

**HACETTEPE UNIVERSITY**  
**DEPARTMENT OF COMPUTER ENGINEERING**  
**BBM-203 PROGRAMMING LAB.**  
**ASSIGNMENT 4**

**Subject:** Login System with Character Tree

**Submission Date:** 17.12.2018

**Deadline:** 06.01.2019 23:55

**Programming Language:** C

**Operation System:** CentOS Linux

**Advisors:** Dr. Sevil Sen Akagündüz, Dr. Mustafa Ege, Dr. Cumhuriyet Yiğit Özcan, R. A. Pelin Canbay

## 1. Introduction / Aim

In this assignment, you are expected to design a Login System with Character Tree. The primary goal of this experiment is to get you to practice on the tree data structure.

## 2. Background Information

Strings can essentially be viewed as the most important and common topics for a variety of programming problems. String processing has a variety of real-world applications too, such as:

- Search Engines
- Genome Analysis
- Data Analytics

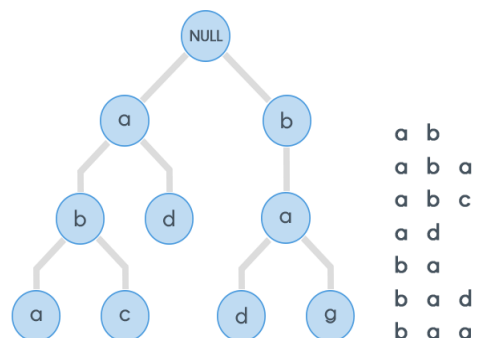
All the content presented to us in textual form can be visualized as nothing but just strings.

A Character tree is a special form of tree data structure that is based on the prefix of a string. The prefix of a string is nothing but any  $n$  letters,  $n \leq |S|$  that can be considered beginning strictly from the starting of a string.

A character tree is a special data structure used to store strings that can be visualized like a graph. It consists of nodes and edges. Each node consists of at max 26 children and edges connect each parent node to its children. These 26 pointers are pointers for each of the 26 letters of the English alphabet. A separate edge is maintained for every node.

Strings are stored in a top to bottom manner on the basis of their prefix in a character tree. All prefixes of length 1 are stored at until level 1, all prefixes of length 2 are sorted at until level 2 and so on.

For example, consider the following diagram :



Now, one would be wondering why to use a data structure such as a character tree for processing a single string? Actually, Character trees are generally used on groups of strings, rather than a single string. When given multiple strings, we can solve a variety of problems based on them. For example, consider an English dictionary and a single string *s*, find the prefix of maximum length from the dictionary strings matching the string *s*. Solving this problem using a naive approach would require us to match the prefix of the given string with the prefix of every other word in the dictionary and note the maximum. This is an expensive process considering the amount of time it would take. Character trees can solve this problem in a much more efficient way. Before processing each Query of the type where we need to search the length of the longest prefix, we first need to add all the existing words into the dictionary. A Character tree consists of a special node called the root node. This node doesn't have any incoming edges. It only contains max 26 outgoing edges for each letter in the alphabet and is the root of the character tree.

So, the insertion of any string into a character tree starts from the root node. All prefixes of length one are direct children of the root node. In addition, all prefixes of length 2 become children of the nodes existing at level one.

### 3. Experiment

In this experiment, you are expected to write an application that constructs a Login System with Character Tree. The application will take the input.txt file in the current directory and read its contents. In order to create the Character tree, there are some commands for adding, deleting and refreshing.

#### Structure of Commands:

- a **username password** #add username to the tree with the given password
- s **username** #search with the given username and return the password if it is
- q **username password** #send query for the password with the given username
- d **username** #delete username and its password
- l #list the tree

**With -a command:** The application will read the given username character by character and add them to the tree. With the last character of the username store the password.

- \* If the first character of the username is not referenced by the root node, the username will be added to the tree starting from the root node.
- \* If the first *n* character of the username exists on the tree, a branch occurs on the *n<sup>th</sup>* node for the last characters
- \*The node which is the last character of the username has to hold the password for the given username.
- \* If there is a username same as the given username, the application will give an output that “reserved username”.

-a pel 123	-a polat 874	-a mert 543
<pre> graph TD     Root((Root)) --&gt; p((p))     p --&gt; e((e))     e --&gt; l((l))     l --&gt; 123[123] </pre>	<pre> graph TD     Root((Root)) --&gt; p((p))     p --&gt; e((e))     p --&gt; o((o))     e --&gt; l1((l))     l1 --&gt; 123[123]     o --&gt; l2((l))     l2 --&gt; a((a))     a --&gt; t((t))     t --&gt; 874[874] </pre>	<pre> graph TD     Root((Root)) --&gt; p((p))     Root --&gt; m((m))     p --&gt; e1((e))     p --&gt; o((o))     e1 --&gt; l1((l))     l1 --&gt; 123[123]     o --&gt; l2((l))     l2 --&gt; a((a))     a --&gt; t1((t))     t1 --&gt; 874[874]     m --&gt; e2((e))     e2 --&gt; r((r))     r --&gt; t2((t))     t2 --&gt; 543[543] </pre>
"was added"	"was added"	"was added"

**With -s command;** The application will read the given username and according to the questioned username it will return an information.

- \* If the first character of the username is not referenced by the root node the application will give an output that "no record".
- \* If the first n character of the username exists on the tree, but the remainder is not, the application will give an output that "incorrect username".
- \* If all characters of the username exist on the tree, but the last character has no password, the application will give an output that "not enough username".
- \* If all characters of the username exist on the tree and the last character has its password, the application will give an output that "password xxx".

-s cihan	-s pel	-s mer	-s mert	-s polet	-s polat
"no record"	"password 123"	"not enough username"	"password 543"	"incorrect username"	"password 874"

**With -q command;** The application will read the username and its password. According to the matching, it will return an information.

- \* If the first character of the username is not referenced by the root node the application will give an output that "no record".
- \* If the first n character of the username exists on the tree, but the remainder is not, the application will give an output that "incorrect username".
- \* If all characters of the username exist on the tree, but the last character has no password, the application will give an output that "not enough username".
- \* If all characters of the username exist on the tree, but the last character has a different password from the given, the application will give an output that "incorrect password".

\* If all characters of the username exist on the tree, and the last character has the same password with the given, the application will give an output that “successful login”.

-q cihan 234	-q pel 123	-q mer 987	-q mert 541	-q polet 874	-q polat 874
“no record”	“successful login”	“not enough username”	“incorrect password”	“incorrect username”	“successful login”

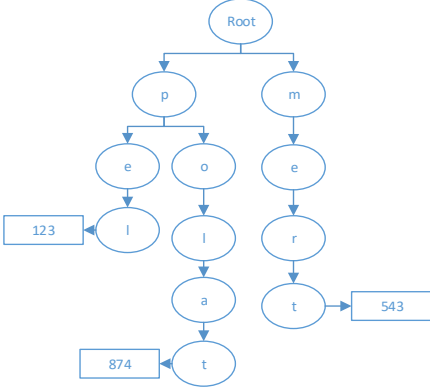
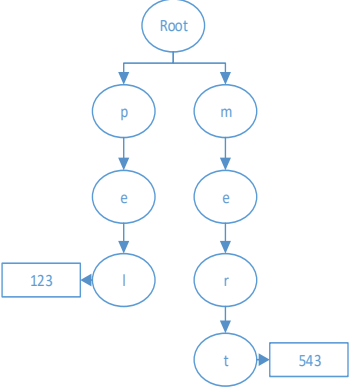
**With -d command;** With the given username the application will delete the username and its password from the tree.

\* If the first character of the username is not referenced by the root node the application will give an output that “no record”.

\* If the first n character of the username exists on the tree, but the remainder is not, the application will give an output that “incorrect username”.

\* If all characters of the username exist on the tree, but the last character has no password, the application will give an output that “not enough username”.

\* If all characters of the username exist on the tree, and the last character has the password, the application will delete all nodes which are not connected to another username. Then it will give an output that “deletion is successful”.

Tree	-d mer	-d polat	-d meryem
	“not enough username”	 <p>“deletion is successful”</p>	“incorrect username”

**With -l command;** The application will list the all username by preorder traversal.

\*After the first level of the tree, each branch will be displayed with a new tab

Tree	-l
<pre> graph TD     Root((Root)) --&gt; p((p))     Root --&gt; m((m))     p --&gt; e1((e))     p --&gt; o((o))     m --&gt; e2((e))     e1 --&gt; l1((l))     e1 --&gt; i1((i))     o --&gt; l2((l))     l2 --&gt; a((a))     e2 --&gt; r((r))     l1 --&gt; n((n))     i1 --&gt; a     a --&gt; t1((t))     a --&gt; v((v))     t1 --&gt; t2((t))     t1 --&gt; e3((e))     v --&gt; e4((e))     n --&gt; 123[123]     l1 --&gt; 123[123]     t2 --&gt; 874[874]     e3 --&gt; 543[543]     e4 --&gt; 888[888] </pre>	<p>-p</p> <p>-pel, pelin</p> <p>-polat</p> <p>-mer</p> <p>-mert</p> <p>-merve</p>

You can see another sample inputs and outputs in ftp site.

### Important Issues

- Projects without a proper Makefile won't be graded.
- All terms in your tree must be in lower case
- Test your program on "dev.cs.hacettepe.edu.tr" before submission. Your submission will be compiled and executed on this machine and this machine only.
- The report will be 30 points
- Please explain your design, data structures, and algorithms.
- Give necessary details without unnecessary clutter.
- Guide: <ftp://ftp.cs.hacettepe.edu.tr/pub/dersler/genel/FormatForLabReports.doc>

### Grading

Your report score will be calculated as follows:

$$ReportFinal = ReportScore * \frac{ExecScore}{70}$$

### Notes

- Regardless of the length, use **UNDERSTANDABLE** names to your variables, classes, and Functions
- Write **READABLE SOURCE CODE** block
  - These will cost you points.
- Deadline is 23:59 pm. No other submission method will be accepted.
- **Save** all your work until the assignment is graded.

- The assignment must be original, **INDIVIDUAL** work. Duplicate or very similar assignments are both going to be punished. General discussion of the problem is allowed, but **DO NOT SHARE** answers, algorithms or source codes.
- Should you have a question, feel free to ask on Piazza. Be aware that the question has not been asked/discussed before.
- Copying without citing will be considered as cheating. Citing does not make it yours; cited work will get 0 from the respective section.
- You will submit your work from <https://submit.cs.hacettepe.edu.tr/index.php> with the file hierarchy as below:

This file hierarchy must be zipped before submission (Not .rar, only .zip files are supported by the system)

- ➔ <student id>.zip
  - Report.pdf
  - \*.c
  - \*.h
  - makefile

## References

- Data Structures Using C++, Second Edition. D. S. Malik
- <http://cglab.ca/~morin/teaching/5408/notes/strings.pdf>
- <https://www.hackerearth.com/practice/data-structures/advanced-data-structures/trie-keyword-tree/tutorial/>