Niveau 4

TP
Champ Mots-clés : thème du référentiel
S4.1. Principes de base : Gestion mémoire :... pointeurs...
Organisation des programmes : ...fonctions, paramètres, valeur de retour...

Théa, Léo, Phil ou Laurent ? Telle est la fonction...

TABLE DES MATIÈRES

1 - Présentation....2
1.1 - Ajout de procédures et fonctions....2
2 - Travail à réaliser....4

CERTAINS DROITS RESERVES

1 - PRÉSENTATION.

Ce TP a pour but de mettre en applications les principes de base concernant le choix entre procédure et fonction et entre les trois passages de paramètres possibles : par valeur, par adresse ou par référence.

1.1 - AJOUT DE PROCÉDURES ET FONCTIONS.

Une fonction ou procédure doit toujours être définie avant de pouvoir l'utiliser. C'est pour cela qu'elle est placée avant le `main()`.

Mais pour éviter d'avoir un seul fichier `.cpp` avec tous nos modules (procédure ou fonction) en premier et le `main()` en dernier, il convient de séparer ces modules du programme principal et de les mettre dans un (ou plusieurs) autre(s) fichier(s) `.cpp`. Ces fichiers sont appelés fichiers **Application**.

Les modules seront ainsi regroupés dans un même fichier s'ils font partie d'un même thème fonctionnel. Par exemple, tous ceux traitant des tableaux une dimension pourront être mis dans le fichier `Tableaux_1D.cpp`.

Ce fichier **application** `.cpp` est à intégrer au projet Qt. Le compilateur connaît alors la **définition** du (ou des) module(s) qu'il contient !

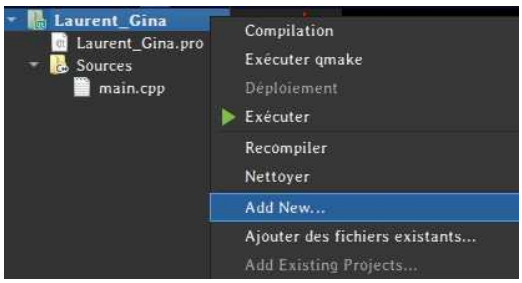
Par contre, on ne sait pas si le compilateur va commencer par compiler ce fichier avant le programme principal (ce qui devrait être fait pour que le `main()` puisse utiliser un des modules !).

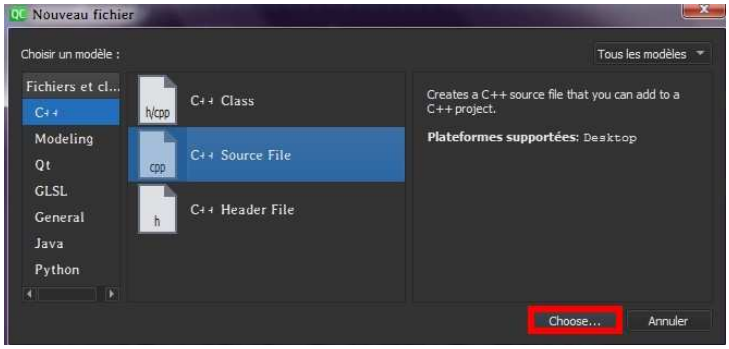
Pour résoudre ce problème, et ainsi s'affranchir de l'ordre de compilation des fichiers, on doit **déclarer** que la fonction existe. Ça se fait grâce à un fichier d'**En-tête**. Ce second fichier est associé au fichier **application** `.cpp` contenant les modules en portant le même nom que le `.cpp` mais d'extension `.h` (pour ***Header***).

Le fichier `.h` ne contiendra que les **déclarations** des modules, ce que l'on appelle le **prototype** du module. Il s'agit de la première ligne de la **définition**, celle qui indique le nom, le type de retour et les éventuels paramètres du module. Dans le fichier `.h`, le **prototype** est toujours finalisé par le **point-virgule** (" `;` ")... mais pas dans le `.cpp`, où il est suivi par l'accolade ouvrante (" `{` ") de **définition** du corps du module.

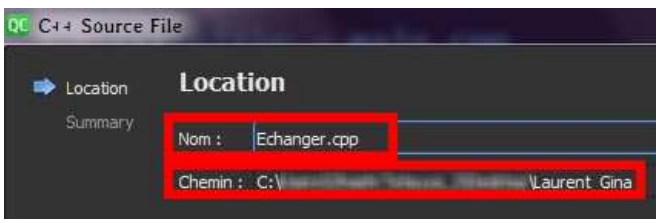
Lorsqu'on souhaite utiliser un module qui se trouve dans un autre fichier **application**, il faut **inclure** la **déclaration** de ce module en début du fichier qui l'utilise. Par exemple dans le `main.cpp`, il faudra mettre un `#include` pour le fichier d'**en-tête** (`.h`) du module.

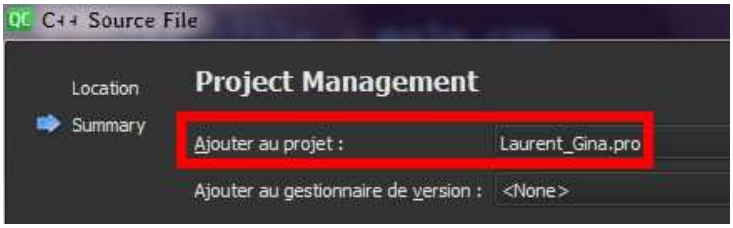
Ne reste qu'à les créer et déclarer dans le projet de compilation qu'il faut utiliser ces fichiers en plus du `main.cpp`. Pour ça, une fois que le nouveau projet est créé, il suffit de cliquer (clic droit) sur le nom du projet dans Qt Creator. On sélectionne alors l'option "`Add new...`" du menu contextuel qui nous ouvre la fenêtre suivante :



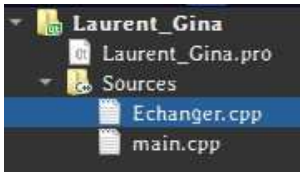


Dans le modèle "C++", on sélectionne "C++ Source File" puis on valide avec le bouton "Choose...". Les fenêtres qui s'affichent nous permettent de définir le nom du fichier .cpp, de sélectionner son emplacement sur le disque et finalement de l'inclure au projet.

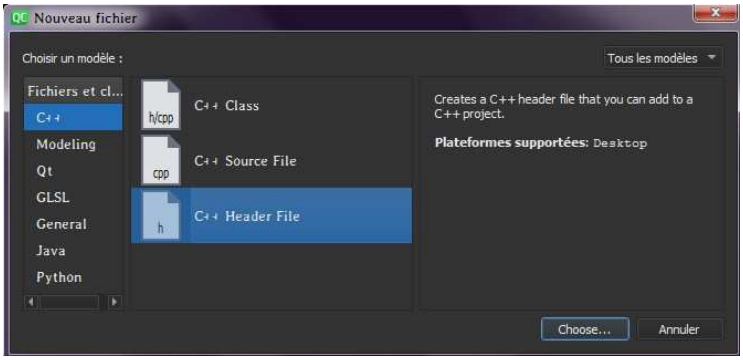




Le nouveau fichier Echanger.cpp est créé sur le disque au même niveau que le main.cpp puisque j'ai sélectionné la racine de mon projet Laurent_Gina. Il est automatiquement inséré dans le projet Qt :

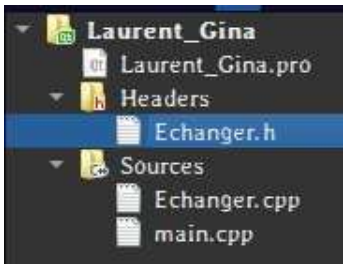


On réitère toute l'opération pour ajouter le fichier d'en-tête associé. Mais ici, on sélectionne l'option "C++ Header File" :



On donne le même nom que le fichier .cpp associé, en remplaçant .cpp par .h ! Ça n'est pas obligatoire, mais ça aide réellement à nous y retrouver dans tous les fichiers si on a un gros projet...

Le fichier Echanger.h est aussi à la racine du dossier du projet et est intégré au projet Qt :



Normalement, les deux fichiers sont vides. On peut tester le projet en validant la compilation !

À vous de remplir chaque fichier selon vos besoins.

Vous pouvez même créer plusieurs paires .cpp/.h pour mieux répartir vos modules selon des thématiques plus précises et avoir au final des projets gérant plusieurs dizaines de fichiers !

> Gare ! Même si c'est le cas pour ce premier TP de prise en main des bonnes pratiques à adopter pour le procédures et fonctions, il est rare de n'avoir qu'un seul module par fichier.

2 - TRAVAIL À RÉALISER

Il vous est demandé, pour chacune des problématiques proposées, de rédiger le programme principal appelant la fonction/procédure puis dans les fichiers `.cpp` et `.h` approprié de coder cette procédure/fonction.

L'architecture des répertoires et les commentaires dOxygen sont respectés !
Pensez à bien créer vos fichiers `.cpp` et `.h` dans les bons dossiers dès le départ !

Dans un nouveau ficher d'en-tête `.h`, il vous faudra mettre le **prototype** (ou **déclaration**) de la procédure/fonction, et dans un fichier `.cpp`, du même nom, la **définition** de la procédure/fonction. Le programme principal `main()`, restant toujours seul dans un fichier nommé `main.cpp`.

Par exemple, pour le premier problème :

Main.cpp

```
#include "Echanger.h"

int main(…)
{
   …
   Echanger(…) ;
   …
}
```

Echanger.h ***(fichier en-tête)***

```
// Déclaration du module
void Echanger(…) ; // Prototype avec ;
```

Echanger.cpp ***(fichier application)***

```
// Définition du module
void Echanger(…)   // Prototype sans ;
{
   … // Code du module
}
```

1) Monsieur Gina doit réaliser un module logiciel nommé `Echanger(…)`, prenant deux paramètres, et devant échanger les valeurs contenues dans les paramètre réels passés pour les afficher ensuite dans le programme principal. Seuls problèmes : doit-il réaliser une procédure ou une fonction ? Et surtout quel type de passage de paramètres doit-il utiliser ? Laurent est tellement secoué par le problème qu'il vous demande conseil ! Réalisez ce module pour lui !

2) Phil Trahuile, le joueur de tiercé du TP qui va avec, n'est pas satisfait du programme que vous lui avez développé : Il trouve que le programme a trop de ligne de codes qui se répètent ! Le calcul de factorielle notamment. Reprenez cet exercice et faites-lui une version utilisant une procédure/fonction `Factorielle(…)` de votre cru !

3) Théa Louest (prof d'un précédent TP) a toujours des problèmes avec ses moyennes : elle n'arrive pas à remettre son tableau de note à 0 en début de chaque trimestre ! Préparez-lui une procédure/fonction `InitTab1D(…)` qui lui remettra toutes les cases du tableau passé en paramètre à 0. Et quitte à y être, proposez-lui un autre module pour l'affichage du tableau !

4) Le Docteur Parre doit faire une étude poussée sur le nombre de taches présentes sur le pelage de certains fauves. Léo sait que la série de **Fibonacci** peut s'appliquer, mais il lui manque un outil de calcul des nombres de cette suite ! Il a besoin d'un module logiciel, réutilisable à volonté, `Fibo(…)`, qui fournit les deux valeurs suivantes aux deux valeurs passées en paramètre.

   Pour information, un membre de la suite de Fibonacci se calcule en faisant la somme des deux membres qui le précèdent.
   Le début de la suite est : 0, 1, 1 (=0+1), 2 (=1+1), 3 (=1+2), 5 (=2+3), 8 (=3+5), 13 (=5+8), 21 (=8+13)...
   Les deux premiers membres (0 et 1) sont immuables et ne sont pas calculés !
   On n'étudiera pas les valeurs négatives de la suite de Fibonacci !
   Par contre, il faudra vérifier que les deux valeurs fournies font bien partie de la suite de Fibonacci !