TP QCM

Programmation C++ Linux

Lycée Diderot, Paris

BTS SNIR 1

TABLE DES MATIERES

TABLE DES ILLUSTRATIONS

INTRODUCTION

Tout au long des deux années qui suivent, nous allons développer nos programmes dans un langage compilé, le **C++**. Pourquoi, parce que c'est celui choisi dans le référentiel du **BTS SN option IR** et qu'il présente une **orientation objet**. Il existe pléthore de langages informatiques, chacun présentant ses avantages et ses inconvénients. Celui-ci n'est ni pire ni meilleur. Le but du jeu est d'acquérir, tout au long des TP proposés, une méthode de pensée et de conception. Passer sur un autre langage ne posera alors que des problèmes de syntaxes, pas de philosophie.

ATTENTION

A chaque séance, vous devrez faire une sauvegarde de votre travail sur Github avec Github Desktop. Ne pas le faire vous vaudra un 0.

Ne pas m'appeler pour valider les différentes étapes vous vaudra un 0.

FORMAT D'UN FICHIER SOURCE EN C++

Un code source peut être constitué de plusieurs **fichiers source** (**.cpp**) et également de plusieurs **fichiers d'en-tête** (**.h**). Il doit avoir obligatoirement la fonction principale **main** pour pouvoir créer un exécutable. Je vous donne un code source simple, le fameux **hello world** que tout débutant en informatique se doit de taper.

```
#include <iostream>
using namespace std;
int main ()
{
    cout<< "Bonjour à vous!" <<endl;
    return 0;
}
```

Figure 1: Structure d'un programme en C++

Ce code est constitué de plusieurs **instructions** et **directives de compilation**. Les instructions sont repérées par le point-virgule qui les termine. Les directives de compilation sont repérées par le dièse qui les précède.

Ici, on commence par une directive, on dit au compilateur que l'on utilise la librairie **iostream** dans notre programme. Elle comprend la définition de l'opérateur **cout** qui permet d'afficher un message à l'écran.

On retrouve également la fonction **main** indispensable à tout code source.

On sauve ce code dans un fichier, par exemple **bonjour.cpp**, qui nous permettra, une fois compilé, d'avoir un fichier exécutable affichant "Bonjour à vous!".

Comme dit plus haut, le **C++** est un langage compilé ce qui veut dire que l'on a besoin d'un compilateur pour transformer notre code source en suite d'instructions simples pour le processeur.

LES ETAPES DE LA COMPILATION

Un langage informatique est un langage précis, avec sa grammaire et ses règles. On observe deux types principaux de langages: le compilé et l'interprété. Un langage dit compilé est un langage dont le code doit subir une transformation et ainsi obtenir un exécutable. A contrario, un langage interprété sera exécuté au fur et à mesure de la lecture du code source.

Le **C++** est un langage compilé. Il faut donc un compilateur c'est à dire un programme capable de traduire le langage informatique (autrement dit humain) en langage machine.

g++ est le compilateur de base du **C++** (pour le **C**, le compilateur se nomme **gcc**). Il évolue en permanence. Certaines choses impossibles il y a seulement 6 mois peuvent l'être aujourd'hui. Cela ne veut pas dire que l'on codait mal 6 mois plus tôt, juste que le compilateur est modifié en vue de simplifier la vie du développeur.

Nous allons donc utiliser **g++** pour compiler nos futurs codes source. Sa syntaxe est simple, nous utiliserons celle qui suit:

```
g++ [fichiers source à compiler] -o [nom de l'exécutable]
```

Sous **Linux**, un exécutable ne porte pas forcément l'extension **.exe** propre à **Microsoft**. Dans la syntaxe ci-dessus, on utilise une option de compilation, le **-o**. Elle est utilisée pour permettre de donner un nom à l'exécutable généré. Si vous ne l'utilisez pas, vous aurez un exécutable répondant au doux nom de **a.out**. Il existe d'autres options, si vous êtes curieux, tapez à l'invite de commande **man g++** et vous obtiendrez la page de manuel concernant le compilateur avec toutes ses options.

Une option également utile pour nous est le **-c**. On oblige notre compilateur à seulement compiler notre code source. Comment ça seulement compiler, me direz-vous? Depuis le début, je dis que la compilation génère un exécutable. En fait, pas tout à fait. La compilation telle qu'on la conçoit se déroule en deux étapes: la compilation et l'édition de lien.

La compilation est en fait l'étape qui va vérifier que votre code source est bien écrit et qui va le traduire dans le langage machine. Si vous n'exécutez que cette étape, vous obtiendrez de votre compilation un fichier objet, autrement dit, un fichier binaire traduisant l'ensemble de vos instructions en **C++** et instructions simples pour le processeur.

Pour obtenir un exécutable, il faut éditer les liens. C'est à dire que votre code source se base sur de l'existant, vous n'inventez pas (pour l'instant), vous utilisez des fonctionnalités déjà connues. Il faut donc que le compilateur intègre ces fonctionnalités issues de librairies dans votre code afin d'obtenir un exécutable.

Qui dit deux étapes dit deux origines d'erreur, celle de compilation et celle d'édition de lien. Les erreurs de compilation concerneront uniquement les erreurs de syntaxe de votre code source (ce qui ne veut pas dire que c'est simple à déboguer). Les erreurs d'édition de liens concerneront l'intégration des liens, à savoir par exemple, une fonction manquante ou carrément un fichier objet manquant (là non plus, ce n'est pas simple à déboguer).

Dans notre exemple plus haut, la ligne de commande deviendrait: **g++ bonjour.cpp -o bonjour**

Pour lancer l'exécutable, il suffit de taper à l'invite de commande **./bonjour**

Vous pouvez également générer le code assembleur à partir de votre fichier source. L'assembleur est le langage de plus bas niveau qui soit, comprendre le langage le plus proche du fonctionnement de la machine mais lisible par un humain. Il n'y a pas un assembleur mais des assembleurs, ce langage dépendant fortement de l'architecture matérielle et logicielle. Pour voir à quoi cela ressemble, il suffit de taper à l'invite de commande **g++ bonjour.cpp -S**

Cela générera un fichier **bonjour.s** au nombre de lignes plus conséquent.

CREATION D'UN QCM

L'objectif va être de créer un QCM tout au long de ce TP. On va se concentrer sur plusieurs étapes pour construire ce programme. Vous savez déjà afficher des messages, il suffit de maintenant voir les différentes structures permettant de créer un QCM en bonne et due forme.

On posera des questions avec 4 propositions de réponse possibles. Une seule de ces propositions sera bonne.

CAS D'UTILISATIONS

Suivant votre formation précédente, vous pouvez ne pas être familier avec le diagramme qui suit. Il s'agit d'un diagramme **UML/SysML** de cas d'utilisations. Il exprime à quoi sert un système en fonctionnement normal.

Le diagramme présenté ci-après a pour but de montrer ce que l'on pourra faire à terme avec ce système. Il n'est pour l'heure pas encore le but à atteindre.

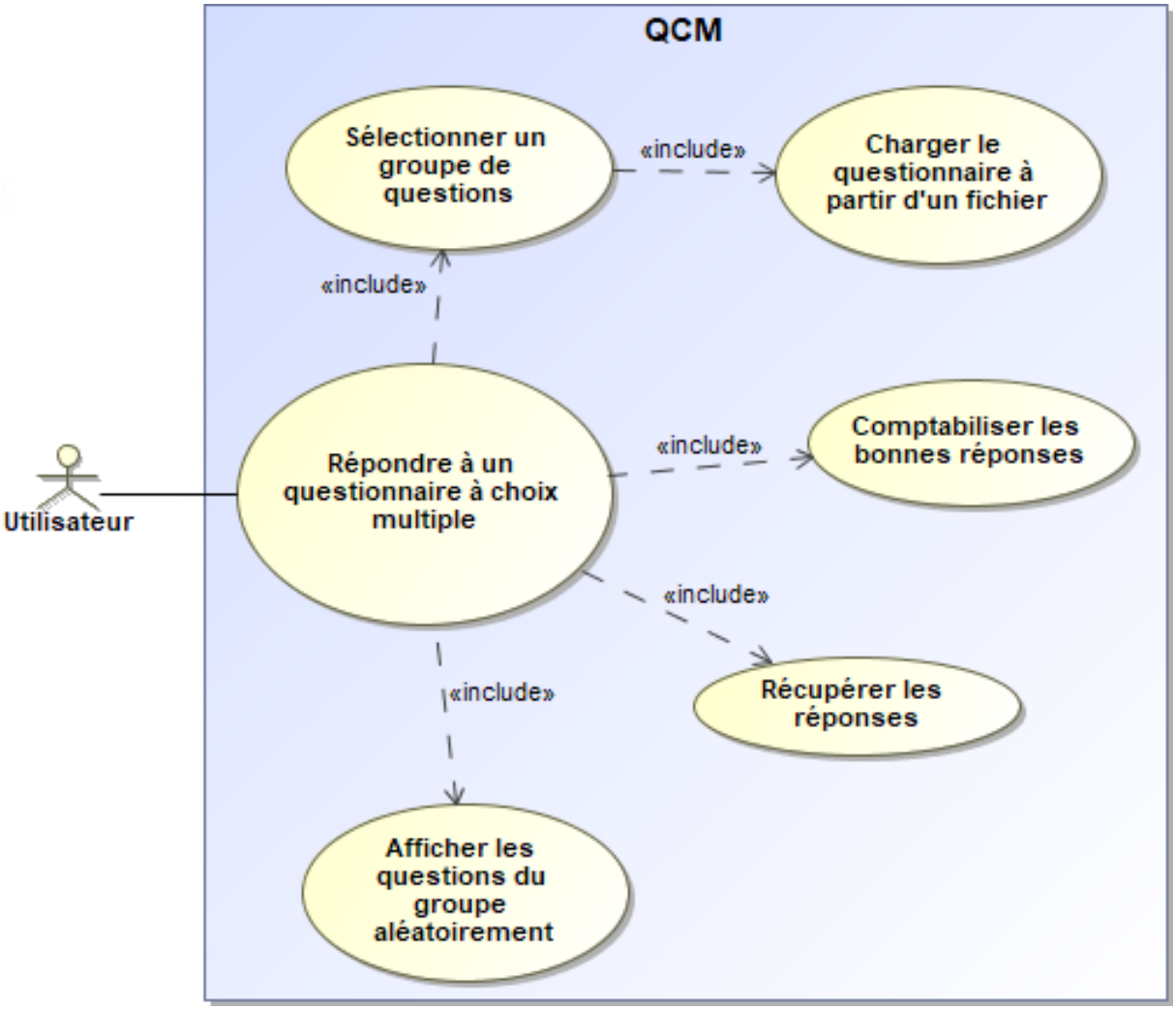

Figure 2: Diagramme des cas d'utilisations

PREMIERE ETAPE

Vous avez fait un programme classique Hello World. Adaptez le pour poser une question et afficher les 4 propositions possibles.

UTILISATION DU PROGRAMME

UML et **SysML** permettent de modéliser entièrement un projet, qu'il soit purement informatique pour le premier ou multi disciplinaire pour le second.

Dans les divers diagrammes disponibles, il en existe un permettant de décrire le déroulé de l'utilisation du système: le diagramme de séquence.

En voici un qui ne prétend surtout pas être complet du QCM:

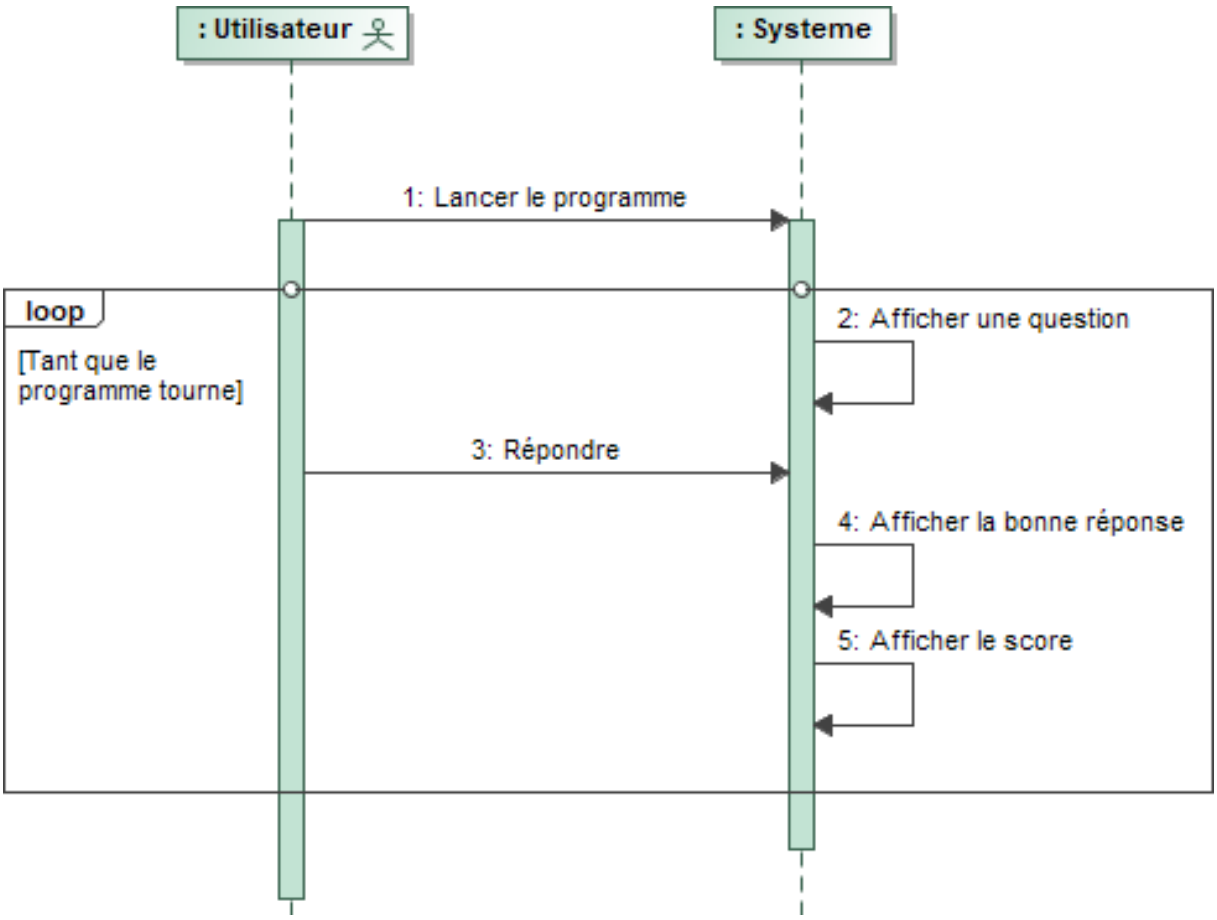

Figure 3: Diagramme de séquence

SECONDE ETAPE

Adaptez votre programme de façon à ce que la question (on se contente d'une seule pour l'instant) et ses propositions associées soient stockées dans des variables. On doit pouvoir récupérer la réponse de l'utilisateur et pour l'instant, juste l'afficher.

CREATION ET UTILISATION DE VARIABLES

La question et les propositions associées doivent forcément être stockées afin de pouvoir être traitées plus facilement. En effet, une fois la réponse de l'utilisateur donnée, il serait intéressant de réafficher la bonne réponse, comme montré dans le diagramme de séquence. Nous n'y sommes pas encore, mais c'est à garder à l'esprit.

Et il faut donc être aussi en mesure de récupérer la réponse de l'utilisateur afin de la traiter. Il faut donc stocker toutes ces informations dans des variables.

Elles permettent de réserver un emplacement mémoire dans la **RAM** (Random Access Memory) afin de pouvoir y stocker tout type d'information.

TYPES DE VARIABLES

Le **C++** est un langage fortement typé: comprenez que contrairement à des langages comme **Python** ou **PHP**, vous devez spécifier le type de données que votre variable va contenir.

Dans notre cas et pour l'instant, nous souhaitons stocker des phrases (questions et réponses) et soit une lettre (réponse a, b, c ou d) ou un chiffre (1,2,3 ou 4) pour la réponse de l'utilisateur. Il faut aussi penser à son score. On a affaire ici à des chaines de caractères et à soit un caractère, soit un nombre entier pour la réponse et bien sûr, un entier pour le score. A noter que 1,2,3 et 4 peuvent être aussi stockés comme des caractères. On choisira cette solution car moins gourmande en place dans la **RAM** pour la réponse de l'utilisateur.

Il est à noter que quel que soit le type de variable que l'on doit utiliser dans un programme, sa déclaration se fait toujours ainsi: `type nomDeLaVariable;`

On utilisera le type **string** pour les chaines, **char** pour la réponse de l'utilisateur et **unsigned short** pour le score.

DIALOGUER AVEC L'UTILISATEUR

Maintenant, on y est: il faut répondre à la question. Si **cout** est l'opérateur de sortie à l'écran, il en faut bien un pour l'entrée au clavier. Il s'agit de **cin**. Son utilisation est simple, il suffit d'écrire l'instruction ainsi: `cin>>maVariable;`

TROISIEME ETAPE

Adaptez votre programme de façon à ce que la réponse de l'utilisateur puisse être testée afin d'afficher si elle est juste ou non.

TRAITER LA REPONSE

Récupérer un caractère à partir du clavier est une chose, encore faut-il en faire quelque chose. Le choix de l'utilisateur se caractérise par un caractère: il faut le tester afin de déterminer le traitement à réaliser.

Pour cela, il faut utiliser une structure conditionnelle: le si en algorithme, **if** en **C++**.

Il s'écrit comme suit:

```
if(maCondition)
{
    instruction1;
    instruction2;
}
else
{
    instruction3;
    instruction4;
}
```

La condition d'un **if** est toujours un booléen (**bool**) c'est à dire soit vrai, soit faux. Les opérateurs de comparaison que l'on utilise sont: `!`, `<`, `>`, `<=`, `>=`, `==`, `!=`

L'objectif est d'afficher un message permettant à l'utilisateur de voir que son choix est bien validé, le message sera donc différencié en fonction de sa réponse, si elle est bonne ou pas. On peut aussi afficher la bonne réponse dans le cas où l'utilisateur répond faux.

QUATRIEME ETAPE

Il faut pouvoir maintenant permettre à l'utilisateur d'arrêter ou de continuer à répondre aux questions. Pour l'instant, on envisagera ça après chaque question (même si il n'y en a qu'une à l'heure actuelle). Donc il faut demander à l'utilisateur s'il veut continuer. Si non, le programme stoppe. Modifiez votre code pour cadrer avec cet énoncé.

STRUCTURE ITERATIVE: WHILE

La structure itérative **do while** en **C++** est comme son nom l'indique une **boucle**. Il en existe de plusieurs sortes mais celle-ci présente l'avantage qu'il n'est pas nécessaire de savoir au préalable combien de fois il faut boucler et de plus, son contenu s'exécutera au moins une

fois. Autrement dit, vous, développeur, ne savez pas au moment où vous coder, combien de fois votre boucle va tourner: cela peut être jamais comme une infinité de fois.

Voici sa syntaxe:

```
do
{
    instruction1;
    instruction2;
} while(maCondition);
```

La traduction de **do while** en français est faire tant que, ce qui veut dire dans notre utilisation, tant que la condition n'est pas vérifiée, on boucle. La condition, comme pour le **if** est un booléen.

INDEX