UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science
Faculty of Engineering, Built Environment & IT
University of Pretoria

COS110 - Program Design: Introduction

Practical 7 Specifications: Templates

Release date: 09-10-2023 at 06:00
Due date: 13-10-2024 at 23:59
Total Marks: 111

Contents

1 General Instructions

- ***Read the entire assignment thoroughly before you start coding.***
- This assignment should be completed individually, no group effort is allowed.
- **To prevent plagiarism, every submission will be inspected with the help of dedicated software.**
- The following imports are allowed ("<iostream>", "<sstream>", "<string>", "Dictionary.cpp", "Dictionary.h", "Tuple.h", "Tuple.cpp", "ScoreBoard.h", "ScoreBoard.cpp","<cstddef>").
- Be ready to upload your assignment well before the deadline, as **no extension will be granted.**
- If your code does not compile, you will be awarded a mark of 0. The output of your program will be primarily considered for marks, although internal structure may also be tested (eg. the presence/absence of certain functions or classes).
- Failure of your program to successfully exit will result in a mark of 0.
- Note that plagiarism is considered a very serious offense. Plagiarism will not be tolerated, and disciplinary action will be taken against offending students. Please refer to the University of Pretoria's plagiarism page at http://www.ais.up.ac.za/plagiarism/index.htm.
- Unless otherwise stated, the usage of c++11 or additional libraries outside of those indicated in the assignment, will not be allowed. Some of the appropriate files that you have submitted will be overwritten during marking to ensure compliance to these requirements. **Please ensure you use c++98**

- We will be overriding your .h files, therefore do not add anything extra to them that is not in the spec
- In your student files you will only find a skeleton main
- If you would like to use *using namespace std*, please ensure it appears in your cpp that you upload

2 Plagiarism

The Department of Computer Science considers plagiarism as a serious offence. Disciplinary action will be taken against students who commit plagiarism. Plagiarism includes copying someone else's work without consent, copying a friend's work (even with consent) and copying material (such as text or program code) from the Internet. Copying will not be tolerated in this course. For a formal definition of plagiarism, the student is referred to http://www.library.up.ac.za/plagiarism/index.htm (from the main page of the University of Pretoria site, follow the Library quick link, and then choose the Plagiarism option under the Services menu). **If you have any form of question regarding this, please ask one of the lecturers, to avoid any misunderstanding.** Also note that the OOP principle of code re-use does not mean that you should copy and adapt code to suit your solution.

3 Outcomes

The goal of this practical is to gain experience with working with templates.

4 Introduction

Implement the UML diagram and functions as described on the following pages.

5 Class Diagram

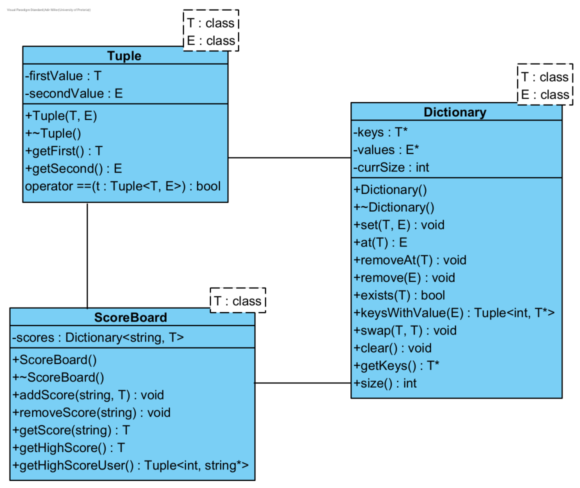

Figure 1: Class diagrams

6 Includes

For this practical, please take note of how we have done our includes in our .h files. This is just for the classes, I am not listing things like <iostream>
Ditionary.h:

```
#include "Tuple.h"
#include "Tuple.cpp"
```

Tuple.h: No class includes
ScoreBoard.h:

```
#include "Dictionary.h"
#include "Dictionary.cpp"
#include "Tuple.h"
#include "Tuple.cpp"
```

main.cpp:

```
#include "Dictionary.h"
#include "Dictionary.cpp"
#include "Tuple.h"
#include "Tuple.cpp"
#include "ScoreBoard.h"
#include "ScoreBoard.cpp"
```

It is **crucial** that you add header guards to your .cpp files. For example:

```
#ifndef DICTIONARY_CPP
#define DICTIONARY_CPP
// The rest of the file
#endif
```

7 Classess

7.1 Tuple<T, E>

- Members
  - firstValue: T
  - secondValue: E
- Functions
  - Tuple(first: T, second: E)
    - The constructor for the Tuple class, it should set the member variables.
  - ~Tuple()
    - The destructor for the Tuple class, it should be present but empty.
  - getFirst(): T
    - Returns the *firstValue* member variable.
  - getSecond(): E
    - Returns the *secondValue* member variable.

    - operator==(rhs: Tuple<T,E>): bool
        - The overloaded operator==, should return true if both the *firstValue* of *rhs* matches the current objects *firstValue* and the *secondValue* of *rhs* matches the current objects value of *secondValue*.

- Example usage of Tuple:

```cpp
Tuple<int,int> t(1,2);
//or
Tuple<int,std::string> t2(1,"a");
std::cout<<t2.getSecond()<<std::end; //outputs "a"
```

7.2 Dictionary<T,E>

Before we speak about the class, what is a Dictionary in programming? It is extremely simple, it is a data structure that holds Key-Value pairs. Essentially, it enables us to associate and retrieve data by using specific and unique keys (they need to be unique to be able to identify a value without ambiguity). For instance, you can think of it as a system that maps a person's student number to their corresponding name. See `https://en.wikibooks.org/wiki/A-level_Computing/AQA/Paper_1/Fundamentals_of_data_structures/Dictionaries`.

- Members
    - keys: T*
        - This is a dynamic array of the keys, notice that it is a pointer.
    - values: E*
        - This is a dynamic array of the values, notice that it is a pointer.
    - currSize: int
        - This is an integer variable that tracks the size of the arrays since they will be resized when adding/removing keys and values. An important note, if the size variable is 0, then the arrays at *keys* and *values* should be deleted.

- Functions
    - Dictionary()
        - The constructor for the Dictionary class, the only thing this should do is set *currSize* to 0 and *keys* and *values* to NULL.
        - You will notice you cannot set *keys* and *values* to NULL without including `#include <cstddef>`
    - ~Dictionary()
        - Destructor for the Dictionary class. It should delete the arrays if they are present.
    - set(T key, E value): void
        - This function should set the value at a particular key. Keep in mind that keys are unique.
        - We will be using the concept of parallel arrays for the keys and values arrays.
        - To set this, loop through *keys* and check if there is a key that exists that matches *key*, set its corresponding value to the new value in the *values* array.
        - For example, if we have a Dictionary<string,int> with the current state being:

          ```
          keys := ["Jack","John","Steve"]
          values := [54,98,32]
          ```

And *set("John",43)* is called, then the arrays should look like the following:

```
keys := ["Jack","John","Steve"]
values := [54,43,32]
```

  * If there is no key in the array that matches *key*, you need to make one. This is done by resizing both arrays to a size one larger than their current size and appending the key and value at the end of their respective arrays.[1] i.e. let's say after the *set("John",43)* is called from earlier, *set("Lee",25)* is called. The array's will look like the following:

```
keys := ["Jack","John","Steve","Lee"]
values := [54,43,32,25]
```

  * If currSize is 0 and there are no arrays, they will need to be created.

- at(key: T): E
  * This function should return the correct value given the key that is passed in. If we go back to the example from earlier:

```
keys := ["Jack","John","Steve","Lee"]
values := [54,43,32,25]
```

    then *at("John")* should return *43*.
  * If a key that does not exist in the array is passed in, you should return the default value. For example, given *template<class E>*, to retrieve the default value, simply use *return E()*.

- removeAt(key: T): void
  * This function should remove the value at the passed key.
  * Once This value has been removed, the array needs to be resized to one smaller than its original size. It is up to you how to arrange the elements in the array to take the space of the missing element.
  * If using the example from earlier:

```
keys := ["Jack","John","Steve","Lee"]
values := [54,43,32,25]
```

    and remove("John") is called, the arrays should look something like the following:

```
keys := ["Jack","Steve","Lee"]
values := [54,32,25]
```

  * If the key does not exist, do nothing.
  * If all items of the Dictionary are removed, delete the arrays and set them to NULL.

- remove(value: E): void
  * This function will remove all instances of a particular value. Remember that only keys are unique, there can be several values that have the same value.
  * Say we have the following:

```
keys := ["Leonard","Jackie","Albert","Bob"]
values := [56,74,91,74]
```

    and *remove(74)* is called, then our arrays would look like the following:

```
keys := ["Leonard","Albert"]
values := [56,91]
```

  * Remember to resize the array after the removal.
  * If no values match, do nothing.
  * If all items of the Dictionary are removed, delete the arrays.

[1]Hint: Create a new array and copy over the data

- exists(key: T): bool
  - This returns true if the passed in key exists in the Dictionary and false otherwise.
- keysWithValue(value: E): Tuple<int,T*>
  - This function will return a Tuple with an array, and the size of the array. Note the order, it is important.
  - This function returns the array of all the keys that have the passed in value.
  - For example:

    ```
    keys := ["Leonard","Jackie","Albert","Bob"]
    values := [56,74,91,74]
    ```

    and *keysWithValue(74)* is called, then a Tuple with an int of 2 and a string array containing "Jackie" and "Bob" should returned.
  - If no values match, return an int of 0 and an empty string array.
  - When deleting this array (like in your main.cpp, you cannot delete like this in your Dictionary destructor), you can do it as such:

    ```
    Tuple<int,string*> test= dict.keysWithValue(22);
    //you can delete the array as follows
    delete[] test.getSecond();
    ```
- swap(firstKey: T, secondKey: T): void
  - Assuming both keys are in the array, this function should swap the values of the keys.
- clear(): void
  - This function simply clears and deletes both arrays and resets *currSize*. The arrays need to be set to NULL.
- getKeys(): T*
  - Getter for *keys*.
- size(): int
  - Getter for *currSize*.

7.3 ScoreBoard<T>

- Members
  - scores: Dictionary<string,T>
    - A Dictionary of usernames and their respective score.
- ScoreBoard()
  - The constructor for ScoreBoard, this should not do anything but must be present.
- ~ScoreBoard()
  - The destructor for ScoreBoard, this should not do anything but must be present.
- addScore(user: string, score: T): void
  - This function adds/updates a score for a particular user.
  - Since this is a ScoreBoard, only the high scores are recorded. Therefore, if the user already has a score, only update it if the new score is larger.
- removeScore(user: string): void

  - This function removes a user from the dictionary.
  - If the user does not exist in the dictionary then do nothing.

- getScore(user: string): T
  - Return the score of a user. If the key is not in the array, then return the default value. [2]

- getHighScore(): T
  - Return the highest score in the dictionary.
  - If there are no scores then, return the default value (done in the same way as *at()*).

- getHighScoreUser(): Tuple<int, string*>
  - Return a Tuple with an array and the size of the array. Note the order, it is important.
  - This array should be all users that have the highest score.
  - If there are no users in the system, return an int of 0 and an empty string array.

8 Common Errors

Undefined Reference:
If you try and declare an object of:

```
Dictionary<int,int> obj;
```

And you get the error:

```
undefined reference to 'Dictionary<int, int>::Dictionary()'
```

It most likely means you forgot to include you .cpp file in the file, remember with templates we need both the .h and the .cpp included.
Prev Declared:
If you see

```
note: 'int Dictionary<T, E>::size()' previously declared here
194 | int Dictionary<T, E>::size()
```

This means that you most likely forgot the guards, since we are including the cpp in multiple files we need

```
#ifndef DICTIONARY_CPP
#define DICTIONARY_CPP
// The rest of the file
#endif
```

NULL
If you see an error about ‘NULL‘ not being declared you need to include

```
#include <cstddef>
```

[2]Remember, to get the default value, use *return T();*

9 Memory Management

Memory management is a core part of COS110 and C++, so this practical memory management is extremely important due to the scope. Therefore each task on FitchFork will allocate approximately 10% of the marks to memory management. The following command is used:

```
valgrind --leak-check=full ./main
```

Please ensure, at all times, that your code *correctly* de-allocates *all* the memory that was allocated.

10 Upload checklist

The following c++ files should be in a zip archive named uXXXXXXXX.zip where XXXXXXXX is your student number:

- main.cpp
- Tuple.cpp
- Dictionary.cpp
- ScoreBoard.cpp

You will notice you do not need to upload a makefile or any h files, you may if you wish, however, FitchFork will overwrite them before running your submission.
The files should be in the root directory of your zip file. In other words, when you open your zip file you should immediately see your files. They should not be inside another folder.

11 Submission

You need to submit your source files, only the cpp files (**including the main.cpp**), on the FitchFork website (https://ff.cs.up.ac.za/). All methods must be implemented (or at least stubbed) before submission. Place the above-mentioned files in a zip named uXXXXXXXX.zip where XXXXXXXX is your student number. There is no need to include any other files or .h files in your submission. **Ensure your files are in the root directory of the zip file**. Your code should be able to be compiled with the C++98 standard
For this practical, you will have 10 upload opportunities. Upload your archive to the Practical 7 slot on the FitchFork website.