FizzBuzz

Mini-Projet à rendre par binôme pour le jeudi 6 octobre 10h

En 2007, Imran Ghory décide de proposer un problème de programmation simple aux candidats à un poste de développeur afin d'éliminer plus rapidement ceux dont il juge que les compétences sont trop faibles. A sa grande surprise, de très nombreux candidats échouent ou ont des difficultés face au test, autant parmi les jeunes diplômés en informatique que chez les senior.

L'exercice qu'il a proposé consistait à écrire un programme qui compte jusqu'à 100 en remplaçant les multiples de trois par le mot « Fizz » et les multiples de 5 par le mot « Buzz ».

Il est directement inspiré d'un jeu pour enfants au cours duquel chaque enfant compte un nombre à son tour. Si le nombre est un multiple de 3, il doit le remplacer par le mot Fizz. Si c'est un multiple de 5, il doit le remplacer par le mot Buzz. Enfin si c'est un multiple de 3 et de 5 simultanément il doit le remplacer par le mot FizzBuzz.

Exemple de séquence de nombre respectant les règles de FizzBuzz :

1, 2, Fizz, 4, Buzz, Fizz, 7, 8, Fizz, Buzz, 11, Fizz, 13, 14, FizzBuzz, 16, 17, Fizz, 19

Le but de ce projet est de réaliser un programme permettant à un joueur d'affronter un ordinateur dans une partie de FizzBuzz.

Vous programmerez ce jeu de façon incrémentale. Vous ne devez pas passer à la question n+1 tant que le programme de la question n n'a pas été testé à partir d'un jeu d'essais le plus complet possible.

Travail à effectuer

1. Créer un programme qui demande son nom au joueur ainsi qu'un intervalle indiquant la valeur minimale et la valeur maximale entre lesquelles il souhaite compter. Ces deux valeurs doivent être strictement positives et la 2ème doit être supérieure à la 1ère. Redemander un nombre tant que les contraintes de saisie ne sont pas respectées.

2. Compléter le programme de façon à ce qu'il demande au joueur qui commence. Si l'ordinateur démarre le joueur saisit 0 et sinon 1. A nouveau, vérifier la valeur saisie et la redemander autant de fois que nécessaire.

3. Modifier le programme pour permettre au joueur et à l'ordinateur de compter de 1 en 1 dans la console chacun à leur tour, entre les 2 bornes fournies par le joueur (pas de vérifications encore nécessaires). Pour cela vous afficherez le nom du joueur suivi du symbole « > » à chaque fois qu'il doit entrer un nombre et «Ordinateur> » suivi d'un nombre chaque fois que l'ordinateur joue. Exemple :

> Ordinateur>1
>
> Joueur> 2
>
> Ordinateur>3
>
> Joueur>

4. Stocker la valeur entrée par le joueur dans une chaîne de caractères (variable de type **`string`**). Vérifier que la valeur du joueur respecte la consigne de compter de 1 en 1. Afficher un message d'erreur sinon.

   NB : le compilateur gcc fourni avec Code::Blocks ne prend pas entièrement en charge le C++11 (**`stoi`**) pour faire des conversions depuis de type **`string`** vers le type **`int`** ou inversement. Vous devez vous inspirer du programme exemple-conversions.cpp fourni et donné en annexe.

5. Remplacer les multiples de 3 par Fizz et les multiples de 5 par Buzz ainsi que les multiples de 3 et de 5 par FizzBuzz lorsque l'ordinateur joue. Utiliser une concaténation pour générer le mot FizzBuzz, ainsi que les booléens **`mult3`** et **`mult5`** pour savoir si une valeur est un multiple de 3 ou de 5.

6. Vérifier que le joueur remplace bien les multiples de 3 par le mot Fizz et les multiples de 5 par le mot Buzz ainsi que les multiples de 3 et de 5 à la fois par le mot FizzBuzz. Continuer à utiliser les booléens **`mult3`** et **`mult5`**. Si le joueur se trompe, afficher un message d'erreur et donner la bonne valeur.

7. Ajouter des vies au joueur. Celui-ci commencera la partie avec 3 vies. Pour chaque erreur, il perdra une vie. Afficher le nombre de vies restantes entre parenthèses après le nom du joueur. Exemple :

   > Ordinateur>1
   >
   > Joueur1 (3 vies) > 2

8. Arrêter la partie lorsque le joueur n'a plus de vies ou lorsque la valeur maximale choisie par le joueur a été atteinte et afficher le message approprié (« Victoire » ou « Défaite »)

9. Proposer au joueur de recommencer une partie ou de quitter le jeu lorsqu'une partie est finie.

ATTENTION :

- Ne pas oublier de commenter son programme ! Chaque boucle, chaque conditionnelle, chaque groupe d'instructions, doit être précédé d'un commentaire décrivant le traitement effectué.

- Afin de coder proprement et efficacement, veillez à toujours nommer correctement vos variables et vos constantes, ainsi qu'à bien indenter votre code. N'oubliez pas que ces noms doivent être intelligibles afin de rendre votre programme lisible

Optionnel (pour les plus rapides)

Attention : continuer seulement, et seulement si tout ce qui précède a été correctement traité !

1. Permettre au joueur de ne pas donner de limite maximale à atteindre.
2. Ajouter une règle qui, immédiatement après que le joueur ou l'ordinateur ait écrit le mot « FizzBuzz » demande que l'on écrive Buzz à la place de chaque multiple de 3, Fizz à la place de chaque multiple de 5 et BuzzFizz à chaque multiple de 3 et de 5 à la fois.
   Compléter cette règle en ajoutant son inverse : immédiatement après que le joueur ou l'ordinateur ait écrit le mot « BuzzFizz », le mot Fizz doit être écrit à la place de chaque multiple de 3, le mot Buzz à la place de chaque multiple de 5 et le mot FizzBuzz à la place de chaque multiple de 5 et de 3 à la fois.

Travail à rendre par binôme avant le jeudi 6 octobre 10h

Vous devrez, dans le cadre de ce mini-projet, écrire un seul programme et constituer un dossier qui contiendra :

1. Une page de titre, incluant le nom des binômes, leur numéro de groupe, le sujet du mini-projet, et une date.
2. Une page contenant une table des matières, incluant la numérotation des pages.
3. Une introduction rappelant brièvement le sujet.
4. Les hypothèses de travail et les choix de programmation que vous avez dû faire.
5. En cas de difficultés rencontrées ou de problème(s) non résolu(s), fournir les messages d'erreurs produits et expliquez les tests effectués pour essayer de trouver votre erreur. Signalez toute partie de votre programme que vous jugez défectueuse.
6. Un jeu d'essais, complet et commenté, respectant les principes décrits dans l'annexe du TD2, et permettant de convaincre votre lecteur que votre programmation est correcte, pour tous les cas possibles.

   Ce jeu d'essais DEVRA être présenté sous forme d'un tableau, comportant les 3 colonnes suivantes :

   - but du test (*pourquoi tester ce cas ?)*
   - la ou les valeurs choisies en entrée *(avec quelles valeurs est fait ce test ?)*
   - les résultats **prévus** pour ce test *(que devrait-il se passer ? quels résultats devrait-on obtenir ?)* Attention : il n'est pas nécessaire de fournir tous les affichages du programme, il ne s'agit pas ici de donner une trace d'exécution.
7. Une conclusion, indiquant l'avancement plus ou moins complet du travail, les difficultés rencontrées, les améliorations et les extensions éventuelles apportées, ainsi que ce que le projet vous a apporté.
8. Une annexe : le code source du programme dûment et pertinemment commenté.

N'essayez pas de nécessairement TOUT traiter. Un programme bien écrit, bien testé, mais ne traitant pas la totalité des questions est plus apprécié qu'un programme qui essaie de tout traiter mais qui ne fonctionne pas ou qui a été partiellement testé.

Votre dossier (en format pdf) ainsi que votre code source devront être déposés sur Moodle au plus tard le jeudi 6 octobre à 10h00. Vous devez également remettre une version papier de votre dossier dans le casier de votre enseignant.

ATTENTION :

- La date/heure de dépôt sur Moodle fait foi pour la date de remise.

- Aucun délai supplémentaire ne sera accordé.

- Un point sera retiré par jour de retard en semaine et deux points par jour de retard le weekend. Travail rendu après jeudi 10h et avant vendredi 7/10 10h : -1, avant samedi 8/10 10h : -2, avant dimanche 9/10 10h : -4, lundi 10/10 10h : -6, etc.)

- Aucun projet ne sera accepté après jeudi 13 octobre.

Bonne chance !

Annexe

```
// L.  Cossou, H. Maynard
// IUT Orsay, sept. 2016
// conversion de string en int et de int en string
// en passant par les bons vieux tableaux de char
// car le compilateur qu'on utilise avec Code::Blocks n'intègre pas la norme C++ 11
// ATTENTION :  si c_str() ne correspond pas à la définition d'un entier, le comportement de atoi()
n'est pas prévisible.

#include <iostream>
#include <string>
#include <cstdlib>  // pour pouvoir utiliser atoi
#include <cstdio>   // pour pouvoir utiliser sprintf
using namespace std;

int main() {
    string s;
    int val=10;
    char tab[250]; // un intermédiaire pour pouvoir utiliser atoi et sprintf
    // conversion de string vers int
    s="25"; // la string initiale
    val=atoi(s.c_str());  // on convertit la string en tableau de char, et on peut utiliser atoi
                          // pour obtenir un entier à partir du tableau de char
    cout << s << " " << val << endl;  // val contient bien l'entier qui était stocké dans s
    // conversion de int vers string
    val=10;  // l'entier initial
    sprintf(tab,"%d",val);     // on utilise la fonction sprintf du C pour envoyer l'entier dans le
                               // tableau de char
    s=tab;  // facile d'envoyer le tableau de char dans l'entier
    cout << s << " " << val << endl;  // s contient bien l'entier qui était stocké dans val
    /* exemple d'utilisation de stoi (mais seulement avec la norme C++ 11)
    s="55";
    val=stoi(s);
    cout <<val << endl;
    */
    return 0;
}
```