2. Software Using Documentation

2.1. Software Usage

The program works by taking one input and after performing the necessary operations it creates an output file named “output.txt”.

Input file includes commands like

- “-a” for adding new user to the trie,
- “-s” for searching an user in trie,
- “-d” for deleting an user from trie,
- “-q” for sending query,
- “-l” for listing the whole trie.

3. Software Design Notes

3.1. Description of the Program

3.1.1. Problem

In this experiment, we are expected to write a program that works with one input file. The commands are explained above. We design a Login System with Character Tree while reading commands from file line by line and each line program prints some information.

3.1.2. Solution

```
struct Node{
    struct Node* child[26];
    char password[10];
};typedef struct Node Node;
```

Since we are expected to write a program that creates a character tree , I created a structure named “Node” which includes a pointer to point another Node and a password. When the program see “-a” command firstly it looks the first character of adding name and search for his character in root child. If there is a branch occurs on the n. node fort he last characters. If there is not programs adds this username to the tree starting from the root node.

For “-s“,”-d“,”-q” commands there are some similar outputs so i wrote a function and i use it for all these commands. This function firstly looks that the first character of the username is referenced by the root node or not. If it is not the programs writes “no record” and exits from the function. If it is then it looks to tree to find the rest of username and these operations goes like this.

```
for(i=0;questionedName[i];i++){
    charIndex = getCharIndex(questionedName[i]);
    if(iterator -> child[charIndex] != NULL){
        control++;
        iterator = iterator -> child[charIndex];
    }
}
```

I will do the operations mentioned above with these variables “i” and “control”. Here this loop, we starts from root and looks every character of questioned name in tree. If we find that character we increase the “control” value.

After this loop is over, we check “control” and “i” values. If they are equal and there is a password on the last node we came, this means questioned name is in tree. If they are equal but there is no password on the last node, this means questioned name is in tree but it is incomplete so program prints “not enough username”. If the values of “control” and “i” are not equal and “control” is not zero this means that the first n character of the questioned name exists on the tree but the remainder is not so program prints “incorrect username”.