Universidad Católica San Pablo

If-Structure Syntax Checker

Computer Science — Programming Languages

Harold Alejandro Villanueva Borda

1. Cplusplus code of the If structure syntax checker

```cpp
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>

bool Check_If_Syntax(const std::string& code) {
    // Find the position of the first '(' and the first ')'.
    size_t Parenthesis_Opens = code.find('(');
    size_t Parenthesis_Closes = code.find(')');

    // Verify that both parentheses exist and that the closing parenthesis is after the opening parenthesis.
    if (Parenthesis_Opens == std::string::npos || Parenthesis_Closes == std::string::npos || Parenthesis_Closes <= Parenthesis_Opens) return false;


    // Extract the expression and sentence from the "if"
    std::string expression = code.substr(Parenthesis_Opens + 1, Parenthesis_Closes - Parenthesis_Opens - 1);
    std::string sentence = code.substr(Parenthesis_Closes + 1);

    // Verify that the expression and sentence have the correct structure
    if (expression.empty() || sentence.empty() || sentence[0] != '{' || sentence[sentence.size() - 1] != '}') return false;


    // Convert the expression to a vector of tokens
    std::vector<std::string> Tokens_Expression;
    std::string token = "";
    for (char c : expression) {
        if (isspace(c)) {
            continue;
        }
        if (c == '(' || c == ')' || c == '{' || c == '}' || c == ';' || c == ',' || c == '.' || c == '[' || c == ']') {
            if (!token.empty()) {
                Tokens_Expression.push_back(token);
                token = "";
            }
            Tokens_Expression.push_back(std::string(1, c));
        } else {
            token += c;
        }
    }
    if (!token.empty()) {
        Tokens_Expression.push_back(token);
    }

    // Verify that the expression has a correct structure
    std::vector<std::string> operators = {"==", "!=", "<=", ">=", "<", ">", "&&", "||", "=", "+=",
     "-=", "*=", "/=", "%=", "++", "--", "+", "-", "*", "/", "%", "!", "or", "and", "xor", "not"};
    bool wait_operand = true;
    bool wait_operator = false;
    bool wait_operator_binary = false;
    bool wait_Parenthesis_Closes = false;
    bool wait_semicolon = false;
    bool wait_Close_Brackets = false;
    bool wait_Unary_Operator = false;
    int num_Parenthesis_Open = 0;
    int num_Brackets_Open = 0;
    for (std::string token : Tokens_Expression) {
        if (wait_operand) {
            if (isdigit(token[0]) || isalpha(token[0]) || token[0] == '_' || token[0] == '-') {
                wait_operand = false;
                wait_operator = true;
                wait_operator_binary = true;
                wait_Parenthesis_Closes = true;
                wait_Unary_Operator = false;
                wait_semicolon = true;
                wait_Close_Brackets = false;
            } else if (token == "(") {
                wait_Parenthesis_Closes = true;
                wait_operator = true;
                wait_operator_binary = true;
                wait_operand = true;
                wait_Unary_Operator = false;
                wait_semicolon = true;
                wait_Close_Brackets = false;
                num_Parenthesis_Open
   ++;
            } else if (token == "[") {
                wait_Close_Brackets = true;
                wait_operator = true;
                wait_operator_binary = true;
                wait_operand = true;
                wait_Unary_Operator = false;
                wait_semicolon = true;
                wait_Parenthesis_Closes = false;
                num_Brackets_Open++;
            } else {
                return false;
            }
        } else if (wait_operator) {
            if (find(operators.begin(), operators.end(), token) != operators.end()) {
                wait_operand = true;
                wait_operator = false;
                wait_operator_binary = false;
                wait_Parenthesis_Closes = false;
                wait_Unary_Operator = true;
                wait_semicolon = false;
                wait_Close_Brackets = false;
            } else {
                return false;
            }
        } else if (wait_operator_binary) {
            if (find(operators.begin(), operators.end(), token) != operators.end()) {
                wait_operand = true;
                wait_operator = false;
                wait_operator_binary = false;
                wait_Parenthesis_Closes = false;
                wait_Unary_Operator = true;
                wait_semicolon = false;
                wait_Close_Brackets = false;
            } else {
                return false;
            }
        } else if (wait_Parenthesis_Closes) {
            if (token == ")") {
                wait_operand = false;
                wait_operator = true;
                wait_operator_binary = true;
                wait_Parenthesis_Closes = false;
                wait_Unary_Operator = false;
                wait_semicolon = true;
                wait_Close_Brackets = false;
                num_Parenthesis_Open
  --;
            } else {
                return false;
            }
        } else if (wait_Close_Brackets) {
            if (token == "]") {
                wait_operand = false;
                wait_operator = true;
                wait_operator_binary = true;
                wait_Parenthesis_Closes = false;
                wait_Unary_Operator = false;
                wait_semicolon = true;
                wait_Close_Brackets = false;
                num_Brackets_Open--;
            } else {
                return false;
            }
        } else if (wait_Unary_Operator) {
            if (token == "++" || token == "--") {
                wait_operand = false;
                wait_operator = true;
                wait_operator_binary = true;
                wait_Parenthesis_Closes = false;
                wait_Unary_Operator = false;
                wait_semicolon = true;
                wait_Close_Brackets = false;
            } else {
                return false;
            }
        } else if (wait_semicolon && token == ";") {
            return true;
        } else {
            return false;
        }
    }

    // Verify that there are no unclosed parentheses or brackets
    if (num_Parenthesis_Open
    != 0 || num_Brackets_Open != 0) {
        return false;
    }

    // Verify that you have not ended up waiting for an operand.
    if (wait_operand) {
        return false;
    }

    return true;
}
```

Implementation

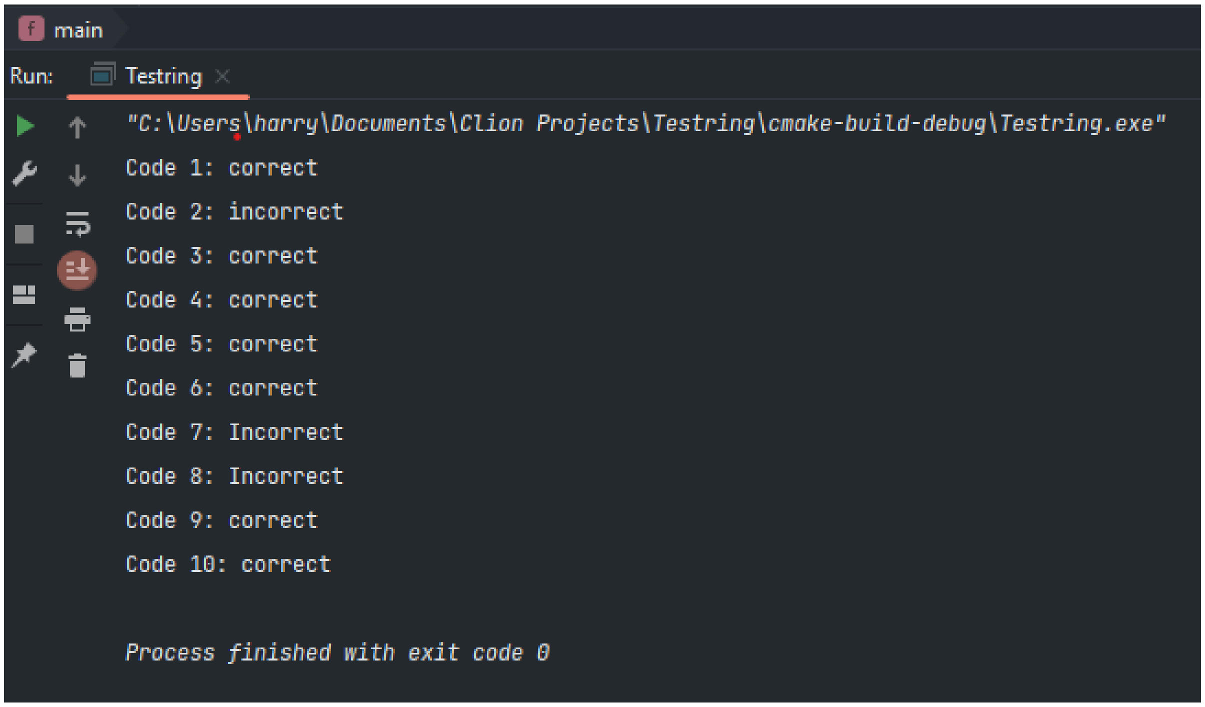
2. Code execution

2.1. Input

```
int main() {

    std::string codigo1 = "if(a == 4){cout << a;}";
    std::cout << (Check_If_Syntax(codigo1) ? "Code 1: correct" : "Code 1: incorrect") << std::endl;
    std::string codigo2 = "if(a == 4;){cout << a;}";
    std::cout << (Check_If_Syntax(codigo2) ? "Code 2: correct" : "Code 2: incorrect") << std::endl;
    std::string codigo3 = "if(a == b){cout << a;}";
    std::cout << (Check_If_Syntax(codigo3) ? "Code 3: correct" : "Code 3: Incorrect") << std::endl;
    std::string codigo4 = "if(a > b){cout << a;}";
    std::cout << (Check_If_Syntax(codigo4) ? "Code 4: correct" : "Code 4: Incorrect") << std::endl;
    std::string codigo5 = "if(a > b && a < c){cout << c;}";
    std::cout << (Check_If_Syntax(codigo5) ? "Code 5: correct" : "Code 5: Incorrect") << std::endl;
    std::string codigo6 = "if(a > b && a a < c || b == a){cout << c;}";
    std::cout << (Check_If_Syntax(codigo6) ? "Code 6: correct" : "Code 6: Incorrect") << std::endl;
    std::string codigo7 = "if(a != b < c ; 1){cout << c;}";
    std::cout << (Check_If_Syntax(codigo7) ? "Code 7: correct" : "Code 7: Incorrect") << std::endl;
    std::string codigo8 = "if(a != b < c) {cout << c}";
    std::cout << (Check_If_Syntax(codigo8) ? "Code 8: correct" : "Code 8: Incorrect") << std::endl;
    std::string codigo9 = "if(a and b){cout << 2;}";
    std::cout << (Check_If_Syntax(codigo9) ? "Code 9: correct" : "Code 9: Incorrect") << std::endl;
    std::string codigo10 = "if(a or b){cout << 20;}";
    std::cout << (Check_If_Syntax(codigo10) ? "Code 10: correct" : "Code 10: Incorrect") << std::
    endl;
    return 0;
}
```

2.2. Output



main

Run: Testring

```
"C:\Users\harry\Documents\Clion Projects\Testring\cmake-build-debug\Testring.exe"
Code 1: correct
Code 2: incorrect
Code 3: correct
Code 4: correct
Code 5: correct
Code 6: correct
Code 7: Incorrect
Code 8: Incorrect
Code 9: correct
Code 10: correct

Process finished with exit code 0
```

Figura 1: Code Execution