AP1 TP

7 Semaine 7 : Tableaux 2/2

L'objet de ce TP est de découvrir la **compilation séparée** et ses intérêts, en l'illustrant à travers l'implémentation de l'ensemble des fonctions vues en Cours et TD de **manipulation et mise à jour de tableaux, triés ou non**.
Pour que l'apport pédagogique soit le plus efficace possible, il vous est demandé de suivre les instructions données dans ce TP **dans l'ordre**.

Exercice 1 : Découpage du programme et Compilation

Le programme qui vous est fourni ci-après (et qui vous sera demandé de compléter) propose de manipuler un tableau, trié ou non, à l'aide du menu suivant :

```
 0. Sortir
 1. Saisie d'un tableau quelconque
 2. Saisie d'un tableau en le triant
11. Recherche séquentielle d'un élément
12. Supression d'un élément
13. Ajout d'un élément
21. Recherche séquentielle d'un élément dans un tableau trié
22. Recherche dichotomique d'un élément
23. Suppression d'un élément dans un tableau trié
24. Ajout d'un élément dans un tableau trié
```

Cela représente un nombre de fonctions très conséquent ; les mettre toutes dans un même fichier devient problématique pour le programmeur... Pour pouvoir s'y retrouver, il devient ici indispensable de découper le programme en plusieurs fichiers.
Nous allons voir à travers cet exemple comment procéder pour découper un programme en plusieurs fichiers, puis comment le compiler.

1. Recopiez dans votre répertoire tous les fichiers se trouvant dans `/net/Bibliotheque/AP1/TPSem07`.
2. Vous y trouverez 2 fichiers `.cc` :
   - `Main.cc`, qui contient la fonction `main`, ainsi qu'un ensemble de fonctions liées à la gestion du menu,
   - `Tableau.cc`, qui contient l'ensemble des fonctions de manipulation des tableaux, triés ou non, vues en Cours et TD.

   On ne demande pas de regarder en détails ces fichiers pour l'instant, seulement les grandes lignes.
3. Il est possible de compiler le programme avec la ligne de commande suivante. Testez la.

```
g++ Main.cc Tableau.cc -o tabTest
```

OUI MAIS... Pourquoi est-ce que cette compilation fonctionne ? Lors du traitement du fichier `Main.cc`, comment le compilateur fait-il pour "connaitre" les fonctions définies dans le fichier `Tableau.cc` et utilisées dans `Main.cc` ?
La bonne solution est de créer un fichier "commun" aux deux fichiers `Main.cc` et `Tableau.cc`. On appelle généralement ce type de fichier commun un fichier d'en-tête, on l'identifie en le notant `nom_fichier.h` (ici `Tableau.h`), et on l'utilise en l'incluant là où il faut grâce à la directive `#include "nom_fichier.h"`.

1. Consultez le contenu du fichier d'en-tête `Tableau.h`. Que contient-il ?
2. Vérifiez la présence de la directive d'inclusion de ce fichier dans le fichier `Main.cc`. Mettez la en commentaire, et voyez les messages d'erreur provoqués par son absence lors de la compilation. Remettez en état après ce test.

Exercice 2 : Précisions sur la compilation

On rappelle les trois étapes de la compilation :

1. le prétraitement du programme par le pré-processeur ;
2. la construction des fichiers objets : cette étape génère les `.o` ;
3. l'édition des liens : cette étape assemble les `.o` pour créer l'exécutable.

En compilant avec la ligne de commande `g++ Main.cc Tableau.cc -o tabTest`, le programme `tabTest` est obtenu en assemblant les fichiers `Main.o` et `Tableau.o`, mais ces fichiers `.o` sont uniquement créés en mémoire, pas sur disque.
En compilant séparément les fichiers avec les lignes de commandes ci-dessous

```
g++ -c Tableau.cc
g++ -c Main.cc
g++ Main.o Tableau.o -o tabTest
```

on crée physiquement sur disque les fichiers `.o` , puis on les assemble pour créer l'exécutable.

1. Testez ces lignes de commandes de compilation séparée.
2. Admettons que vous modifiez le fichier `Main.cc`, quelle(s) ligne(s) de commande de compilation devez-vous réexécuter pour obtenir le nouvel exécutable de votre programme ?

Exercice 3 : Make et Makefile

Il existe une commande `make` qui permet de faciliter la compilation d'un programme réparti sur plusieurs fichiers. La commande `make` recherche dans le répertoire de travail un fichier de nom `Makefile` contenant la description modulaire (c.a.d une description du découpage) du programme. Le fichier `Makefile` du programme `tabTest` doit contenir les informations suivantes :

➤ `tabTest` est construit avec `Main.o` et `Tableau.o`

➤ `Tableau.o` est construit avec `Tableau.cc` et `Tableau.h`

➤ `Main.o` est construit avec `Main.cc` et `Tableau.h`

Dans le "langage de `make`" cela s'écrit :

```
CC=g++

OBJECTS= Tableau.o Main.o

TARGET= tabTest

$(TARGET) : $(OBJECTS)
        $(CC) $(OBJECTS) -o $(TARGET)

Tableau.o : Tableau.cc Tableau.h
        $(CC) -c Tableau.cc

Main.o : Main.cc Tableau.h
        $(CC) -c Main.cc
```

Les trois premières lignes sont des définitions de variables.

➤ La variable `CC` est une variable réservée contenant le nom de la commande de compilation.

➤ La variable `OBJECTS` est initialisée avec la liste des fichiers objet du programme.

➤ La variable `TARGET` est initialisée avec le nom du programme exécutable.

Les lignes suivantes sont à lire deux par deux et comportent :

➤ une spécification de dépendance, par exemple
`Main.o : Main.cc Tableau.h`,
qui indique de "quoi dépend" le `.o` correspondant (appelé cible)

➤ une action associée à cette spécification de dépendance (ici `$(CC) -c Main.cc`) qui décrit l'action à exécuter pour construire le fichier cible (ici `Main.o`).

➤ Les “espaces” précédant la commande sont en fait des **tabulations** !

Un `Makefile` contenant les commandes ci-dessus vous a été fourni dans le répertoire de départ.

1. Avec quelle ligne de commande pourrait-on traduire la ligne `$(CC) $(OBJECTS) -o $(TARGET)` ?
2. Effacez tous les fichiers `.o` de votre répertoire.
3. Lancez la commande `make`.
4. Utilisez la commande `ls -l` et notez, pour chaque fichier `.o` la date de création.
5. Modifiez le fichier `Tableau.cc` (sans introduire d'erreur ...) et relancez la compilation avec `make`. Utilisez la commande `ls -l` et notez, pour chaque fichier `.o` la date de création. Que constatez-vous ?
6. Notez ce que sont, à votre avis, les divers avantages de la compilation séparée. Et les inconvénients ?

make clean Pour éviter le gaspillage de l'espace disque, et favoriser en même temps une bonne organisation de vos répertoires, il est bon de ne conserver que

➤ les fichiers sources (`.cc` et `.h`)

➤ le fichier `Makefile`.

Vous pouvez automatiser le nettoyage de votre répertoire en rajoutant dans le `Makefile` une cible qui ne dépend de rien et à laquelle est associée la commande `rm` avec les arguments appropriés. Il suffit ensuite de taper la commande

```
make nom_cible
```

pour supprimer les fichiers indésirables.

Remarque : traditionnellement, cette cible est nommée `clean`. Elle a été définie dans le fichier `Makefile` fourni ici. Utilisez la pour faire du ménage de temps en temps.

Exercice 4 : Corriger le programme de manipulation d'un tableau

Le programme fourni compile, mais n'est pas correct... Des erreurs se sont glissées dans certaines fonctionnalités

1. Fonctionnalité `2. Saisie d'un tableau en le triant`
   Testez-la et découvrez ses erreurs.
   C'est la fonction `saisirTableauTrie` qui est fausse, pourquoi ? Corrigez la.
2. Fonctionnalité `24. Ajout d'un élément dans un tableau trié`
   Testez-la et découvrez ses erreurs.
   Pourtant la fonction `ajouterTrie` utilisée aussi dans la fonctionnalité précédente semble correcte... Donc, où se trouve l'erreur ?
3. Fonctionnalité `22. Recherche dichotomique d'un élément`
   L'implémentation de la fonction `cherchePosDicho` semble erronée, cherchez l'erreur !

Les autres fonctionnalités sont correctes ou restent à faire. Consultez bien l'ensemble du code avant de continuer.

Exercice 5 : Compléter le programme de manipulation d'un tableau

Rajoutez une à une les fonctionnalités manquantes dans le programme. Dans un soucis pédagogique, respectez **rigoureusement l'ordre** de traitement proposé ci-dessous.

1. Fonctionnalité `12. Supression d'un élément`
2. Fonctionnalité `21. Recherche séquentielle d'un élément dans un tableau trié`
3. Fonctionnalité `23. Suppression d'un élément dans un tableau trié`
4. Amélioration : la fonction `cherchePosAjout` a été écrite en adaptant un algorithme de recherche séquentielle. Or, étant donné que le tableau dans ce cas est trié, il serait sans doute plus judicieux de faire cette recherche de position par dichotomie.
   Adaptez l'algorithme de recherche dichotomique pour proposer une nouvelle version de `cherchePosAjout`.