TP 1 — Premières compilations

Le but de ce TP n'est pas de programmer « avec des objets » mais se familiariser avec certaines particularités de C++ et utiliser l'EDI QtCreator.

1. Séquence de couleurs (1) : Structure et fonctions

Projet QtCreator, Passage par référence, Références constantes, struct, enum class, std::array, using.

Dans un premier temps, nous allons créer une structure de données, accompagnée de fonctions sur cette structure de données. Nous n'allons pas créer de classes dans ce premier exercice dont le but est de se familiariser avec certains aspects de C++. Plus tard dans le semestre, nous réécrirons cette structure de données sous la forme d'une classe et créerons une interface graphique.

On veut créer un type « séquence de couleurs » permettant de représenter une suite ordonnée de couleurs parmi un ensemble prédéfini : rouge, bleu, jaune, vert. Toutes les déclarations de types et fonctions demandés ici seront écrites dans un fichier `sequence.hh` et le code des fonctions sera écrit dans un fichier `sequence.cc`. Le `main` de test (présenté plus bas) sera écrit dans un fichier `main.cc`.

Les types et fonctions étant destinés à être utilisés aussi bien dans le `main` de test que nous allons écrire ici que depuis d'autres fichiers (par exemple une interface graphique que nous créerons plus tard), ils doivent être déclarés dans un fichier d'entêtes qui sera inclus aussi bien dans `main.cc` que dans le fichier qui définira les fonctions (`sequence.cc`) et éventuellement d'autres fichiers source (interface graphique par exemple).

1. Déclarer un type `couleur` comme une énumération fortement typée (`enum class`) composée des 4 valeurs `rouge, bleu, jaune, vert`.
2. Déclarer un type `sequence` comme un type structuré composé d'un tableau (`std::array`) de 100 `couleur` (100 étant la taille maximale d'une séquence ici) et d'un entier représentant la taille actuelle de la séquence. Plutôt que d'utiliser `unsigned int` pour typer cet entier, définir un nouveau type `indicesequence` pour représenter (au mieux) les indices (ainsi que la taille) d'une telle séquence.

Nous allons maintenant déclarer et définir des fonctions afin de fournir des traitements sur ces séquences. Ces fonctions devront être déclarées et définies de façon à ce que le fichier suivant (`main.cc` http://www.info.univ-angers.fr/pub/genest/fichiers/l3_poo/tp/1/main.cc) puisse être compilé « tel quel » et produire les résultats attendus. Il vous est **fortement** conseillé de tester votre code au fur et à mesure : mettez en commentaire dans le `main` les appels aux fonctions que nous n'avez pas encore définies et faites un test à chaque fois que vous avez écrit une fonction. Les numéros en commentaire ci-dessous correspondent aux numéros des questions dans lesquelles les fonctions sont décrites.

```
#include "sequence.hh"
#include <iostream>

int main() {
    sequence a, b;
    sc_initialiservide(a); // 3
    sc_initialiservide(b);
    sc_ajouter(a, couleur::rouge); // 4
    sc_ajouter(a, couleur::bleu);
    sc_ajouter(a, couleur::rouge);
    sc_ajouter(a, couleur::vert);
    sc_copier(b, a); // Copie a dans b // 5
    sc_afficher(sc_acces(a, 0)); std::cout << "\n"; // 6 et 7
    for (indicesequence i=0; i<a.taille; ++i)
          sc_afficher(sc_acces(a,i));
    sc_vider(a); // 8
    std::cout << "A: "; sc_afficher(a); // 9
    std::cout << "B: "; sc_afficher(b);
    std::cout << sc_comparer(a,b) << "\n"; // 10
    sc_copier(a,b);
    std::cout << sc_comparer(a,b) << "\n";
    return 0; }
```

Le résultat fourni par l'exécution du programme doit être le suivant :

```
rouge
rougebleurougevert
A:
B: rouge bleu rouge vert
0
1
```

3. Écrire une fonction `sc_initialiservide` se chargeant d'initialiser à vide une séquence. Réfléchir à l'intérêt de cette fonction, et au comportement du programme si elle n'est pas utilisée.
4. Écrire une fonction `sc_ajouter` rajoutant une couleur à une séquence. On supposera que la séquence ne dépasse jamais sa taille limite.
5. Écrire une fonction `sc_copier` prenant comme arguments deux séquences et copiant (en écrasant) dans son premier argument le contenu de son deuxième argument.
6. Écrire une fonction `sc_acces` prenant comme paramètre une séquence et un indice et retournant la couleur se trouvant à cet indice dans la séquence. On supposera que l'`indicesequence` passé en paramètre sera toujours « correct ».
7. Écrire une fonction `sc_afficher` prenant comme paramètre une couleur et affichant sur le flux de sortie standard l'intitulé de cette valeur.
8. Écrire une fonction `sc_vider` vidant une séquence.
9. Écrire une fonction `sc_afficher` prenant comme paramètre une séquence et affichant sur une ligne le contenu de la séquence.
10. Écrire une fonction `sc_comparer` prenant comme paramètres deux séquences et retournant un booléen valant vrai si les deux séquences sont identiques, faux sinon.

2. Chaînes de caractères

std::string.

Consulter la documentation de `std::string` https://en.cppreference.com/w/cpp/string/basic_string pour répondre aux questions suivantes. Les 4 fonctions demandées ici seront écrites dans un seul fichier source contenant aussi le `main` de test. Il n'est pas question ici d'écrire du code réutilisable, mais simplement de tester certaines fonctionnalités des chaînes de caractères C++.

1. Écrire une fonction `nbvoyelles` retournant le nombre de voyelles dans un mot. Ce mot sera lu au clavier dans la fonction `main`.
2. Écrire une fonction vérifiant si une chaîne est un palindrome.
3. Le *Chiffre de César* est une méthode de chiffrement de texte dans laquelle le message chiffré est obtenu en « décalant vers la droite » d'un certain nombre de rangs dans l'alphabet chacune des lettres du message. Le déchiffrement étant évidemment obtenu en décalant du même nombre de rangs dans l'autre sens. Par exemple avec un décalage de 3, le mot « mot » devient « prw ». Quand on arrive au bout de l'alphabet, on revient au début, ainsi « y », avec un décalage de 3 devient « b ».
   Écrire une fonction `chiffrer` prenant comme paramètre une chaîne et un décalage et retournant le forme chiffrée de la chaîne. Seules les lettres seront prises en compte (majuscules et minuscules), les autres caractères (chiffres, ponctuation, etc) seront ignorés et laissés « en clair » dans la valeur retournée. Écrire une fonction `dechiffrer`.
4. Écrire une fonction récursive qui, à partir d'une chaîne contenant un nombre entier positif, retourne la valeur de cet entier sous la forme d'un `unsigned int`. On rappelle que chaque caractère d'une `std::string` est représenté sous la forme d'un `char` qui n'est rien d'autre qu'un entier compris entre 1 et 255, et que les caractères qui correspondent aux chiffres utilisent des codes entiers consécutifs, `'0' = 48, '1' = 49, '2' = 50 ... '9' = 57` (`man ascii` si vous voulez en savoir plus). Notez que `std::stoi` fait exactement ce qui est demandé ici, mais le but de cet exercice est de manipuler les `std::string` (et vous n'avez donc pas le droit d'appeler cette fonction ou autres fonctions comparables). Par contre, on a tout intérêt en temps normal à faire appel à la bibliothèque standard plutôt que réinventer la roue.

3. Séquence de couleurs (2) : Allocation dynamique

Pointeurs, allocation dynamique, new[], delete[].

L'utilisation de `std::array` dans l'exercice 1 nous a permis d'écrire le code de façon simple, mais cette solution a un défaut majeur : quelle que soit la taille « réelle » de la séquence, il y a toujours 100 cases dans le tableau. Ce qui peut être inutilement trop… ou pas assez. On a besoin d'une structure de données de taille dynamique *et non* fixe.

Le code qu'il vous est demandé d'écrire ici résidera dans deux fichiers `sequence2.hh` et `sequence2.cc` et devra être écrit en faisant en sorte que la fonction `main` de l'exercice 1 puisse être utilisée « telle quelle » (en changeant simplement le nom du fichier inclus, évidemment). Ainsi ce que nous écrirons ici sera totalement interchangeable avec ce que nous avons écrit à l'exercice 1.

1. Définir `sequence` comme un type composé d'un pointeur pointant sur une zone de `couleur` allouée dynamiquement et d'un entier représentant la taille de cette zone. À tout moment, la zone allouée dynamiquement sera exactement de la taille présente dans la structure.

```
using indicesequence = std::size_t;
struct sequence {
    couleur * couleurs;
    indicesequence taille;
};
```

2. Écrire les fonctions décrites dans l'exercice 1 en faisant tout particulièrement attention à la gestion de la mémoire, y compris aux fuites de mémoire, c'est-à-dire des allocations qui ne sont jamais désallouées. Il faudra peut-être déclarer une nouvelle fonction.
3. Le principal défaut de ce type est le manque d'efficacité de `sc_ajouter` : à chaque ajout, une nouvelle zone est allouée, les couleurs sont copiées dans la nouvelle zone, et l'ancienne zone est détruite. Sur des séquences de grande taille, cela est totalement inefficace. Pour améliorer cela, on pourrait (par exemple) augmenter la capacité de la séquence de 5 cases quand elle est « pleine » plutôt que de rajouter seulement une case. Dans ce cas, il faut mémoriser, en plus de la taille réellement utilisée, la taille de la zone allouée. Modifier le type ainsi que les fonctions pour gérer cette modification.