Sabancı University

Faculty of Engineering and Natural Sciences

CS204 Advanced Programming

Fall 2019-2020

Homework 5

Iterator and Operator overloading

Due: **12/11/2019, 11:55pm**

PLEASE NOTE:

Your program should be a robust one such that you have to consider all relevant programmer mistakes and extreme cases; you are expected to take actions accordingly!

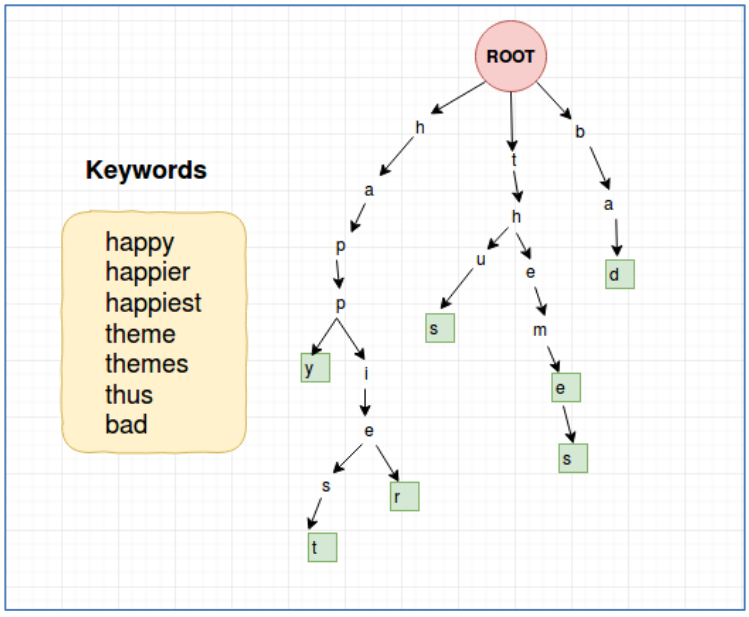
You HAVE TO write down the code on your own.

You CANNOT HELP any friend while coding.

Plagiarism will not be tolerated!

Background

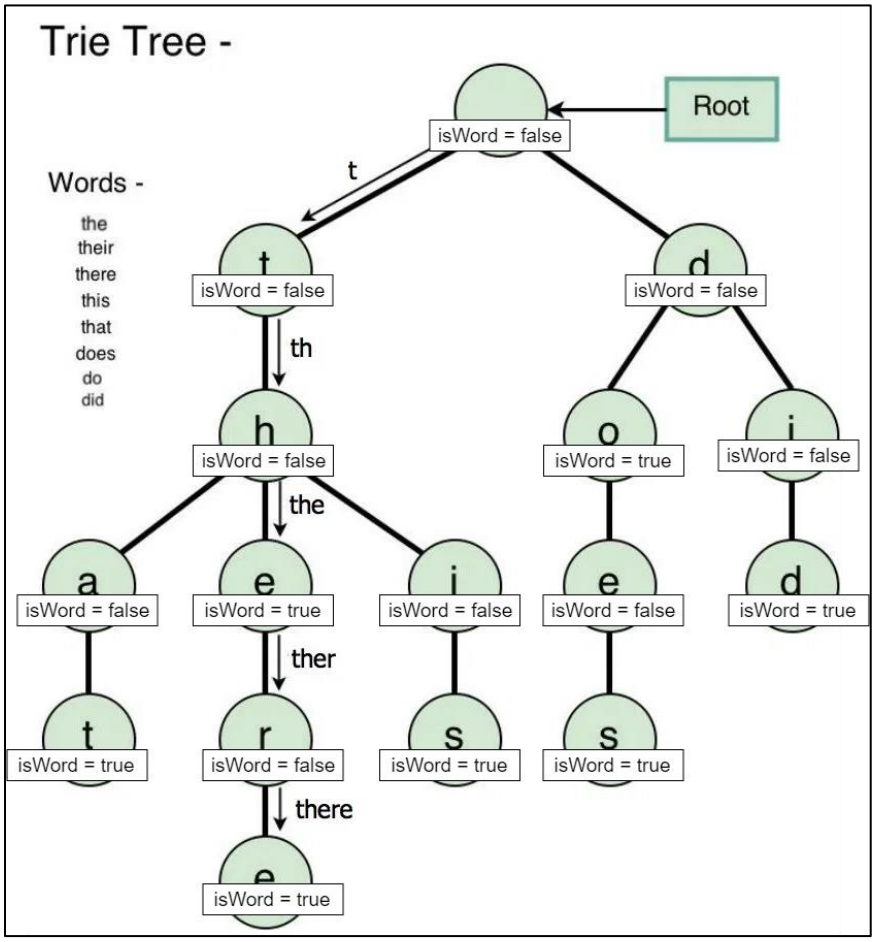
Trie is an efficient information storage/querying data structure commonly used to store words in a dictionary, document etc. As the name suggests, it is like a **tree**, which we have covered in the lectures. In fact, we only used binary trees, the most common ones one can encounter in Computer Science. However in practice, tree nodes can have an arbitrary number of children. In a **Trie**, all nodes will have the same number of children but unlike binary trees it is more than two. Every **TrieNode** in a **Trie** has 26 child nodes (one child for every character in the English alphabet). As new keys (words) are inserted into the **Trie**, the corresponding child node is generated. A sample **Trie** can be seen in the following image.



Introduction

You are given a `Trie` class, which has a private root of the Trie (`TrieNode *root`). Some of the overloaded operators (e.g. functions) will be part of the class itself, while the others will be free one, thus not as a member of the `Trie` class. You are also asked to complete the **copy constructor**, **move constructor** and **destructor** for `Trie`.

Every Trie node has a Boolean variable named `isWord` stating the end of a word in the Trie. Words start from the root and end at a node which has `isWord` set to **True.** In the following example trie you can see the words in the trie and the structure of it.



In addition to operators and constructors/destructors, you will implement an iterator class for the `Trie`. The files containing a significant code portion for that class (`TrieIterator.h` and `TrieIterator.cpp`) are given. In this homework, you only need to complete the necessary code for `Trie` and `TrieIterator`. For almost all missing parts, you will find a comment in these files.

In addition to these, you are given a helper Stack class (`Stack.h` and `Stack.cpp`). You won't change anything in them. You are also given a tester file `TrieDemo.cpp`. This code also does not need any modifications. It is a sample tester file and while grading, we will use different tester files. Note that you are not required to read input files in this homework. The output of the `main` function in `TrieDemo.cpp` is given in the homework assignment.

Operators

The operators to be implemented are listed below in detail. In this part, please keep in mind that `Trie` functionalities (insertion and traversal [with `Iterator`]) are already handled fully – of course, do not forget the parts you have to complete.

<<	This operator (**free function**) takes an `ostream` object as the left operand, and an instance of `Trie` class as the right operand. It prints the trie's content in sorted order by sending it to an output (`ostream`) stream.
==	This operator (**member function**) takes an instance of trie as parameter. The operator should return true if and only if both the current trie **(*this)** and the parameter trie contain the same elements/words. (Hint: for a very simple implementation, use the same helper function you use for **<<**)
!=	This operator (**member function**) is the exact opposite of **==** (Hint: just use the previous operator)
=	This operator (**member function**) will assign (more accurately make replica-clone of) the right hand side trie to the left one. In order to allow multiple assignments (e.g. trie1 = trie2 = trie3), you should carefully pay attention to this operator.
+=	This operator (**member function**) takes a trie as parameter and adds its nodes to the current trie instance. Note that this operation should alter the nodes of current trie.
+	This operator (**member function**) takes a trie instance as a parameter and combines the content within the current one and the one on the parameter inside a new trie instance. After the operation, content of the current trie object must not have been changed. instead, another instance of a trie should be returned.
+=	This operator (**member function**) takes a string as parameter on the right hand side and inserts it to trie. Note that this operation should alter the nodes of current trie.
+	This operator takes a trie and a string **(either as a left hand side operand or a right hand side!)** and returns another instance where the string is added to the trie. (Note that you need more than one version of **+** operator. They differ in terms of function signatures. One must be implemented as a member function; the other must be a free function.)

Output

Output of the **TrieDemo.cpp** is also given in the **output.txt** file.

```
Content of tr1:
compute
computer
process
program
progress
uni
unido
universe
university


Creating tr2 with: tr2(tr1)
Copy constructor called
Content of tr1:
```

```
compute
computer
process
program
progress
uni
unido
universe
university

Content of tr2:
compute
computer
process
program
progress
uni
unido
universe
university


Creating tr3 with: tr3 = tr1
Copy constructor called
Content of tr3:
compute
computer
process
program
progress
uni
unido
universe
university

Content of tr4:
computing
header
headphone
saramago
zapatista


Creating tr5 with: tr5(tr1 + tr4)
Move constructor called
Content of tr5:
compute
computer
computing
header
headphone
process
program
progress
saramago
uni
unido
```

```
universe
university
zapatista


Deleting the tr1
Content of tr1:
Trie is empty.

Iterator for tr5 is starting:
[1] compute
[2] computer
[3] computing
[4] header
[5] headphone
[6] process
[7] program
[8] progress
[9] saramago
[10] uni
[11] unido
[12] universe
[13] university
[14] zapatista

tr2 += tr4
Content of tr2:
compute
computer
computing
header
headphone
process
program
progress
saramago
uni
unido
universe
university
zapatista

Content of tr5:
compute
computer
computing
header
headphone
process
program
progress
saramago
uni
unido
universe
university
zapatista
```

```
tr5 and tr2 are equal.

tr4 += "gloves"
Content of tr4:
computing
gloves
header
headphone
saramago
zapatista


tr3 = tr3 + tr4
Move constructor called
Content of tr3:
compute
computer
computing
gloves
header
headphone
process
program
progress
saramago
uni
unido
universe
university
zapatista

Content of tr5:
compute
computer
computing
header
headphone
process
program
progress
saramago
uni
unido
universe
university
zapatista

tr5 and tr3 are not equal.

tr4 = tr4 + "helmet"
Move constructor called
Content of tr4:
computing
gloves
header
headphone
```

```
helmet
saramago
zapatista


tr4 = "jacket" + tr4
Move constructor called
Content of tr4:
computing
gloves
header
headphone
helmet
jacket
saramago
zapatista

Press any key to continue . . .
```

Some Important Rules

In order to get a full credit, your programs must be efficient and well presented, presence of any redundant computation or bad indentation, or missing, irrelevant comments are going to decrease your grades. You also have to use understandable identifier names, informative introduction and prompts. Modularity is also important; you have to use functions wherever needed and appropriate.

When we grade your homeworks we pay attention to these issues. Moreover, in order to observe the real performance of your codes, we are going to run your programs in *Release* mode and **we may test your programs with very large test cases**.

What and where to submit (PLEASE READ, IMPORTANT)

You should prepare (or at least test) your program using MS Visual Studio 2012 C++. We will use the standard C++ compiler and libraries of the abovementioned platform while testing your homework. It'd be a good idea to write your name and last name in the program (as a comment line of course).

Submissions guidelines are below. Some parts of the grading process are automatic. Students are expected to strictly follow these guidelines in order to have a smooth grading process. If you do not follow these guidelines, depending on the severity of the problem created during the grading process, 5 or more penalty points are to be deducted from the grade.
Name your cpp file that contains your program as follows:

"SUCourseUserName_YourLastname_YourName_HWnumber.cpp"

Your SUCourse user name is actually your SUNet username that is used for checking sabanciuniv e-mails. Do NOT use any spaces, non-ASCII and Turkish characters in the file name. For example, if your SUCourse user name is valent, name is Valentina, and last name is Tereşkova, then the file name must be:

Valent_Tereskova_Valentina_hw1.cpp

Do not add any other character or phrase to the file name. Make sure that this file is the latest version of your homework program. Compress this cpp file using WINZIP or WINRAR programs. Please use "zip" compression. "rar" or another compression mechanism is NOT allowed. Our homework processing system works only with zip files. Therefore, make sure that the resulting compressed file has a zip extension. Check that your compressed file opens up correctly and it contains your cpp file.

You will receive no credits if your compressed zip file does not expand or it does not contain the correct file. The naming convention of the zip file is the same as the cpp file (except the extension of the file of course). The name of the zip file should be as follows:

SUCourseUserName_YourLastname_YourName_HWnumber.zip

For example zubzipler_Zipleroglu_Zubeyir_hw1.zip is a valid name, but

hw1_hoz_HasanOz.zip, HasanOzHoz.zip

are **NOT** valid names.

Submit via SUCourse ONLY! You will receive no credits if you submit by other means (e-mail, paper, etc.).

Successful submission is one of the requirements of the homework. If, for some reason, you cannot successfully submit your homework and we cannot grade it, your grade will be 0.

Good Luck!
CS204 Team (M. Yusa Erguven, Kamer Kaya)