# Programmation Orientée Objet

Introduction et généralités

**Pr Imane DAOUDI**

**2016-2017**

# Plan

1. Introduction à la programmation orientée objet
2. Généralités
   - Types de base, opérateurs et expressions
   - Les instructions de contrôles
   - Les entrées-sorties
3. Opérateurs sur tableaux, pointeurs et objets
4. Les fonctions
5. Programmation des classes et objets
6. Appels et surcharges
7. L’ héritage
8. Polymorphisme

# 1. Introduction à la programmation orientée objet

➢ Principe de Programmation OO.

Un programme informatique comporte toujours **des traitements** (des fonctions), mais aussi et surtout **des données**.

- ***la programmation structurée*** s'intéresse **aux traitements** puis **aux données**,

- ***la conception objet*** s'intéresse d'abord **aux données**, auxquelles elle s'associe ensuite **les traitements**.

**- Les données** sont ce qu'il y a de plus stable dans la vie d'un programme, il est donc normal d'architecturer le programme autour de ces données

d'où ⇒Le concept de l'orienté objet repose sur les notions :

**« d'objet » et « classe »**

# 2. Généralités

➢ Programme C++.

- Un programme **C++** est réparti dans un ou plusieurs fichiers.
- Chacun peut contenir des définitions/déclarations de fonctions, des définitions de types et des définitions de variables globales.
- Il existe une seule fonction main: c'est la fonction qui sera exécutée après la compilation.
- Le profil de main est : **int main()** ou **int main( int argc, char ** argv )** pour passer des arguments.

# 2. Généralités

## ➢ Programme C++.

- Le canevas minimal à utiliser pour réaliser un programme C++ se présente ainsi.

```
include <iostream>
Using namespace std;
Main()              // en-tête

{…                  // corps du programme
}
```

- Toute variable utilisée dans un programme doit avoir fait l'objet d'une déclaration en précisant le type et, éventuellement la valeur initiale

```
int i;                      //i est une variable de type int nommée x
float x=5.25:               // x est une variable de type float nommée x initialisée avec la
                            //valeur 5.25
Const int Nfois=5;          // NFOIS est une variable de type int dont la valeur, fixée à 5, ne
                            //peut plus être modifiée
```

# 2. Généralités

- Types de base, opérateurs et expressions

- **int** : entiers (au min 16 bits) (d'autres formats existent: **long int** (min 32 bits), **short int, unsigned int**) . . .
- **(float), double et long double** : nombres à virgule flottante (en général 15 chiffres pour double). Par ex. 23.3 ou 2.456e-12 ou 23.56e − 4
- **Char** : caractères ('a','b',. . . 'A',. . . ,':',. . . ).
- **Bool** : booléens ('true' ou 'false').
- (Et aussi: des types particuliers pour les tailles de tableaux, de chaines de caractères, d'objets etc. **size t, ::size type**)

# 2. Généralités

- Types de base, opérateurs et expressions
- Définition de variables

- Syntaxe : type v;

```
int p ;

double x ;
```

- Toute variable doit être définie **avant** d'être utilisée !
- Une définition peut apparaître **n'importe où** dans un programme.
- Une variable est définie **jusqu'à la fin de la première instruction composée** (marquée par }) qui contient sa définition.
- (Une variable définie en dehors de toute fonction est une **variable globale**).

# 2. Généralités

- Types de base, opérateurs et expressions
- Définition de variables

- Une variable peut être initialisée lors de sa déclaration, deux notations sont possibles :

- **int p=34 ;** **int p (34) ;**
- **double x=12.34 ;** **double x (12.34) ;**

- Une variable d'un type élémentaire qui n'est pas initialisée, n'a pas de valeur définie: elle peut contenir n'importe quoi.

# 2. Généralités

- Types de base, opérateurs et expressions
- Définition de variables: constantes symbolique

- **Syntaxe** : const type nom = val ;
- **Par exemple**: const int Taille = 100 ;
- Il ne sera pas possible de modifier Taille dans le reste du programme (erreur à la compilation). . .

# 2. Généralités

- Types de base, opérateurs et expressions
- Chaînes de caractères

• Il existe une classe **string**, ce n'est un pas un type élémentaire. Pour l'utiliser, il faut placer en tête du fichier :

```
# include <string>

    string t ;                  //définit t comme une variable...

    string s(25,'a') ;

    string mot = "bonjour" ;

    s.size() ;                  //représente la longueur de s

    s[i];                       //est le i-ème caractère de s ( i = 0,1,. . . s.size()-1)

     s+t;                       //est une nouvelle chaîne correspondant à la

                                //concaténation de s et t.
```

# 2. Généralités

- Types de base, opérateurs et expressions
- Tableaux

- Pour utiliser la classe **vector,** il faut placer en tête du fichier :

  **# include <vector>**

- Un tableau est typé:

  **vector<int> Tab(100,5)** ;

  **vector<int> Tab(50)** ;

  **vector<double> T** ;

- Structure générale: **vector< type > Nom(n,v)** ;
  - ***vector< type > Nom1 = Nom2 ; //*** les valeurs de Nom2 sont alors recopiées dans Nom1
  - **T.size()** correspond à la taille de T.

- NB: **size()** renvoie en fait un entier non signé, son type exact est **vector<type>::size type**

# 2. Généralités

- Types de base, opérateurs et expressions
- Tableaux

- **T[i] :** désigne le i-ème élément avec **i = 0, . . . T.size()-1**.
- **vector<vector<int> > T:**T définit un tableau à deux dimensions.
- Pour l'initialiser, on peut utiliser l'instruction suivante :
  - **vector<vector<int> > T2**
  - **T2(100,vector<int>(50,1))** ;
- On initialise chacune des 100 cases de T1 avec un tableau de taille 50 rempli de 1.

# 2. Généralités

- Types de base, opérateurs et expressions
- Les expressions: affectation

- En C/C++, l'affectation est une expression:
- Soient **v** une variable et **expr** une expression.
- **v = expr** affecte la valeur de **expr** à la variable v et retourne la valeur affectée à **v** comme résultat.
- Par exemple**, i = (j = 0)** affecte 0 à j puis à i et retourne 0 !!

# 2. Généralités

- Types de base, opérateurs et expressions
- Les expressions: opérateurs classiques

- Opérateurs arithmétiques:

  ***, +, -, /** (division entière et réelle), **%** (modulo)

- Opérateurs de comparaison

  **<** (inferieur), **<=** (inférieur ou égal), **==** (égal), **>** (supérieur),

  **>=** (supérieur ou égal) et **!=** (diffèrent)

- Opérateurs booléens

  **&&** représente l'opérateur "**ET**", **||** représente le "**OU**", et **!** représente le "**NON**".

- Par exemple, **((x<12) && ((y>0) || !(z>4)))**

# 2. Généralités

- Types de base, opérateurs et expressions
- Les expressions: Pré et Post incrément (Décrément)

- **++var** incrémente la variable var et retourne la nouvelle valeur.

  (++i équivaut à i=i+1)

- **var++** incrémente la variable var et retourne l'ancienne valeur.

  (i++ équivaut à (i=i+1)-1)

- L'expression **--var** décrémente la variable **var** et retourne la nouvelle valeur.
- L'expression **var--** décrémente la variable **var** et retourne l'ancienne valeur.

# 2. Généralités

- Types de base, opérateurs et expressions
- Evaluation des expressions booléennes

- Dans **e1 && e2**, la sous-expression **e2** n'est évaluée que si **e1** a été évaluée à 'true'.

```
if (i >=0 && T[i] > 20) blabla
```

- Dans **e1 || e2**, la sous-expression **e2** n'est évaluée que si **e1** a été évaluée à 'false'.

```
if (i<0 || T[i] > 20) blabla
```

# 2. Généralités

- Types de base, opérateurs et expressions
- Evaluation des expressions arithmétiques

- Si une (sous-)expression mélange plusieurs types, c'est le type le plus large qui est utilisé.

```
int i=3,j=2,m ;
double r=3.4 ;
m = (i/j)*r ;
```

- D'abord l'expression **(i/j)** est évaluée: **/** désigne ici la division entière, cela donne donc 1.
- Pour évaluer le produit **1*r**, il faut convertir **1** en double **(1.0)** et faire le produit sur les doubles, cela donne **3.4**
- Pour l'affectation, comme **m** est entier, **3.4** est converti en **int**. Finalement on a **m = 3**.

# 2. Généralités

- Types de base, opérateurs et expressions
- Evaluation des expressions arithmétiques

- Pour éviter les erreurs, il est possible de convertir explicitement des données d'un certain type en un autre.
- Par exemple:

```
int i=3,j=2,m ;
double r=3.4 ;
m = (double(i)/j)*r ;
Donne... 5 !
```

- L'évaluation d'une expression arithmétique ne se fait pas toujours de gauche à droite !
- **Ex: (i/j)*(r/3)**

# 2. Généralités

## ➢ Les instructions de contrôle

- Définition de variables, fonctions, types etc.

**expr** ;

**{ liste d'instructions }** : instruction composée.

**if (expr) instr**

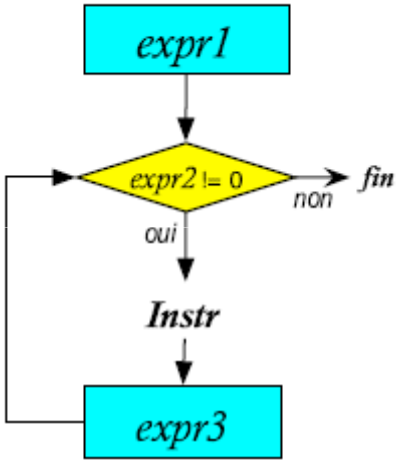
**if (expr) instr1 else instr2**

```
if (v == 3) i =i+4 ;

if ((v==3) && (i<5))

{ i=i+4 ;

v=v*2 ;}

else v=i ;

r = r*3 ;
```

# 2. Généralités

➢ Les instructions de contrôle : Boucle For

- for (expr1 ;expr2 ;expr3) instr.

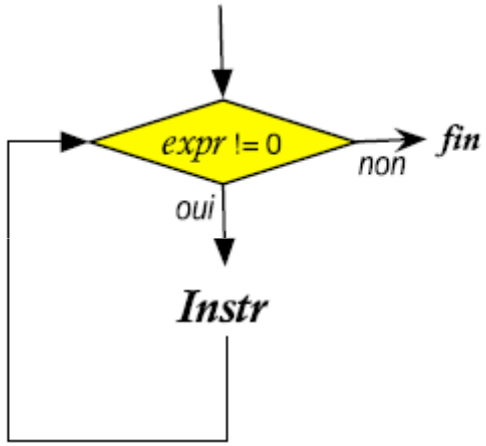



- Ex: **for(i=0 ;i<235 ;i=i+1) cout << T[i]** ;
- **for(i=0,j=1 ;i<235 ;i=i+1,j=j+3) cout << T[i][j]** ;

# 2. Généralités

➢ Les instructions de contrôle : Boucle While

- while (expr) instr.



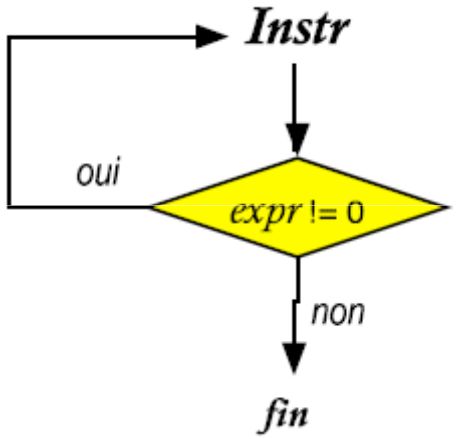

- Ex:

```
i=0 ;
While (i<235) {cout << T[i] ;i=i+1} ;
```

# 2. Généralités

➢ Les instructions de contrôle : Boucle do While

- do instr while (expr) ;




- Ex:

```
i=0 ;
Do {cout << T[i] ;i=i+1} ;
While (i<235)
```

# 2. Généralités

➢ Les instructions de contrôle : switch, break, continue et go to

- **Switch**(expr) {bloc d'instructions};
- **Break**: permet d'interrompre le déroulement de la boucle, en passant à l'instruction qui suit. En cas de boucle imbriquées, break fait sortir de la boucle la plus interne
- **Goto**: permet le branchement en un emplacement quelconque du programme

# 2. Généralités

## ➢ Les entrées –sorties: Afficher à l'écran

- Pour utiliser les entrées/sorties, il faut ajouter:

  **# include <iostream>**

- Syntaxe : **cout < < expr1 < < . . . < < exprn ;**
- Cette instruction affiche expr1 puis expr2. . .
- Afficher un saut de ligne se fait au moyen de **cout << endl**.

  **int i=45 ;**

  **cout << "la valeur de i est " << i << endl ;**

# 2. Généralités

## ➢ Les entrées –sorties: Afficher à l'écran

- Syntaxe : **cout < < expr1 < < . . . < < exprn ;**
- **cout (ou std::cout)** désigne le "**flot de sortie**" standard.
- **< <** est un opérateur binaire:
- le premier opérande est **cout** (de type "**flot de sortie**")
- le second opérande est l'expression à afficher
- le résultat est de type "**flot de sortie**"
- **< <** est associatif de gauche à droite
- **< <** est surchargé (ou sur-défini): on utilise le même opérateur pour afficher des caractères, des entiers, des réels ou des chaînes de caractères etc.

# 2. Généralités

## ➢ Lire au clavier

- Syntaxe : **cin > > var1 > > . . . > > varn** ;
- Cette instruction lit (au clavier) des valeurs et les affecte à var1 puis var2 . . .
- cin est le flot d'entrée standard, et **> >** est un opérateur similaire à **< <.**
- Les caractères tapés au clavier sont enregistrées dans un buffer dans lequel les cin viennent puiser des valeurs. Les espaces, les tabulations et les fins de lignes sont des séparateurs.

# 3. Opérateurs sur tableaux, pointeurs et Objets

- Pointeurs et tableaux

- Les deux opérations essentielles concernant les pointeurs sont:
  - L'accès à l'objet pointé noté *
  - L'évaluation de la valeur de l'adresse d'un objet **notée &** qui fournit un pointeur
- On considère qu'il y a pas d'objet pointé si le pointeur vaut 0.

- Exemple:

- Char c, *pc;
- Pc=&c; // fait pointer pc sur c
- *pc='a'; // le caractère pointé par pc reçoit la valeur 'a'

# 3. Opérateurs sur tableaux, pointeurs et Objets

- Pointeurs et tableaux

- Le seul opérateur disponible sur les tableaux est la sélection d'un élément, par indexation:

- Exemple:

- Int i;

- Char tab[100], *pc; // fait pointer pc sur c

- tab[i] // Désigne le i+1$^{ème}$ élément de tab

- En c++ pointeurs et tableaux sont étroitement liés : tab et équivalent à &tab[0]. Donc tab est de type pointeur de caractères, et par conséquent pc=tab est autorisé

# 3. Opérateurs sur tableaux, pointeurs et Objets

- Opérations arithmétiques sur les pointeurs

- pc+n; //pointe le n ième objet suivant celui sur lequel pointe pc
- pc-n; // pointe le n ième objet précédent celui sur lequel pointe pc
- Après l'expression pc=tab (ou pc=&tab[0])
  - *tab et *pc sont équivalents à tab[0]
  - tab+3 et pc+3 sont équivalents à &tab[3]
  - pc[3] et *(pc+3) sont équivalents à tab[3]
- Après l'expression pc=&tab[2] (ou pc=tab+2)
  - pc[-1] est équivalent à tab[1]

# 3. Opérateurs sur tableaux, pointeurs et Objets

- Opérations post et pré (incrémentation)

- pc++; //équivaut à pc=pc+1 fait pointé pc sur le caractère suivant
- *pc++='a'; // range 'a' dans le caractère pointé par pc, puis fait pointer pc sur le suivant
- C=*--pc; // fait pointer pc sur le caractère précédent, qui est rangé dans c