TP 6 — Exceptions

1. Tableau à intervalle d'indices

Exceptions, std::vector, Surcharge d'opérateurs, Types emboîtés (chapitre 7) (pour la question 7).

Les éléments des tableaux C ou C++ sont repérés par des indices qui sont des entiers positifs commençant à 0. On veut ici écrire une classe permettant d'utiliser des tableaux dans lesquels l'intervalle des indices est quelconque, par exemple [-10,10] ou [10000,10100].

1. Déclarer une classe `arrayint` pour manipuler des tableaux d'entiers à indices quelconques. Vous utiliserez un attribut de type `std::vector` pour représenter les éléments d'un `arrayint`.
   Munir cette classe d'un constructeur permettant de choisir l'indice minimum et l'indice maximum du tableau, ces indices étant des entiers signés.
   Définir deux accesseurs permettant d'accéder à l'indice minimum `imin` et à l'indice maximum du tableau `imax`.
   Définir un constructeur par recopie.
   Une déclaration d'un `arrayint` pourra ainsi se faire sous cette forme :
   ```
   arrayint tab(-10,10);
   arrayint tab2(tab);
   ```
   Les deux `arrayint` ainsi déclarés sont composés de 21 cases repérées par des indices compris entre -10 et 10.
2. Définir une classe `exceptionarrayint`, sous-classe de `std::exception`. Cette classe représentera les accès incorrects à un `arrayint`, c'est-à-dire l'accès à une valeur d'indice qui n'est pas incluse dans l'intervalle d'indices. Une instance d'`exceptionarrayint` mémorisera un message d'erreur (`std::string`) et l'indice erroné. Définir un constructeur permettant d'initialiser ces deux attributs. Définir la méthode `what`.
3. Écrire une méthode `at` dans `arrayint` prenant comme paramètre un indice et retournant l'entier situé à cet indice dans le tableau. Si l'indice passé ne fait pas partie de l'intervalle d'indices, lever une `exceptionarrayint`. Écrire l'opérateur d'accès [] faisant la même chose que `at` et permettant d'accéder au contenu d'un `arrayint` de la même façon qu'à un tableau C/C++. Par exemple sur l'`arrayint` déclaré ci-dessus, on pourra écrire : `std::cout << tab[-10];`.
4. Écrire une méthode `set` prenant comme paramètre un indice et une valeur et modifiant la valeur mémorisée dans l'`arrayint` pour l'indice passé.
5. Faire en sorte qu'il soit possible de faire une copie par affectation (=) et une comparaison d'égalité (==) sur des `arrayint`.
6. Écrire l'opérateur de sortie sur un flux <<. À titre d'exercice, cet opérateur utilisera l'accesseur à l'indice minimum du tableau, mais **n'utilisera pas** l'accesseur à l'indice maximum (et ne sera pas `friend` d'`arrayint`, et on ne rajoutera pas non plus de méthode retournant la taille d'un `arrayint`).
7. Définir un type emboîté `const_iterator` dans `arrayint` et munir `arrayint` de méthodes `begin` et `end` retournant un `const_iterator` afin de pouvoir parcourir un `arrayint` de la même façon qu'un conteneur de la bibliothèque standard : (soit `tab` un `arrayint`)
   ```
   for (arrayint::const_iterator i=tab.begin(); i!=tab.end(); ++i)
     std::cout << (*i) << "\n";
   ```
8. Écrire une sous-classe d'`arrayint` appelée `arrayintbis` qui propose les mêmes fonctions, mais avec une fonction supplémentaire : Les instances de cette sous-classe mémoriseront à tout moment dans quelles cases des valeurs ont déjà été stockées, les autres cases contenant des « valeurs non initialisées ». Lors de l'accès en lecture à une case non initialisée (par `at` ou `[]`), une exception sera levée, ce qui facilitera le débogage du programme (même si Valgrind est capable de faire la même

chose, sans requérir la moindre modification du code source du programme). Attention lors de la copie : Copier un tableau contenant des cases non initialisées créera un tableau contenant des cases non initialisées, mais ne lèvera pas d'exception. Même chose pour la comparaison : comparer deux tableaux comparera uniquement les cases initialisées.

2. Résultats de loterie, utilisation de `std::map`

std::list, std::map, Exceptions, Surcharge d'opérateurs, Algorithmes de la bibliothèque standard.

`std::map` est un conteneur associatif, permettant d'associer des **valeurs** à des **clés**. Les types correspondant aux clés et aux valeurs sont les paramètres du modèle de classes : par exemple `std::map<std::string, int>` est un conteneur dans lequel les clés sont des chaînes et les valeurs des entiers. Sur un tel conteneur `m`, on pourra écrire `m["abc"] = 3;` car l'opérateur `[]` permet d'accéder à la valeur associée à la clé. Un parcours (par itérateur ou par boucle `for` d'intervalle) d'un `std::map` permet d'accéder à des valeurs qui sont de type `std::pair<clé,valeur>`, qui est une `struct` disposant de deux champs `first` (clé) et `second` (valeur). Ainsi `for (auto i : m) std::cout << i.first;` affichera toutes les clés d'un `std::map m`.
Le type utilisé pour la clé devra posséder un opérateur de comparaison < afin que `std::map` puisse comparer les clés et insérer un nouvel élément à l'endroit convenu dans la structure de données interne (qui est un arbre binaire de recherche afin de fournir des traitements avec une bonne complexité : insertion et accès en temps logarithmique).
Plus d'informations : http://en.cppreference.com/w/cpp/container/map . Notez que ce n'est que dans la classe `ensembleresultat` demandée ci-dessous que l'utilisation de `std::map` est requise.

1. Écrire une classe `resultat` qui mémorise le résultat d'un tirage de loterie. Un `resultat` sera formé de 5 entiers compris entre 1 et 49. Les entiers seront stockés dans une liste d'entiers, qui sera un attribut de la classe. Définir un constructeur sans argument (construisant une `resultat` vide), et faire en sorte que les `resultat` soient copiables.
2. Définir une méthode `ajouternumero` qui ajoute au `resultat` l'entier passé en paramètre. Cette méthode lèvera une exception dans le cas où l'entier passé en paramètre n'est pas compris entre 1 et 49 ou s'il figure déjà dans la liste des numéros du `resultat` (pour cela, vous n'écrirez pas une boucle de recherche mais utiliserez un algorithme de la bibliothèque standard http://en.cppreference.com/w/cpp/algorithm), ou si le `resultat` contient déjà 5 numéros. Vous utilisez une classe exception fournie par la bibliothèque standard afin d'éviter de définir une nouvelle classe : `std::invalid_argument` http://en.cppreference.com/w/cpp/error/invalid_argument
3. Écrire une méthode `trier` de `resultat` qui trie dans l'ordre croissant les numéros stockés dans la liste.
4. Écrire un opérateur de sortie de `resultat` sur un flux.
5. Écrire une classe `date` formée de trois attributs entiers : année, mois, jour. Définir constructeur, accesseurs, aucun mutateur (la date doit être fixée à la construction), opérateur d'affectation = et opérateur de sortie sur un flux.
6. Faire en sorte que deux dates puissent être comparées par les opérateurs classiques de comparaison : ==, !=, <, <=, >, >=.
7. Définir une classe `ensembleresultat` permettant de mémoriser un ensemble de résultats avec la date du tirage, en ayant un accès efficace au résultat d'une date donnée.
   Écrire une méthode `ajout` prenant comme paramètre une `date` et un `resultat` et ajoutant à l'ensemble le résultat associé à la date passée. S'il y avait déjà un résultat à cette date, il est remplacé par celui passé en paramètre.
   Écrire une méthode `resultatdu` qui en fonction d'une `date`, retourne le `resultat` correspondant à cette date (et lève une exception s'il n'y a pas de résultat mémorisé pour cette date).
   Écrire une méthode retournant l'ensemble trié des `dates` pour lesquelles un `resultat` est mémorisé.