UMONS - Faculté Polytechnique
Service d'Informatique, Logiciel
et Intelligence Artificielle (ILIA)
www.ig.Umons.ac.be

POLYTECH MONS
Rue de Houdain, 9
B- 7000 Mons
www.umons.ac.be

et Programmation Objet

Travaux Pratiques de Méthodologie Orientée Objet
BAB2

par
Mohammed BENJELLOUN

Avec la participation de :
A. Cools
M. Hani
M. Delehouzée
N. Rabie
E. Malengreau
A. Guttadauria

Année académique 2023-2024

POLY TECH. MONS

ILIA UMONS

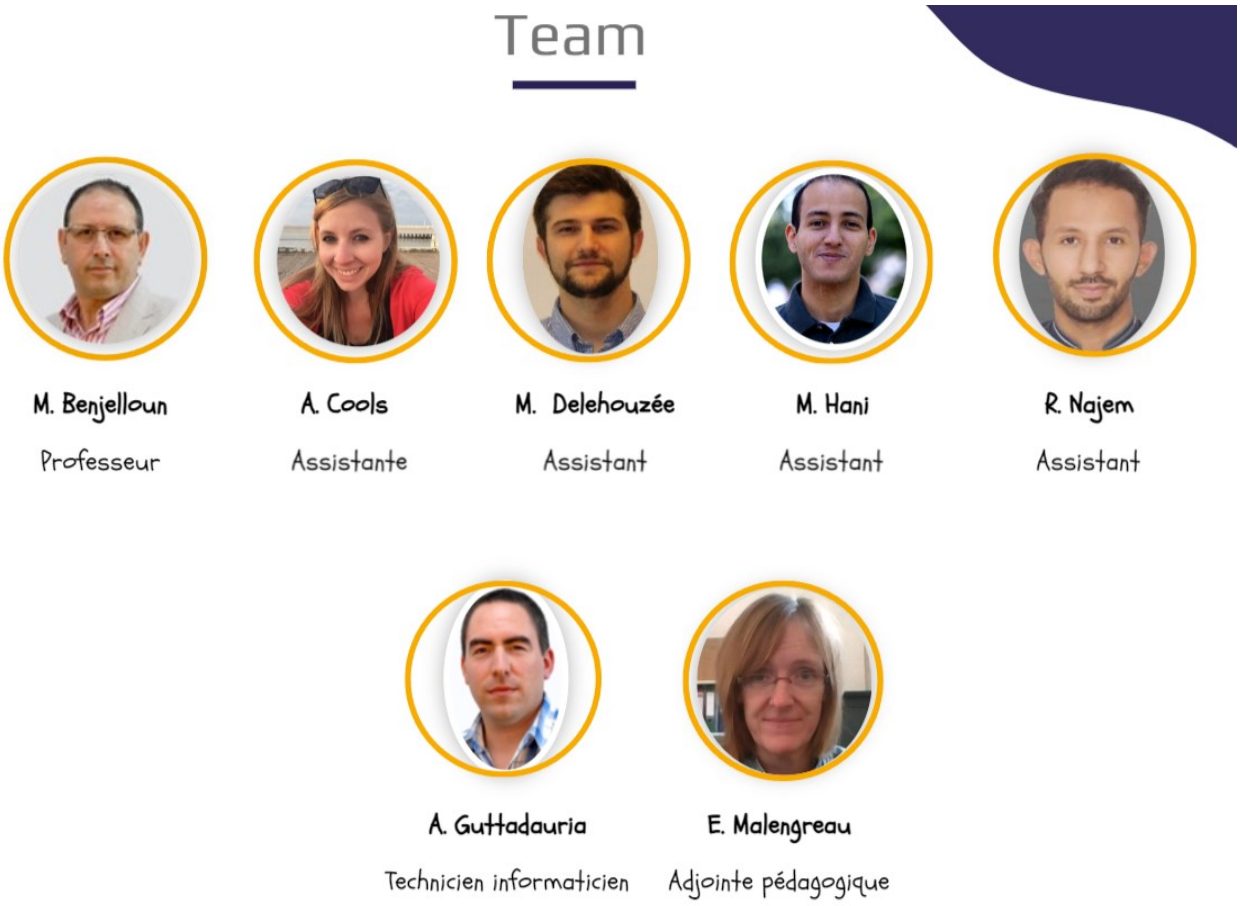
UMONS Université de Mons

Table des matières

Consignes

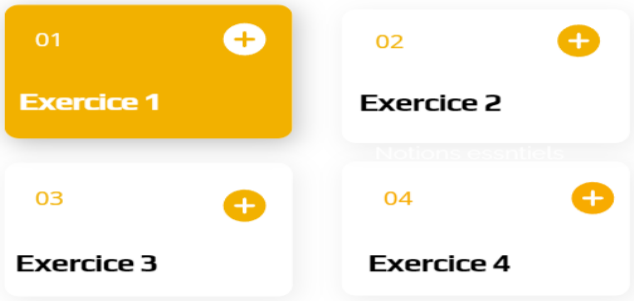
Si malgré la préparation de chaque TP, vous n'avez pas terminé à la fin de la séance, vous devrez le continuer seul et poser vos éventuelles questions avant le TP suivant. Vous pouvez venir nous les poser directement ou les poser via le forum. Cette remarque est valable pour tous les travaux pratiques de ce cours.

Pour rappel, les travaux pratiques sont obligatoires.



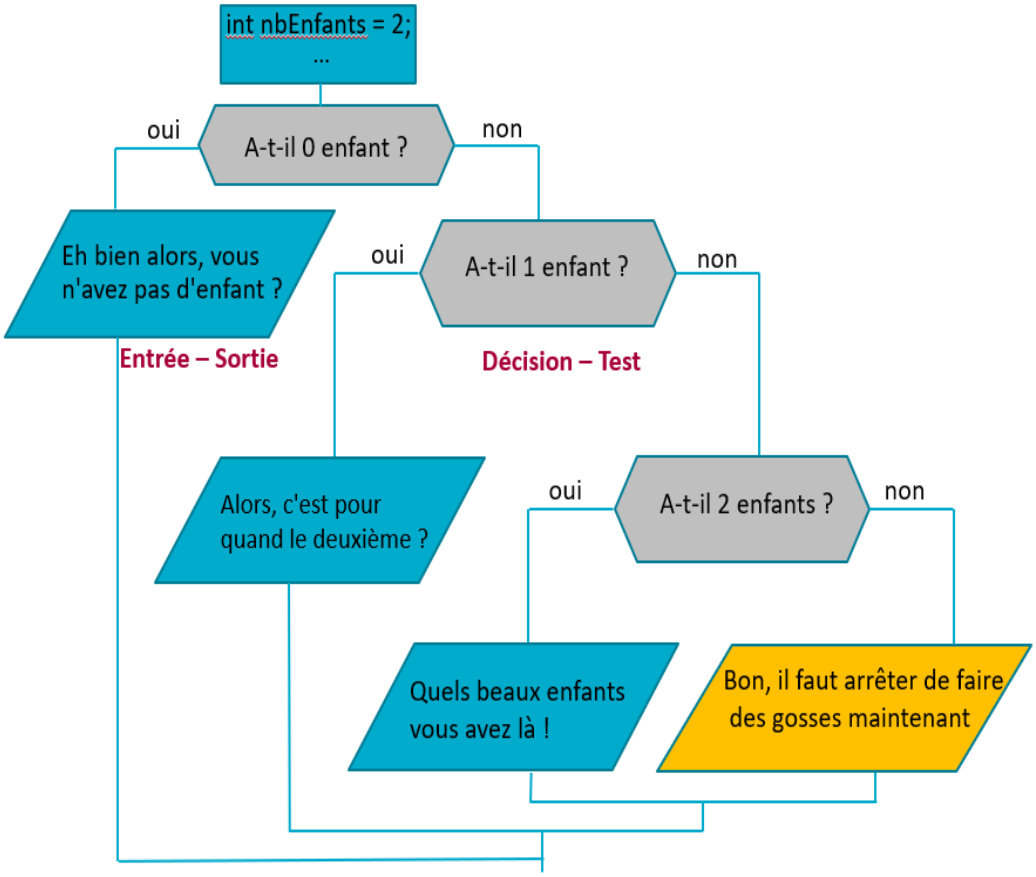
TP1

But : 4 exercices vous sont proposés. Ces exercices portent sur les tests et les structures de contrôle (boucles) ainsi que sur la manipulation des tableaux.



Exercice 1.1 :

On vous demande de traduire l'organigramme ci-dessous, en C ++ :



Exercice 1.2 :

Nous considérons un tableau qui contient les entiers 5, 3, 11, 9, 2. Ecrire un programme qui permet de faire un ***choix*** entre les 3 options suivantes :

a) **Ajouter** au **début** du tableau une valeur introduite par l'utilisateur
b) **Ajouter** à la **fin** du tableau une valeur introduite par l'utilisateur
c) **Supprimer** la valeur de l'élément à **la deuxième** position du tableau

Pour améliorer votre programme, nous vous demandons de gérer ce ***choix*** à l'intérieur d'une boucle.

- Cela revient à ajouter un choix **(d) : Quitter. Tant que** l'utilisateur n'a pas choisi l'option (d), vous devez exécuter l'un des trois autres choix.
- On peut aussi tenir compte du fait que si le nombre d'éléments est inférieur à 2, il ne faut pas lancer l'action (c).

Exercice 1.3 :

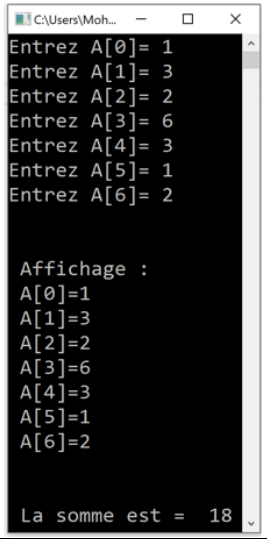
Demander à l'utilisateur d'introduire le matricule de l'étudiant E ainsi que deux notes, supposées entières, Ea et Eb.

1) Afficher les détails de l'étudiant, qui comprennent son matricule et ses notes Ea et Eb, puis calculer et afficher la somme SE = Ea + Eb et la moyenne.
2) Adapter le programme pour qu'il fonctionne sur un ensemble de N d'étudiants. La valeur de N doit être introduite par l'utilisateur.
3) Intégrer la contrainte que l'entier N $\in$ [1, Nmax]. Nmax est une constante = 10. Et tant que N $\notin$ [1, Nmax], redemander une nouvelle valeur de N.
4) Adapter le programme pour stocker les informations dans des tableaux : TabMat[Nmax] pour les matricules ; TabEa[Nmax] pour les premières notes ; TabEb[Nmax] pour les secondes notes ; TabSom[Nmax] pour les sommes des notes et TabMoy[Nmax] pour les moyennes des notes.

Exercice 1.4 :

Compléter le code suivant pour que l'exécution de ce code génère ce résultat :

```
#include …


void  main()
{
 const int dim = 7;
 int i, A[dim], S=0;
 Saisie (??? )
 Affichage (???)

}
```



fonction Saisie(??)

fonction Affichage (????)

Le code doit contenir une fonction Saisie(??) pour l'acquisition des données et une fonction Affichage (????) pour afficher à la fois les données et la somme des données.

TP2

But : 4 exercices vous sont proposés. Ces exercices portent essentiellement sur l'utilisation et la manipulation des fonctions.

Exercice 2.1 :

Compléter le programme suivant pour que les fonctions suivantes respectent ces contraintes:

Fct1(? ??) : affiche ce texte « Bonjour »
Fct2(? ? ?) : demande la valeur d'un entier A
Fct3(? ? ?) : affiche le double de la valeur de l'entier A demandé dans la fonction Fct2(? ? ?)
Fct4(? ? ?) : demande la valeur d'un float B, d'un char C et d'un string D
Fct5(? ? ?) : affiche les valeurs de B, C et D demandées dans la fonction Fct4(? ? ?)
Fct6(? ? ?) : quand on appel cette fonction avec les paramètres B et C dans le **cout**, affiche les valeurs de B, C et D demandées dans la fonction Fct4(? ? ?)

« Adapter si nécessaire la fonction Fct5 en une fonction Fct6 pour que, à l'appel de cette fonction Fct6 dans un cout, on puisse afficher à la fois B, C et D. »

```
void main()
{
        .. .. ;
        Fct1( ? ?? ) ;
        Fct2( ? ?? ) ;
        Fct3( ? ?? ) ;
        Fct4( ? ?? ) ;
        Fct5( ? ?? ) ;
          cout << "Voici la solution de cette fonction"<< Fct6( ??? ) ;
}
```

Exercice 2.2 :

Calculer la puissance nième d'un nombre entier X (X^n n entier >=0)

Transformer le programme suivant en utilisant une fonction permettant de calculer et de retourner (sortir de la fonction) la puissance nième d'un nombre entier.

- (X^n n entier >= 0) : $X^3 = X * X * X$; 5^3 ; 7^9 ; 3^{21} ???

```
void main ( )
{
      int i, P1=1;
      for (i=0; i< 3 ; i++) {
         P1= P1 * 5;
       }

      int  P2=1;
      for (i=0; i< 9 ; i++) {
         P2= P2 * 7;
       }

      int  P3=1;
      for (i=0; i< 21 ; i++) {
         P3= P3 * 3;
       }


       cout << "\n La puissance P1 est = "<< P1 ;
       cout << "\n La puissance P2 est = "<< P2 ;
       cout << "\n La puissance P3 est = "<< P3 ;
}
```

Exercice 2.3 :

Compléter les fonctions Saisie(?) et Affichage (?) pour qu'un programme puisse saisir les données d'un tableau dont la dimension est demandée dans la fonction Saisie. Cette fonction calcule la somme et la moyenne des éléments et doit les transmettre à la fonction Affichage.

```
void Saisie ( ??? ) {
   cout<<"Entrer la valeur de N <= Nmax "<<endl;
   cin>> N ;

     for(int i=0;i<N; i++) {
        // remplissage d'un tableau de N éléments
     }
     // Calcul de la somme des éléments et de la moyenne
     // retour (sortie) de la somme et de la moyenne

}
void Affichage ( ? ? ? )
{
     for(int i=0; i<N; i++)  {
         // Affichage des éléments du tableau
     }
   // Affichage de la somme et la moyenne calculées dans la fonction Saisie.
}
void main ()
{
 .. ..
 int Tab[Nmax]
 Saisie ( ??? ) ;
 Affichage( ? ? ) ;
}
```

Exercice 2.4 :

Le même programme que Exercice 2.3 mais appliqué à une matrice.

```
…
const int Nmax=10;

void main()  {
        int N1,N2;   . . .
        int C[Nmax][Nmax];
        Saisie ( ? ? ?);
        Affichage ( ? ? ? );
}
```

TP3

But : Manipulation d'un tableau de structures contenant des fonctions membres et utilisation d'une variable « static », constructeur et opérateur.

3.1) Écrivez un programme qui gère un ensemble de PROFs (Nmax=10). Chaque PROF a une structure de type **struct PROF** :

```
struct PROF {
        int Id;
        string nom;
        string service;
};
```

Votre programme doit gérer les PROFs en utilisant en boucle la fonction menu suivante :

```
int menu() {
  int choix;
  cout << "Entrez votre choix : \n"
       << "   1.  Saisir + Afficher\n"
       << "   2.  Ajouter un PROF au début + Afficher\n"
       << "   3.  Chercher si PROF1 existe ou pas dans le tableau\n"
       << "   4.  Quitter " << endl;
  cin >> choix;
  return choix;
}
```

Remarque : pour comparer si PROF1 existe dans le tableau, la comparaison se fait uniquement sur le nom et le service.

3.2) Reprenez le même énoncé (3.1), mais en définissant les fonctions membres **saisie()** et **afficher()** pour gérer l'ensemble des informations relatives à un PROF. La structure est modifiée comme suit :

```
struct PROF {
        int Id;
        string nom;
        string service;
        void saisie();
        void afficher() ;
};
```

3.3) Améliorez le programme de l'énoncé (3.2) de manière à ce que :

a) l'**Id** ne soit plus saisi manuellement, mais qu'il s'incrémente automatiquement à l'aide d'une **variable statique** dans une fonction membre : le **constructeur.**
b) pour chercher si un PROF existe, utilisez un **operateur d'égalité** (==).

```
struct PROF    {
        int Id;
        string nom;
        string service;

        PROF();
        void saisie();
        void afficher();
        bool operator == (PROF P)
        {
              ???
         }
};
```

TP4

But : **Lire des données dans un fichier. Manipulation d'un tableau de classes contenant des fonctions membres. Se familiariser avec l'utilisation de constructeurs, destructeurs. Optimisation et création de code générique.**

Déclarez une classe **ETUD** et une seconde classe **PROF** dont les objets sont respectivement des étudiants et des enseignants. La déclaration de ces classes est :

```
class ETUD {
    int Id;
    string nom;
    public :  void saisie();
              void affiche();
 };
```

```
class PROF {
    int Id;
    string nom, service;
    public :   void saisie();
               void affiche();
 };
```

Dans les deux classes, en plus des deux méthodes Saisie() et Affichage(), on déclarera au moins :

- un constructeur permettant l'incrémentation automatique de l'Id (Prof1 : Id=1, Prof2 : Id=2, …) et (Etud1 : Id=1, Etud2 : Id=2, …).
- un destructeur montrant la disparition d'instances d'objets.

Ce programme doit d'abord gérer les Etudiants en utilisant en boucle le menu suivant :

```
int menu() {
  int choix;
  cout << "Entrez votre choix : \n"
       << "  1. Lire fichier PROF + Lire fichier ETUD +Afficher les deux tableaux\n"
       << "  2. Ajouter un PROF au début du tableau + Afficher\n"
       << "  3. Ajouter un ETUD au début du tableau + Afficher\n"
       << "  4. Chercher PROF1 dans le tableau des PROF et le supprimer s'il existe\n"
       << "  5. Quitter " << endl;
  cin >> choix;
  return choix;
}
```

Le case 1 consiste à stocker les données du fichier PROF.txt dans un tableau. Puis lire/saisir les données du fichier ETUD.txt dans un autre tableau.

Pour vous aider, le fichier **TP4_Lec_Fichiers.cpp** sur Moodle contient un code de démarrage de ce TP : définition d'une partie des deux classes ; code pour lire les données à partir des fichiers **ETUD.txt** (Nom des étudiants) et **PROF.txt** (Nom et Service des Profs). Les informations contenues dans ces fichiers sont séparées par un délimiteur **';'** .

Nous vous conseillons de déposer les fichiers .txt dans le même dossier que le code source de votre programme.

Vous devez absolument optimiser votre code (template de fonctions) afin de permettre la réutilisation des morceaux de code. Vous pouvez également utiliser switch-case, les opérateurs, ...

S'il vous reste du temps:

- Que deviendra le programme si les tableaux sont remplacés respectivement par les conteneurs ***Vector*** et ***List***.

Fichiers pour ce TP :

ETUD.txt

```
7
Etud1;
Etud2;
Etud3;
Etud4;
Etud5;
Etud6;
Etud7;
```

PROF.txt

```
5
Prof1;Informatique;
Prof2;Informatique;
Prof3;Mathematique;
Prof4;Physique;
Prof5;Chimie;
```

Exemple d'exécution :

Nmax = 10 et Tab[Nmax] ==>
ETUD = : 1 ; 2 ; 3 ; 4 ; 5 ; 6 ; 7 ; 8 ; 9 ; 10 ;
PROF = : 1 ; 2 ; 3 ; 4 ; 5 ; 6 ; 7 ; 8 ; 9 ; 10 ;
Entrez votre choix :
1. Lire fichier PROF + Lire fichier ETUD +Afficher les deux tableaux
2. Ajouter un PROF au début du tableau + Afficher
3. Ajouter un ETUD au début du tableau + Afficher
4. Chercher PROF1 dans le tableau des PROF et le supprimer s'il existe
5. Quitter
1

Id	Nom
1	Etud1
2	Etud2
…	
7	Etud7

Id	Nom	Service
1	Prof1	Informatique
2	Prof2	Informatique
3	Prof3	Mathematique
4	Prof4	Physique
5	Prof5	Chimie

Entrez votre choix :
1. Lire fichier PROF + Lire fichier ETUD +Afficher les deux tableaux
2. Ajouter un PROF au début du tableau + Afficher
3. Ajouter un ETUD au début du tableau + Afficher
4. Chercher PROF1 dans le tableau des PROF et le supprimer s'il existe
5. Quitter
2

11 ; Quel est le nom ? ProfPlus
Quel est le service ? Meca

Id	Nom	Service
11	ProfPlus	Meca
1	Prof1	Informatique
…		
4	Prof4	Physique
5	Prof5	Chimie

Entrez votre choix :
1. Lire fichier PROF + Lire fichier ETUD +Afficher les deux tableaux
2. Ajouter un PROF au début du tableau + Afficher
3. Ajouter un ETUD au début du tableau + Afficher
4. Chercher PROF1 dans le tableau des PROF et le supprimer s'il existe
5. Quitter
3

11 ; Quel est le nom ?
EtudPlus

Id	Nom
11	EtudPlus
1	Etud1
2	Etud2
…	
7	Etud7

Entrez votre choix :
1. Lire fichier PROF + Lire fichier ETUD +Afficher les deux tableaux
2. Ajouter un PROF au début du tableau + Afficher
3. Ajouter un ETUD au début du tableau + Afficher
4. Chercher PROF1 dans le tableau des PROF et le supprimer s'il existe
5. Quitter
4

Quel est le nom du prof cherché ?
SuperProf
Quel est le service du prof cherché ?
Info
Le prof cherché n'existe pas !

Entrez votre choix :
1. Lire fichier PROF + Lire fichier ETUD +Afficher les deux tableaux
2. Ajouter un PROF au début du tableau + Afficher
3. Ajouter un ETUD au début du tableau + Afficher
4. Chercher PROF1 dans le tableau des PROF et le supprimer s'il existe
5. Quitter
4

Quel est le nom du prof cherché ?
Prof4
Quel est le service du prof cherché ?
Physique

Le prof cherché existe en position 5 et sera supprimé !

Id	Nom	Service
11	ProfPlus	Meca
1	Prof1	Informatique
2	Prof2	Informatique
3	Prof3	Mathematique
5	Prof5	Chimie

TP5

But : **Lecture des données dans un fichier. Manipulation des conteneurs Vector, List. Optimisation et création de code générique.**

Pour faire ce TP, vous pouvez récupérer une partie du code du TP précédent.

Déclarez une classe **ETUD** et une seconde classe **PROF** dont les objets sont respectivement des étudiants et des enseignants. La déclaration de ces classes est la suivante :

```
class ETUD {
      int Id;
      string nom;
      public :  void saisie();
                void affiche();
 };
```

```
class PROF {
      int Id;
      string nom, service;
      public :    void saisie();
                  void affiche();
 };
```

Dans les deux classes (ETUD, PROF), on déclarera au moins deux méthodes, l'une pour saisir et l'autre pour afficher chaque objet de la classe (saisie() et affichage()).

Si nécessaire, des fonctions membres peuvent être ajoutées aux classes. On prévoira au moins :

- un **constructeur** par classe permettant l'incrémentation automatique de l'Id (Prof1 : Id=1, Prof2 : Id=2, … et Etud1 : Id1=1, Etud2 : Id2=2, ….).
- un **destructeur** montrant la disparition d'instances d'objets.

La manipulation des étudiants se fera en utilisant un conteneur ***List*** alors que les PROFs seront rangés dans un conteneur ***Vector***.
Ce programme doit d'abord gérer les Etudiants et les Profs en utilisant en boucle le menu suivant :

```
int menu() {
  int choix;
  cout << "Entrez votre choix : \n"
       << "  1. Lire fichier PROF et stocker les données dans un Vector + Affichage\n"
       << "  2. Lire fichier ETUDB et stocker les données dans une List   + Affichage\n"
       << "  3. Ajouter un PROF au début du Vector + Afficher\n"
       << "  4. Ajouter un ETUD au début de List    + Afficher\n"
       << "  5. Supprimer le PROF et ETUD au début de Vector et List + Affichage\n"
       << "  6. Créer une List2 contenant uniquement les noms des ETUD et PROF \n"
       << "  7. Quitter " << endl;
  cin >> choix;
  return choix;
}
```

Les choix seront traités via l'instruction case. Votre programme utilisera une série de fonctions permettant de séparer les tâches.

Vous devez absolument optimiser votre code (Templates de fonctions) afin de permettre la réutilisation des morceaux de code.

Le Template de la fonction Saisie (lire fichier et stocker) doit insérer les éléments **au début**. Par exemple, si on saisit Etud1, Etud2, Etud3, la List contiendra Etud3 en position 0 et Etud1 en position 2. Le même raisonnement sera appliqué pour PROF.

S'il vous reste du temps :

Que deviendra le programme si les classes ETUD (+ int Note[3]) et PROF héritent d'une classe Personne caractérisée par : int Id string nom. ?

Fichiers pour ce TP :

ETUDB.txt

```
7
Etud 1;
Etud 2;
Etud 3;
Etud 4;
Etud 5;
Etud 6;
Etud 7;
```

PROF.txt

```
5
Prof1;Informatique;
Prof2;Informatique;
Prof3;Mathematique;
Prof4;Physique;
Prof5;Chimie;
```

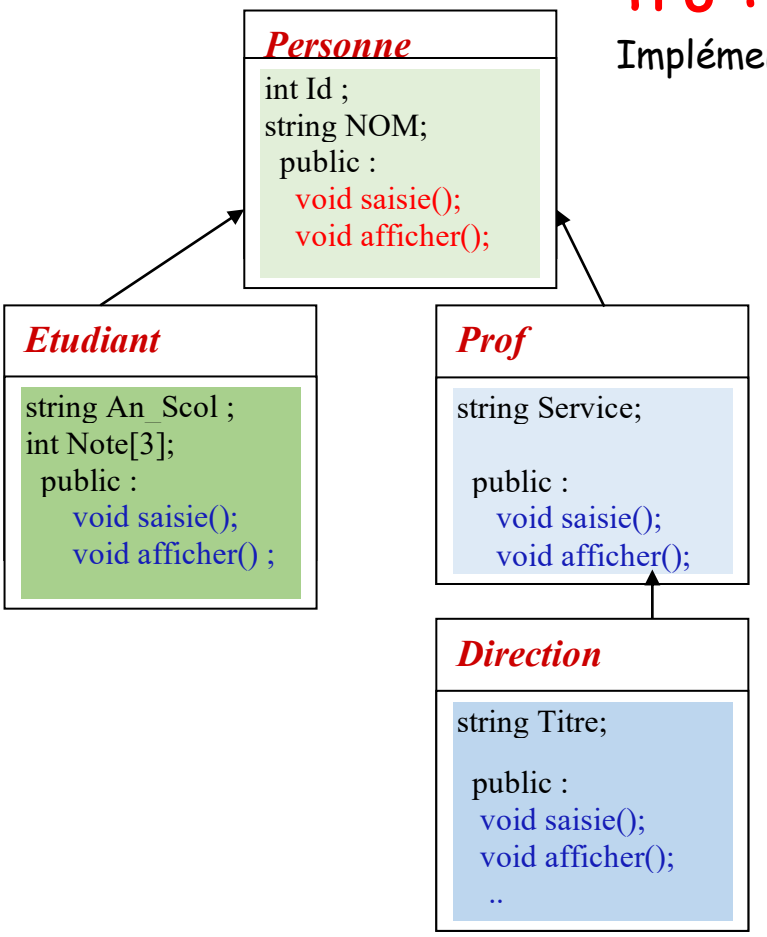
Exemple d'exécution :

```
Entrez votre choix:
1. Lire fichier PROF et stocker les donnees dans un Vector + Affichage
2. Lire fichier ETUD et stocker les donnees dans une List + Affichage
3. Ajouter un PROF au debut du Vector + Afficher
4. Ajouter un ETUD au debut de List + Afficher
5. Supprimer le PROF et ETUD au debut de Vector et List + Affichage
6. Creer une List2 contenant uniquement les noms des ETUD et PROF
7. Quitter
Entrez votre choix:  1

Ce conteneur contient :
1     Prof1   Informatique
2     Prof2   Informatique
3     Prof3   Mathematique
4     Prof4   Physique
5     Prof5   Chimie

Entrez votre choix:  2

Ce conteneur contient :
1     Etud 1
2     Etud 2
3     Etud 3
4     Etud 4
5     Etud 5
6     Etud 6
7     Etud 7

Entrez votre choix:  3

Veuillez saisir les informations suivantes :
Nom :PROFFF
Service :Math
Ce conteneur contient :
6     PROFFF  Math
1     Prof1     Informatique
2     Prof2     Informatique
3     Prof3     Mathematique
4     Prof4     Physique
5     Prof5     Chimie

Entrez votre choix:  4

Veuillez saisir le nom :ETUD  DD
Ce conteneur contient :
8     ETUD  DD
1     Etud 1
2     Etud 2
3     Etud 3
4     Etud 4
5     Etud 5
6     Etud 6
7     Etud 7
```

```
Entrez votre choix:  5

Ce conteneur contient :
1     Etud 1
2     Etud 2
3     Etud 3
4     Etud 4
5     Etud 5
6     Etud 6
7     Etud 7

Ce conteneur contient :
1     Prof1   Informatique
2     Prof2   Informatique
3     Prof3   Mathematique
4     Prof4   Physique
5     Prof5   Chimie

Entrez votre choix:  6

Voici la liste des noms des etudiants et des PROFs:
Etud 1
Etud 2
Etud 3
Etud 4
Etud 5
Etud 6
Etud 7
Prof1
Prof2
Prof3
Prof4
Prof5

Entrez votre choix:
1. Lire fichier PROF et stocker les donnees dans un Vector + Affichage
2. Lire fichier ETUD et stocker les donnees dans une List + Affichage
3. Ajouter un PROF au debut du Vector + Afficher
4. Ajouter un ETUD au debut de List + Afficher
5. Supprimer le PROF et ETUD au debut de Vector et List + Affichage
6. Creer une List2 contenant uniquement les noms des ETUD et PROF
7. Quitter
```

TP6 :

Implémentez la hiérarchie de classes



But :

Manipulation des conteneurs Vector & List et les algorithmes de la librairie STL. Utilisation de l'héritage qui est un principe propre à la programmation orientée objet. Optimisation et réutilisation de déclarations et de code.

Ensuite, implémentez un programme qui doit gérer en boucle le menu suivant :

1 : **Saisir N Etudiants et affichage ;**
2 : **Saisir M Profs et affichage ;**
3 : **Doyen membre de Direction qui Calcule la Moyenne de chaque étudiant + affichage ;**
4 : **Chercher selon le NOM si un Etud ou Prof existe ou pas et affichage ;**
5 : **Trier selon le Nom (Etud ou Prof) par ordre croissant et affichage ;**
6 : **Afficher la liste des étudiants par année scolaire (*An_Scol*) et par ordre décroissant de leur moyenne.**

An_Scol = Ba1 ou Ba2 ou Ba3 ou Ma1 ou Ma2.

Case 1 et 2 ; Saisie au clavier.
Les Etudiants sont rangés dans une ***List*** alors que les PROFs sont rangés dans un ***Vector***.

Il est possible d'ajouter des paramètres aux fonctions. Si nécessaire, d'autres fonctions membres peuvent être ajoutées aux classes.

Optimisez votre code afin de permettre la réutilisation des morceaux de code.

S'il vous reste du temps :

Que deviendra le programme si l'on suppose que :

- Chaque étudiant peut choisir le nombre de matières ainsi que leur titre. Tous les étudiants n'ont pas forcément les mêmes matières ni le même nombre (exp : Etud1 : Math et Info ; Etud2 : Chimie, Info, MécaRa, Anglais).
- Un Prof ne peut enseigner qu'une seule matière (exp : Math ou Anglais, …)

Chercher tous les étudiants qui ont cours avec le Prof X.

Et si vous avez encore du temps :

Que deviendra le programme si un Prof peut enseigner une ou plusieurs matières (exp : Math et Anglais) et qu'il faille chercher tous les étudiants qui ont au moins un cours avec le Prof X.

Justifiez votre choix du type pour ranger les matières.

Exemple d'exécution :

```
Entrer votre choix
-------------------------
  1. Saisir N Etudiants et afficher
  2. Saisir M Profs et affichage
  3. Doyen membre de Direction Calcule la moyenne de chaque étudiant et affichage
  4. Chercher selon le NOM si un Etud ou Prof existe et affichage
  5. Trier selon le Nom (Etud ou Prof) par ordre croissant et affichage
  6. Afficher la liste des étudiants par année scolaire et par ordre décroissant de leur moyenne
  7. Quitter

Choix : 1
Nombre d'étudiants   : 6
Nombre de matière <= 3: 3
Martin   BA2  10 11 15
Maxime BA2   17 16 13
Jean      BA3   15 13 14
Joy       MA1  17 18 15
Marie   BA2    17 13 13
Helena BA1    11 16 12

Choix : 2
Nombre de Profs : 5
Claude   Chimie
Theo     Info
Pierre   Elec
Laurent Math
Samuel  Info

Choix : 3
Alice Mine Doyen
== Moyennes ==
Martin   BA2   10 11 15    M: 12
Maxime  BA2   17 16 13    M: 15.33
Jean       BA3   15 13 14   M: 14
Joy        MA1  17 18 15   M: 16.66
Marie     BA2    17 13 13   M: 14.33
Helena   BA1    11 16 12   M: 13

Choix : 4
Entrer un NOM : Pierre
Pierre existe. Il s'agit d'un Prof

Choix : 4
Entrer un NOM : Maxime
Maxime existe. Il s'agit d'un Etudiant

Choix : 4
Entrer un NOM : Mathis
Mathis n'existe pas !!

. . .
```

TP7

But : **Manipulation des conteneurs associatifs MAP/Multi-MAP.**
Utilisation des algorithmes de la librairie STL.
Optimisation et création de code générique.

Réalisez un programme qui permet de charger et d'analyser la base de données du classement des entreprises selon leurs valeurs marchandes (ou *Market Value*[1], *en anglais),* et de charger et d'interagir avec des données de densité de population.

I-Formats des Bases de données

Les deux bases de données, à exploiter, dans le cadre de ce TP, sont disponibles sur Moodle, *dans le répertoire TP7_MAP*. Il s'agit des deux fichiers :

1- **"entreprises.txt"** qui contient les données des 150 plus grandes entreprises mondiales, les mieux classées, e, 2015, selon le quotidien économique britannique, Financial Times (FT)[2].
2- **"countries.txt"** qui contient les données de densité de population par région et superficie[3].

Format du fichier "entreprises.txt"

Company; Country; Market_Value; Sector; Turnover; Employees; Rank_2015; Rank_2014

Le tableau ci-dessous vous montre un extrait de la base de données, contenue dans entreprises.txt

Company	Country	Market value (m$)	Sector	Turnover (m$)	Employees	Global rank 2015	Global rank 2014
Apple	US	724,773.1	Technology hardware & equipment	182,795.0	92,600	1	1
Exxon Mobil	US	356,548.7	Oil & gas producers	364,763.0	75,300	2	2
Berkshire Hathaway	US	356,510.7	Nonlife insurance	N/R	316,000	3	5
Google	US	345,849.2	Software & computer services	66,001.0	53,600	4	4
Microsoft	US	333,524.8	Software & computer services	86,833.0	128,000	5	3
PetroChina	China	329,715.1	Oil & gas producers	367,853.7	534,652	6	16
Wells Fargo	US	279,919.7	Banks	N/R	264,500	7	7

[1] Market value is the value of a company according to the stock market. ~ investopedia.com
[2] source : https://www.ft.com/content/1fda5794-169f-11e5-b07f-00144feabdc0#axzz3mMra30FA
[3] source : http://gsociology.icaap.org/dataupload.html

Format du fichier "countries.txt"

Region; Nom; Population; Area

Liste des régions : ASIA (EX. NEAR EAST), EASTERN EUROPE, NORTHERN AFRICA, OCEANIA, WESTERN EUROPE, SUB-SAHARAN AFRICA, LATIN AMER. & CARIB, C.W. OF IND. STATES, WESTERN EUROPE, NEAR EAST, BALTICS, NORTHERN AMERICA

II- Construction des classes

Vous utiliserez deux classes, à savoir :

1- La classe « **Entreprise** » caractérisée par :

```
string m_company;        //(Nom de la Société) clé
string m_country;        //(Nom du Pays)
float m_market_value;    //(Valeur Marchande)
string m_sector;         //(Secteur d'Activité)
float m_turnover;        //(Chiffre d'Affaire)
float m_employees;       //(Nombre d'Employés)
int m_rank_2015;         //(Rang en 2015)
int m_rank_2014;         //(Rang en 2014)
```

2- La classe « **PAYS** » caractérisée par :

```
string Region;           // (Localisation) clé
string Nom;              // (Nom)
float Population;
float Area;              // (Superficie)
```

III- Consignes

- ✓ Déclaration, d'au moins, deux méthodes saisie() et affichage(), dans chacune de vos classes, qui vous permettront de saisir et d'afficher **chaque objet** de classe.
- ✓ La manipulation des **Entreprises** et des **Pays** se fera en utilisant les conteneurs associatifs ***MAP/Multi-MAP.***
- ✓ La gestion des choix au niveau du main() sera traitée à l'aide de la fonction ***switch()*** et l'instruction ***case***.
- ✓ Votre programme utilisera une série de fonctions permettant de ***séparer les tâches***.
- ✓ Si nécessaire, des fonctions membres peuvent être ajoutées aux classes ainsi que la définition d'accesseurs et d'opérateurs.
- ✓ Vous devez réaliser un ***programme optimal***, en optant pour l'utilisation des modèles de fonctions (Templates), à chaque fois ou cela est profitable.

IV- Instructions

Les étapes de réalisation, proposées, sont les suivantes :

1. Déclare les **classes** Entreprise et PAYS.
2. Récupére les méthodes saisie() et affichage(), de chaque **Classe**, mises à votre disposition sur Moodle.
3. Crée respectivement une map<string, Entreprise> et une multimap<string, PAYS>, pour contenir respectivement la base de données des entreprises et des PAYS. Indice : Le nom de l'entreprise est la clé de la map et le nom de la région est la clé de la multimap.
4. Définissez la fonction chargement() qui :
   a. Ouvre les fichiers **entreprises.txt** ou **contries.txt** à l'aide du flux de lecture **ifstream (un code que vous pouvez éventuellement utiliser, vous est fourni dans le répertoire TP7_MAP, nommé : TP7_lecture_affichage.cpp)**
   b. Crée un objet **Entreprise** ou **PAYS** à partir de chaque ligne du fichier, à l'aide de void Entreprise::saisie(ifstream &read) ou void PAYS::saisie(ifstream &read).
   c. Insère l'objet **Entreprise** dans la map map<string, Entreprise> ou de l'objet PAYS dans la multimap multimap<string, PAYS>.

5. Définissez la fonction **Affichage** de la map ou de la multimap (qui fait appel à la méthode affichage() de la **classe**).
   Nota Bene : Vérifiez que la map et la multimap sont ordonnées alphabétiquement selon la clé (Ceci est garanti si les conteneurs sont bien définis!).
6. Permettez, à l'utilisateur, de rechercher si une *entreprise existe*.
7. Trouvez et listez les deux entreprises ayant réalisé la plus grosse progression et la plus grosse chute.
8. Pour chaque entreprise, calculez le chiffre d'affaires moyen généré par employé, c'est-à-dire Turnover/Employee, et trouver le max.
9. Pour une région donnée, affichez l'ensemble des pays qui s'y trouvent.

V- (Optionnel) Organisation du code

Votre programme **peut être** organisé comme suit :

Entreprise.h :
- contiendra la définition, les prototypes et les attributs de la classe Entreprise

Entreprise.cpp :
- contiendra la définition des méthodes de la classe Entreprise.

Pays.h :
- contiendra la définition, les prototypes et les attributs de la classe PAYS.

Pays.cpp :
- contiendra la définition des méthodes de la classe PAYS.

main.cpp :
- contiendra la fonction main et le reste du programme.

VI- Exemple d'exécution

```
-----MENU--------

1. Lire le fichier Entreprises et affichage
2. Lire le fichier Pays et affichage
3. Chercher une entreprise par son nom et affichage
4. Cherchez les entreprises ayant réalisé la plus grosse progression/chute et affichage
5. Calculez le Turnover/Employee, le chiffre d'affaire moyen généré par employé, de toutes les entreprises, et trouver le max
6. Pour une région donnée, affichez tous les pays dans cette région
7. Quitter
```

Choix : 1
Entrez le nom du fichier Entreprises: entreprises.txt

Key	Value							
3M	US	104795.3984	General industrials	31821	89800	73	87
ABBOTT LABORATORIES	US	69910.89844	Pharmaceuticals & biotechnology	20247	77000	127	157
ABBVIE	US	93204.10156	Pharmaceuticals & biotechnology	19960	26000	84	97
ACTAVIS	US	120536.1016	Pharmaceuticals & biotechnology	13062.2998	21600	54	285
AGRICULTURAL BANK OF CHINA	China	189297.4062	Banks	0	493583	27	38
AIA GROUP	Hong Kong	75815.79688	Life insurance	0	20000	112	165
... ...							
...							
WESTPAC BANKING	Australia	93870.39844	Banks	0	36373	82	70

------MENU---------
..
Choix : 2
Entrez le nom du fichier Pays: countries.txt

Key	Value		
ASIA (EX. NEAR EAST)	Afghanistan	31056996	6475
ASIA (EX. NEAR EAST)	Bangladesh	147365344	144000
ASIA (EX. NEAR EAST)	Bhutan	2279723	47000
ASIA (EX. NEAR EAST)	Brunei	379444	5770
..			
..			
WESTERN EUROPE	Spain	40397840	504782
WESTERN EUROPE	Sweden	9016596	449964
WESTERN EUROPE	Switzerland	7523934	41290
WESTERN EUROPE	UK	60609152	244820

------MENU---------

Choix : 3

Entrez le nom de l'Entreprise cherchée: Walt Disney

Key	Value						
WALT DISNEY	US	178267.0938	Media	48813	180000	30	39

------MENU---------

Choix : 4

L'entreprise avec la plus grosse chute est : Actavis
L'entreprise avec la plus grosse progression est : Eni

------MENU---------

Choix : 5

Key	Value	
3M	1.166986585	0.3543541133
ABBOTT LABORATORIES	0.9079337716	0.262948066
ABBVIE	3.584773064	0.7676923275
ACTAVIS	5.580375195	0.6047360897

Le turnover maximum est : 724773.125 pour l'entreprise

------MENU----------
Choix : 6

Entrez le nom de la région cherchée: OceaNIa

Key	Value		
OCEANIA	American Samoa	57794	199
OCEANIA	Australia	20264082	7686850
OCEANIA	Cook Islands	21388	240
OCEANIA	Fiji	905949	18270
OCEANIA	French Polynesia	274578	4167
OCEANIA	Guam	171019	541
OCEANIA	Kiribati	105432	811
...			
...			
OCEANIA	Tonga	114689	748
OCEANIA	Tuvalu	11810	26
OCEANIA	Vanuatu	208869	12200
OCEANIA	Wallis and Futuna	16025	274

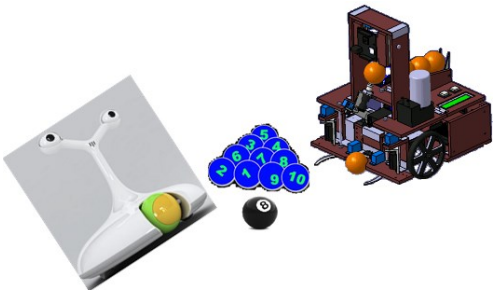
TP8 : Combat de robots 2023

Dans une arène nous disposons d'un certain nombre de boules noires (N = 8 pts) et autres boules numérotées (X), rouges et bleues. Nous considérons un duel entre Robot1 qui ne doit normalement ramasser que les boules rouges (pénalité (-X) si couleur autre que rouge) et le Robot2 qui ne doit ramasser que les bleues (pénalité (-X) si couleur autre que bleue). Par exemple si Robot1 ramasse les boules (R; 7) – (R; 3) – (N; 8) – (B; 1) – (N; 8) - (R; 9), le total sera de 7 + 3 – 8 – 1 – 8 + 9 = 2

Écrire un programme **optimal** permettant de gérer un duel en manipulant une List L1 et un Vector V1 de boules ramassées respectivement pour le Robot1 et le Robot2. La saisie se fera en lisant les données stockées dans des fichiers comme par exemple **Robot1.txt** et **Robot2.txt**. Ces fichiers donnent les boules ramassées par le robot. Chaque boule est caractérisée par sa couleur (char) et son numéro (int).

Ce programme, utilisant les classes et une **programmation optimale avec des templates de fonctions** quand c'est nécessaire, doit gérer en boucle le menu suivant :

1- **LectureSaisieL et Affichage**
2- **LectureSaisieV et Affichage**
3- **Compter le nombre de boules pénalisantes dans L et V**
4- Qui est le **gagnant?**
5- Déposer les boules du Robot perdant dans une **MMap**
6- **Afficher** L, V et **MMap**
7- **Supprimer un élément sur deux de** L, V et **MMap**



Exemple d'exécution :

Votre choix? … = **(1)**

Lecture du fichier : **Robot1.txt**

Affichage(..)

R 7	R 3	N 8	B 1	N 8	R 9

Votre choix? … = **(2)**

Lecture du fichier : **Robot2.txt**

Affichage(..)

B 3	B 4	N 8	R 4	B 2

Votre choix? … = **(3)**

Ici il faut compter le nombre de boules pénalisantes pour chaque contenaire.

Pour Robot1 -> L : N et B
Pour Robot2 -> V : N et R

→ L : **Nbre N = 2; Nbre B = 1**
→ V : **Nbre N = 1; Nbre R = 1**

Votre choix? … = **(4)**

Une fonction **Som(…)** calculera et affichera la somme des valeurs des boules de chaque robot. L'affichage du nom du robot gagnant se fera dans la fonction main().Dans le cas de **(1)** et **(2)** :

Som_{Robot1} = **7+3-8-1-8+ 9 = 2**
Som_{Robot2} = **3+4-8-4+ 2 = -3**

Le gagnant est : Robot1

Votre choix? … = **(5)**

Ici on vide les boules du robot **perdant** dans le bac (**MMap : multimap**) de récupération.

Chaque fois que l'on transfert une boule du robot perdant au bac, il faut afficher les résultats intermédiaires. Pour MMap, tri selon couleur et val.

Voici un exemple d'affichage d'exécution :

Robot Perdant	Bac (MMap)
B 3 B 4 N 8 R 4 B 2	Vide
B 4 N 8 R 4 B 2	B 3
N 8 R 4 B 2	B 3 B 4
R 4 B 2	B 3 B 4
B 2	B 3 B 4 N 8
Vide	B 2 B 3 B 4 N 8

Votre choix? … = **(6)**

L :

R 7	R 3	N 8	B 1	N 8	R 9

V : **Vide**

MMap :

B 2	B 3	B 4	N 8	R 4

Votre choix? … = **(7)**

L :

R 7	**N 8**	**N 8**

V : **Vide**

Map :

B 2	**B 4**	**R 4**

Votre choix? … = **(4)**

Votre choix? … = **(1)**

Votre choix? … = **(2)**

Les données des fichiers :

Robot1.txt	Robot2.txt
R;7; R;3; N;8; B;1; N;8; R;9;	B;3; B;4; N;8; R;4; B;2;

TP9_1

But : **Création d'une application avec interface graphique sur Qt Creator. Manipulation de conteneurs List et Vector. Manipulation de fichiers (txt).**

Partie 1 : Introduction à Qt

Qt est un framework de développement d'applications multi-plateformes. Qt n'est pas un langage de programmation, c'est un framework développé en C++ qui va étendre les fonctionnalités du C++. Qt est accompagné de son interface de développement, **Qt Creator**, disposant d'outils permettant par exemple de générer des interfaces graphiques de manière simplifiée.

1.1 Qt Creator et création de projet

Voici à quoi ressemble Qt Creator après la création d'un projet.

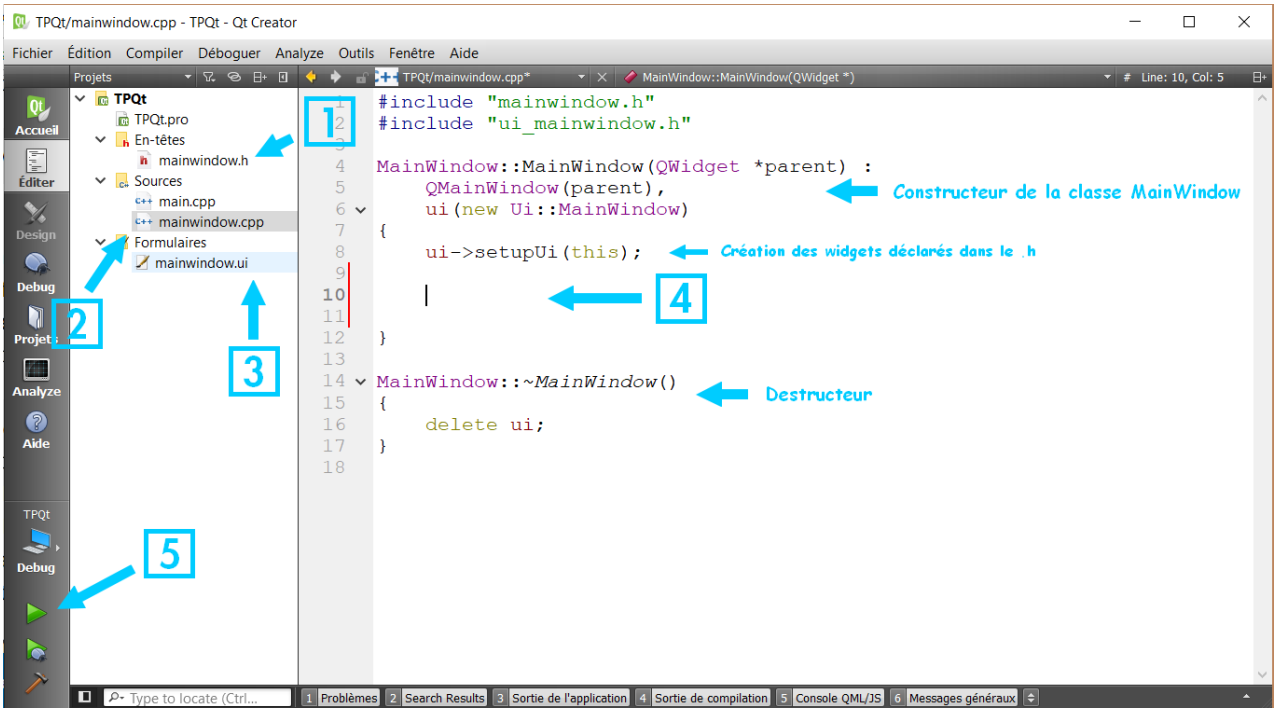


Figure 1 - MainWindow.cpp

Une classe MainWindow (votre fenêtre principale pour l'interface) est automatiquement créée. Il y a donc un fichier MainWindow.cpp (image ci-dessus) et un fichier MainWindow.h (image ci-dessous).

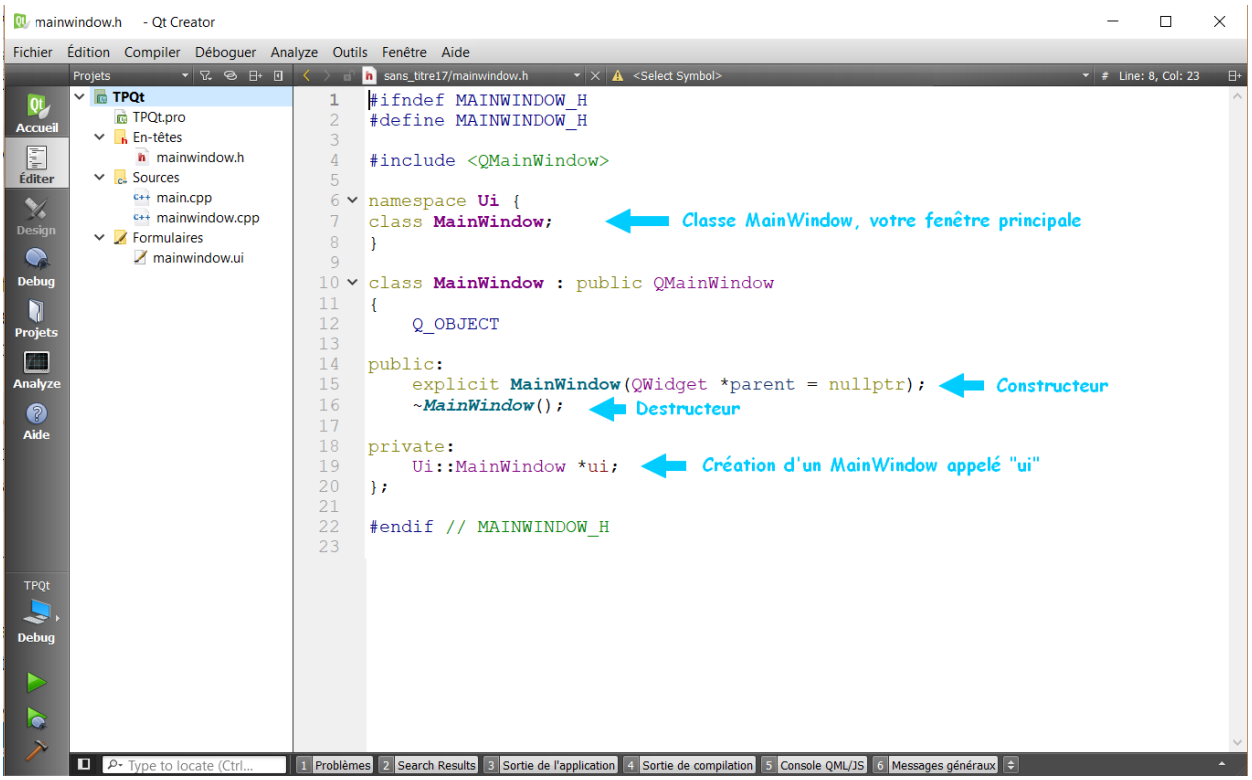


Figure 2- MainWindow.h

1 - Il s'agit du header de notre projet (le .h de la classe créée).

2 – Les fichiers sources du projet. Le premier fichier .cpp (main.cpp) permet de gérer les différentes interfaces graphiques. Le deuxième fichier .cpp (mainwindow.cpp) est lié à la fenêtre principale c'est-à-dire votre interface graphique. Ce fichier contiendra la majorité du code de l'application.

3 – On retrouve ici les fichiers ***.ui*** (user interface). Chaque fichier .cpp gérant une interface graphique est lié à son fichier ***.ui*** dans lequel, vous pourrez modifier l'apparence de l'interface.

4 – Vous pouvez commencer à coder dans cette partie qui est l'équivalent de l'intérieur d'une fonction main().

5 – C'est le bouton d'exécution du projet.

Voici la page permettant de gérer votre interface graphique.

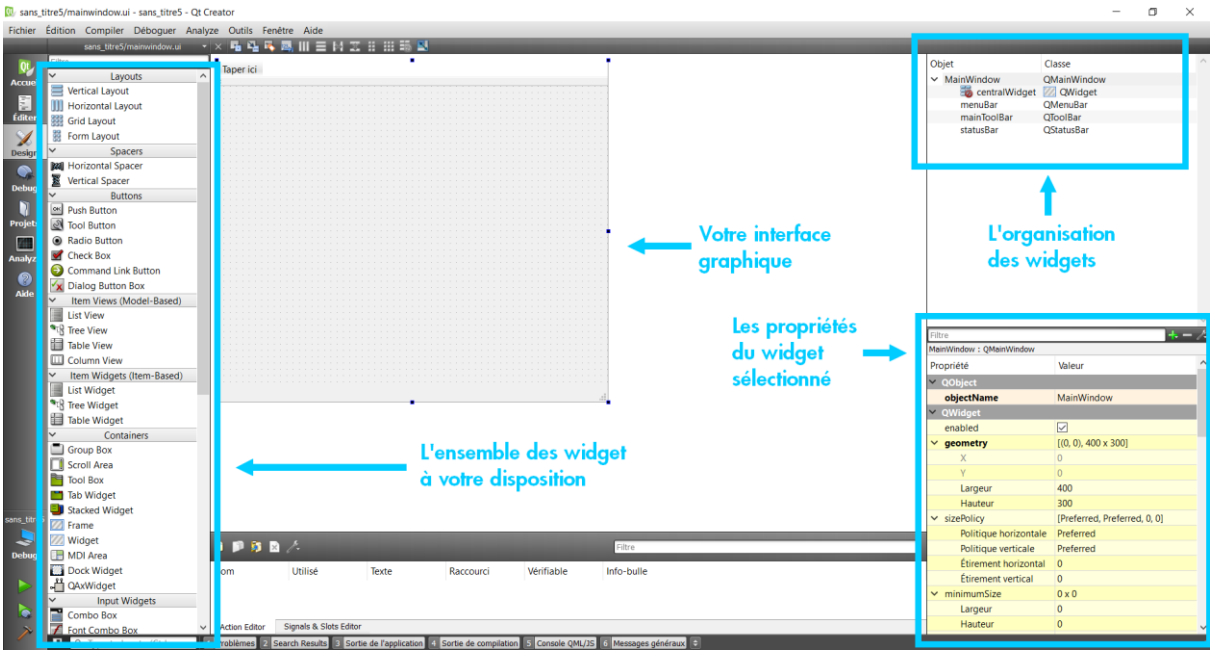


Figure 3- MainWindow.ui

1.2 Création de l'interface avec les widgets

Pour créer une interface sur Qt Creator, il vous suffit de faire du drag and drop (glisser – déplacer) des widgets (un widget est un objet d'interface, un bouton par exemple), vers votre interface graphique. La partie supérieure droite vous permet d'organiser vos widgets. La partie inférieure droite permet de modifier certaines propriétés des widgets (Taille du widget, nom de variable, …).

Voici l'ensemble des widgets utilisés durant ce TP :

Nom du widget :	**Description :**
Stacked Widget	Système de page
Label	Affichage de texte sur l'interface
Line Edit	Zone pour pouvoir rentrer du texte
Push Button	Bouton
Combo Box	Menu déroulant
List Widget	Tableau permettant d'afficher une liste d'objet
Check Box	Case à cocher

Concernant le fonctionnement des widgets, il y a deux choses à connaitre, les slots et les méthodes.

1.2.1 Slots

Un slot est une fonction qui est appelée lorsqu'un évènement s'est produit. Par exemple, cliquer sur un bouton est un évènement et va donc appeler une fonction. Pour créer ces slots, il vous suffit de faire un clic droit sur le widget, de choisir l'option « Aller au slot … » et de sélectionner l'évènement qui vous intéresse. La fonction sera automatiquement créée et vous n'aurez qu'à y rajouter votre code.

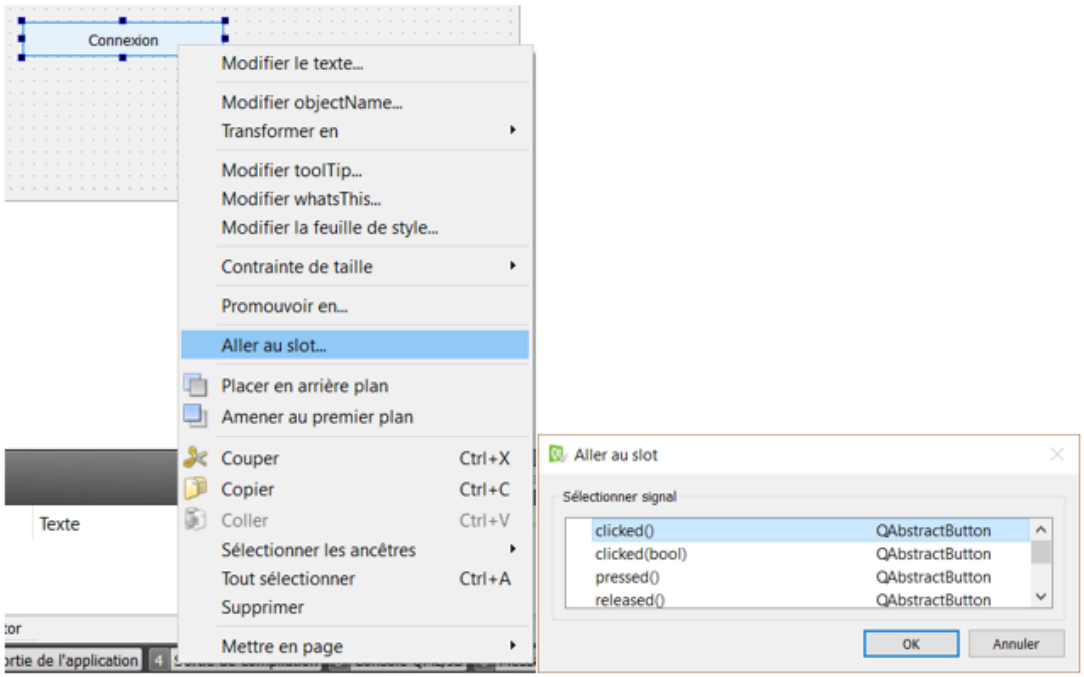


Figure 4- Création du slot lié au clic sur un bouton

Voici l'ensemble des slots utilisés pour ce TP ainsi que les widgets auxquels ils sont liés.

Nom du widget :	**Slot :**	**Description :**
Push Button	Clicked()	Appelé quand l'utilisateur clique sur le bouton
Combo Box	Activated(int)	Appelé quand l'utilisateur choisi un élément du menu déroulant
List Widget	itemClicked()	Appelé quand on clique sur un des éléments du tableau
Check Box	Clicked()	Appelé quand on coche ou décoche la case

1.2.2 Méthodes

Chaque widget possède des méthodes, ce sont des fonctions qui vont vous permettre de travailler avec ces widgets. Il y a par exemple, une méthode qui va permettre de récupérer le texte qu'un utilisateur a encodé dans une zone de texte, ou bien une méthode pour modifier le texte affiché sur un bouton.
La syntaxe pour l'utilisation d'une méthode d'un widget sera toujours la même.

```
ui->nomDuWidget->nomDeLaMéthode();
```

Par exemple, si on veut récupérer le texte qu'un utilisateur aurait rentré dans une zone de texte (Widget : Line Edit) voici le code :

```
ui->lineEdit->text();
```

Le « ui » est l'objet « user interface », c'est votre interface graphique, on pointe ensuite (->) vers un certain widget de cette interface.

Voici l'ensemble des méthodes utilisées dans ce TP :

Nom du widget :	**Nom de la méthode :**	**Description :**
Line Edit	text()	Récupère le texte présent
	setText()	Modifie le texte
Combo Box	addItem()	Ajoute un élément dans le menu déroulant
	currentText()	Récupère le texte de l'élément sélectionné dans le menu déroulant
Stacked Widget	setCurrentIndex()	Change l'index du Stacked Widget (chaque index représente une page de votre interface)
List Widget	currentIndex()	Récupère l'index de l'objet sélectionné dans le tableau
	text()	Récupère le texte d'un objet du tableau
	addItem()	Rajoute un objet dans le tableau
Check Box	isChecked()	Renvoie un booléen en fonction l'état de la case à cocher

1.3 Particularité de Qt

Comme mentionné précédemment, Qt permet d'étendre les fonctionnalités du C++ et utilise une version modifiée du C++ dans certain cas.
On peut prendre l'exemple du type de variable *string*.

Dans Qt, ce type de variable devient *QString*. Il sera donc important d'ajouter ***#include <QString>*** au début de votre projet. Il est toujours possible de travailler avec des variables de type *string* mais certaines fonctionnalités spécifiques à Qt (comme par exemple afficher un texte sur votre interface) va demander une variable de type *QString*, il est donc intéressant de travailler avec ce type de variable, ou au minimum être capable de faire la conversion.

```
string texte = "Affichage texte";   //Type string

int chiffre = 1;                    //Type integer

QString texteQt;                    //Type QString

QString chiffreQt;

//Fonction de conversion d'un string vers un QString
texteQt = QString::fromStdString(texte);

//Fonction de conversion d'un int vers un QString
chiffreQt = QString::number(chiffre);
```

Partie 2 : Création de l'application

L'objectif est de créer une interface graphique pour permettre à un professeur de gérer ses élèves. L'interface possède une page de connexion pour le professeur ainsi qu'une page affichant l'ensemble des classes et des élèves du professeur (avec les informations liées à l'élève). Le professeur pourra aussi ajouter un élève.
Voici à quoi ressemblera votre interface :

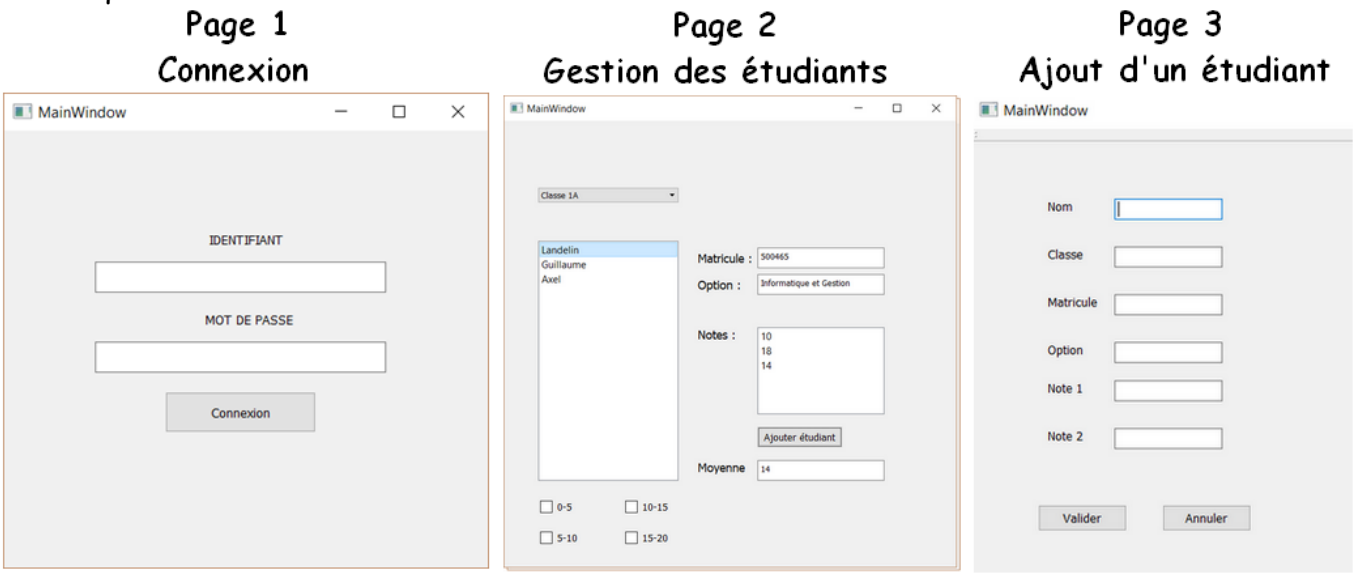


Figure 5- Pages de l'interface

Déclarer deux classes :

Une classe Etudiant dont les objets sont des étudiants caractérisés par

```
QString nom ;
QString matricule ;
QString option ;
QString classe ;
Int notes[nbNotes] ;
```

Une classe Prof dont les objets sont des enseignants caractérisés par

```
QString id ;
QString mdp ;
QString classes[nbClasses] ;
```

Dans les deux classes, on déclarera des méthodes pour récupérer les caractéristiques ainsi qu'un constructeur (prenant en paramètres les caractéristiques de la classe) et un destructeur.

La manipulation des étudiants se fera en utilisant un conteneur List alors que les Profs seront rangés dans un conteneur Vector. Le code pour les deux classes est fourni avec le TP.

Les fichiers profs.txt et etudiants.txt sont à placer dans le dossier « Build » de votre application.

Voici le code pour la lecture de ces fichiers :

```
ifstream lecture;

//Déclaration des variables

string id;
string mdp;
string classes[nbClassesMax_Prof];    //La lecture donne le tableau en string
QString qClasses[nbClassesMax_Prof];  //et on le convertira en QString
string nbP;
string nbCl;

nbProf = 0;
nbClasse = 0;

lecture.open("profs.txt");

getline(lecture, nbP, ';');
lecture >> nbCl;
lecture.ignore();

nbProf = atoi(nbP.c_str());      //La lecture donne nbCl et nbP en string
nbClasse = atoi(nbCl.c_str());  //et il faut les convertir en entier

for(int i = 0 ; i < nbProf ; i++){

    getline(lecture, id, ';');
    getline(lecture, mdp, ';');

    for(int j = 0 ; j < nbClasse ; j++){

        if(j!=nbClasse-1){
            getline(lecture, classes[j], ';');
        }else{
            lecture >> classes[j];
            lecture.ignore();
            }

             //Converison du tableau de string en QString

            qClasses[j] = QString::fromStdString(classes[j]);

        }

       professeurs.push_back(Prof(QString::fromStdString(id),
QString::fromStdString(mdp), qClasses, nbClasse));

    }

lecture.close();
```

2.1 Page de connexion

Commencez par extraire les données des professeurs depuis le fichier **professeurs.txt**.
Créez une page de connexion dans laquelle l'utilisateur (prof) peut rentrer son identifiant et son mot de passe. Récupérez ensuite les valeurs rentrées par l'utilisateur pour les comparer au fichier prof et valider (ou non) la connexion du professeur.

2.2 Page de gestion des étudiants

Commencez par créer un menu déroulant dans lequel les différentes classes du prof vont apparaître.
Créez ensuite un tableau pour afficher l'ensemble des étudiants présents dans une classe. Ce tableau doit se remplir quand l'utilisateur sélectionne une classe dans le menu déroulant.
Créez ensuite différentes zones de textes pour afficher les caractéristiques des étudiants (Option, Matricule, Notes, Moyenne). Ces valeurs doivent se remplir lorsque l'utilisateur clique sur un élément du tableau.

Pour terminer cette page de gestion des étudiants, créer plusieurs check box permettant de n'afficher que certains étudiants en fonction de leur moyenne (check box avec les notes de : 0 à 5, 5 à 10, 10 à 15, 15 à 20). La liste d'étudiants affichée dans le tableau doit s'actualiser quand l'utilisateur clique sur une check box.

2.3 Page ajout d'un étudiant

Sur votre page de gestion des étudiants, créez un bouton « Ajouter un étudiant » qui dirigera l'utilisateur vers une nouvelle page dédiée à la création d'un étudiant.
Cette page doit permettre à l'utilisateur de rentrer dans des zones de texte les différentes caractéristiques de l'étudiant. En validant la création de l'étudiant via un bouton, vous ramènerez l'utilisateur à la page de gestion des étudiants.

2.4 S'il vous reste du temps …

Affichage d'un message d'erreur :

Si l'identifiant ou le mot de passe est incorrect, on veut afficher un message d'erreur pour prévenir l'utilisateur. Pour ce faire il faut utiliser un objet de type *QMessageBox*, qui est un popup permettant d'afficher des messages à l'utilisateur (ne pas oublier ***#include <QMessageBox>***).

On commence par déclarer un objet de type *QMessageBox*. On définit ensuite le texte lié à ce *QMessageBox* via la méthode *setText(QString)* qui prend un *QString* en paramètre. Pour terminer, il faudra appeler la méthode *exec()* de votre *QMessageBox* lorsque vous souhaitez faire apparaitre ce message.

Valider les valeurs des champs :

Pour assurer le bon fonctionnement du programme, il ne faudrait pas que l'utilisateur rentre des valeurs erronées, comme par exemple rentrer un nom (*QString*) à la place d'une note (*Int*). Pour ce faire, on va utiliser des objets de type *QRegExpValidator*. Le constructeur d'un objet de type *QRegExpValidator* prend en paramètre un objet de type *QRegExp*, qui est lui-même un simple *QString*. La valeur de ce *QString* va définir le type de caractères qui seront acceptés dans les champs de texte.

Voici des exemples :

[a-zA-Z\\s]{0,64} = Uniquement des lettres de A à Z (majuscules et minuscules), des espaces (\\s) et on autorise un nombre de caractères entre 0 et 64.

[5]\\d{5} = Un 5, suivi de 5 chiffres (pour assurer un matricule entre 500 000 et 599 999).

[1][ABCD] = un 1 suivi d'un A, B, C ou D (pour avoir les classes 1A, 1B, …).

[0-9]|1[0-9]|20 = Un chiffre de 0 à 9, ou (le « ou » est représenté par |) un chiffre 1 suivi d'un autre chiffre de 0 à 9, ou un 20 (pour assurer une note de 0 à 20).

Une fois le *QRegExpValidator* correctement créé, on utilise la méthode *setValidator(QRegExpValidator)* de notre widget *lineEdit*.

TP9_2

But : **Manipuler des classes, constructeurs/destructeurs**
Utiliser les templates/ héritage, Vector et réaliser un programme (jeux) graphique.

N.B. : L'étudiant qui souhaite obtenir le code / solution de ce TP, peut en faire la demande à M. BENJELLOUN.

Partie 1 : Introduction à la librairie SFML[4] et Manipulation des classes

1.1 Prise en main de SFML

SFML est une librairie multimédia qui fournit une interface simple pour le développement de jeux vidéo et d'applications interactives. Elle est multiplateforme et permet une programmation orientée objet. Ses modules principaux sont : Graphique, Son, Réseau etc.

Afin de vous initier à l'utilisation de SFML, nous vous proposons en premier lieu de compiler et tester trois programmes du fichier **mainStart.cpp**

a. Le **CODE 1** montre la structure d'un programme basic de la librairie graphique SFML, il permet d'afficher un cercle dans une fenêtre. Le programme se compose de deux parties. Création de la fenêtre et de l'objet cercle, puis une boucle while pour l'affichage et l'actualisation de la fenêtre. La création d'une fenêtre (ou autre objet) se fait suivant un constructeur par paramètre selon le code :

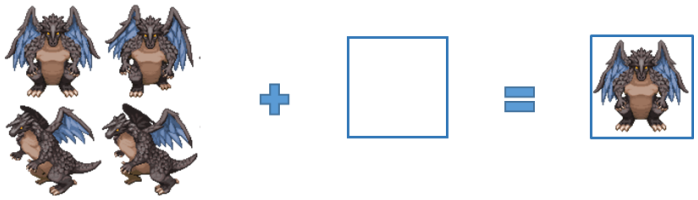
Classe fenêtre de SFML	Le nom de l'objet fenêtre	Paramètres de la fenêtre	
		La taille	Le titre
RenderWindow	maFenetre(	VideoMode(200,200),	"Fenetre SFML");

NB : **Lisez bien les commentaires liés à chaque ligne de code pour comprendre la structure du programme.**

Le passage du CODE 1 au CODE 2 se fait en commentant CODE 1 et en dé-commentant CODE 2.

[4] **S**imple and **F**ast **M**ultimedia **L**ibrary **[Gomila, 2013] :** www.sfml-dev.org

b. Le CODE 2 permet d'afficher un Sprite au lieu du cercle. Le Sprite est définie par le graphique suivant :



Texture Rectangle Sprite

c. Le CODE 3 permet d'animer un Sprite avec les touches du clavier à l'aide de la fonction **update()** qui tourne à l'intérieur de la boucle. Terminer la fonction afin de bouger le Sprite dans tous les sens (Up et Right).

1.2 Manipulation d'une classe: Définition du constructeur et du destructeur

L'objectif du CODE 4 est d'arriver au même résultat que CODE 3 mais avec l'utilisation d'une classe appelée **MyWarrior**

```
class MyWarrior {
public:
 MyWarrior();
 ~MyWarrior();
 void update();
 void getAttack(int damage);
 Sprite m_sprite

private:
 Texture m_texture;
 int m_health;
};
```

La définition de la classe est fournie, vous devez accomplir les points suivants :

a. Implémentez le constructeur pour initialiser les quatre attributs de la classe et le destructeur pour qu'il affiche " Monster Destruction ! " à la fin du programme.

Remarque : Inspirez-vous du code 3 pour initialiser la texture et le sprite.

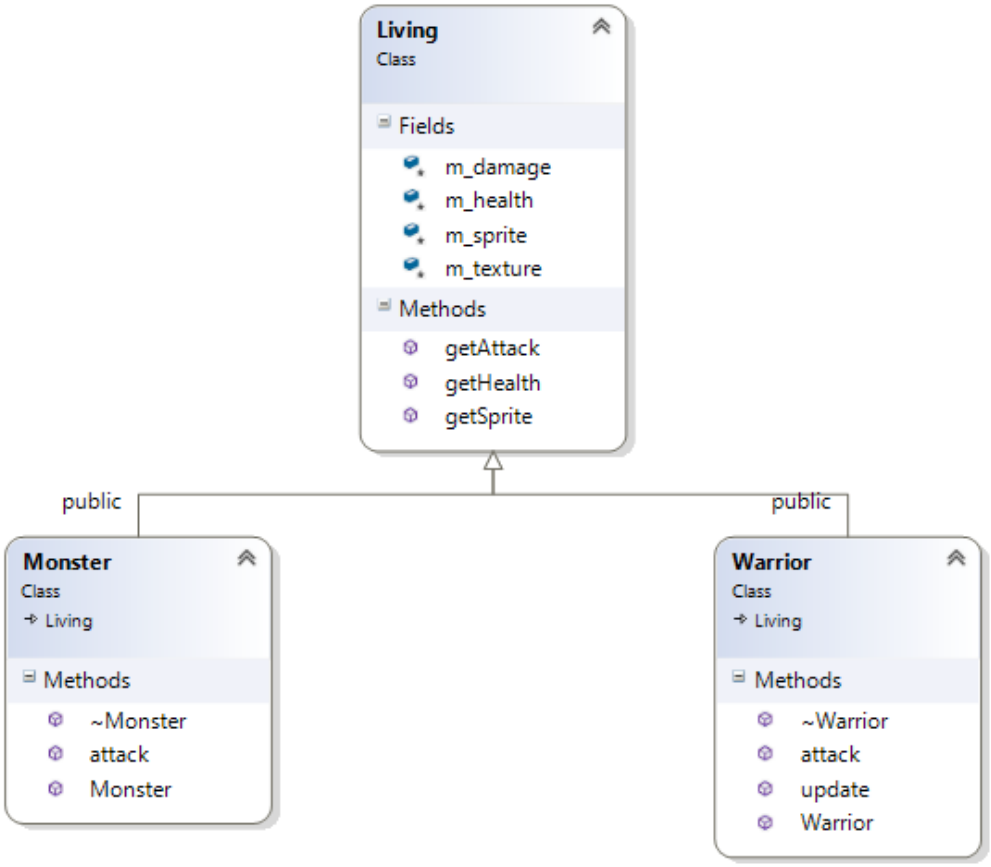
b. Implémentez la fonction getAttack() qui permet de modifier (diminuer) le niveau de vie (m_health), et la fonction Update() cette fois fonction membre de la classe Mywarrior

c. Une fois les fonctions implémentées, déclarez un objet de la classe dans la fonction main du CODE 4. Ensuite, appelez la fonction membre **update()** à l'intérieur de la boucle pour animer l'objet.

Partie 2 : Héritage

Durant cette partie, vous devez réaliser un jeu de rôle RPG entre un guerrier (utilisateur) et un monstre (ordinateur). La création et la gestion de l'interface graphique se fera à l'aide de la bibliothèque SFML. Vous allez appliquer la notion d'héritage qui est un principe propre à la programmation orientée objet.

Commencez par l'implémentation de la hiérarchie suivante :



1. Implémentez la foction getAttack() de la classe Living, ainsi que les accesseurs getHealth() et getSprite().
2. Ensuite implémentez les constructeurs des classes Monster et Warrior. Pour initialiser la position d'un sprite. Utilisez sprite.setPosition(x, y) pour déterminer la position du Monster
3. Dans les deux classes Monster et Warrior, implémenter la fonction attack() qui permet d'infliger des dégâts à l'adversaire, cette fonction fait appelle à la méthode getAttack() de l'adversaire.
4. Afin de lancer le combat, l'attaque sera déclenchée une fois le Warrior est proche d'une certaine distance du Monster (par exp : d = 100 px).
   a. Implémentez la fonction globale close(warrior,monster) qui calcule la distance entre les deux.

   PS : Il est possible d'utiliser la notion de Template pour cette fonction !

   Autres solutions existent grâce à l'héritage.

b. Vous avez certainement remarqué que l'attaque se fait à une très grande fréquence. Afin de réduire cette fréquence, vous allez ajouter la condtion suivante :

Si le Warrior est proche du guerrier **et** le temps passé > 100 ms
Lancer une attaque du Monstre
Réinitialiser le compteur de temps

Pour tester si la valeur du compteur est supérieure à « 100ms », utilisez :

```
sf::Clock clock; // lancer le chrono à l'extérieur de la boucle
sf::Time elapsed1 = clock.getElapsedTime();
   // elapsed1.asMilliseconds() // retourner la valeur du temps
clock.restart();
```

c. De même

Si le Warrior est proche du guerrier **et** la touche espace appuyée
lancer une attaque du Warrior

Pour tester si le bouton « espace » est appuyé, utilisez :
Keyboard::isKeyPressed(Keyboard::Key::Space)

5. Pour gérer la fin du combat :
   a. Si vie du Monster est nulle, affichez « **You Win !** »
   b. Si la vie du Warrior est nulle, affichez « **Game Over !** »

Partie 3 : Un vecteur de Monsters

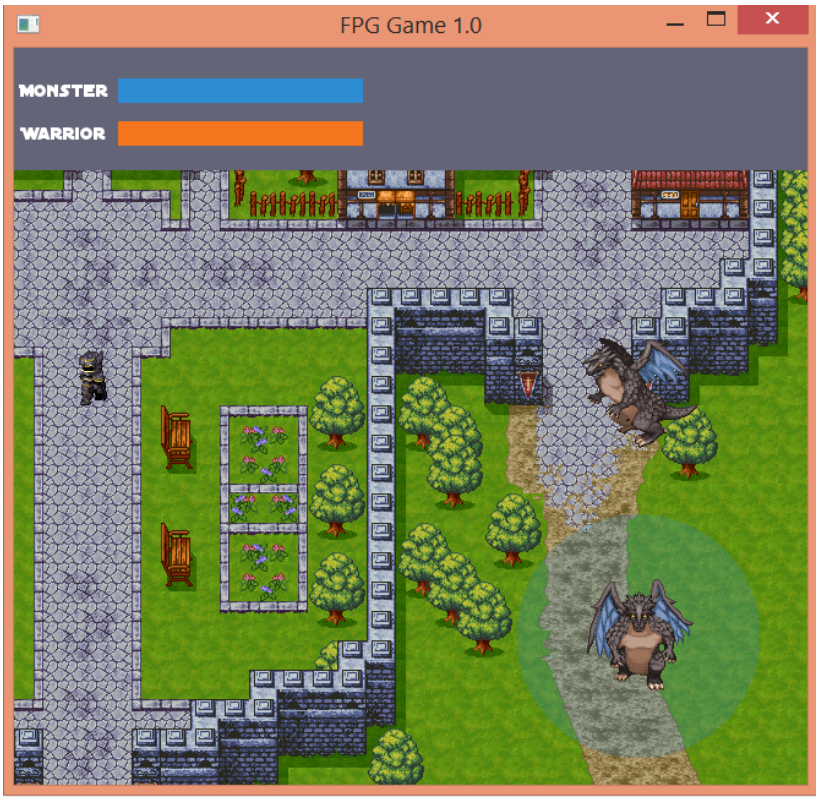
Le jeu peut être étendu pour le cas de plusieurs Monsters organisés dans un vecteur, dans ce cas la fonction "Close" doit être appelé pour chaque monstre :

1- Commencez par créer un vecteur de monstre vector<Monster> monsters
2- Utilisez une boucle sur les éléments du vecteur, et remplacer le Monster par un itérator dans les appels des fonctions close() et les différentes conditions.
3- Pour que le code marche correctement il faut changer également le constructeur de la classe Monster. Utilisez une variable statique dans le constructeur pour donner une position différente à chaque Monster de du vecteur.
4- Un Monster doit disparaitre quand il meurt.
L'interface du jeu à ce stade est la suivante :

Partie 4 : Des améliorations du jeu

Afin d'améliorer la version actuelle du jeu, on peut imaginer d'autres points supplémentaires :

- Multi-joueurs : crée un 2ème Warrior qui sera contrôlé par une autre fonction update (utilise d'autres boutons de contrôles par exp : 4, 8, 6 et 2 pour la direction et + pour les frappes).
- Un Monster qui a le niveau de vie très bas fusionne avec un autre proche de lui (s'il existe).
- Des éléments graphiques seront fournis dans un fichier annexe pour atteindre le visuel suivant



D'autres exemples de jeux avec SFML :

1. https://goo.gl/SyaR7g
2. https://goo.gl/hREldW

TP9_3

But : Se familiariser à la définition et l'utilisation d'opérateurs
Utiliser la récursivité et Réaliser un programme graphique

N.B. : L'étudiant qui souhaite avoir le code / solution de ce TP, peut en faire la demande à M. BENJELLOUN.

Durant ce TP, vous devez réaliser un programme graphique permettant de dessiner des fractales à partir de segments de droite.

Un segment de droite entre les coordonnées de deux points peut être tracé à partir de la fonction « **drawLine** » de la librairie **Imagine++**. Cette librairie est accessible en ligne librement à l'adresse : http://imagine.enpc.fr/~monasse/Imagine++/.

Le prototype de la fonction est le suivant :

```
void drawLine (int x1, int y1, int x2, int y2, const Color& c, int pen_w) ;
```

Le quatrième paramètre de cette fonction sert à indiquer la couleur du trait. Des couleurs de base sont définies dans la librairie **Imagine++** (ex : BLUE, RED, etc.). Le dernier paramètre permet de préciser la largeur du trait. Cette fonction est utilisable à condition d'inclure le fichier **<Imagine/Graphics.h>** en début du programme. De même, l'espace de nom « Imagine » doit être utilisé (**using namespace Imagine**).

Ces dessins seront réalisés dans une fenêtre graphique définie à l'aide d'Imagine++. Il sera nécessaire d'appeler la fonction **openWindow**(int largeur, int hauteur) au début du main() et une fonction **endGraphics()** à la fin du main().

Dans un premier temps, il est demandé de :

1. Définir une classe « Vecteur » de sorte à pouvoir manipuler des vecteurs. Cette classe contiendra au minimum les attributs et les méthodes décrits ci-après :

```
class Vecteur {
        double  x,  y;
 public:
        Vecteur operator+(Vecteur b);
        Vecteur operator-(Vecteur b);
        Vecteur operator*(double lambda);
        Vecteur operator/(double lambda);
        Vecteur Rotate(double angle);
};
```

2. Implémenter une fonction **dessinerSegment** comprenant au moins deux paramètres de type Vecteur. Le rôle de cette fonction est de dessiner un segment de droite entre les extrémités de deux vecteurs.

Sur base du travail réalisé, le dessin d'un arbre fractal peut être réalisé. Un tel arbre possède la structure de la figure 6_3.a. Le résultat final attendu est celui de la figure 6_3.b ou 6_3.c.

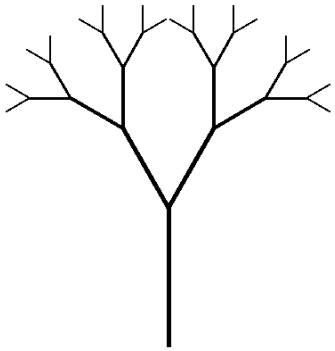


figure 6_3.a

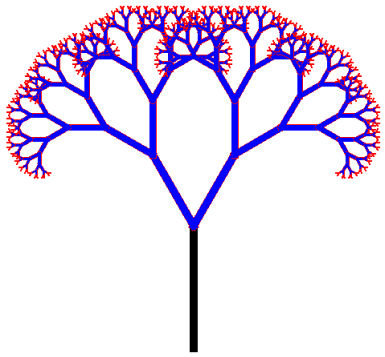


figure 6_3.b

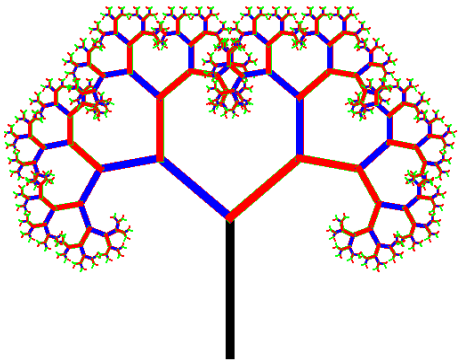
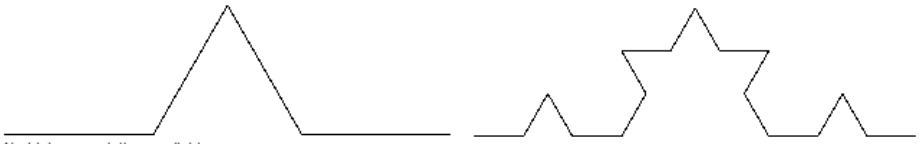


figure 6_3.c

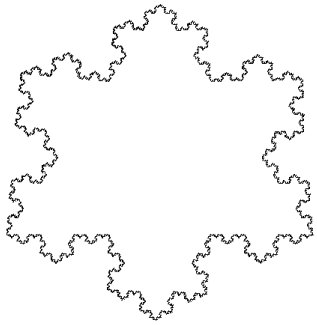
À partir de deux vecteurs initiaux, il est demandé de :

3. Implémenter la fonction ***récursive* dessinerArbre** permettant de tracer un arbre fractal à partir de deux vecteurs. Un paramètre concernant la « profondeur » de l'arbre courbe sera à intégrer à la fonction dessinerArbre.

De même, une courbe de Koch se construit en remplaçant le deuxième tiers d'un segment par deux segments formant la pointe d'un triangle équilatéral, comme illustré à la figure ci-dessous.



Il est donc demandé de construire un flocon de neige en dessinant trois courbes de Koch à partir des trois segments formant un triangle équilatéral (voir illustration à la figure).



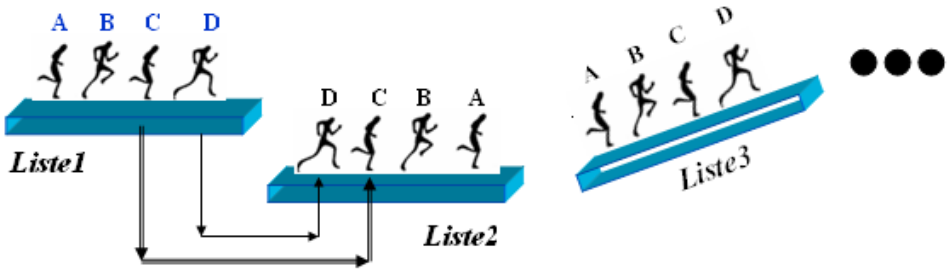
Les derniers points du TP consistent donc à :

4. Implémenter la fonction ***récursive* dessinerKoch** permettant de tracer une courbe de Koch entre les extrémités définies par deux vecteurs. Un paramètre concernant la « profondeur » de la courbe sera à intégrer à la fonction dessinerKoch.

5. Utiliser la fonction dessinerKoch pour réaliser un flocon de neige de Koch entre trois extrémités définies par des vecteurs.

Exemples d'examens

Parcours Vita



Ecrire un programme permettant de manipuler plusieurs listes dont le nombre est **NL< Nmax=15**. Il s'agit d'un Parcours Vita avec **NL** activités et un nombre de participants **NP<Nmax.** Les informations caractérisant chaque participant dans la liste sont les suivantes: **Nom** : chaîne de caractères
Code : un caractère (A, B, …)
Créer la (les) **structure**(s) nécessaire(s) à ce problème.
Les fonctions *Saisie(..)* et *Affichage(..)* pour chaque élément (participant) sont déclarées dans la structure participant.

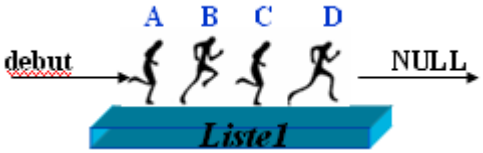
Ce programme doit gérer en boucle le menu suivant :

8- SaisieL Liste1 et Affichage
2- Liste1 TO Liste2 et Affichage
3- Parcours et Affichage
4- Fin

1- Saisie Liste1 et Affichage:

Est constitué au moins de *deux fonctions* :
SaisieL (…….) ;
Dans cette fonction on demandera et **on testera** le nombre de participants (**NP**) à saisir et on effectuera la saisie des participants dans **une liste par insertion en tête**.

Par exemple si **NP=4** et l'utilisateur introduit au clavier : (NomD D) puis (NomC C) puis (NB B) et enfin (NA A)
La liste est :



Affichage(…….) ;

(NA A) --- (NB B) --- (NomC C) --- (NomD D)

2- Liste1 → Liste2 et Affichage:

Ici, comme si les participants sautent, un par un, de la poutre1 (Liste1) à la poutre2 (Liste2). On applique (**FIFO**), c'est-à-dire, on retire d'abord le D de Liste1, on insère D dans Liste2 et on affiche liste1 et liste2.
Liste1 : (NA A) --- (NB B) --- (NomC C)
Liste2 : (NomD D)

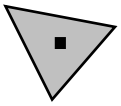
Puis on transfert **(NomC C)** de **Liste1** vers **Liste2.** Ce dernier sera inséré **dans Liste2 après** (NomD D).

Liste1 : (NA A) --- (NB B)
Liste2 : (NomD D) --- (NomC C)
…
Tant que liste1 n'est pas vide, on continue.

A la fin on aura l'affichage suivant :

Liste1 : Vide
Liste2 : (NomD D) --- (NomC C) --- (NB B) ----(NA A)

Attention : Ici on affiche l'état des deux listes chaque fois qu'un participant passe de la liste1 à la liste2. Il faut soigner tous les affichages afin de bien suivre et comprendre les différentes séquences.



3- Parcours et Affichage:

Ici on demandera à l'utilisateur le nombre **NL** et on y fera traverser les **NP** participants comme le montre la figure générale (début).

On affiche toutes les listes chaque fois que toute l'équipe passe d'une liste à l'autre. Si par exemple **NL=3** on affichera:

Liste1 : (NA A) --- (NB B) --- (NomC C) --- (NomD D)
Liste2 : Vide
Liste3 : Vide
- - - - - - - - -
Liste1 : Vide
Liste2 : (NomD D) --- (NomC C) --- (NB B) ----(NA A)
Liste3 : Vide
- - - - - - - - -
Liste1 : Vide
Liste2 : Vide
Liste3 : (NA A) --- (NB B) --- (NomC C) --- (NomD D)

RQs :

Utilisez les règles de bonnes pratiques étudiées durant les travaux pratiques telles que l'indentation syntaxique, la programmation modulaire, libérer la mémoire d'un élément supprimé …

Le poulailler

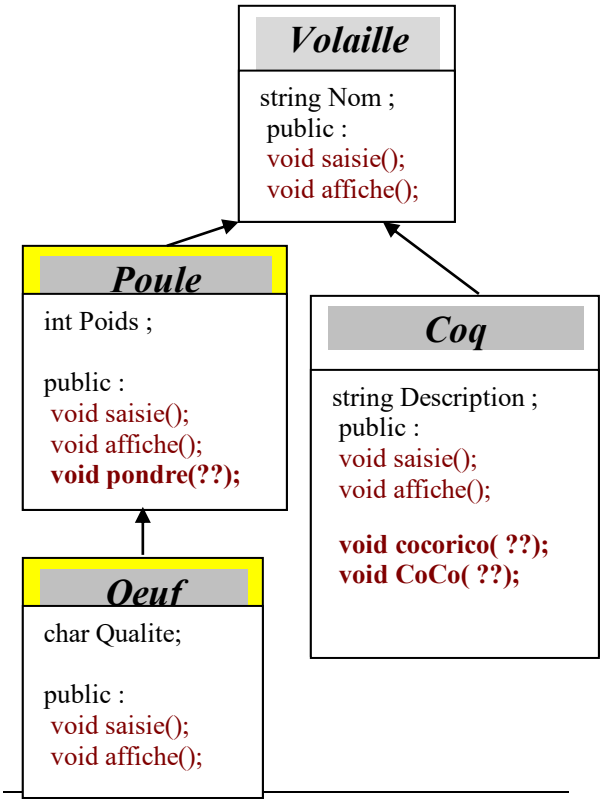
Implémentez la hiérarchie de classes en face : la classe de base ***Volaille*** et ses classes dérivées : ***Poule, Coq, Œuf*** ainsi que les différentes ***fonctions membres***. On considère que le **poids** d'un **œuf = 0,1 poids** de la **poule**. Il est possible d'ajouter des paramètres aux fonctions et/ou d'ajouter d'autres fonctions.

Ce programme doit gérer en boucle le Menu suivant :

9- ***Créer un Tableau Dynamique*** **de poules + Affichage**
10- ***Créer un coq*** **+ Affichage**
5- ***Le coq fait cocorico et toutes les poules pondent***
6- ***Le coq dit CoCo et une poule/2 pond***
7- Fin (on tue tout le poulailler !!)

1- Créer Tableau Dynamique de Poules et Affichage:

Est constitué au moins de *deux fonctions* :
SaisiePoule (…….) ;
Dans cette fonction on demandera et **on testera le nombre** de Poules (NP<Max=10) à saisir et on effectuera la saisie dans **un Tableau Dynamique.**



Voici un exemple d'exécution du programme

Menu :
****
1. Créer un Tb Dyn poule + Af.
2. Créer un coq **+ Affichage**
3. Le coq fait cocorico et toutes les poules pondent
4. Le coq dit CoCo poule/2
5. Fin

Choix : 1
Combien de poules ? : **4**

Le Nom ? : Poule1
Le poids ? : 200
Le Nom ? : Poule2
Le poids ? : 180
Le Nom ? : Poule3
Le poids ? : 1200
Le Nom ? : Poule4
Le poids ? : 1000
→
Poule4 (1000)
Poule3 (1200)
Poule2 (180)
Poule1 (200)

Menu :
****
Choix : 2
Le Nom ? : M_CoQ
Description ? : BeauCoQ

→
Son Nom : M_CoQ
Description : BeauCoQ

Menu :
****
Choix : 3

L'oeuf a bien ete pondu et pese : 100(gr)
Qualite de l'oeuf (A,B ou C)? : A

L'oeuf a bien ete pondu et pese : 120(gr)
Qualite de l'oeuf (A,B ou C)? : B

L'oeuf a bien ete pondu et pese : 18(gr)
Qualite de l'oeuf (A,B ou C)? : A

L'oeuf a bien ete pondu et pese : 20(gr)
Qualite de l'oeuf (A,B ou C)? : C

Menu :
****
Choix : 4 (Une poule sur 2)

L'oeuf a bien ete pondu et pese : 100(gr)
Qualite de l'oeuf (A,B ou C)? : A

L'oeuf a bien ete pondu et pese : 18(gr)
Qualite de l'oeuf (A,B ou C)? : A

Rqs : **Normalement pas de Classe amie.**

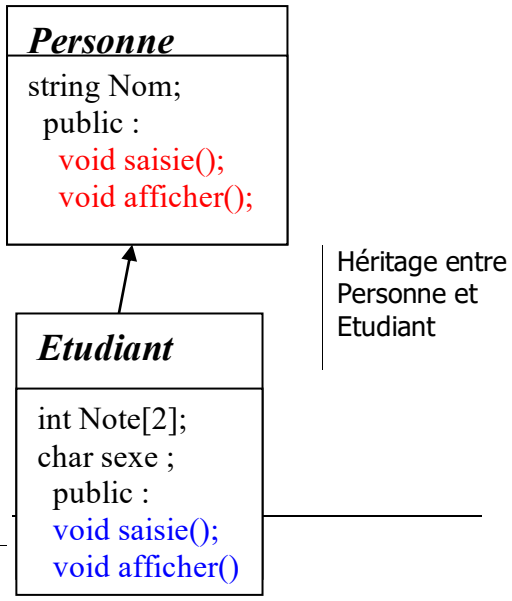
Utilisez les règles de bonnes pratiques étudiées durant les travaux pratiques telles que l'indentation syntaxique, la programmation modulaire, libération de mémoire.

Les étudiants !

Écrivez les classes Personne et Etudiant. Si nécessaire, des fonctions membres peuvent être ajoutées aux classes.

Ce programme doit gérer en boucle le Menu suivant :

1) Saisie d'1 **List** d'Etudiant de 1ere (insertion au **Fin**) + Affichage
2) Saisie d'1 **Vector** d'Etudiant de 2d (insertion au **début**) +Affichage
(Si insertion en fin = **0** point)
3) Calculez et afficher le nombre de filles (F) dans la liste **List**.
4) Calculez et afficher la moyenne de l'Etudiant à la position P ds **List**.
5) Créez 1 Etudiant et chercher dans **List** puis **Vector** s'il **existe** et sa **position** selon le **Nom** et le **sexe.**
6) Transférez les filles de List et Vector dans un tableau **Dynamique TD**.
(Si TD est un Vector ou list = moitié de points).
7) Questions Subsidiaires



Voici un exemple d'exécution du programme

Votre choix? … = 1

Nbr Etudiants a saisir = **3**

Son nom ? : L1
Note1 ?: 10
Note2 ?: 15
Son sexe ? : F

Son nom ? : L2
Note1 ?: 11
Note2 ?: 12
Son sexe ? : M

Son nom ? : L3
Note1 ?: 15
Note2 ?: 16
Son sexe ? : F

Affichage Etudiants ...

L1	L2	L3
10	11	15
15	12	16
F	M	F

(insertion en fin)

Votre choix? … = 2

Nbr Etudiants a saisir = **3**

Son nom ? : V1
Note1 ?: 19
Note2 ?: 0
Son sexe ? : M

Son nom ? : V2
Note1 ?: 15
Note2 ?: 2
Son sexe ? : M

Son nom ? : V3
Note1 ?: 18
Note2 ?: 3
Son sexe ? : F

Affichage Etudiants ...

V3	V2	V1
18	15	19
3	2	0
F	M	M

(insertion au début)

Votre choix? … = 3

Le nombre de filles = 2

Votre choix? … = 4

Position dans la liste = **9**
La position doit etre <= 3.

Position dans la liste = **2**
Moyenne = 11,5

Votre choix? … = 5

Nom : V2
Sexe : F

L'Etudiant n'est pas dans **List**.
L'Etudiant n'est pas dans **Vector**.

Votre choix? … = 5

Nom : V2
Sexe : M

L'Etudiant n'est pas dans **List**.
L'Etudiant est dans **Vector**.
Position = 2.

Votre choix? … = 6

Tableau Dynamique :

L1	L3	V3
10	15	18
15	16	3
F	F	F

List :

L2
11
12
M

Vector :

V2	V1
15	19
2	0
M	M

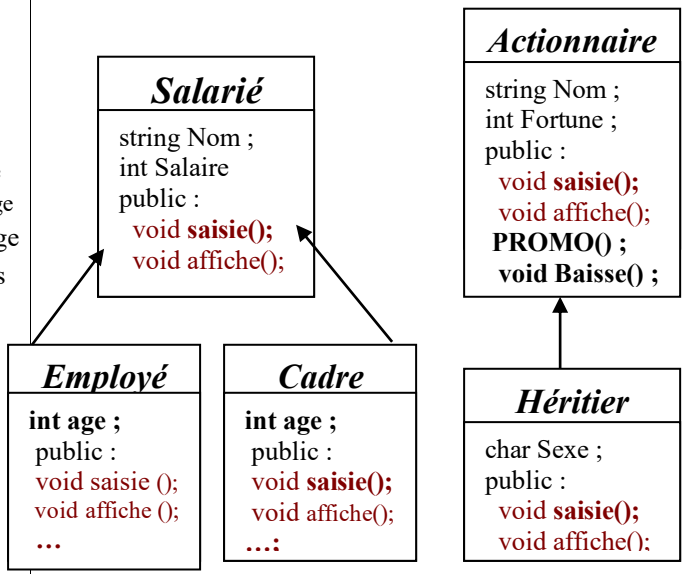
L'entreprise

Écrivez les différentes classes: les classes de base ***Actionnaire***, ***Salarié*** et ses classes dérivées ***Employé et Cadre.*** Si nécessaire, des fonctions membres peuvent être ajoutées aux classes.

Ce programme doit gérer en boucle le Menu suivant :

1) Saisie d'1 **liste** d'Employés (**insertion en Fin**) + Affichage
2) Saisie d'1 **Vector** de Cadres (insertion au **Début**)+ Affichage
3) Créer Dynamiquement l'Actionnaire (1 seul)+Affichage
4) L'Actionnaire décide de promouvoir 2 Employés en Cadres (Fct membre : PROMO() et le choix : les 2 plus âgés) **(Commentez les fonctions qui participent a ce point 4)**
5) L'Actionnaire décide (Fct membre : Baisse()) de diminuer les salaires des Employés de 5. (Donc modification des salaires de la liste) Afficher les modifications.
6) Grève (une fonction) et l'Actionnaire est remplacé par ses héritiers. Sa fortune est partagée à part égale.

On suppose que cette modélisation est juste.



Voici un exemple d'exécution du programme

Votre choix? … = 1

Nbr Employés a saisir = 4

Son nom ? : E1
Son Salaire?: 100
Son Age ? : 30

Son nom ? : E2
Son Salaire?: 102
Son Age ? : 35

Son nom ? : E3
Son Salaire?: 80
Son Age ? : 20

Son nom ? : E4
Son Salaire?: 105
Son Age ? : 38

Affichage Employés ...

E1	E2	**E3**	E4
100	1102	80	105
30	35	20	38

Votre choix? … = 2

Nbr de Cadre a saisir = 2

Son nom ? : **C1**
Son Salaire?: 130
Son Age ? : 35

Son nom ? : **C2**
Son Salaire?: 150
Son Age ? : 40

Affichage des Cadres ...

C2	**C1**
150	130
40	35

C2 avant C1 (insert début)

Votre choix? … = 3

Nom Actionnaire : Coco
Fortune : 1000000

Affichage :
Nom actionnaire : Coco
Sa Fortune : 1000000

Votre choix? … = 4

PROMO() de l'actionnaire ➔

Affichage des Employés :

E1	**E3**
100	80
30	20

Affichage des Cadres :
Par exemple :

E4	E2	**C2**	**C1**
105	1102	150	130
38	35	40	35

Votre choix? … = 5

Votre choix? … = 6

…

RQs :

Optimisez et réutilisez votre code. Utilisez les règles de bonnes pratiques étudiées durant les travaux pratiques telles que l'indentation syntaxique, la programmation modulaire, libération de mémoire.