DAUPHINE UNIVERSITÉ PARIS | TUNIS

Le langage C++

M1 Actuariat

Fatma CHAKER
chakerfatma@yahoo.fr

Séance 6 : Structures, listes chaînées et classes

1

Plan

DAUPHINE UNIVERSITÉ PARIS | TUNIS

- Les structures
- Les listes chaînées
- Les Classes

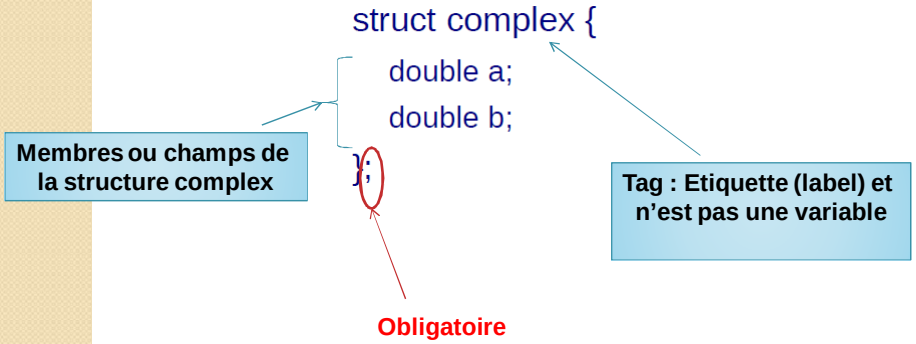
Les structures

- Une structure (**struct**) permet de regrouper des informations hétérogènes. Elle peut être composée de données de types différents (contrairement aux tableaux).
- **Déclaration d'une structure** :

```
struct structure_name
{
    type1 field1;
    type2 field2;
      ...
    typen fieldn;
};
```


Les structures : exemple

- Le Langage C ne possède pas un type pour les nombres complexes !
- **Rappel** : Un nombre complexe est z = a + ib



struct = nouveau type

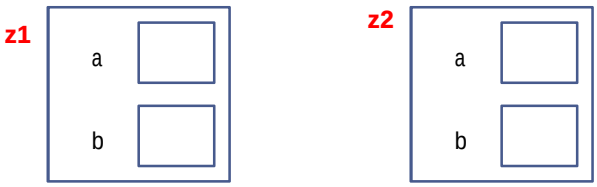
Idée : Lorsqu'on définit une structure on définit alors un nouveau type d'objet

➔ Déclarer une variable de type structure :

<struct-type> <identifier_list>;

- **Exemple**:

struct complex Z1, Z2; ***// ou encore complex Z1,Z2;***



Note: La déclaration **struct** ne permet pas d'allouer de la mémoire ou de créer des variables.

5

Déclaration des variables struct

struct complex z1,z2;

- Declare 2 variables z1 et z2 de type struct complex.

struct complex Z[25];

- Declare un tableau de 25 éléments de structure complexe.

struct complex *z;

- Declare un pointeur sur un objet de type struct complex

Accès aux membres d'une structure

Pour accéder à un membre de la structure on utilise l'opérateur (.) :

<struct-variable>.<member_name>;

```
struct complex z1;
z1.a = 7.2; // ou encore cin>>z1.a;
z1.b =5;

struct complex M[10];
M[2].a=10.3; // ➔ la valeur de a du 3ème complexe =10.3
M[2].b=20.2; // ➔ la valeur de b du 3ème complexe =20.2

struct complex *p;

(*p).a = 10.1; // ➔ la valeur de a du complexe pointé par p
 p->b = 20.3; // ➔est équivalent à (*p).b = 20.3;
```


Afficher les membres d'une structure

Pour afficher le contenu d'une variable **struct**, il faut afficher chaque champs (membre) séparemment en utilisant l'opérateur (**.**).

Faux :

```
cout << z1; // ne marche pas!
```

Correct:

```
cout << z1.a << endl;
cout << z1.b << endl;
```

Les structures : Opérateur =

L'opérateur = est définit automatiquement pour les structures

```
struct Complex z1,z2;
z1.a = 7.2;
z1.b = 5.3;
z2 = z1; //This is OK, will copy value-by-value
```

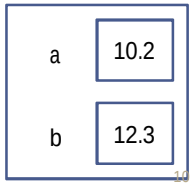

Les structures : Initialisation

On ne peut pas initialiser les membres d'une structure lors de sa déclaration parce que la mémoire n'est pas encore allouée

```
struct complex        // Initialisation
  {                    //  non autoriése
    double a=10.2;
    double b=12.3
  };
```

Les membres d'une structure members sont initialisés au moment de la création d'une variable type structure en utilisant une liste ordonnée d'initialisation :

```
struct complex z1 {10.2,12.3};
```



Fonctions et Structures

- On peut passer les membres d'une structure à une fonction.

```
somme_complexe(z1.a,z1.b);
```

- On peut passer une **structure** entière à une fonction.

```
afficher_complexe(z1);
```

- On peut utiliser un passage par **valeur** ou par **adresse**.

Fonctions et Structures

Passage par valeur

```
void affiche_complexe(struct complex);  // proptotype

int main()
{
....
struct complex z1;
affiche_complexe(z1); // appel
…..
}
void affiche_complexe(struct complex C1) //entête
{
cout<<"Z="<<C1.a<<"+i "<<C1.b<<endl;
}
```

Fonctions et Structures

Passage par adresse

```
void affiche_complexe(struct complex*); // proptotype
int main()
{
....
struct complex z1;
affiche_complexe(z1); // appel
…..
}
void affiche_complexe(struct complex *C1)
{
cout<<“Z=“<<C1->a<<“+i “<<C1->b<<endl;
// c1->a équivalent à (*C1).a
}
```


Retourner une structure

- Une fonction peut retourner une **struct**

```
struct complex getComplex();  // prototype
s1 = getComplex();      // appel
```

- La fonction **doit définir** une variable locale de type structure.
  - Pour une utilisation interne
  - Pour être utilisée avec le **return**

Retourner une structure

```
struct complex getComplex()
 { struct complex s1;//variable locale
   cin >> s1.a;
   cin >> s1.b;
return s1;
 }
```


Les membres d'une structure

✓ Les membres d'une structure peuvent être hétérogènes et de différents types (char, int, double, string,…).

✓ Un membre peut être un tableaux (de taille prédéfinie)
→ L'opérateur = permet de copier les éléments du tableau (valeur par valeur)

✓ Un membre peut être une structure.

Variables du même type peuvent être déclarées en les séparant par virgule.

```
struct PersonInfo
       { string name,
                address,
                city;
       };
struct Student
       { int      studentID;
         PersonInfo pData;
         short    year;
         char     nameS[20];
         double   gpa;
       };
```

✋ Doit être définie auparavant !!!

```
Student s5;

S5.studentID=20;
 s5.pData.name =“Karim";
getline(cin,S5.pData.address);
 s5.pData.city = “Tunis";
strcpy(S5.NameS=“MedBen Foulen”);
……
```

Les membres d'une structure

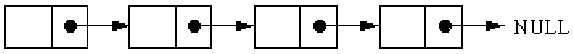
```
struct myStruct {
    int a;
    struct myStruct s;
} obj1;
```

```
struct myStruct {
    int a;
    struct myStruct *s;
} obj1;
```

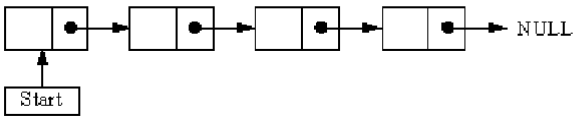
Erreur de compilation !!
myStruct non encore définie !!!

Les listes chaînées : Rappel

Une **liste chaînée** (*linked list*) désigne une **structure de données** représentant une collection ordonnée et de taille arbitraire d'éléments (nœuds) de même type.
L'accès aux éléments d'une liste se fait de manière séquentielle : chaque élément permet l'accès au suivant. Le dernier élément pointe vers NULL.



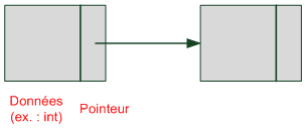
Par convention, l'accès à une liste simplement chaînée est simplement un pointeur vers le premier élément de cette liste.



Structures de données : Liste

✓On considère la structure suivante :

```
struct nœud
{ int sommet;
  struct nœud *suivant;
};
```



✓ Chaque objet contient un nombre et un pointeur vers l'objet suivant

✓ On peut construire une liste d'entiers

✓ Sa taille est illimitée

Création d'une liste simplement chaînée

```
noeud *start = new noeud; // Sauvegarde du premier élément start de la
// chaîne, ceci permet de retourner au début de la liste
noeud *current;
current = start;
int n = 0;
while (n <= 20)
  {
    if (n == 20)
      current->suiv = NULL;
    else
      current->suiv = new noeud;
    current->sommet = n + 1;
    current = current->suiv;
    n++;
  }
```

Exercice :

Ecrire une fonction « Afficher_liste » qui reçoit en argument un pointeur sur le 1er élément de la liste chaînée créée précédemment et qui permet d'afficher tous les nombres stockés dans cette liste.

Prototype : void Afficher_liste(struct noeud *);

Solution:

```
void Affiche_liste(struct noeud *head)
 {  while(head->suiv != NULL) // ou while(head)
    {
      cout << head->sommet <<endl;
      head = head->suiv;
    }
}
//Programme principal
int main()
{
…….
Affiche_liste(start);
….
}
```

POO : Terminologies

Classe: Extension des structures (***struct***) *avec différents niveaux* de visibilité (*protected, private et public)*

- Permet de regrouper les variables (données membre) et les fonctions qui opèrent sur ces variables (fonctions membre)
- Décrit les propriétés que toutes les instances de la classe auront

Objet : Instance d'une **classe**, de la même manière qu'une variable est une instance d'une **struct**

Attributs: Données membre d'une classe

Méthodes : Fonctions membre d'une classe.

Déclaration d'une classe

```
class className
   {
      declaration;
      declaration;
   };
```

Exemple :

```
class Complex {
   double a;
   double b;
};
class Complex z1,z2;
```

Qu'elle est la différence ?

✓ En C++, lors de la définition d'un objet on peut ne pas préciser le mot clé **class**.

```
Complex z1,z2;   // correct
```

NB : Ceci est aussi vrai pour les structures en C++ !! Mais sera considéré comme une erreur en C.

✓ Restrictions d'accès aux données: La classe protège l'accès à ses données

```
class Complex{
        double a;
        double b;
};
Complex z1;
Z1.a=5.2; // Erreur de compilation !! Accès à une
                // donnée privée
```

25

Qu'elle est la différence ? : Public et Private

✓ Utilisés pour contrôler l'accès aux membres d'une classe (variables ou méthodes). Chaque membre peut être déclaré :

public: les membres publics peuvent être utilisés et modifiés par tous

ou

private: les membres privés ne peuvent être utilisés (modifiés pour les données, appelés pour les méthodes) que par les méthodes de la classe elle-même

✓ Par défaut, tous les membres sont privés

✓ On peut spécifier la partie publique en utilisant le label public.

✓ Les variables sont le plus souvent **private , tandis que les méthodes sont** usuellement **public.**

Exemple :

```
class Complex{
        public :
                double a;
                double b;
        };
        Complex z1;
        Z1.a=5.2; // Correct Pas d'erreur de compilation
                  // a est une variable publique.
```


Qu'elle est la différence ? : Fonctions membres

✓ D'une façon très simple et en faisant le parallèle avec le C, on peut dire qu'une classe est une structure à laquelle on ajoute des fonctions permettant de gérer cette structure.

✓ Par défaut, les méthodes d'un objet peuvent accéder à tous les champs d'une classe, même ceux qui sont déclarés en Private.

Exemple :

```
class Complex{
        public :
                double a;
                double b;
        public :
                double abs();
        };
        Complex z1;
        cout<<Z1.abs(); // OK
```


Accès aux fonctions membres

Soit *z1* une instance de la classe **Complex**

```
Complex z1;
```

✓ L'opérateur « **.** » permet un **accès direct** aux méthodes de l'instance.

```
z1.abs();   //
```

✓ L'opérateur « **->** » permet l'accès aux méthodes de l'instance **via un pointeur**.

Ex: Si **C** est un pointeur sur Complex.

```
Complexe *C=&z1; // ou encore Complex *C;
                 // C=&z1;
```

Alors on accède aux membres de *z1* par C->abs() qui est équivalent à z1.abs() ou encore à (*C).abs();

Définition des fonctions membres

- ✓ Les fonctions membres font partie de la déclaration d'une classe.
- ✓ La définition de la fonction peut être placée :
  - à l'intérieur de la déclaration de la classe (inline)

Ou

  - Après la déclaration de la classe. Dans ce cas le prototype de la fonction doit être déclaré à l'intérieur de la déclaration de la classe.

Fonctions membres inline

```
class Complex {
   double a,b;
   public:
   double abs() { return a*a+b*b; };
};
```

Utilisée uniquement avec des fonctions très simples et courtes.

Définition de la fonction après la déclaration de la classe

- Pour **définir une fonction membre** après la déclaration de la classe
  - Déclarer un **prototype de la fonction** dans la partie déclaration de la classe
  - Lors de la définition de la fonction, précéder le nom de la fonction par le nom de la classe et par l'opérateur de résolution de portée (**::**)

Définition de la fonction après la déclaration de la classe

```
class Complex {
   double a,b;
   public:
   double abs(); //only declaration
};
…
double Complex::abs( )
{ return a*a + b*b; }
```

Prototype de la fonction

Fin de la déclaration de la classe Complex

Programmation d'une classe : Stratégie DDU

En C++, la programmation d'une classe se fait en trois phases : ***déclaration*****,** ***définition et utilisation*** *(en abrégé : D.D.U).*

❑ **Déclaration : c'est la partie interface de la classe. Elle se fait dans un fichier dont le nom se termine** par .h Ce fichier se présente de la façon suivante :

```
class Maclasse
{
public:
      déclarations des données et fonctions-membres publiques

private:
      déclarations des données et fonctions-membres privées
};
```


Programmation d'une classe : Stratégie DDU

❑ **Définition : c'est la partie implémentation de la classe. Elle se fait dans un fichier dont le nom se** termine par **.cpp** Ce fichier contient les définitions des fonctions-membres de la classe, c'est-`a-dire le code complet de chaque fonction.

❑ **Utilisation : elle se fait dans un fichier dont le nom se termine par .cpp**

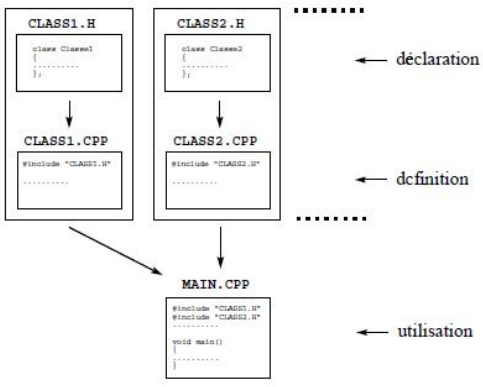
Structure d'un programme C++

Nos programmes seront généralement composés d'un nombre impair de fichiers :

– **Pour chaque classe :**
un **fichier .h** contenant sa déclaration,
un **fichier .cpp** contenant sa définition,

– **Un fichier .cpp** contenant le traitement principal. Ce dernier fichier contient la fonction **main**, et c'est par cette fonction que commence l'exécution du programme.

Structure d'un programme C++



Exemple de la classe Complex

Header file : complex.h

```
//ce fichier défini la classe Complex
#ifndef COMPLEX_H
#define COMPLEX_H
Class Complex {
        double _a;
        double _b;
public:
        void set(double,double);
        double abs();
};
#endif
```

Pour gérer les inclusions multiple du fichier header

Par convention, les noms de classe commencent par une majuscule, les données membres par _ ou m_, les fonctions membres par une minuscule.

Exemple de la classe Complex

complex.cpp

```
//ce fichier contient la définition de chaque méthode de la classe
// Complex
#include "complex.h"
void Complex::set(double newa,double newb)
{
        _a=newa;
        _b=newb;
}
double Complex::abs()
{
        return _a*_a + _b*_b

}
```

Ne pas oublier d'inclure le header complex.h

NB: On fait précéder chaque méthode de Complex::

Exemple de la classe Complex

main.cpp

```
#include <iostream>
#include "complex.h"
using namespace std;
int main()
{
      Complex z1;
      double a,b;
      cout << "Enter a,b" << endl;
      cin >> a >> b;
      z1.set(a,b);
      cout << z1.abs() << endl;
}
```