CSC 265: Assignment 1

Dates

- Demo 1: in lab the week of 14 September
- Demo 2: in lab the week of 21 September
- Demo 3: in lab the week of 28 September
- Final submit deadline: Saturday 3 October at 10:00 AM

Summary

Develop a C++ implementation of the `string_set` class using a hash table.

Download and unpack the assignment files

1. Download. Use the web browser to download assignment_01.zip and save it in your home directory.
2. Unpack. Right click on the zip file and select extract.
3. Check files.

   Use the file manager to check that directory `assignment_01` contains the files:

   `string_set.h` and `string_set.cpp`
   : a declaration and a skeleton implementation of the `string_set` class.

   `test_A.cpp`
   : a test program focusing on the `string_set` constructor, and the `add` and `contains` methods.

   `test_A_gold.txt`
   : contains the correct output of `test_A.cpp`.

   `test_B.cpp`
   : a test program focusing on the `string_set` destructor, and the `remove` method.

   `test_B_gold.txt`
   : contains the correct output of `test_B.cpp`.

   `test_C.cpp`
   : a test program focusing on the `reset` and `next` methods.

   `test_C_gold.txt`
   : contains the correct output of `test_C.cpp`.

Deliverable: **`string_set.cpp`**

Specification

Each instance of the `string_set` class stores a set of strings.

Strings may be added, removed and checked for membership. An *iterator* is also provided, allowing the user to retrieve the stored elements one at a time.

The correct behaviour of the public methods is specified in the file `string_set.h`. Your code must be correct with respect to those specifications.

Implementation

Your implementation must be contained completely in the eight functions contained in `string_set.cpp`.

Do not modify any other part of `string_set.cpp`.

Do not modify `string_set.h` in any way.

`hash_table`
: The variable `hash_table` stores the head element of `HASH_TABLE_SIZE` linked lists. For *i* between 0 and `HASH_TABLE_SIZE-1`, the list headed by `hash_table[`*i*`]` must contain only strings which `hash_function` maps to *i*.

`iterator_index` and `iterator_node`
: The iterator uses `iterator_index` and `iterator_node` to maintain the current position of the iterator. While elements remain to be returned, the value of `iterator_index` must be between 0 and `HASH_TABLE_SIZE`-1. The variable `iterator_node` must be either NULL or the address of a node in the list headed by `hash_table[iterator_index]`. When no more elements remain to be returned, the value of `iterator_index` must be `HASH_TABLE_SIZE`.

`hash_function`
: Implement `hash_function` to map a string to an integer between 0 and `HASH_TABLE_SIZE`-1. Follow the specification contained in `string_set.h`.

`string_set` constructor
: Initialize `hash_table`, `iterator_index` and `iterator_node`.

`add(`*s*`)`
: If *s* is present, throw `duplicate_exception`.

  Otherwise, allocate space for a new `node` and for *s*. Insert the new node at the front of the list headed by `hash_table[`*i*`]`, where *i* is `hash_function(`*s*`)`.

`remove(`*s*`)`
: If *s* is not present, throw `not_found_exception`.

  Otherwise, remove the node containing *s*, freeing the space for the node and *s*.

`contains(`*s*`)`
: If *s* is present, return 1; otherwise return 0;

`reset`
: Reset the iterator to the first element.

`next`
: Return the next available string, or `NULL` if no more elements remain.

  The `next` function first returns the elements in the list headed by `hash_table[0]`, in the order they appear in the list. It then returns the elements in the list headed by `hash_table[1]`, in the order they appear in the list and so on.

`string_set`destructor
: Delete all dynamic storage allocated for nodes and strings.

Development steps

Step A

- Implement the `string_set` constructor, and the `hash_function`, `add` and `contains` functions.
- Compile and run the code:

```
g++ -g -Wall -o test_A test_A.cpp string_set.cpp
./test_A
```

- Compare your output to the correct output in `test_A_gold.txt`.

Step B

- Implement the `string_set` destructor and the `remove` method.
- Compile and run the code:

```
g++ -g -Wall -o test_B test_B.cpp string_set.cpp
./test_B
```

- Compare your output to the correct output in `test_B_gold.txt`.

Step C

- Implement the `reset` and `next` functions.
- Compile and run the code:

```
g++ -g -Wall -o test_C test_C.cpp string_set.cpp
./test_C
```

- Compare your output to the correct output in `test_C_gold.txt`.