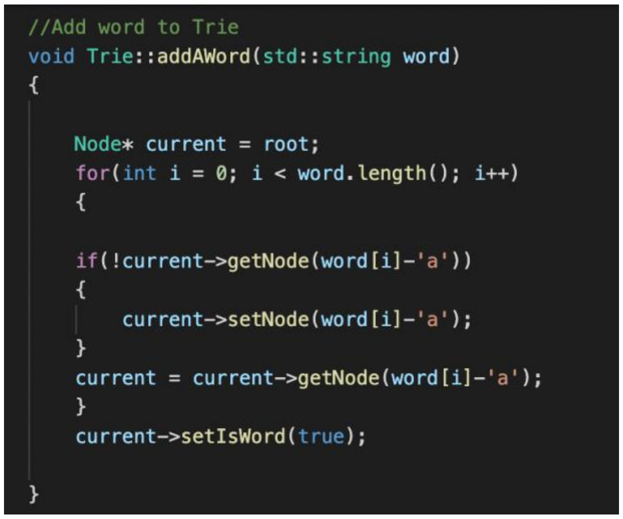
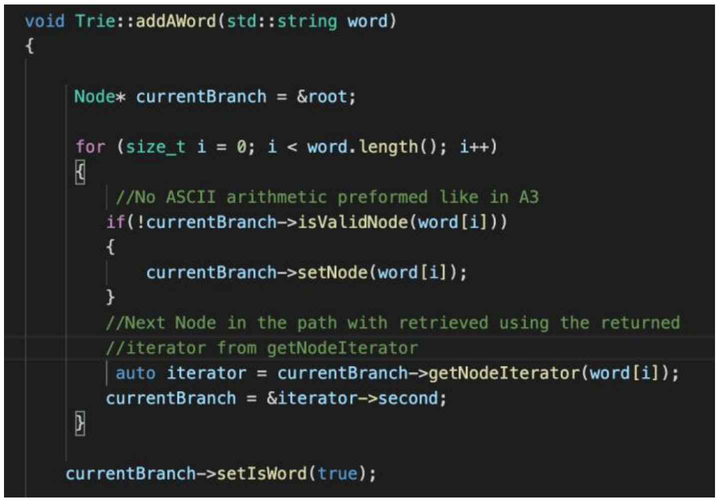
Osee Pierre
CS 3505
A4: Refactoring and Testing
2/22/2022

Refactoring Decisions

Given the requirement to change the branches structure from a Node* branches[26] to a map<char, Node>, the implementation details of how the Trie structure functioned changed. In the addAWord method, it was useful to use Node* to traverse the tree while making changes to the Trie like in A3 but the characters of the word that is being added to the Trie are used directly to interact with the Nodes instead of preforming ASCII arithmetic like in A3. Also, addAWord uses the iterator received from newly added getNodeIterator method to get the next Node in its path. Since a Node* is being used, getNodeIterator is used instead of the regular getNode method because getNode returns an address of a temporary object of the type Node.

A3 addAWord method	A4 addAWord method
	

```
//Add word to Trie
void Trie::addAWord(std::string word)
{

    Node* current = root;
    for(int i = 0; i < word.length(); i++)
    {

    if(!current->getNode(word[i]-'a'))
    {
        current->setNode(word[i]-'a');
    }
    current = current->getNode(word[i]-'a');
    }
    current->setIsWord(true);

}
```

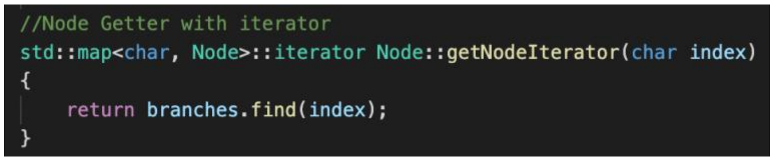
```
void Trie::addAWord(std::string word)
{

    Node* currentBranch = &root;

    for (size_t i = 0; i < word.length(); i++)
    {
        //No ASCII arithmetic preformed like in A3
        if(!currentBranch->isValidNode(word[i]))
        {
            currentBranch->setNode(word[i]);
        }
        //Next Node in the path with retrieved using the returned
        //iterator from getNodeIterator
        auto iterator = currentBranch->getNodeIterator(word[i]);
        currentBranch = &iterator->second;
    }

   currentBranch->setIsWord(true);
```

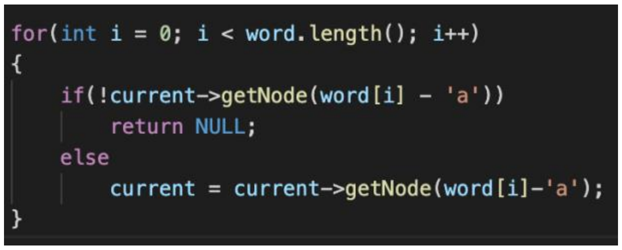
A4: getNodeIterator uses the find feature of a map to check if a key exists and returns with an iterator so the location of the Node can be accessed



```
//Node Getter with iterator
std::map<char, Node>::iterator Node::getNodeIterator(char index)
{
    return branches.find(index);
}
```

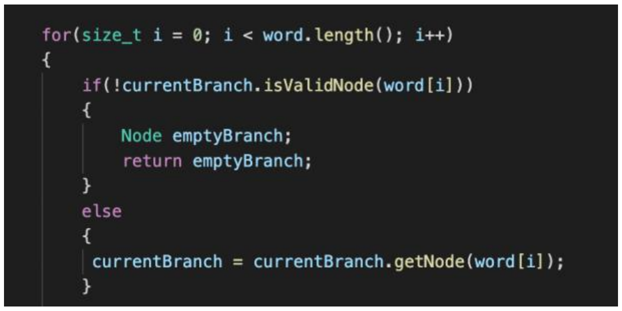
Since a Node* branches[26] structure is not used, null pointers are not utilized. So, in searchTrie, which is a helper function that searches for a word, if a branch is not valid it returns an empty Node instead of NULL. The allWordsStartingWithPrefix method, which uses the result from searchTrie the determine whether to proceed, checks if the Node has any branches instead of checking if its NULL like in A3.

A3 searchTrie



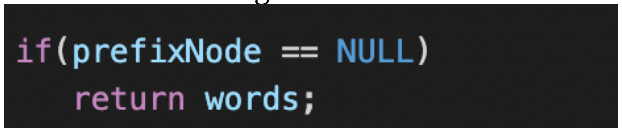
```
for(int i = 0; i < word.length(); i++)
{
    if(!current->getNode(word[i] - 'a'))
        return NULL;
    else
        current = current->getNode(word[i]-'a');
}
```

A4 searchTrie



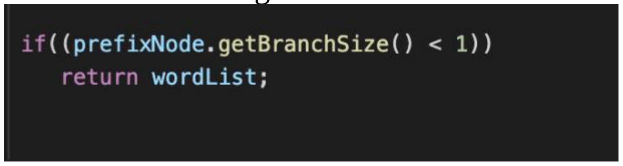
```
for(size_t i = 0; i < word.length(); i++)
{
    if(!currentBranch.isValidNode(word[i]))
    {
        Node emptyBranch;
        return emptyBranch;
    }
    else
    {
     currentBranch = currentBranch.getNode(word[i]);
    }
```

A3 allWordsStartingWithPrefix



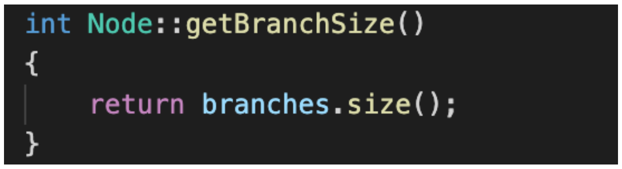
```
if(prefixNode == NULL)
    return words;
```

A4 allWordsStartingWithPrefix



```
if((prefixNode.getBranchSize() < 1))
    return wordList;
```

A4: method that gets the amount of branches a Node has



```
int Node::getBranchSize()
{
    return branches.size();
}
```

Additionally, in allWordsStartingWithPrefix and traverseTrie methods, which are used to return a list of words that start with a particular set of letters, no pointers are used in their implementation giving that the map structure is used, and changes are not required when the Node being considered is sent to the traverseTrie helper function to search down the Trie structure.

Lastly, unlike A3, A4 doesn't have any resources that it creates and control. The map structure is self-containing and manages all of its resources and thus implementation of Rule of three is not required.