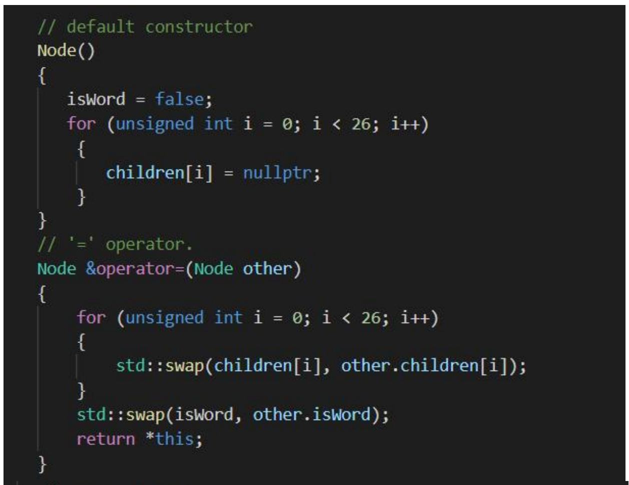
<div align="right">

A4: Refactoring and Testing
Robert Li and Wenlin Li
CS 3505
2022/02/22

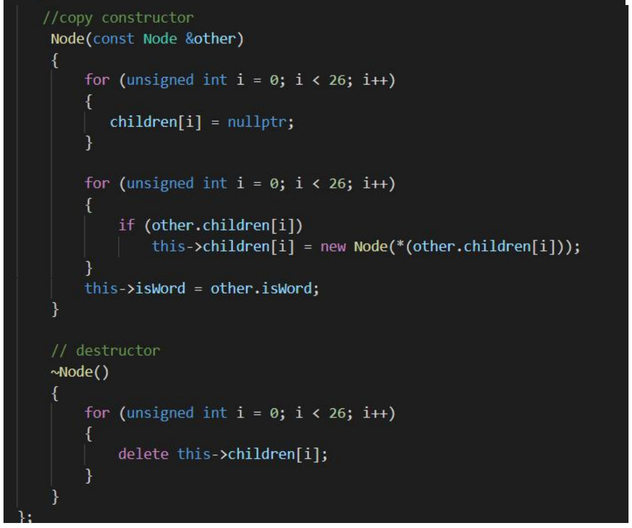
</div>

The most challenging part of refactoring this code is making change of Node to Map, which construct the code goes different ways. We stuck on the rule-of-three using at first, we hardly find out a way to make the map work with rule-of-three constructors, by the hint from TA, the point's changing, we figured out we do not need to implement rule-of-three in our class. We figured out there was pointers managing everything in A3, after we change node and branch to map, the map can manage all the copies and old nodes' words by char and Node parameter, which is safer when we creating new copies, the old node will stay. Since we no longer need to worry about the pointers and references in our classes, our code can be much cleaner and easier to works, since the map can easily manage the location of the words in nodes.

The Old code:



```
// default constructor
Node()
{
    isWord = false;
    for (unsigned int i = 0; i < 26; i++)
     {
         children[i] = nullptr;
     }
}
// '=' operator.
Node &operator=(Node other)
{
    for (unsigned int i = 0; i < 26; i++)
    {
        std::swap(children[i], other.children[i]);
    }
    std::swap(isWord, other.isWord);
    return *this;
}
```

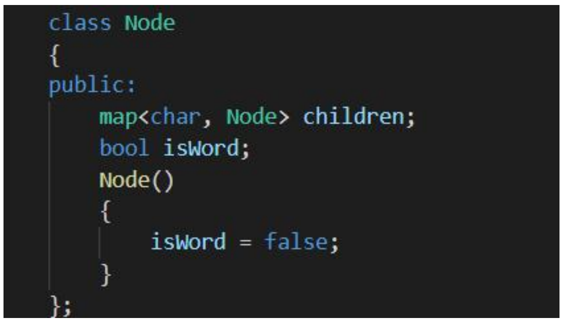


```
  //copy constructor
  Node(const Node &other)
  {
      for (unsigned int i = 0; i < 26; i++)
      {
          children[i] = nullptr;
      }

      for (unsigned int i = 0; i < 26; i++)
      {
          if (other.children[i])
              this->children[i] = new Node(*(other.children[i]));
      }
      this->isWord = other.isWord;
  }

  // destructor
  ~Node()
  {
      for (unsigned int i = 0; i < 26; i++)
      {
          delete this->children[i];
      }
  }
};
```

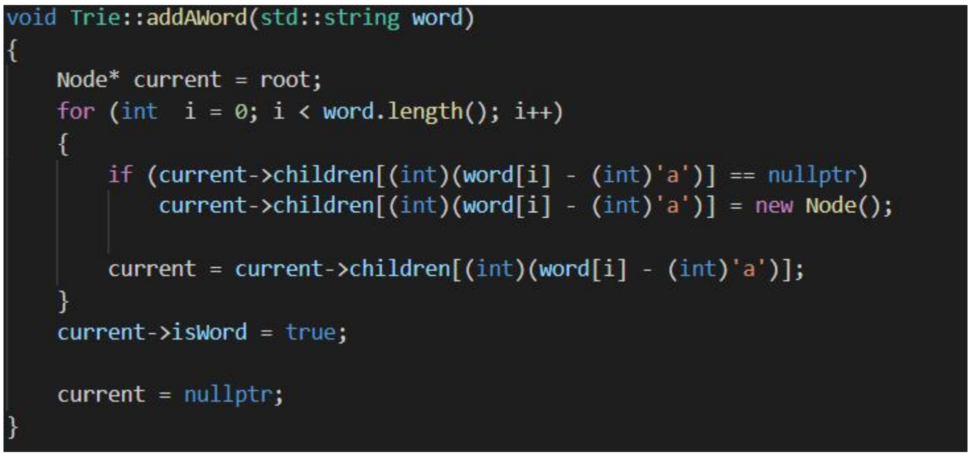
The new code:



```
class Node
{
public:
    map<char, Node> children;
    bool isWord;
    Node()
    {
        isWord = false;
    }
};
```

The second one is about addAWord method, after we figured out how to refactor this method, the rest 2 methods is same process as the addAWord method. The main process of refactoring our method is thinking about we do not need to worry about pointers and references in our methods(map is great). We find out the c++ iterator function on google, rather than using pointers and references to mange our nodes. The code after refactoring is easier to understand and cleaner, the second keyword helps us keep track of the node, rather than recursion with pointers.

The old code:

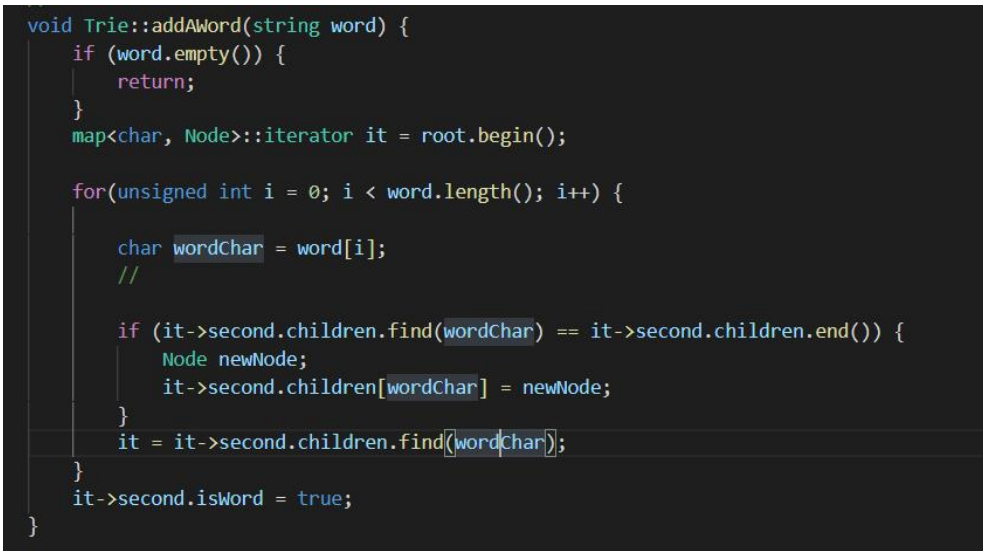


```
void Trie::addAWord(std::string word)
{
    Node* current = root;
    for (int  i = 0; i < word.length(); i++)
    {
        if (current->children[(int)(word[i] - (int)'a')] == nullptr)
            current->children[(int)(word[i] - (int)'a')] = new Node();

        current = current->children[(int)(word[i] - (int)'a')];
    }
    current->isWord = true;

    current = nullptr;
}
```

The new code:



```
void Trie::addAWord(string word) {
    if (word.empty()) {
        return;
    }
    map<char, Node>::iterator it = root.begin();

    for(unsigned int i = 0; i < word.length(); i++) {

        char wordChar = word[i];
        //

        if (it->second.children.find(wordChar) == it->second.children.end()) {
            Node newNode;
            it->second.children[wordChar] = newNode;
        }
        it = it->second.children.find(wordChar);
    }
    it->second.isWord = true;
}
```