Laboratoire 3

GLO-2100 Algorithmes et structures de données

Jonathan Gingras

1 La STL : standard template library

1.1 Introduction

La STL contient plusieurs classes et fonctions utilitaires.

Nous allons principalement nous concentrer sur les conteneurs, car ces derniers implémentent des structures de données standards que vous verrez en classe.

- Références des conteneurs

1.2 Namespace `std`

L'intégralité de la STL se trouve dans le namespace `std`.

Par exemple:

```
std::vector<...>
std::list<...>
std::map<...>
std::set<...>
```

etc.

1.3 Mémoire

L'allocation de la mémoire par les conteneurs de la STL utilise des classes qu'on appelle les allocateurs, qui allouent la mémoire sur le tas.

Vous n'avez donc pas besoin de gérer la mémoire vous même si vous storer des objets par valeur.

Mais n'oublier pas de libérer la mémoire si vous storez des pointeurs!

1.4 `std::vector`

La classe vector est une classe utilitaire permettant de manipuler un vecteur d'éléments en mémoire contiguë.

Par exemple:

```
std::vector<int> v; // vecteur de int

v.push_back(7); // ajoute l'élément 7 à v

v[0]; // accède à l'élément à l'index 0
```

Attention à ne pas accéder à un indice qui n'existe pas (qui dépasse la mémoire allouée), vous risquer d'accéder à de la mémoire illégale et faire planter le programme.

1.5 `std::list`

Une liste est une liste doublement chaînée. Elle est utile lorsque que vous n'avez pas énormément d'éléments et qu'ils n'ont pas besoin d'être contiguës en mémoire.

Par exemple:

```
std::list<int> l; // liste de int

l.push_back(7);
// ajoute un élément 7 à la fin

l.push_front(7);
// ajoute un élément 7 au début de la liste
```

Une liste n'a pas d'opérateur [], car cet opérateur signifie l'accès contiguë.

1.6 Les itérateurs

Pour itérer sur les éléments d'un conteneur de la STL, on utilise les classes membres `iterator`.

Par exemple:

```
std::vector<int> v {1, 2, 3, 4};

for(std::vector<int>::iterator it = v.begin();
    it != v.end(); ++it) {
  std::cout << *it << std::end;
}
```

1.7 Les itérateurs (suite)

- opérateur ++ pour avancer au prochain élément
- begin(): un itérateur vers le premier éléments
- end(): un itérateur vers *après* le dernier, donc ne pas le déréférencer! utile principalement pour utiliser `!=` sur lui.
- la déréférence (`*`) d'un itérateur est la référence vers l'élément itéré

1.8 Les itérateurs (raccourcis en C++11)

En C++11, il existe un raccourcis: les `for-range loops`

Par exemple (même effet que page précédente):

```
std::vector<int> v {1, 2, 3, 4};

for(int &i : v) {
  std::cout << i << std::end;
}
```

1.9 Les itérateurs (raccourcis en C++11) (suite)

- La déréférence est automatique dans les for-range loops.
- Notez que `i`, ici, est une référence (&), car on ne veut pas de construction par copie de chaque élément, donc on écrit un (&).
- Le dernier point reste valide si vous utilisez l'inférence de type («`auto &`» et non «`auto`»).
- Vous pouvez aussi ajouter un `const` devant le type si vous ne modifiez pas les itérés.

1.10 Exercice

Voir `src/iterating.cc`.

Voir `src/iterable_class.cc`.

1.11 Considérations importantes sur l'héritage

- N'héritez **pas** des classes de la STL!
- Ça va compiler, mais si vous vous servez de façon polymorphique de ce genre de classe, vous allez leaker : **aucun destructeur** des conteneurs de la STL **n'est virtuel**.
- Allez-y par composition à la place.

1.12 Autres structures de données

D'autres structures de données spécialisées existent dans la STL:

- `std::map` : association clé-valeur, généralement implémenté en arbre
- `std::unordered_map` (C++11) : association clé-valeur, implémenté en table de hashage
- `std::set` : ensemble, généralement implémenté en arbre
- `std::unordered_set` (C++11) : ensemble, implémenté en table de hashage
- `std::stack` : pile FILO, non itérable
- `std::queue` : file FIFO, non itérable

1.13 `std::map`

Pour instantier une `std::map`, il faut sélectionner deux types (clé et valeur). Le type clé doit implémenter l'opérateur `<` pour ses comparaisons internes.

Par exemple, une `std::map<std::string, int>` associe des strings à des ints.

Seulement une valeur peut être associée à une clé.

Si on veut avoir plusieurs valeurs par clé, il faut utiliser la classe `std::multimap`.

1.14 Exemple `std::map`

```
std::map<std::string, int> map;

map["one"] = 1;
// assigne 1 au string "one"

int two = map["two"];
// assigne le int two au int assigné au string "two"
// dans ce cas, ce sera 0, car l'opérateur []
// doit toujours retourner une référence valide
// même si elle n'a pas été assignée avant
```

1.15 Exemple `std::map` (suite)

```
map.at("tree");
// va lancer une exception, car .at agit
// comme l'opérateur [], mais ne crée pas l'élément
// s'il n'existe pas

std::map<std::string, int> map2 {
```

```
  {"one", 1},
  {"two", 2},
  {"three", 3}
};
// on peut initialiser une map avec
// une liste de valeurs de paires en C++11
```

1.16 Exemple `std::map` (suite)

```
for(auto &pair: map2) {
  std::cout << "key: " << pair.first
  << ", " << "value: " << pair.second;
}
// on peut itérer sur une map avec une
// std::pair
```

1.17 Exemple `std::unordered_map`

Une `std::unordered_map` s'utilise comme une `std::map`, toutefois, elle est implémentée en table de hachage et non en arbre de recherche.

Elle est donc plus rapide en accès direct, mais plus lente en itération.

Le type clé ici n'a pas besoin d'implémenter l'opérateur `<`, mais il doit spécialiser le type `std::hash` et y implémenter l'opérateur parenthèse (qui représente la fonction de hachage).

1.18 Exemple `std::str`

```
#include <unordered_map>

class MyClass {};

namespace std {
template <>
struct hash<MyClass> {
  size_t operator () () {
    return 0;
    // mauvaise fonction de hashage,
    // mais fonctionne
  }
};
}
```

1.19 Exercice

Voir `src/maps.cc`

Voir `src/stacks_queues.cc`

1.20 Construction par copie

Lors du dernier lab, on a vu que l'a construction par copie coûte cher sur les conteneurs de la STL, car leurs constructeurs par copie implémentent une copie profonde de leur contenu.

Nous allons faire des comparaisons de temps pour illustrer la situation.

1.21 Exercice

Voir `src/time_comparison.cc`