SUMMER TRAINING/INTERNSHIP

PROJECT REPORT

(Term June-July 2025)

Multilingual Dictionary App using Trie (C++)

Submitted by

|  |  |
| --- | --- |
| Name | Registration No |
| Vatam Prudviswar Reddy | 12316480 |
| Peddineni Chandra Vardhan | 12305894 |
| Kyatham Fanindra Reddy | 12320622 |
| Shikar Shukla | 12310049 |
| Uttam Singh | 12320989 |

Course Code: CSE343 / CSE443

Under the Guidance of

Dr. Anshu Sharma

Dr. Chirag Sharma

Nitish Kumar   
School of Computer Science and Engineering

# CERTIFICATE

This is to certify that Mr. Vatam Prudvi Swar Reddy (Registration Number: 12316480) has successfully completed the Summer Training/Internship project titled "Multilingual Dictionary App using Trie (C++)" in partial fulfillment of the requirements for the award of the degree of B.Tech in Computer Science and Engineering.

Signature of Guide  
  
Signature of HOD

**ACKNOWLEDGEMENT**

I would like to express my heartfelt gratitude to my mentor and the School of Computer Science and Engineering for their guidance and support throughout the training period. Their encouragement helped me complete this project successfully.

**TABLE OF CONTENTS**

CHAPTER 1: INTRODUCTION

CHAPTER 2: PROJECT DETAILS

CHAPTER 3: IMPLEMENTATION

CHAPTER 4: RESULTS AND DISCUSSION

CHAPTER 5: CONCLUSION

**CHAPTER 1: INTRODUCTION**

**Overview of Training Domain**

The training domain for this project involves the implementation of data structures in C++, specifically focusing on the Trie (Prefix Tree) for fast string operations.

**Objective of the Project**

The primary objective was to design and implement a multilingual dictionary application using a Trie data structure in C++. Speciﬁc goals included:

**1.** Efﬁcient storage and retrieval of words with translations in multiple languages.

**2**. Support for case-insensitive word searches and preﬁx-based autocomplete.

**3**. Functionality for inserting, searching, deleting, and retrieving words by preﬁx.

**4**. A user-friendly console interface for seamless dictionary management.

**CHAPTER 2: PROJECT DETAILS**

**Problem Definition**

To develop a multilingual dictionary using the Trie data structure in C++ that enables efficient insertion, search, translation, and autocomplete functionality for words in multiple languages.

**Scope and Objectives**

- To implement a Trie data structure supporting multilingual translation.  
- To design a C++ application that stores and retrieves word translations.  
- To allow autocomplete suggestions for user input based on prefix.

**System Requirements**

Hardware:  
- Processor: 1.8 GHz or faster  
- RAM: 4GB or more  
  
Software:  
- OS: Windows/Linux  
- Compiler: C++  
- IDE: Visual Studio Code or any C++ supported Online Complier.

Areas Covered During Training

* Data structures: Arrays, linked lists, trees, and tries.
* Algorithms: Searching, sorting, recursion, and tree traversal.
* C++ programming: Classes, memory management, STL usage.
* Software development practices: SDLC, testing, and documentation.
* Optimization techniques: Time and space complexity analysis.

**Daily/Weekly Work Summary**

* **Week 1:** Studied Trie data structure and C++ STL; analyzed project requirements.
* **Week 2:** Designed TrieNode and MultilingualDictionary classes.
* **Week 3:** Implemented insertion, search, and autocomplete functionalities.
* **Week 4:** Added word deletion and case-insensitive handling; developed console interface, conducted testing, resolved bugs, and prepared project documentation.

**Title of the Project**

Multilingual Dictionary Application Using Trie

Problem Definition

Traditional dictionary applications often struggle with efﬁcient word storage and retrieval, particularly for preﬁx-based searches and multilingual support. Linear or hash-based storage can result in slower search times for large datasets or complex queries. This project addresses these

issues by using a Trie data structure to enable fast word lookups, preﬁx-based autocomplete, and multilingual translation storage.

**Scope and Objectives**

· Develop a console-based dictionary application.

· Support multiple languages (e.g., English, Spanish, French).

* + - Implement core functionalities: insert, search, delete, and autocomplete.
    - Ensure case-insensitive word handling and efﬁcient memory usage.
    - Achieve O(m) time complexity for word operations (m = word length).
    - Provide a scalable solution for large dictionaries.

**Enable users to manage translations interactively.**

# CHAPTER 3: IMPLEMENTATION

## C++ Implementation

#include <iostream>

#include <unordered\_map>

#include <vector>

#include <string>

#include <algorithm>

#include <cctype>

// Trie node structure

class TrieNode {

public:

std::unordered\_map<char, TrieNode\*> children; // Children nodes

bool isEndOfWord; // Marks end of a word

std::unordered\_map<std::string, std::string> translations; // Language code -> translation

TrieNode() : isEndOfWord(false) {}

~TrieNode() {

for (auto& child : children) {

delete child.second;

}

}

};

// Trie class for the multilingual dictionary

class MultilingualDictionary {

private:

TrieNode\* root;

// Helper function to collect all words with a given prefix

void collectWords(TrieNode\* node, std::string prefix, std::vector<std::pair<std::string, std::unordered\_map<std::string, std::string>>>& results) {

if (node->isEndOfWord) {

results.push\_back({prefix, node->translations});

}

for (auto& child : node->children) {

collectWords(child.second, prefix + child.first, results);

}

}

// Helper function to convert string to lowercase

std::string toLower(const std::string& str) {

std::string result = str;

std::transform(result.begin(), result.end(), result.begin(), [](unsigned char c) { return std::tolower(c); });

return result;

}

public:

MultilingualDictionary() {

root = new TrieNode();

}

~MultilingualDictionary() {

delete root;

}

// Insert a word with its translations

void insert(const std::string& word, const std::unordered\_map<std::string, std::string>& translations) {

TrieNode\* current = root;

std::string lowerWord = toLower(word);

// Traverse or create nodes for each character

for (char c : lowerWord) {

if (current->children.find(c) == current->children.end()) {

current->children[c] = new TrieNode();

}

current = current->children[c];

}

current->isEndOfWord = true;

current->translations = translations;

}

// Search for a word and return its translations

std::unordered\_map<std::string, std::string> search(const std::string& word) {

TrieNode\* current = root;

std::string lowerWord = toLower(word);

for (char c : lowerWord) {

if (current->children.find(c) == current->children.end()) {

return {};

}

current = current->children[c];

}

return current->isEndOfWord ? current->translations : std::unordered\_map<std::string, std::string>();

}

// Get all words starting with a prefix

std::vector<std::pair<std::string, std::unordered\_map<std::string, std::string>>> startsWith(const std::string& prefix) {

std::vector<std::pair<std::string, std::unordered\_map<std::string, std::string>>> results;

TrieNode\* current = root;

std::string lowerPrefix = toLower(prefix);

// Traverse to the prefix node

for (char c : lowerPrefix) {

if (current->children.find(c) == current->children.end()) {

return results;

}

current = current->children[c];

}

// Collect all words in the subtree

collectWords(current, lowerPrefix, results);

return results;

}

// Delete a word from the trie

bool deleteWord(const std::string& word) {

std::string lowerWord = toLower(word);

return deleteHelper(root, lowerWord, 0);

}

private:

// Helper function for deletion

bool deleteHelper(TrieNode\* current, const std::string& word, int index) {

if (index == word.length()) {

if (!current->isEndOfWord) {

return false; // Word not found

}

current->isEndOfWord = false;

current->translations.clear();

return current->children.empty(); // Return true if node can be deleted

}

char c = word[index];

auto it = current->children.find(c);

if (it == current->children.end()) {

return false; // Word not found

}

bool shouldDeleteCurrentNode = deleteHelper(it->second, word, index + 1);

if (shouldDeleteCurrentNode) {

delete it->second;

current->children.erase(c);

return current->children.empty() && !current->isEndOfWord;

}

return false;

}

};

// Main function with console interface

int main() {

MultilingualDictionary dict;

int choice;

std::string word, lang, translation, prefix;

// Sample data

dict.insert("cat", {{"en", "cat"}, {"es", "gato"}, {"fr", "chat"}});

dict.insert("dog", {{"en", "dog"}, {"es", "perro"}, {"fr", "chien"}});

dict.insert("car", {{"en", "car"}, {"es", "coche"}, {"fr", "voiture"}});

do {

std::cout << "\nMultilingual Dictionary Menu:\n";

std::cout << "1. Insert a word\n";

std::cout << "2. Search for a word\n";

std::cout << "3. Find words by prefix\n";

std::cout << "4. Delete a word\n";

std::cout << "5. Exit\n";

std::cout << "Enter your choice: ";

std::cin >> choice;

std::cin.ignore(); // Clear newline from input buffer

switch (choice) {

case 1: {

std::cout << "Enter word: ";

std::getline(std::cin, word);

std::unordered\_map<std::string, std::string> translations;

std::cout << "Enter translations (lang:translation, type 'done' to finish):\n";

while (true) {

std::cout << "Language code (e.g., en, es, fr) or 'done': ";

std::getline(std::cin, lang);

if (lang == "done") break;

std::cout << "Translation: ";

std::getline(std::cin, translation);

translations[lang] = translation;

}

dict.insert(word, translations);

std::cout << "Word inserted successfully!\n";

break;

}

case 2: {

std::cout << "Enter word to search: ";

std::getline(std::cin, word);

auto translations = dict.search(word);

if (translations.empty()) {

std::cout << "Word not found!\n";

} else {

std::cout << "Translations:\n";

for (const auto& t : translations) {

std::cout << t.first << ": " << t.second << "\n";

}

}

break;

}

case 3: {

std::cout << "Enter prefix: ";

std::getline(std::cin, prefix);

auto results = dict.startsWith(prefix);

if (results.empty()) {

std::cout << "No words found with prefix '" << prefix << "'\n";

} else {

std::cout << "Words with prefix '" << prefix << "':\n";

for (const auto& result : results) {

std::cout << result.first << ": ";

for (const auto& t : result.second) {

std::cout << t.first << "=" << t.second << " ";

}

std::cout << "\n";

}

}

break;

}

case 4: {

std::cout << "Enter word to delete: ";

std::getline(std::cin, word);

if (dict.deleteWord(word)) {

std::cout << "Word deleted successfully!\n";

} else {

std::cout << "Word not found!\n";

}

break;

}

case 5:

std::cout << "Exiting...\n";

break;

default:

std::cout << "Invalid choice! Try again.\n";

}

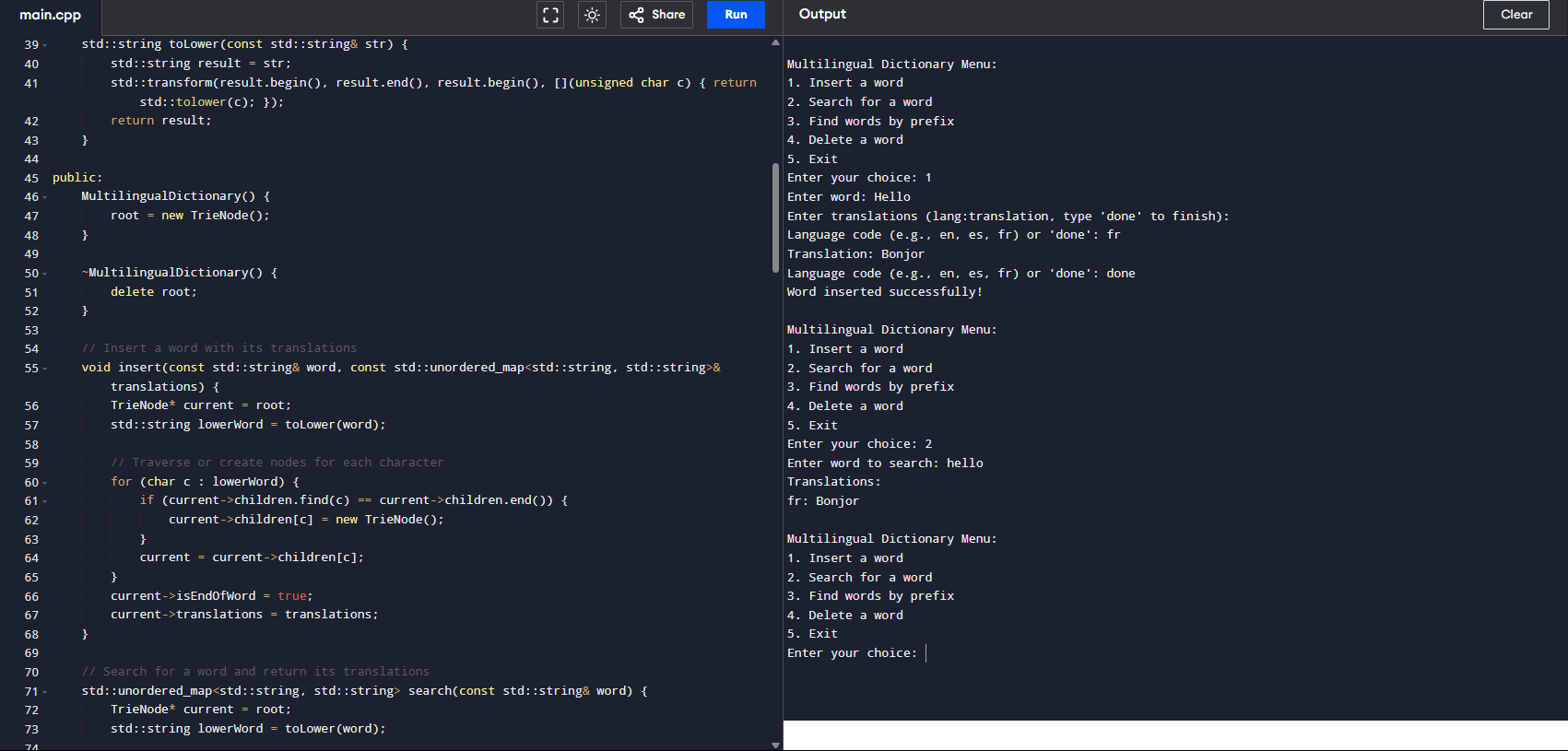
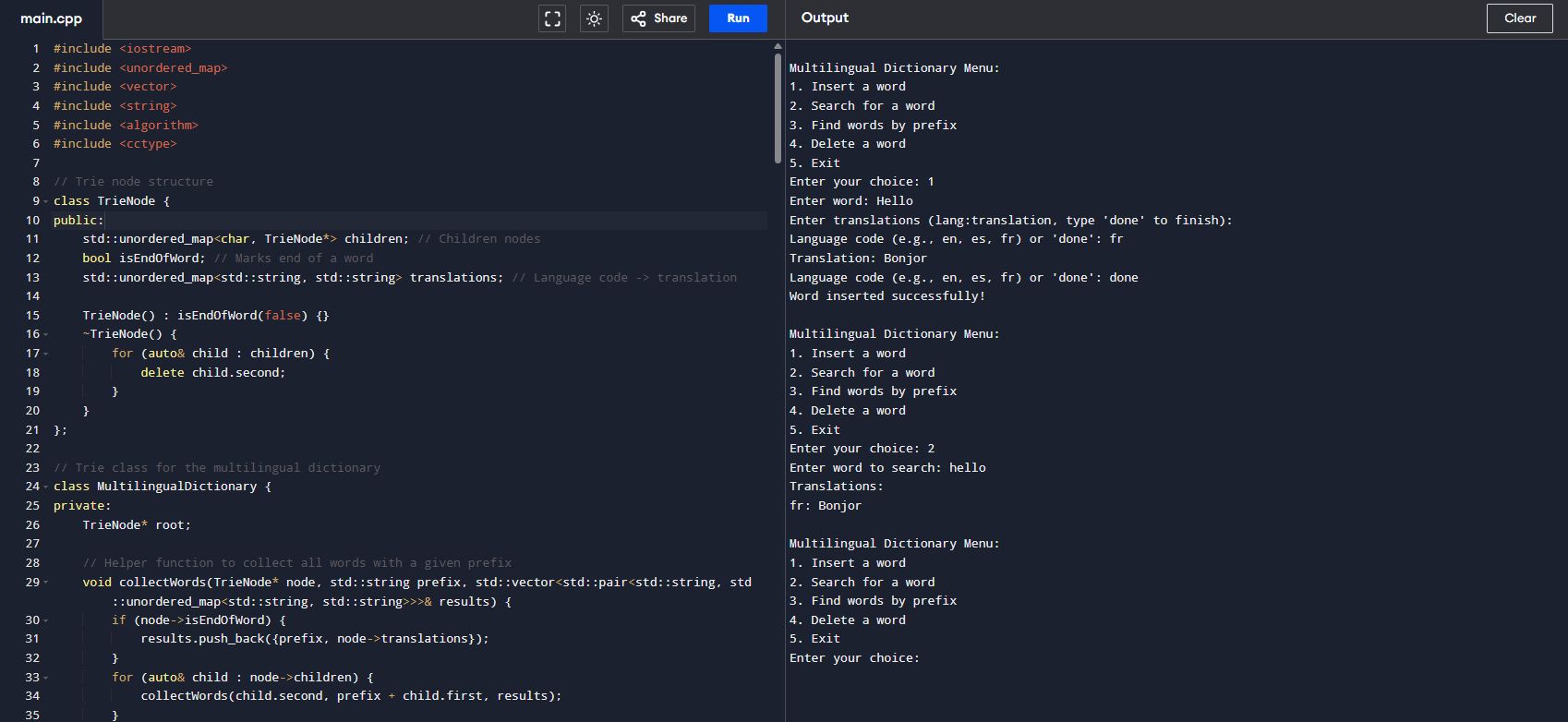
} while (choice != 5);

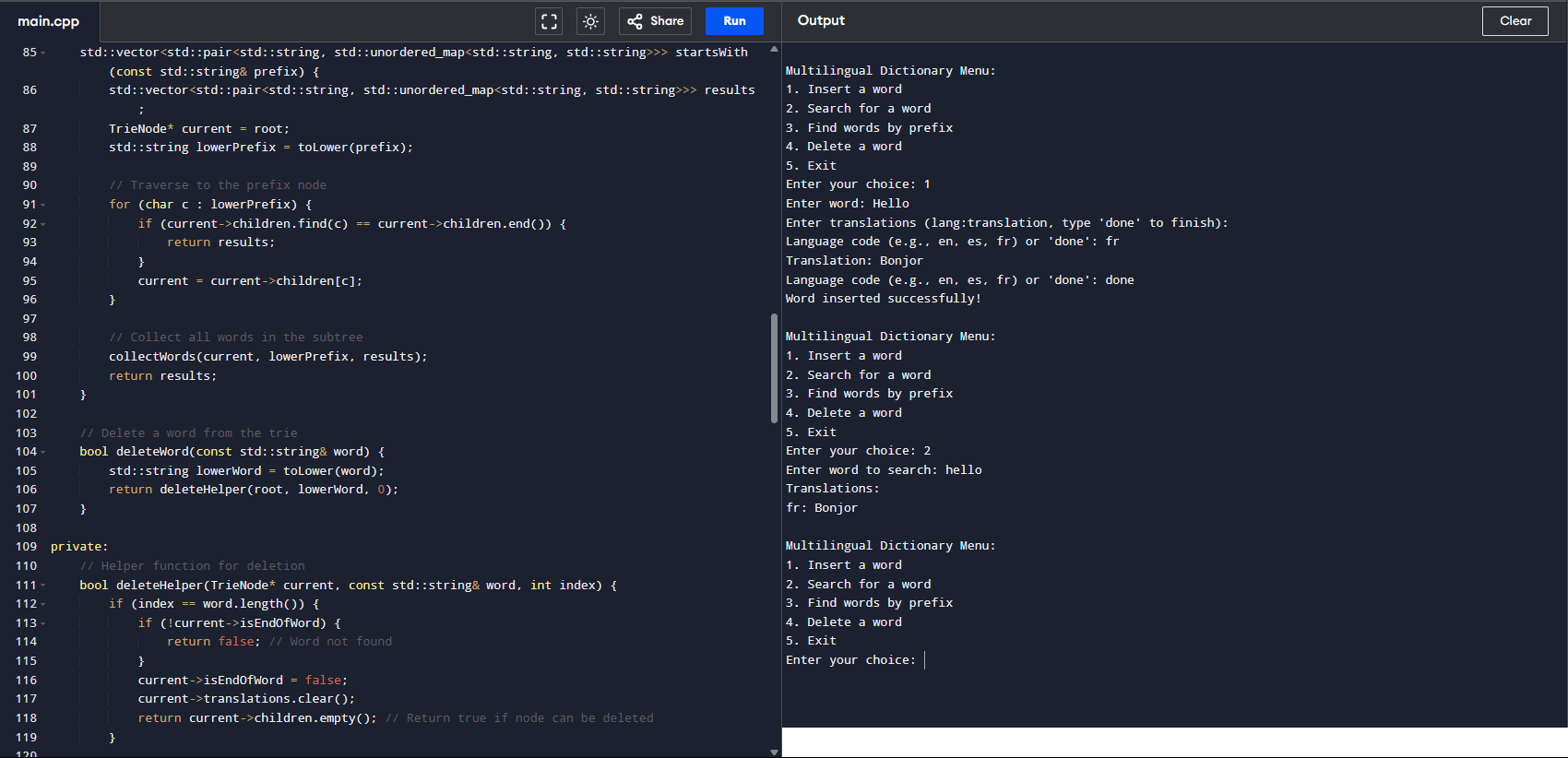
return 0;

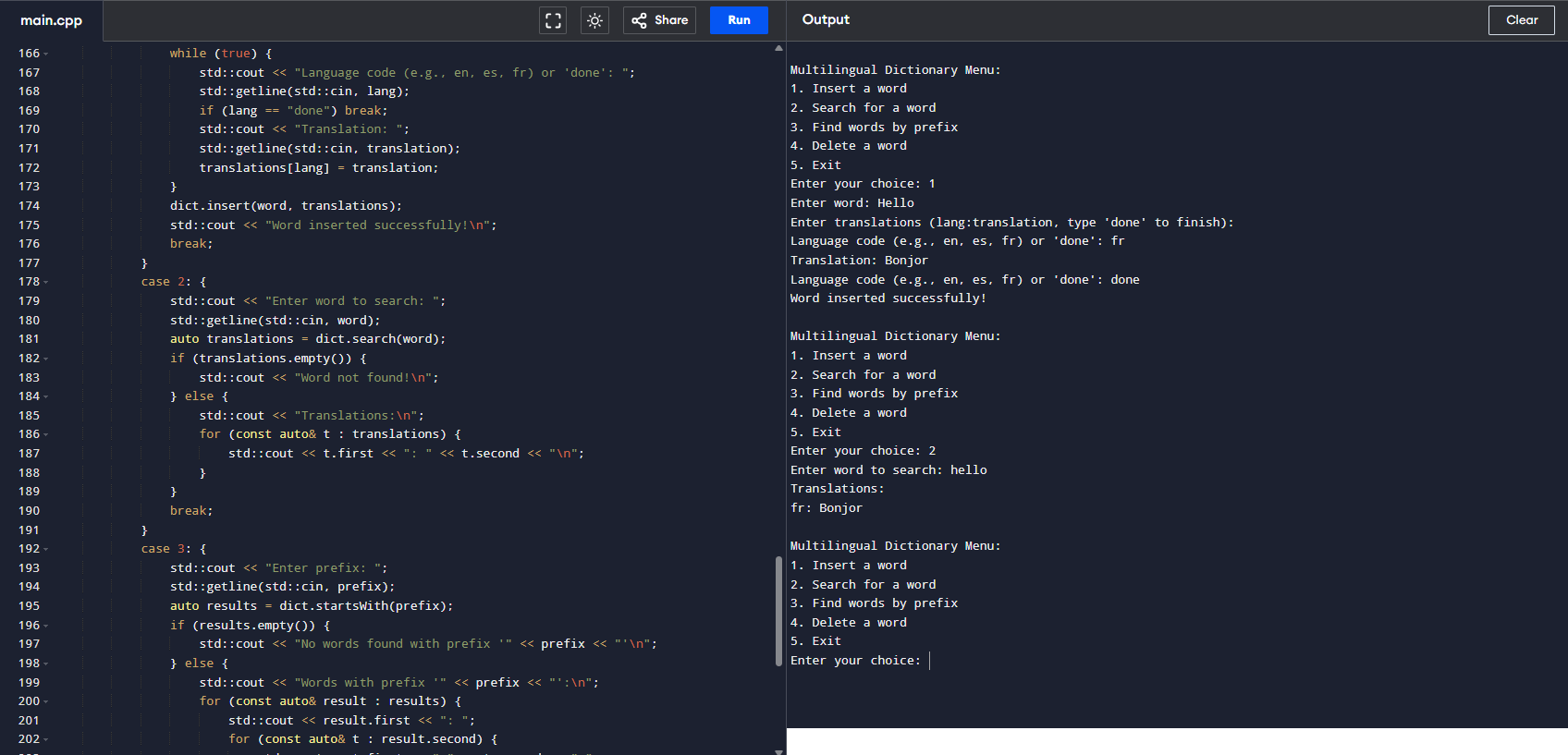
}

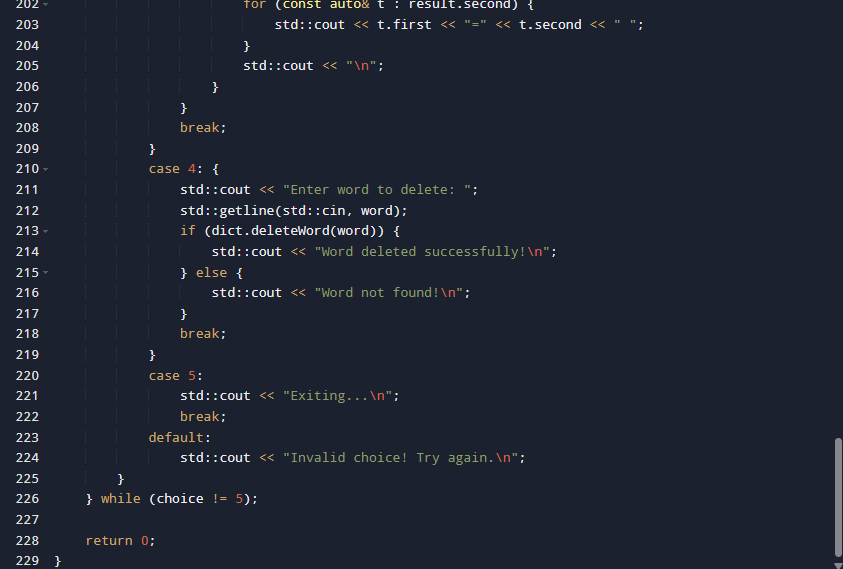
# CHAPTER 4: RESULTS AND DISCUSSION

The implemented application was able to:  
- Store words and their translations in a Trie.  
- Search and retrieve multilingual translations.  
- Provide autocomplete suggestions based on prefixes.

**Input and Output**

****

****

****

## Challenges Faced

- Designing the TrieNode to handle multiple languages.  
- Managing complex input and output formatting in C++.

**CHAPTER 5: CONCLUSION**

This project demonstrated the use of the Trie data structure for building an efficient multilingual dictionary in C++. The knowledge gained during its implementation strengthened the understanding of strings, maps, and class structures in C++.