

## Refactoring A4

One of the big refactorings we did with our Trie.cpp was using map in place of an array. The most significant change that we made was with the helper method findPrefix for the allWordsStartingWithPrefix. Since we did not work with an array of pointers anymore, we no longer needed to check if a pointer was a nullptr in the array. All we needed to do was iterate through the existing value pairs of the map. If it didn't exist, it would simply not be within the map.

```

148 /**
149  * This helper function iterates through the trie to get in position to start building words
150  *
151  * @param inputTrie A reference to the trie that will be iterated through.
152  * @param prefix The prefix.
153  */
154 Trie *Trie::findPrefix(Trie &inputTrie, string prefix)
155 {
156     for (Trie *pointer : letArray)
157     {
158         if (pointer)
159         {
160             if (prefix.length() == 1 && pointer->currentLetter[0] == prefix[0])
161             {
162                 inputTrie = *pointer;
163                 break;
164             }
165             if (pointer->currentLetter[0] == prefix[0])
166             {
167                 return pointer->findPrefix(inputTrie, prefix.erase(0, 1));
168             }
169         }
170     }
171     return this;
172 }

```

```

111 /**
112  * This helper function iterates through the trie to get in position to start building words using the prefix.
113  *
114  * @param prefix The prefix.
115  */
116 Trie Trie::findPrefix(string prefix)
117 {
118     unsigned int counter = 0;
119     for (auto &child : children)
120     {
121         if (prefix.length() == 1 && child.first == prefix[0])
122         {
123             *this = child.second;
124             counter++;
125             break;
126         }
127         if (child.first == prefix[0])
128         {
129             counter++;
130             return child.second.findPrefix(prefix.erase(0, 1));
131         }
132     }
133     if (counter == prefix.length())
134     {
135         return *this;
136     }
137     else
138     {
139         Trie emptyTrie = Trie();
140         return emptyTrie;
141     }
142 }

```

The second biggest refactor we made was getting rid of the member variable currentLetter. This variable was a string that stored the letter of the current Tire (node), and in the case of the root was simply just "root". We got rid of this because it wasn't needed because in the map the key was the current letter, so to store it in a variable is redundant. A lot of our methods relied on currentLetter, so we had to go through and make changes in a lot of places. Most being simple changes from "currentLetter" to the key of the value. However, reimplementing the allWordsStartingWithPrefix method took a while to redesign without using currentLetter. As representing the root node took some clever thinking.

```

120 /**
121  * This function returns all words starting with a given prefix.
122  *
123  * @param input The prefix.
124  */
125 vector<string> Trie::allWordsStartingWithPrefix(string input)
126 {
127     vector<string> words = vector<string>();
128     // Need newTrie so that the current trie doesn't get messed up when trying to find the prefix.
129     Trie newTrie = Trie(*this);
130     findPrefix(newTrie, input);
131     // Making sure an empty input is still valid.
132     if (newTrie.currentLetter == "root" && !input.empty())
133     {
134         newTrie.currentLetter = "noWordsExistWithThisInput";
135     }
136     newTrie.buildWords(words, input);
137     return words;
138 }
139
140 /**
141  * This helper function iterates through the trie to get in position to start building words using the prefix.
142  *
143  * @param inputTrie A reference to the trie that will be iterated through.
144  * @param prefix The prefix.
145  */
146 Trie *Trie::findPrefix(Trie &inputTrie, string prefix)
147 {
148     for (Trie *pointer : letArray)
149     {
150         if (pointer)
151         {
152             if (prefix.length() == 1 && pointer->currentLetter[0] == prefix[0])
153             {
154                 inputTrie = *pointer;
155                 break;
156             }
157             if (pointer->currentLetter[0] == prefix[0])
158             {
159                 return pointer->findPrefix(inputTrie, prefix.erase(0, 1));
160             }
161         }
162     }
163     return this;
164 }

```

```

87 * This function returns all words starting with a given prefix.
88 *
89 * @param input The prefix.
90 */
91 vector<string> Trie::allWordsStartingWithPrefix(string input)
92 {
93     vector<string> words = vector<string>();
94     Trie saveRoot = Trie(*this);
95     // Convert the first child of the root 'node' to: " << children.
96     // Need newTrie so that the current trie doesn't get messed up when trying to find the prefix.
97     *this = findPrefix(input);
98     if (endOfWord)
99     {
100         words.push_back(input);
101     }
102     buildWordsAndAddToVector(words, input);
103     *this = saveRoot;
104     return words;
105 }
106
107 /**
108  * This helper function iterates through the trie to get in position to start building words using the prefix.
109  *
110  * @param prefix The prefix.
111  */
112 Trie Trie::findPrefix(string prefix)
113 {
114     unsigned int counter = 0;
115     for (auto &child : children)
116     {
117         if (prefix.length() == 1 && child.first == prefix[0])
118         {
119             *this = child.second;
120             counter++;
121             break;
122         }
123         if (child.first == prefix[0])
124         {
125             counter++;
126             return child.second.findPrefix(prefix.erase(0, 1));
127         }
128     }
129     if (counter == prefix.length())
130     {
131         return *this;
132     }
133     else
134     {
135         Trie emptyTrie = Trie();
136         return emptyTrie;
137     }
138 }

```