**Mini Project:**

**Mini-Compiler-> Language Detection (6 lang.) & Syntax Checker**

Made By-

Aryan Khandelwal (E215)

Vaibhav Tayal (E221)

Pranay Nepalia (E226)

Bhavya Verma (E265)

**Code:**

**Language Detection:**

#include <iostream>

#include <string>

#include <unordered\_set>

using namespace std;

bool contains(const string &code, const string &keyword) {

    return code.find(keyword) != string::npos;

}

bool containsExact(const string &code, const string &keyword) {

    return code.find(keyword) != string::npos &&

           (code.find(keyword + " ") != string::npos ||

            code.find(keyword + "(") != string::npos ||

            code.find(keyword + ":") != string::npos);

}

bool isPythonConditional(const string &code) {

    return contains(code, "for ") && contains(code, "in range") || contains(code, "while ") ||

           contains(code, "if ") || contains(code, "elif ") || contains(code, "else:");

}

bool isCppConditional(const string &code) {

    return (contains(code, "for (") || contains(code, "while (") ||

            contains(code, "do {") || contains(code, "if (") ||

            contains(code, "else if (") || contains(code, "else {")) &&

           contains(code, "cout");

}

bool isJavaConditional(const string &code) {

    return (contains(code, "for (") || contains(code, "while (") ||

            contains(code, "do {") || contains(code, "if (") ||

            contains(code, "else if (") || contains(code, "else {")) &&

           (contains(code, "System.out.println") || contains(code, "public class") || contains(code, "static void main"));

}

bool isJavaScriptConditional(const string &code) {

    return (contains(code, "for (") || contains(code, "while (") ||

            contains(code, "if (") || contains(code, "else if (") || contains(code, "else {")) &&

            contains(code, "console.log");

}

void detectGeneralLanguage(const string &code, unordered\_set<string> &detectedLanguages) {

    if (contains(code, "def") || containsExact(code, "print")) {

        detectedLanguages.insert("Python");

    }

    if (contains(code, "#include <iostream>") || contains(code, "cout") ||

        contains(code, "int main") || contains(code, "struct")) {

        detectedLanguages.insert("C++");

    }

    if (contains(code, "#include <stdio.h>") &&

        (contains(code, "printf") || contains(code, "scanf"))) {

        detectedLanguages.insert("C");

    }

    if (contains(code, "public class") || contains(code, "System.out.println") ||

        contains(code, "static void main")) {

        detectedLanguages.insert("Java");

    }

    if (contains(code, "<html>") || contains(code, "<body>") ||

        contains(code, "<div>") || contains(code, "<table>")) {

        detectedLanguages.insert("HTML");

    }

    if (contains(code, "function") || contains(code, "console.log") ||

        contains(code, "let") || contains(code, "const") || contains(code, "var")) {

        detectedLanguages.insert("JavaScript");

    }

}

string detectLanguage(const string &code) {

    unordered\_set<string> detectedLanguages;

    detectGeneralLanguage(code, detectedLanguages);

    if (isPythonConditional(code)) {

        detectedLanguages.insert("Python");

    }

    if (isCppConditional(code)) {

        detectedLanguages.insert("C++");

    }

    if (isJavaConditional(code)) {

        detectedLanguages.insert("Java");

    }

    if (isJavaScriptConditional(code)) {

        detectedLanguages.insert("JavaScript");

    }

    if (detectedLanguages.size() > 1) {

        return "Error: Mixed Language Code";

    } else if (detectedLanguages.empty()) {

        return "Unknown Language";

    }

    return \*detectedLanguages.begin();

}

int main() {

    string code;

    string line;

    cout << "Enter the block of code (type 'END' on a new line to finish):\n";

    while (true) {

        getline(cin, line);

        if (line == "END")

            break;

        code += line + "\n";

    }

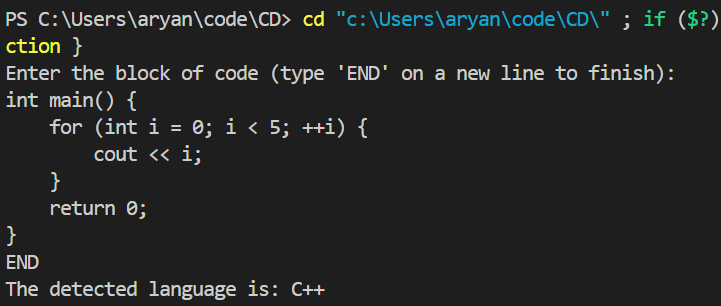
    string language = detectLanguage(code);

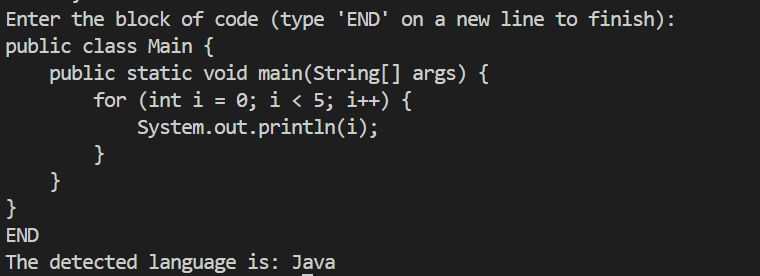
    cout << "The detected language is: " << language << endl;

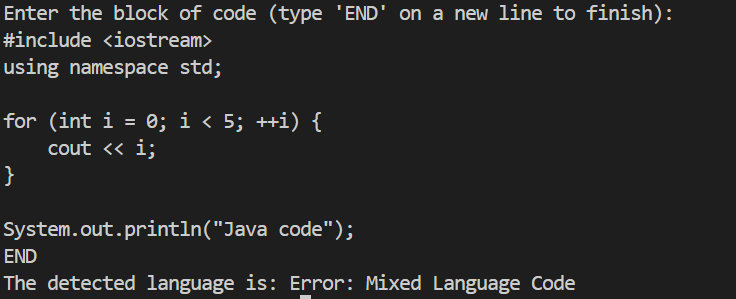
    return 0;

}

**Output:**

****

****

****

**Syntax Checker:**

#include <iostream>

#include <stack>

#include <regex>

#include <string>

#include <unordered\_set>

#include <sstream>

#include<unordered\_map>

using namespace std;

unordered\_set<string> declaredVariables;

unordered\_set<string> validHeaders = {

    "iostream", "vector", "string", "map", "unordered\_map", "algorithm",

    "stack", "queue", "set", "cmath", "cstdlib", "cstdio", "climits", "cfloat"

};

bool checkHeaderSyntax(const string &code) {

    stringstream ss(code);

    string line;

    regex headerPattern(R"(#\s\*include\s\*[<\"](\w+)\.h?[>\"])");  // Match both `<...>` and `"..."`

    bool hasErrors = false;

    smatch match;

    while (getline(ss, line)) {

        if (regex\_search(line, match, headerPattern)) {

            string headerName = match[1];

            if (validHeaders.find(headerName) == validHeaders.end()) {

                cout << "Error: Invalid or unsupported header: " << headerName << endl;

                hasErrors = true;

            }

        } else if (line.find("#include") != string::npos) {

            cout << "Error: Incorrect header syntax: " << line << endl;

            hasErrors = true;

        }

    }

    return hasErrors;

}

unordered\_set<string> validKeywords = {

    "while", "for", "if", "else", "int", "float", "double", "char",

    "string", "bool", "class", "struct", "return", "void", "public",

    "private", "protected", "const", "static", "virtual"

};

unordered\_map<string, string> keywordMisspellings = {

    {"iff", "if"}, {"elseif", "else if"}, {"elif", "else if"},

    {"wile", "while"}, {"fore", "for"}, {"mian", "main"},

    {"naim", "main"}, {"pubic", "public"}, {"privat", "private"},

    {"proteced", "protected"},{"cuot","cout"}

};

bool checkBalancedBrackets(const string &code) {

    stack<char> brackets;

    for (char ch : code) {

        if (ch == '{' || ch == '(' || ch == '[') {

            brackets.push(ch);

        } else if (ch == '}' || ch == ')' || ch == ']') {

            if (brackets.empty()) return false;

            char top = brackets.top();

            if ((ch == '}' && top != '{') || (ch == ')' && top != '(') || (ch == ']' && top != '[')) {

                return false;

            }

            brackets.pop();

        }

    }

    return brackets.empty();

}

bool checkMainFunction(const string &code) {

    return regex\_search(code, regex(R"(\bint\s+main\s\*\(\s\*\))"));

}

bool checkKeywordMisspellings(const string &code) {

    stringstream ss(code);

    string word;

    bool hasError = false;

    while (ss >> word) {

        if (keywordMisspellings.find(word) != keywordMisspellings.end()) {

            cout << "Error: Possible misspelling of keyword '" << word

                 << "'. Did you mean '" << keywordMisspellings[word] << "'?" << endl;

            hasError = true;

        }

    }

    return !hasError;

}

bool checkVariableDeclarations(const string &line) {

    regex varPattern(R"(\b(?:int|float|double|char|string|bool)\s+(\w+)\s\*(?:=|;|,|\s))");

    smatch match;

    bool valid = false;

    string tempLine = line;

    while (regex\_search(tempLine, match, varPattern)) {

        declaredVariables.insert(match[1]);

        tempLine = match.suffix().str();

        valid = true;

    }

    return valid;

}

bool checkUndeclaredVariables(const string &line) {

    regex varUsePattern(R"(\b(\w+)\s\*=)");

    smatch match;

    bool hasErrors = false;

    string tempLine = line;

    while (regex\_search(tempLine, match, varUsePattern)) {

        string varName = match[1];

        if (declaredVariables.find(varName) == declaredVariables.end()) {

            cout << "Error: Undeclared variable: " << varName << endl;

            hasErrors = true;

        }

        tempLine = match.suffix().str();

    }

    return !hasErrors;

}

bool checkCinCoutOperators(const string &code) {

    regex coutMisuse(R"(cout\s\*>>)");

    regex cinMisuse(R"(cin\s\*<<)");

    bool hasErrors = false;

    if (regex\_search(code, coutMisuse)) {

        cout << "Error: Incorrect operator '>>' used with 'cout'. Use '<<' instead." << endl;

        hasErrors = true;

    }

    if (regex\_search(code, cinMisuse)) {

        cout << "Error: Incorrect operator '<<' used with 'cin'. Use '>>' instead." << endl;

        hasErrors = true;

    }

    return hasErrors;

}

bool checkOperators(const string &code) {

    regex assignmentInCondition(R"(\bif\s\*\(\s\*\w+\s\*=\s\*\w+\s\*\))");

    bool hasErrors = false;

    if (regex\_search(code, assignmentInCondition)) {

        cout << "Warning: Possible use of assignment (=) instead of comparison (==) in condition." << endl;

        hasErrors = true;

    }

    return hasErrors;

}

bool checkSemicolons(const string &code) {

    stringstream ss(code);

    string line;

    int lineNumber = 1;

    bool hasErrors = false;

    regex conditionalPattern(R"((if|while|for)\s\*\(.+\)\s\*;)");

    while (getline(ss, line)) {

        line = regex\_replace(line, regex(R"(^\s+|\s+$)"), "");

        if (regex\_search(line, conditionalPattern)) {

            cout << "Error: Conditional or loop statement should not end with a semicolon on line "

                 << lineNumber << ": " << line << endl;

            hasErrors = true;

        }

        else if (!line.empty() && line.back() != ';' && line.back() != '{' && line.back() != '}') {

            cout << "Error: Missing semicolon on line " << lineNumber << ": " << line << endl;

            hasErrors = true;

        }

        lineNumber++;

    }

    return !hasErrors;

}

void runSyntaxChecks(const string &code) {

    if (!checkKeywordMisspellings(code)) return;

    if (!checkBalancedBrackets(code)) {

        cout << "Error: Mismatched or unbalanced brackets." << endl;

    }

    if (!checkMainFunction(code)) {

        cout << "Error: 'int main()' function is missing or incorrectly declared." << endl;

    }

    checkHeaderSyntax(code);

    stringstream ss(code);

    string line;

    while (getline(ss, line)) {

        checkVariableDeclarations(line);

        checkUndeclaredVariables(line);

    }

    checkSemicolons(code);

    checkOperators(code);

    checkCinCoutOperators(code);

    cout << "Syntax check completed." << endl;

}

int main() {

    string code;

    string line;

    cout << "Enter C++ code to check for syntax errors (type 'END' to finish):\n";

    while (true) {

        getline(cin, line);

        if (line == "END") break;

        code += line + "\n";

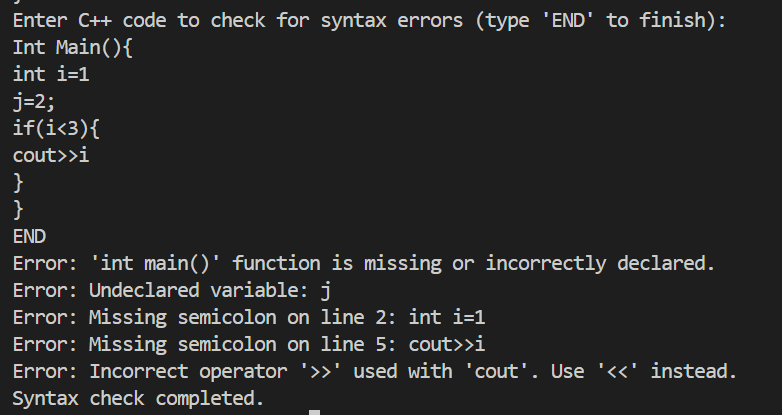
    }

    runSyntaxChecks(code);

    return 0;

}

**Output:**

****