Github: LoreDN
Author: Lorenzo Di Napoli
Repository: https://github.com/LoreDN/code-C++

myHash — Documentation

Contents

Library Description

This library contains the definition of the *"myHash"* class, which allows to works with a set implementation of an Hash-Table in **C++**.
Since an Hash-Table can be used following two different criterias (open/close hashing), there are two implementations, the *"myHash_ Open"* one and the *"myHash_ Close"* one.

CONTENTS OF THE LIBRARY

The library consists of two files:

- **"myHash.hpp":** the template library, it is the header file where you can find a brief description on how to use the library in practice, together with all the classes definitions.
- **"myHash.cpp":** the core of the library, it contains all the methods definitions for each class; it is the file that has to be given to the compiler in order to be able to use the library.

INHERITARY STRUCTURE

The library starts with the definition of a simple abstract class: *"myHash"*.
Starting from this class, there are two derivated classes, *"myHash_ Open"* and *"myHash_ Close"*, which allow to use the core class with open/close hashing, implementing both the types of Hash-Tables.

TEMPLATES IMPLEMENTATION

In order to use the Hash-Tables with all possibles types and custom classes, have been used **templates**.
Due to the library structure, the correct way to use them is via **explicit template instantiations**, which have to be manually added for each wanted custom type/class.
This process has been made as easy as possible; the only thing you need to do is to add the wanted **explicit template instantiation** at the start of the *"myHash.cpp"* file. Have been already added the *int* and *float* types ones, so you can copy one of these and change the type with the wanted one.

Classes Documentation

In the *"myHash"* library have been implemented a total of three classes:

R. **myHash:** the Root abstract class, from which all the others derive from.

1. **myHash_Open:** Hash-Table implementation following open hashing criteria (Bucket implemented as Linked Lists).
2. **myHash_Close:** Hash-Table implementation following close hashing criteria (Bucket implemented as single elements).

ROOT. myHash

```
template <typename TYPE>
class myHash {

     protected:

          // attributes
          size_t dimension;
          size_t elements = 0;

     public:

          // destructor
          virtual ~myHash() = default;

          // methods prototypes
          virtual void add(const TYPE key) = 0;
          void scan(const size_t num);
          void scan_file(const std::string path);

          virtual void print(const size_t start, const size_t end, const bool
              flag_user_interface) = 0;
          virtual void print_file(const std::string path, const size_t start, const
              size_t end, const bool flag_user_interface) = 0;

          virtual void remove(const TYPE key) = 0;
          virtual void remove_Bucket(const size_t bucket) = 0;
          inline float load_factor() { return (float)this->elements /
              (float)this->dimension; }

};
```

`size_t dimension;`

The Hash-Table size, it is a *protected* member, since the user is not able to modify it.

```
size_t elements = 0;
```

The number of elements curently stored in the Hash-Table, it is a *protected* member, since the user is not able to modify it.

```
virtual  myHash() = default;
```

This is the *destructor*, it has been left as a *virtual method* in order to allow each derivated class to implement its own one.

```
virtual void add(const TYPE key) = 0;
```

Method to add an element to the Hash-Table, it has been left as a *virtual method* since each derivated class has a different hashing criteria.

```
void scan(const size_t num);
```

Method to scan a total of `num` elements from terminal.

```
void scan_file(const std::string path);
```

Method to scan all the elements from a file via std::ifstream.

```
virtual void print(const size_t start, const size_t end, const bool flag_user_interface) = 0;
```

Method to print the Hash-Table to the terminal, it has been left as a *virtual method* since each derivated class has a different hashing criteria.

```
virtual void print_file(const std::string path, const size_t start, const size_t end,
const bool flag_user_interface) = 0;
```

Method to print the Hash-Table to a file, it has been left as a *virtual method* since each derivated class has a different hashing criteria.

```
virtual void remove(const TYPE key) = 0;
```

Method to remove an element from the Hash-Table, it has been left as a *virtual method* since each derivated class has a different hashing criteria.

```
virtual void remove_Bucket(const size_t bucket) = 0;
```

Method to remove a Bucket from the Hash-Table, it has been left as a *virtual method* since each derivated class has a different hashing criteria.

```
inline float load_factor() { return (float)this->elements / (float)this->dimension; }
```

Method to calculate the load factor of the Hash-Table.

1. myHash_Open

```
template <typename TYPE>
class myHash_Open :  public myHash<TYPE> {

    public:

        // Bucket definition
        typedef struct myHash_Node {

            TYPE value;
            myHash_Node *next;

        }myBucket;

    private:

        // attributes
        myBucket **table;

        // methods prototypes
        inline size_t hash(const TYPE key) { return abs(key) % this->dimension; }
        myBucket *add_Bucket(const TYPE key);

    public:

        // constructor
        myHash_Open(const size_t dim);

        // destructor
        ~myHash_Open() override;

        // methods prototypes
        void add(const TYPE key) override;

        void print(const size_t start, const size_t end, const bool
            flag_user_interface) override;
        void print_file(const std::string path, const size_t start, const size_t end,
            const bool flag_user_interface) override;

        myBucket *find(const TYPE key);
        void remove(const TYPE key) override;
        void remove_Bucket(const size_t bucket) override;
        void copy(myHash_Open *destination);

};
```

```
// Bucket definition
typedef struct myHash_Node {

     TYPE value;
     myHash_Node *next;

}myBucket;
```

Definition of the struct *"myHash_Node"*, used as the Node of a Linked List, wich represents the Bucket of the Hash-Table following the open hashing criteria.

```
myBucket **table;
```

The Hash-Table, it is a *private* member, since the user is not able to modify it directly.

```
inline size_t hash(const TYPE key) { return abs(key) % this->dimension; }
```

Method for the Hash function, it is a *private* member, since the user is not able to call it directly.

```
myBucket *add_Bucket(const TYPE key);
```

Method to add a Bucket to the Hash-Table, it is a *private* member, since the user is not able to call it directly.

```
myHash_Open(const size_t dim);
```

This is the *constructor*, which assignes a size variable `dim` to the protected member `myHash.dimension`.

It also sets the Hash-Table `myHash_Open.table` to **nullptr**.

```
~myHash_Open() override;
```

This is the *destructor*, which frees the Hash-Table `myHash_Open.table`.

```
void add(const TYPE key) override;
```

Method to add a key to the Hash-Table.

```
void print(const size_t start, const size_t end, const bool flag_user_interface) override;
```

Method to print Hash-Table to the terminal.
The `const bool flag_user_interface` is used in order to choose the type of print:

- **flag = 1** —> `- Bucket:  i "\n" Element j :  myHash_Open.table[ i ] "\t" ...  "\n"`.
- **flag = other values** —> `myHash_Open.table[ i ] "\t" ...  "\n" myHash_Open.table[ i + 1 ] ...`.

```
void print_file(const std::string path, const size_t start, const size_t end, const
bool flag_user_interface) override;
```

Method to print the Hash-Table to file via std::ofstream.
The `const bool flag_user_interface` is used in order to choose the type of print:

- **flag = 1** —> `- Bucket:  i "\n" Element j :  myHash_Open.table[ i ] "\t" ...  "\n"`.
- **flag = other values** —> `myHash_Open.table[ i ] "\t" ...  "\n" myHash_Open.table[ i + 1 ] ...`.

```
myBucket *find(const TYPE key);
```

Method to find a key in the Hash-Table.

```
void remove(const TYPE key) override;
```

Method to remove a key from the Hash-Table.

```
void remove_Bucket(const size_t bucket) override;
```

Method to remove an entire Bucket from the Hash-Table, then sets it to **nullptr**.

```
void copy(myHash_Open *destination);
```

Method to copy the Hash-Table to another object `myHash_Open`.

2. myHash_Close

```
template <typename TYPE>
class myHash_Close :  public myHash<TYPE> {

     // set constants
     TYPE const EMPTY = __INT_MAX__;
     TYPE const TOMBSTONE = __INT_MAX__ - 1;

     private:

          // attributes
          TYPE *table;
          bool flag_probing;

          // methods prototypes
          int hash(const TYPE key);
          int linear_probing(const int key_abs);
          int quadratic_probing(const int key_abs);

     public:

          // constructor
          myHash_Close(const size_t dim, const bool flag);

          // destructor
          ~myHash_Close() override;

          // methods prototypes
          void add(const TYPE key) override;

          void print(const size_t start, const size_t end, const bool
              flag_user_interface) override;
          void print_file(const std::string path, const size_t start, const size_t end,
              const bool flag_user_interface) override;

          int find(const TYPE key);
          void remove(const TYPE key) override;
          void remove_Bucket(const size_t bucket) override;
          void copy(myHash_Close *destination);

};
```

```
TYPE const EMPTY = __INT_MAX__;
```

Constant to mark an Empty Bucket.

```
TYPE const TOMBSTONE = __INT_MAX__ - 1;
```

Constant to mark a Bucket which has been removed.

```
myBucket **table;
```

The Hash-Table, it is a *private* member, since the user is not able to modify it directly.

```
bool flag_probing;
```

Flag used in order to chose the proibing method (if necessary), it is a *private* member, since the user is not able to modify it directly.

```
int hash(const TYPE key);
```

Method for the Hash function, it is a *private* member, since the user is not able to call it directly.

```
int linear_probing(const int key_abs);
```

Method to find an avaible Bucket with *linear probing*, it is a *private* member, since the user is not able to modify it directly.

```
int quadratic_probing(const int key_abs);
```

Method to find an avaible Bucket with *linear probing*, it is a *private* member, since the user is not able to modify it directly.

```
myHash_Close(const size_t dim, const bool flag);
```

This is the *constructor*, which assignes a size variable `dim` to the protected member `myHash.dimension` and a boolean `flag` to the private member `myHash_Close.flag_probing`.

It also sets the Hash-Table `myHash_Close.table` Buckets to **EMPTY**.

```
~myHash_Close() override;
```

This is the *destructor*, which frees the Hash-Table `myHash_Close.table`.

```
void add(const TYPE key) override;
```

Method to add a key to the Hash-Table.

```
void print(const size_t start, const size_t end, const bool flag_user_interface) override;
```

Method to print Hash-Table to the terminal.
The `const bool flag_user_interface` is used in order to choose the type of print:

- **flag = 1** —> `- Bucket i:  myHash_Close.table[ i ] "\n"`.
- **flag = other values** —> `myHash_Close.table[ i ] "\n"`.

```
void print_file(const std::string path, const size_t start, const size_t end, const
bool flag_user_interface) override;
```

Method to print the Hash-Table to file via std::ofstream.
The `const bool flag_user_interface` is used in order to choose the type of print:

- **flag = 1** —> `- Bucket i:  myHash_Close.table[ i ] "\n"`.
- **flag = other values** —> `myHash_Close.table[ i ] "\n"`.

```
int find(const TYPE key);
```

Method to find a key in the Hash-Table.

```
void remove(const TYPE key) override;
```

Method to remove a key from the Hash-Table, then sets the Bucket to **TOMBSTONE**.

```
void remove_Bucket(const size_t bucket) override;
```

Method to remove an entire Bucket from the Hash-Table, then sets it to **TOMBSTONE**.

```
void copy(myHash_Close *destination);
```

Method to copy the Hash-Table to another object `myHash_Close`.