2.1 Trie Data Structure Implementation

Node Structure

```
struct Node {
    // Array to store links to child nodes
    Node* links[26];

    // End of word flag
    bool flag = false;
    // Check if node contains key
    bool containsKey(char ch) {
        return links[ch - 'a'] != NULL;
    }
    // Insert new node with key
    void put(char ch, Node* node) {
        links[ch - 'a'] = node;
    }
    // Get node for key
    Node* get(char ch) {
        return links[ch - 'a'];
    }
    // Mark end of word
    void setEnd() {
        flag = true;
    }
    // Check if end of word
    bool isEnd() {
        return flag;
    }
};
```

Trie Implementation

```
class Trie {
private:
    Node* root;
public:
    Trie() {
        root = new Node();
    }
    // Insert word - O(len)
    void insert(string word) {
        Node* node = root;
        for (int i = 0; i < word.length();
            i++) {
            if (!node->containsKey(word[i]))
                {
                node->put(word[i], new
                    Node());
            }
            node = node->get(word[i]);
        }
        node->setEnd();
    }
    // Search word - O(len)
    bool search(string word) {
        Node* node = root;
        for (int i = 0; i < word.length();
            i++) {
            if (!node->containsKey(word[i]))
                {
                return false;
            }
            node = node->get(word[i]);
        }
        return node->isEnd();
    }
    // Check prefix - O(len)
    bool startsWith(string prefix) {
        Node* node = root;
        for (int i = 0; i < prefix.length();
            i++) {
            if
                (!node->containsKey(prefix[i]))
                {
                return false;
            }
            node = node->get(prefix[i]);
        }
        return true;
    }
};
// Usage Example:
// Trie trie;
// trie.insert("striver");
// trie.search("striver");    // true
// trie.startsWith("str");    // true
```