Algorithmes et structures de données 2
Laboratoire n°4 : Tables de hachage

06.12.2017

Introduction

Dans ce laboratoire nous allons étudier les tables de hachage et plus particulièrement différentes fonctions de hachages utilisées pour remplir une structure du type `std::unordered_set`.

Théorie

La structure `std::unordered_set`[1], qui est une table de hachage de la librairie *stl* en C++, a besoin d'une fonction de hachage pour fonctionner. Il y a 2 possibilités de la définir : en la passant comme second argument template de `std::unordered_set` ou en spécialisant l'opérateur `()` de `std::hash()` pour le type que nous souhaitons mettre dans la table. Il est aussi nécessaire de définir une fonction d'égalité selon les mêmes principes ou encore en surchargeant l'opérateur ==.

Exemple par template :

```
template <typename Number>
class MyValueByTemplate {
private:
   Number n;
public:
   MyValueByTemplate(Number n) : n(n) {}
   Number getN() const { return this->n; }
   //equal operator
   bool operator ==(const MyValueByTemplate<Number> &that) const {
      return this->n == that.n;
   }
};
```

[1] Documentation : http://www.cplusplus.com/reference/unordered_set/unordered_set/

```
#include <unordered_set>

//functor
template <typename Number>
struct HashFunctionForMyValueByTemplate {
   size_t operator() (const MyValueByTemplate<Number> val) const {
      return val.getN();
   }
};

int main(int argc, const char * argv[]) {
   unordered_set<MyValueByTemplate<double>, HashFunctionForMyValueByTemplate<double>> set;
   MyValueByTemplate<double> v (23);
   set.insert(v);
   return EXIT_SUCCESS;
}
```

Exemple par spécialisation :

```
#include <functional>

template <typename Number>
class MyValueBySpecialization {
private:
   Number n;
public:
   MyValueBySpecialization(Number n) : n(n) {}
   Number getN() const { return this->n; }
   // equal operator
   bool operator ==(const MyValueBySpecialization<Number> &that) const {
      return this->n == that.n;
   }
};

//hash function
namespace std {
   template <typename Number>
   struct hash<MyValueBySpecialization<Number>>  {
      size_t operator()(const MyValueBySpecialization<Number>& s) const {
         return hash<Number>()(s.getN()); *
      }
   };
}
```

```
#include <unordered_set>

int main(int argc, const char * argv[]) {
    unordered_set<MyValueBySpecialization<double>> set;
    MyValueBySpecialization<double> v (23);
    set.insert(v);
    return EXIT_SUCCESS;
}
```

La librairie STL fourni déjà les implémentations des spécialisations de `std::hash()` pour les types le plus courants[2]. *C'est d'ailleurs la raison pour laquelle dans le second exemple ci-dessous, nous avons pu faire un *return hash<Number>()(s.getN());* qui a l'exécution fera appel à *hash<double>()* qui est défini dans le header <functional>. Dans ce laboratoire nous allons utiliser la spécialisation de la fonction template `std::hash()`.

Durée

- 4 périodes
- A rendre le dimanche **17.12.2017** à **23h55** au plus tard

Donnée

Vous trouverez les structures et exemples fournis sur la page Moodle du cours :
https://cyberlearn.hes-so.ch/course/view.php?id=10182

A faire :

- **Partie 1**
  Nous vous fournissons 6 classes équivalentes qui définissent 6 fonctions de hachage de différents types ainsi qu'une fonction de test permettant de créer et remplir des tables des hachages, utilisant ces classes avec des données et affichant quelques statistiques.
  Pour cette première partie, vous documenterez dans votre rapport les différences constatées entre ces différentes fonctions, leurs avantages et inconvénients ainsi que votre point de vue sur quelle fonction vous semble la plus adaptée à ce jeu de donnée. N'hésitez pas à ajouter des graphiques dans votre rapport pour étayer vos propos. Vous pouvez passer d'un jeu de données à l'autre et modifier le max_load_factor[3] des tables de hachage à partir des #define du fichier `main.cpp`. Vous discuterez aussi des effets de la variation de ces 2 paramètres.
  Vous êtes bien entendu autorisés (et même encouragés) à ajouter d'autres classes, implémentant d'autres types de fonctions de hachage ou d'autres indicateurs statistiques et de les intégrer à votre discussion. La classe

[2] Documentation : http://www.cplusplus.com/reference/functional/hash/
[3] Quelle est la valeur par défaut pour l'implémentation de l'`unordered_map` fournie par la *STL* ? Comment est-ce que les collisions sont gérées dans cette implémentation ? Quelle est la valeur conseillée pour ce type de résolution des collisions ?

`DirectoryWithoutAVS` fait partie de la seconde partie et n'a pas besoin d'être intégrée à votre discussion.

- **Partie 2**
  Le but de cette partie est que vous définissiez vous-même une fonction de hachage pour une structure similaire à celle de la première partie de ce laboratoire, mais cette fois-ci sans le numéro AVS (nous avons gardé la même signature pour le constructeur de cette classe `DirectoryWithoutAVS` afin de simplifier les fonctions de tests, merci de considérer que vous n'y avez pas accès).
  Le but de cette seconde partie est que vous amélioriez la fonction de hachage fournie dans cette classe. Vous débuterez par expliquer pourquoi les résultats obtenus par cette fonction sont généralement moins bons que la plupart de ceux obtenus dans la première partie. Ensuite vous identifierez théoriquement les points à améliorer et les implémenterez. Vous discuterez ensuite du résultat obtenu. Vous indiquerez toutes vos démarches et réflexions dans votre rapport.

Compilation

Ce laboratoire nécessite l'utilisation de librairies externes afin de pouvoir utiliser les fonctions de hachage *cityhash*[4] et *sha256*[5]. La compilation a été testée avec succès sous *Windows* avec l'environnement de référence pour le cours (*Netbeans 8* + *MinGW* + *GCC 4.8.1*), sous *Ubuntu* 14.04 LTS, 64-bit, avec *GCC 4.8.4* et sous *Mac* avec *Xcode 7*.

Le code utilisant *sha256* aura un comportement différent sous un environnement 32-bit ou 64-bit, pour cela un warning sera levé à la compilation pour vous indiquer celui qui vous utilisez. Vous indiquerez dans votre rapport : le système d'exploitation utilisé, la version de votre compilateur, votre environnement et les options de compilation utilisées. Par exemple, voici une commande de compilation fonctionnant sous *Windows* :

```
g++ -std=c++11 -Wall *.cpp libs\*.c* -o labo4.exe
```

Rendu/Evaluation

En plus de votre code, vous remettrez un **rapport** comportant au minimum une introduction, vos réflexions et analyses pour les deux parties ainsi qu'une conclusion. Merci de soigner la présentation et l'orthographe de votre rapport.

Merci de rendre votre travail sur *CyberLearn* dans un fichier *zip* unique, dont le nom respecte les consignes.

Bonne chance !

[4] *CityHash*, family of hash functions for String. Google implementation. MIT license
http://google-opensource.blogspot.ch/2011/04/introducing-cityhash.html
[5] *Sha2* implementation. BSD license
http://www.ouah.org/ogay/sha2/