

R1.01 - Initiation au développement - Contrôle final

16/01/2024

Durée : 1 h 30 min — Tout document et calculatrice interdits ; brouillon(s) autorisé(s) — Barème donné à titre indicatif — Seules les réponses écrites en C++ seront considérées. Toutes autres formulations, pseudo code, code python, ou autres formes issues de votre imagination, seront simplement ignorées.

Introduction

Un questionnaire à choix multiple (QCM), est un type d'évaluation dans lequel les participants doivent choisir une réponse correcte parmi plusieurs options proposées. En principe, chaque question du QCM est accompagnée de plusieurs réponses possibles, mais une seule de ces réponses est correcte.

Il existe différentes variantes de QCM en fonction du type de question utilisé. Parmi celles-ci, on peut citer les questions à choix-multiples et réponses-multiples. Ces questions se distinguent des questions à réponse unique habituellement utilisées par le fait qu'il est nécessaire dans ce cas de choisir plusieurs réponses pour répondre à la question.

L'objectif de cet exercice est de programmer un QCM utilisant des questions de type *choix-multiples* et *réponses-multiples*.

Afin de fixer le travail à réaliser, le programme principal (cf. listing 1) que vous devez utiliser, ainsi qu'un extrait de la trace d'exécution correspondante (cf. listing 2), vous sont donnés. Lors de la réponse aux questions qui vont suivre, gardez en tête que votre programme doit permettre de reproduire à l'identique cette trace.

Le programme principal (cf. listing 1) est décomposé en deux parties. Pour commencer, on initialise les données du QCM (ligne 4), puis, on lance le QCM (ligne 29). Cette dernière opération se fait par le biais du sous-programme `run` (cf. listing 3).

Le déroulé de ce sous-programme est le suivant :

- Il affiche les informations concernant le QCM
- Puis, pour chaque item, il affiche :
 - Le nombre de points associés
 - la question (le *stem*)
 - Appelle le sous-programme `show` (cf. listing 4) qui s'occupe de brasser les réponses possibles, les afficher, et gérer la saisie de l'utilisateur.
- Affiche le score obtenu par l'utilisateur

Listing 1 Programme principal

```
1  #include "mc.hpp"
2
3  int main() {
4      multiple_choice::Assessment test{
5          "Questionnaire sur le code et la route", {{
6              "Où se situe le « point kilométrique zéro » ? ",
7              1.5,
8              {{ "Sur le Mont-Blanc", false }, { "Au début de la règle", false }, { "À
              ↳ Notre Dame de Paris", true }, { "Sous la tour Eiffel", false }}
9          }, {
10             "Quels types de véhicule n'ont pas le droit de circuler sur une autoroute
              ↳ ?",
11             0.5,
12             {{ "Un tracteur", true }, { "Un car", false }, { "Un cheval de course", true
              ↳ }, { "Une moto", false }}
13         }, {
14             "De quel côté de la route roule-t-on en France ?",
15             0.5,
16             {{ "À droite", true }, { "Au milieu", false }, { "À gauche", false }}
17         }, {
18             "Quels équipements sont obligatoires dans une voiture ?",
19             1.0,
20             {{ "Un gilet jaune", true }, { "Une console de jeu", false }, { "Un GPS",
              ↳ false }, { "Un triangle de sécurité", true }}
21         }, {
22             "Quel est le numéro de téléphone des secours ?",
23             1.5,
24             {{ "15", true }, { "119", false }, { "112", true }, { "17", false }, {
              ↳ "123", false }}
25         }
26     };
27
28     multiple_choice::run(test);
29     return 0;
30 }
31
```

Listing 2 Extrait de la trace d'exécution du programme

Questionnaire sur le code et la route

Question 1 (1.5) : Où se situe le « point kilométrique zéro » ?

A: Sur le Mont-Blanc

B: Au début de la règle

C: À Notre Dame de Paris

D: Sous la tour Eiffel

choix > C

Question 2 (0.5) : Quels types de véhicule n'ont pas le droit de circuler sur une
↪ autoroute ?

A: Un car

B: Un tracteur

C: Un cheval de course

D: Une moto

choix > B,C

[...]

Question 5 (1.5) : Quel est le numéro de téléphone des secours ?

A: 123

B: 119

C: 17

D: 15

E: 112

choix > E

Score total: 3.5

Listing 3 ss-programme qui lance le QCM

```
void run(Assessment& assessment) {
    std::cout << assessment.title <<std::endl;
    std::cout << "-----" << std::endl;

    std::size_t k = 0;
    for(Item& q: assessment.items) {
        std::cout << "Question " << ++k << " (" << q.point << ") : ";
        std::cout << q.stem << std::endl;
        show(q);
        std::cout << std::endl;
    }

    std::cout << "Score total: " << computeScore(assessment) << std::endl;
}
```

Listing 4 sous-programme qui affiche chaque item et gère la saisie utilisateur

```
void show(Item& item) {
    randomizeIt(item.options);
    char letter = 'A';
    for (Option& item : item.options)
        std::cout << letter++ << ": " << item.text << std::endl;
    readAnswers(item);
}
```

Questions

Ne restez pas bloqué(e) sur une question, vous pouvez traiter les questions séparément ! Ce n'est pas parce que vous n'y avez pas répondu que vous ne pouvez pas utiliser les résultats d'une question précédente. Il ne vous est pas interdit d'écrire d'autres sous-programmes que ceux explicitement demandés. Pensez à lire tout le sujet avant de commencer afin d'avoir une vision plus globale du problème !

Question 1 (4 points) : Structure de données

Dans votre programme, le QCM sera représenté par trois entités différentes :

- **Assessment** : L'évaluation à proprement parler ;
- **Item** : chacune des questions de l'évaluation ;
- **Option** : Chacune des réponses possibles aux questions.

Le détail des éléments constituant chacune de ces entités est donné ci-dessous.

Assessment

- **title** : Un titre
- **items** : Un tableau d'**Item**

Item

- **stem** : Un texte, appelé en principe *tronc* (*stem* en anglais) car cela n'est pas nécessairement une question (affirmation, phrase à compléter, etc.).
- **point** : Un nombre de points qui représente ce que rapporte une réponse parfaitement correcte.
- **options** : Un tableau d'**Option** pour stocker les réponses possibles.
- **answers** : Un tableau pour stocker le ou les indices qui correspond à ou aux réponses de l'utilisateur dans **options**.

Option

- **text** : Le texte de la réponse possible.
- **type** : Un *drapeau* pour savoir si cette réponse est juste ou non.

Écrivez les déclarations des trois structures de données permettant de représenter ces éléments.

Question 2 : Saisie des réponses

Question 2.1 (3 points) : Saisie utilisateur

Écrire un sous programme dont le prototype est `std::vector<std::string> getUserInput(std::string msg)`. Ce sous-programme doit afficher à l'utilisateur le message `msg` et lui permettre de saisir une chaîne de caractères `s`. Ce sous programme retourne un tableau contenant l'ensemble des sous-éléments de `s` découpée par rapport au symbole `' '`.

Ainsi, si l'utilisateur saisit "A,C,BD" le vecteur contiendra "A", "C", "BD".

Question 2.2 (5 points) : Gestion des réponses

Écrire le sous-programme `readAnswers` (cf. listing 4). Ce sous-programme prend en paramètre un `Item` et permet de mettre à jour son champ `answers` à partir du retour obtenu par l'appel à la fonction `getUserInput` de la question précédente.

Attention, `answers` ne doit contenir que des indices valides ! Vous trouverez ci-dessous quelques exemples du comportement attendu du programme en fonction de différentes saisies de l'utilisateur.

Ci-dessous, l'utilisateur n'a saisi qu'une réponse.

```
Question 1 (1.5) : Où se situe le « point kilométrique zéro » ?
A: Au début de la règle
B: À Notre Dame de Paris
C: Sur le Mont-Blanc
D: Sous la tour Eiffel
choix > C
```

Dans cet exemple, l'utilisateur a appuyé sur entrée directement et n'a ainsi saisi aucune réponse.

```
Question 1 (1.5) : Où se situe le « point kilométrique zéro » ?
A: À Notre Dame de Paris
B: Au début de la règle
C: Sur le Mont-Blanc
D: Sous la tour Eiffel
choix >
```

Dans cet exemple, l'utilisateur a saisi pour l'un de ces choix un choix invalide. Comme la réponse de l'utilisateur ne contient pas uniquement des choix valides, le programme lui redemande la saisie complète.

```
Question 1 (1.5) : Où se situe le « point kilométrique zéro » ?
A: Sur le Mont-Blanc
B: À Notre Dame de Paris
C: Au début de la règle
D: Sous la tour Eiffel
choix > A,E
Erreur, votre choix n'est pas valide
choix >
```

Question 3 (4 points) : Calcul du score d'un item

Il existe de nombreuses manières de calculer le score d'un `Item` de QCM. Cela dépend de ce que l'on veut évaluer. Comme dans notre cas nous avons la possibilité de créer des questions pour lesquelles il faut choisir plusieurs réponses, il nous faut un système de score adapté.

Pour calculer le score de chaque item, on va répartir les points (`point`) de l'item sur les réponses valides et leur opposé ($-1 \times \text{point}$) sur les réponses non valides. Afin de simplifier la création de l'item pour l'évaluateur, on ne va pas lui demander la répartition mais on va la répartir équitablement.

Ainsi, si un item possède 5 réponses possibles (Option) dont 2 à choisir et qu'il vaut 1.5 point alors on aura :

- pour chaque réponse juste choisie : $1.5/2 = 0.75$
- pour chaque réponse fausse choisie : $-1.5/3 = -0.5$

Par exemple, si on prend le cas ci-dessous.

```
Question 5 (1.5) : Quel est le numéro de téléphone des secours ?
A: 123
B: 119
C: 15
D: 17
E: 112
choix > C,D,E
```

L'utilisateur obtient un score de 1 point sur 1.5 car il a choisi deux bonnes réponses et une mauvaise soit un total de $2 \times 0.75 - 0.5 = 1$.

Écrire la définition du sous-programme `computeScore` qui prend en paramètre un `Item` et retourne son score calculé en fonction du contenu de `options`, `answers` et de la valeur de `point`.

Question 4 (1 points) : Calcul du score du QCM

Écrire la définition de la fonction `computeScore` (cf. listing 3) qui prend en paramètre un QCM et retourne le score total obtenu par l'utilisateur.

Question 5 (2 points) : Brassage des réponses possibles

Le sous-programme `show` (cf. listing 4) permet d'afficher un item et gère l'interaction avec l'utilisateur. Celui-ci utilise un sous-programme nommé `randomizeIt` qui prend en paramètre le tableau d'`options` de l'item et va réaliser un brassage de ses éléments. Cela permet d'un appel à l'autre de `show` de modifier aléatoirement l'ordre d'affichage des réponses possibles.

Après une rapide recherche dans les forums, vous avez trouvé un extrait de code (cf. listing 5) qui ressemble à votre besoin.

Listing 5 Extrait de code C++ permettant de brasser le contenu d'un vecteur.

```
#include <algorithm>
//...
static std::random_device rd {};
static std::mt19937 gen(rd());
std::vector<std::string> v = {"a", "b", "c"};
std::shuffle(v.begin(), v.end(), gen);
```

A partir des informations qui précèdent, écrire la définition de la fonction `randomizeIt`.

Question 6 (1 points) : Fichier d'entêtes

Compte tenu de tout ce qui précède, donnez le contenu minimal et suffisant du fichier d'entête `mc.hpp`.