Alexandre

Le Bourch

Vendredi 2 Juin 2023

ROBOTIQUE & INGÉNIERIE SYSTÈMES

Rapport de Projet C++ :

Projet Arbre Généalogique

#define Introduction:

Lors du cours de C++ avons été amenés à réaliser un projet en C++ QT. Le but était de réunir toutes les connaissances acquises pendant le cours pour en faire un projet complexe. Dans mon cas j'ai choisi de faire un projet d'arbre généalogique, le but est de pouvoir créer des personnes et établir des liens entres elles. Une fois qu'un arbre est créé, nous pourrons demander la généalogie d'une personne en particulier, demander les parents, frère, sœurs et enfants d'une personne

Mon but va être de créer un projet ordonné et lisible, je vais devoir créer et organiser mes classes de manière à ce que cela soit compréhensible.

Pour commencer j'ai créé la classe 'Personne' qui va nous permettre de créer des gens pour ensuite créer des liens. J'ai ajouté à cette classe 11 méthodes pour créer des liens entres les personnes.

Ensuite J'ai créé une classe 'Couple' qui va nous permettre de créer des couples de personne. Ainsi au lieu d'ajouter des enfants de manière individuelle à chaque personne, on pourra ajouter un enfant au couple directement, ou bien de pouvoir demander les parents d'un enfant au lieu de demander le père puis la mère. Cette classe contient 5 méthodes pour réaliser les actions citées ci-dessus.

Etant seul pour réaliser ce projet, je me contenterais d'un affichage textuel et non graphique, cela me prendrait beaucoup plus de temp pour avoir un résultat satisfaisant, cependant je souhaite que toute mes fonctions puissent être afficher facilement et de manière récursive.

#include 1.La classe Personne :

Pour créer un objet de la classe Personne, il faut lui fournir quelques informations pour les attributs de base :

- m_nom de type QString
- m_prenom de type QString
- sexe de type Sexe
- birth de type QString
- m_pere pointeur d'un objet de type Personne
- m_mère pointeur d'un objet de type Personne
- death de type QString

On remarque que m_mère et m_pere sont des pointeurs vers des objets de type Personne, cela nous permet de relier les objets entre eux et de naviguer dans la structure de la généalogie. Le type Sexe a été définit dans le header de la classe :

```
enum Sexe { Masculin, Feminin };
```

Ces attributs nous permettent de définir les caractéristiques de la personne. Certains attributs comme death, m_mere et m_pere ne sont pas nécessaires pour la création de l'objet. Pour créer une Personne il faudrait donc écrire :

```
Personne *Titouan_Dupont = new Personne("Titouan", "Dupont", Personne::Masculin, "01/01/1950");
```

C'est bien nous avons une base pour commencer l'ajout des différentes méthodes, les voici :

- `QString toString();`
- `void ajouterEnfant(Personne *enfant);`
- `QList<Personne*> getEnfants();`
- `void setPere(Personne *pere);`
- `Personne* getPere();`
- `void setMere(Personne *mere);`
- `Personne* getMere();`
- `QList<Personne*> getFreresSoeurs();`
- `QString Genealogie();`
- `void ajouterFrereSoeur(Personne* frereSoeur);`
- `QString afficheFrerSoeurs ();`

La majorité de ces méthodes font simplement une assignation de valeurs ou de l'affichage de valeurs peu complexes, hormis les fonctions Genealogie(), ajouterFrereSoeur() et afficheFrerSoeurs():

- La méthode Genealogie() va afficher tous les membres de l'arbre (3 générations) au-dessus de la personne sélectionnée ex : `cout << enfant1Dupont2->Genealogie().toStdString() << endl;`

- La méthode ajouterFrereSoeur() va vérifier que les deux personnes que l'on souhaite lier ont le même père sinon elle ne les lies pas

- La méthode afficheFrerSoeurs() va afficher tous les frère et sœurs de la personne sélectionnée en utilisant une boucle for.

#include 2.La classe Couple :

Pour créer un objet de la classe Couple, il faut simplement lui fournir les deux membres du couple, soit 2 objets Personne. L'initialisation d'un couple ressemble à ça :

```
Personne *pereDupont = new Personne("Jean", "Dupont", Personne::Masculin, "01/01/1950");
Personne *mereDupont = new Personne("Marie", "Dupont", Personne::Feminin, "01/01/1955");
Couple *couple1 = new Couple(pereDupont, mereDupont);
```

Comme Pour la classe Personne nous utilisons des pointeurs pour les deux membres du couple :

- homme pointeur d'un objet de type Personne
- femme pointeur d'un objet de type Personne

La classe Couple comporte 5 méthodes :

- `void ajouterEnfant(Personne* enfant);`
- `QList<Personne*> getEnfants();`
- `Personne* getHomme();`
- `Personne* getFemme();`
- `QString toString();`

Comme pour la classe Personne quasiment toutes ces méthodes font de l'assignation ou de l'affichage de valeurs peu complexe, hormis les méthodes ajouterEnfant() et toString() qui sont plus complexe :

```
void Couple::ajouterEnfant(Personne* enfant)
{
m_enfants.append(enfant);
enfant->setPere(m_homme);
enfant->setMere(m_femme);
}
```

En plus d'ajouter l'enfant à la liste des enfants du couple, cette méthode va lier les deux parents et l'enfant en faisant un setpere() et un setMere() des Personnes déclarées comme couple.

```
QString Couple::toString() const
{
     QString result = "Couple :\n";
     result += "  Membre 1 : " + m_homme->toString() + "\n";
     result += "  Membre 2 : " + m_femme->toString() + "\n";
     result +="Enfants du couple : \n" ;
     for (Personne* enfant : m_enfants) {
          result +="  - " + enfant->toString() + "\n";
     }
     return result;
}
```

Cette méthode va afficher les deux Personnes qui forment le couple, mais aussi tous les enfants reliés à ce couple grave à une boucle for.

#include 3.Le main.cpp :

Le fichier main.cpp est le générateur de l'arbre, c'est dans ce fichier que sont stockés les ordres de création de personne, de couple et de relations. C'est aussi dans ce fichier que l'on demande d'afficher les relations. Le but de ce fichier et de réunir toutes les fonctionnalités créées dans les classes pour les faire fonctionner ensemble.

```
Personne *pereDupont = new Personne("Jean", "Dupont", Personne::Masculin, "01/01/1950");
Personne *mereDupont = new Personne("Marie", "Dupont", Personne::Feminin, "01/01/1955");

Couple *couple1 = new Couple(pereDupont, mereDupont);

Personne *enfant1Dupont = new Personne("Paul", "Dupont", Personne::Masculin, "01/01/1980");
Personne *enfant2Dupont = new Personne("Sophie", "Dupont", Personne::Feminin, "01/01/1985");

couple1->ajouterEnfant(enfant1Dupont);
couple1->ajouterEnfant(enfant2Dupont);
enfant1Dupont->ajouterFrereSoeur(enfant2Dupont);

Personne *pereSmith = new Personne("John", "Smith", Personne::Masculin, "01/01/1950");
Personne *mereSmith = new Personne("Emily", "Smith", Personne::Feminin, "01/01/1955");

Couple *couple2 = new Couple(pereSmith, mereSmith);

Personne* enfant1Smith = new Personne("arthur", "Smith", Personne::Masculin, "01/01/1980");
Personne* enfant2Smith = new Personne("louise", "Smith", Personne::Feminin, "01/01/1985");

couple2->ajouterEnfant(enfant1Smith);
couple2->ajouterEnfant(enfant2Smith);
enfant1Smith >ajouterFrereSoeur(enfant2Smith);

Personne *enfant1Dupont2 = new Personne("toto", "Dupont", Personne::Masculin, "01/01/1980");

Couple* coupleDupont2 = new Couple(enfant1Dupont, enfant2Smith);

coupleDupont2->ajouterEnfant(enfant1Dupont2);
```

Cette première partie du code initialise tout l'arbre généalogique sur 3 générations, en n'oubliant aucune des relation possibles (possible dans le code, donc pas de relation de cousin etc). On remarque que comme indiqué précédemment, à chaque création de personne il n'y a pas d'attribut pour m_pere, m_mère ou death. Selon moi, mon but de rendre le code ordonné et lisible est accomplie, car dans le code ci-dessus on peut clairement comprendre la création de Personne, couple et l'ajout d'enfant.

```
cout << couple1->toString().toStdString() << endl;

cout << enfant1Dupont->afficheFrerSoeurs().toStdString() << endl;

cout << couple2->toString().toStdString() << endl;

cout << coupleDupont2->toString().toStdString() << endl;

cout << enfant1Dupont2->Genealogie().toStdString() << endl;
```

Les 5 lignes de code ci-dessus nous permettent d'afficher toutes les informations que l'on voit sur l'image ci-contre. Cette image est le résultat du main.cpp.

```
Couple :
  Membre 1 : M. Dupont Jean
  Membre 2 : Mme Dupont Marie
Enfants du couple :
  - M. Dupont Paul
  - Mme Dupont Sophie

Frères/soeurs de M. Dupont Paul :
  - Mme Dupont Sophie

Couple :
  Membre 1 : M. Smith John
  Membre 2 : Mme Smith Emily
Enfants du couple :
  - M. Smith arthur
  - Mme Smith louise

Couple :
  Membre 1 : M. Dupont Paul
  Membre 2 : Mme Smith louise
Enfants du couple :
  - M. Dupont toto

généalogie de M. Dupont toto :
- Mère :Mme Smith louise
- Père :M. Dupont Paul
-- Grandpère Paternel : M. Dupont Jean
-- Grandmère paternel : Mme Dupont Marie
-- Grandpère maternel : M. Smith John
-- Grandmère maternel : Mme Smith Emily
```

#include 4.Conclusion :

Ce projet n'est pas totalement complet, il y a pleins d'améliorations possibles, en voici quelques-unes :

- Pouvoir exporter et importer un arbre généalogique avec un fichier style .json
- Pouvoir générer un rapport en fichier texte
- Créer une interface graphique pour visualiser et interagir avec le code
- Améliorer des fonctionnalités déjà existantes comme la méthode Genealogie() qui n'affiche la généalogie d'une personne que sur 3 générations
- Pour rendre le code encore plus lisible il faudrait changer les noms des personnes en mettant leurs initiales plutôt que pere_dupont etc.

Ce projet a été très intéressant car il regroupe plusieurs sujets que nous avons abordé pendant le cours de C++, d'autant que depuis longtemps je souhaite réaliser un arbre généalogique en code.

Le Résultat final :

```
Couple :
  Membre 1 : M. Dupont Jean
  Membre 2 : Mme Dupont Marie
Enfants du couple :
  - M. Dupont Paul
  - Mme Dupont Sophie

Frères/soeurs de M. Dupont Paul :
  - Mme Dupont Sophie

Couple :
  Membre 1 : M. Smith John
  Membre 2 : Mme Smith Emily
Enfants du couple :
  - M. Smith arthur
  - Mme Smith louise

Couple :
  Membre 1 : M. Dupont Paul
  Membre 2 : Mme Smith louise
Enfants du couple :
  - M. Dupont toto

généalogie de M. Dupont toto :
- Mère :Mme Smith louise
- Père :M. Dupont Paul
-- Grandpère Paternel : M. Dupont Jean
-- Grandmère paternel : Mme Dupont Marie
-- Grandpère maternel : M. Smith John
-- Grandmère maternel : Mme Smith Emily
```