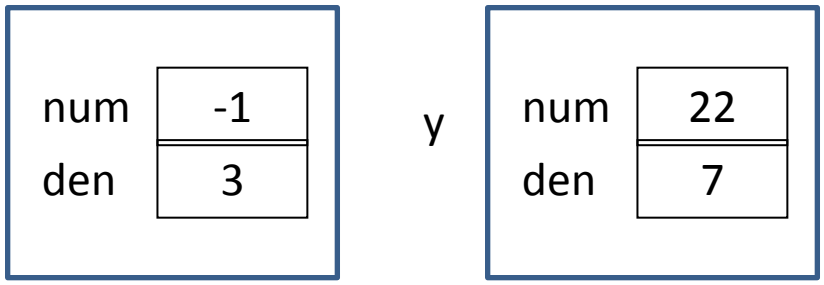
protected : concerne la notion d'héritage et les classes dérivées. Une sous classe peut accéder aux membres *protected* d'une classe parent, ce qui n'est pas le cas avec les membres *private*.

# Constructeurs

Un constructeur est une méthode appelée, automatiquement, dès qu'un objet est déclaré. Il porte le nom de la classe et ne renvoie pas de résultat.

```
#include <iostream>
using namespace std;
class Ratio
{ public:     Ra tio(int n, int d) { num = n; den = d; }
                    void print() { cout << num << '/' << den; }
  private:          int num, den;
};
main()
{ Ratio x(-1,3), y(22,7);
  cout << "x = " ;  x.print();
  cout << "  et y = " ;   y.print();
}
```

Constructeur




x=-1/3

et y=22/7

# Listes d'initialisations

```
#include <iostream>
using namespace std;
class Ratio
{ public:        Ratio() : num (0), den(1) { }
                 Ratio(int n) : num (n), den (1) { }
                 Ratio(int n, int d) : num (n), den (d) { }
                 void print() { cout << num << '/' << den; }
  private:       int num, den;
};
main()
{Ratio x, y(4), z(22,7);
  cout << "x = " ;  x.print();
  cout << "\ny = " ;  y.print();
  cout << "\nz = " ;  z.print();
}
```

x=0/1

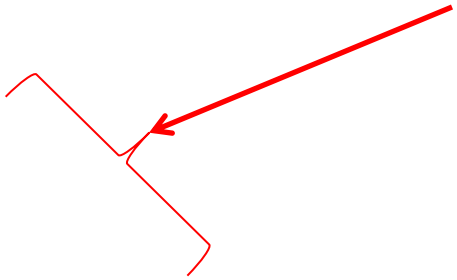
y=4/1

z=22/7

# Constructeurs multiples

```
#include <iostream>
using namespace std;
class Ratio
{ public:          Ratio() { num = 0; den = 1; }
                   Ratio(int n) { num = n; den = 1; }
                   Ratio(int n, int d) { num = n; den = d; }
                   void print() { cout << num << '/' << den; }
  private:         int num, den;
};
main()
{Ratio x, y(4), z(22,7);
  cout << "x = " ;  x.print();
  cout << "\ny = " ;  y.print();
  cout << "\nz = " ;  z.print();
}
```

3Constructeurs



x=0/1

y=4/1

z=22/7

# Constructeur de copie

```
#include <iostream>
using namespace std;
class Ratio
{ public: Ratio(int n=0, int d=1) : num(n), den(d) { }
          Ratio(const Ratio& r) : num(r.num), den(r.den) { }
          void print() { cout << num << '/' << den; }
  private:     int num, den; };
main()
{  Ratio x(100, 360);
    cout << "x = ";     x.print();
     Ratio y(x);
     cout << " y ="  ;  y.print();
     cout << "\n";
     system("pause");
}
```

**Constructeur de copie**

x=100/360

y=100/360

# Destructeur de classe

```
#include <iostream>
using namespace std;
class Ratio
{ public:     Ratio() { cout << " Objet est né. \n"; }
              ~Ratio()  { cout << " Objet meurt. \n"; }   // ~ =  tilde
  private:     int num, den;
};
int main()
{ { Ratio x;        // Début de portée pour x
    cout << " Maintenant, x est vivant \n";
  }       // Fin de portée de x

   cout << " Entre blocs.\n";
  { Ratio y;
    cout << " Maintenant, y est vivant.\n";
  }
```

**Destructeur**

Objet est né.

Objet meurt.

Objet est né.

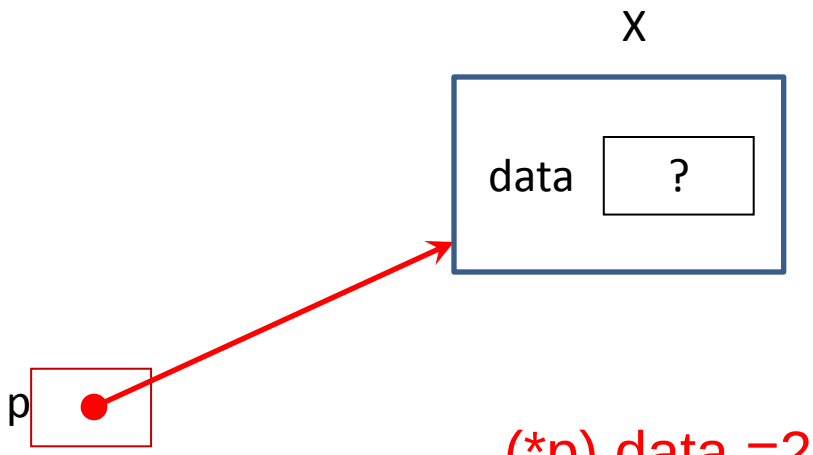
Objet meurt.

# Poineturs d'objets

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

class X
{ public:
    int data;
};
int main()
{ X* p = new X;
  (*p).data = 22;              // équivalent à: p->data = 22;
   cout << "(*p).data = " << (*p).data << " = " << p->data << endl;

 p->data = 44;
  cout << "  p->data = " << (*p).data << " = " << p->data << endl;
}
```




(*p).data =22=22

P->data =44=44

# Composition ou agrégartion

Utilisation d'une ou de plusieurs classes dans la définition d'une autre classe.

```
#include <iostream>
#include <cstring>
using namespace std;
class date {
      private : int day, month, year;
     public : date(int d=0, int m=0, int y=0): day(d), month(m),
year(y){}
              void getDate(){cin>>day>>month>>year;}
              void setDate(int d, int m, int y){day=d; month=m;
year=y;}
              void printDate(){cout <<day<<"  "<<month<<"  "<<year;}
      };
```

```
class Person
{public:
     Person (char* n="", int s=0, char* nat="algeria"): name(n),
sex(s), nation(nat){}
     void setDOB(int d, int m, int y) {dob.setDate(d, m, y);}
     void setDOD(int d, int m, int y) {dod.setDate(d, m, y);}
     void printname(){cout<<name;}
     void printnation(){cout<<nation;}
     void printDOB(){dob.printDate();}
     void printDOD(){dob.printDate();}
protected:
       string name, nation;
       date dob, dod;  // date of birth, date of death
       int sex;
};
```

```
main (){

    Person author ("T J", 1);
    author.setDOB(13, 4, 1743);
    author.setDOD(4, 7, 1826);
    cout<<"lauteur est :=";
    author.printname();
    cout<<endl;
    cout<<"Ne le : ";
    author.printDOB();
    cout<<endl;
    cout<<"mort le : ";
    author.printDOD();
    cout<<endl;     cout<<endl;     cout<<endl;

   system("pause");
}
```

# Héritage

L'héritage permet de donner à une classe **mère (de base)** toutes les caractéristiques d'une ou de plusieurs autres classes **filles (descendantes)**.

```
class Person
{public:
     Person (char* n="", int s=0, char* nat="algeria"): name(n), sex(s),
nation(nat){}
     void setDOB(int d, int m, int y) {dob.setDate(d, m, y);}
     void setDOD(int d, int m, int y) {dod.setDate(d, m, y);}
     void printname(){cout<<name;}
     void printnation(){cout<<nation;}
     void printDOB(){dob.printDate();}
     void printDOD(){dob.printDate();}
protected:
       string name, nation;
       date dob, dod;  // date of birth, date of death
       int sex;
};
```

```
class Student : public Person
{public: Student (char* n, int s=0, char* i=""): Person(n, s),
id(i), credits(0){}
void setDOINS(int d, int m, int y){doins.setDate(d, m, y);}
void printDOINS(){doins.printDate(); cout<<endl<<"name in
class Student="<<name;}
    private:
        string id;
        date doins;
        int credits;
        float moy;
};
```

```
main (){
Student x("D M", 1, "219360061");
 x.setDOB(13, 5, 1977);
 x.setDOINS(29, 9, 1995);
 x.printname();
 cout<<endl;
 cout<<"Ne le : ";
 x.printDOB();
 cout<<endl;
 cout<<"Inscrit le : ";
 x.printDOINS();
 cout<<endl;
 cout<<endl;
 cout<<endl;

 system("pause")      ;
 }
```

# Membres de données static

Appartiennent à la classe, et non pas aux objets de cette classe. Ils sont donc communs à tous ces objets.

```
#include <stdio.h>
class test
{ public: int n() ; } ;

int test::n(void)
{ static int compte=0 ; // Donnée de la classe test
  return compte++ ; }

int main(void)
{  test objet1, objet2 ;
   printf("%d ", objet1.n()) ;                // Affiche 0
   printf("%d\n", objet2.n()) ;  // Affiche 1
   return 0 ;
}
```

# TD N° 01 – Orienté Objet

## Exercice 01:

Réaliser une classe point pour manipuler un point tridimensionnel (x, y, z). La classe doit contenir un *constructeur par défaut* , un *constructeur de copie*, une fonction *negate()* pour rendre le point négatif, une fonction *norm()* pou caclculer la distance séparant le point de son origine (0, 0, 0), et une fonction *print()*.