



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

```

1  #include <iostream>
2  #include <string>
3  #include <vector>
4  #include <algorithm>
5
6  bool Check_If_Syntax(const std::string& code) {
7      // Find the position of the first '(' and the first ')'.
8      size_t Parenthesis_Opens = code.find('(');
9      size_t Parenthesis_Closes = code.find(')');
10
11     // Verify that both parentheses exist and that the closing parenthesis is after the opening
12     // parenthesis.
13     if (Parenthesis_Opens == std::string::npos || Parenthesis_Closes == std::string::npos ||
14         Parenthesis_Closes <= Parenthesis_Opens) return false;
15
16     // Extract the expression and sentence from the "if"
17     std::string expression = code.substr(Parenthesis_Opens + 1, Parenthesis_Closes -
18         Parenthesis_Opens - 1);
19     std::string sentence = code.substr(Parenthesis_Closes + 1);
20
21     // Verify that the expression and sentence have the correct structure
22     if (expression.empty() || sentence.empty() || sentence[0] != '{' || sentence[sentence.size() -
23         1] != '}') return false;
24
25     // Convert the expression to a vector of tokens
26     std::vector<std::string> Tokens_Expression;
27     std::string token = "";
28     for (char c : expression) {
29         if (isspace(c)) {
30             continue;
31         }
32         if (c == '(' || c == ')' || c == '{' || c == '}' || c == ';' || c == ',' || c == '.' || c
33         == '[' || c == ']') {
34             if (!token.empty()) {
35                 Tokens_Expression.push_back(token);
36                 token = "";
37             }
38             Tokens_Expression.push_back(std::string(1, c));
39         } else {

```

```

37         token += c;
38     }
39 }
40 if (!token.empty()) {
41     Tokens_Expression.push_back(token);
42 }
43
44 // Verify that the expression has a correct structure
45 std::vector<std::string> operators = {"==", "!=", "<=", ">=", "<", ">", "&&", "||", "=", "+=",
46     "-=", "*=", "/=", "%=", "++", "--", "+", "-", "*", "/", "%", "!", "or", "and", "xor", "not"};
47 bool wait_operand = true;
48 bool wait_operator = false;
49 bool wait_operator_binary = false;
50 bool wait_Parenthesis_Closes = false;
51 bool wait_semicolon = false;
52 bool wait_Close_Brackets = false;
53 bool wait_Unary_Operator = false;
54 int num_Parenthesis_Open = 0;
55 int num_Brackets_Open = 0;
56 for (std::string token : Tokens_Expression) {
57     if (wait_operand) {
58         if (isdigit(token[0]) || isalpha(token[0]) || token[0] == '_' || token[0] == '-') {
59             wait_operand = false;
60             wait_operator = true;
61             wait_operator_binary = true;
62             wait_Parenthesis_Closes = true;
63             wait_Unary_Operator = false;
64             wait_semicolon = true;
65             wait_Close_Brackets = false;
66         } else if (token == "(") {
67             wait_Parenthesis_Closes = true;
68             wait_operator = true;
69             wait_operator_binary = true;
70             wait_operand = true;
71             wait_Unary_Operator = false;
72             wait_semicolon = true;
73             wait_Close_Brackets = false;
74             num_Parenthesis_Open
75             ++;
76         } else if (token == "[") {
77             wait_Close_Brackets = true;
78             wait_operator = true;
79             wait_operator_binary = true;
80             wait_operand = true;
81             wait_Unary_Operator = false;
82             wait_semicolon = true;
83             wait_Parenthesis_Closes = false;
84             num_Brackets_Open++;
85         } else {
86             return false;
87         }
88     } else if (wait_operator) {
89         if (find(operators.begin(), operators.end(), token) != operators.end()) {
90             wait_operand = true;
91             wait_operator = false;
92             wait_operator_binary = false;
93             wait_Parenthesis_Closes = false;

```

```
93         wait_Unary_Operator = true;
94         wait_semicolon = false;
95         wait_Close_Brackets = false;
96     } else {
97         return false;
98     }
99 } else if (wait_operator_binary) {
100     if (find(operators.begin(), operators.end(), token) != operators.end()) {
101         wait_operand = true;
102         wait_operator = false;
103         wait_operator_binary = false;
104         wait_Parenthesis_Closes = false;
105         wait_Unary_Operator = true;
106         wait_semicolon = false;
107         wait_Close_Brackets = false;
108     } else {
109         return false;
110     }
111 } else if (wait_Parenthesis_Closes) {
112     if (token == ")") {
113         wait_operand = false;
114         wait_operator = true;
115         wait_operator_binary = true;
116         wait_Parenthesis_Closes = false;
117         wait_Unary_Operator = false;
118         wait_semicolon = true;
119         wait_Close_Brackets = false;
120         num_Parenthesis_Open
121 --;
122     } else {
123         return false;
124     }
125 } else if (wait_Close_Brackets) {
126     if (token == "]") {
127         wait_operand = false;
128         wait_operator = true;
129         wait_operator_binary = true;
130         wait_Parenthesis_Closes = false;
131         wait_Unary_Operator = false;
132         wait_semicolon = true;
133         wait_Close_Brackets = false;
134         num_Brackets_Open--;
135     } else {
136         return false;
137     }
138 } else if (wait_Unary_Operator) {
139     if (token == "+" || token == "--") {
140         wait_operand = false;
141         wait_operator = true;
142         wait_operator_binary = true;
143         wait_Parenthesis_Closes = false;
144         wait_Unary_Operator = false;
145         wait_semicolon = true;
146         wait_Close_Brackets = false;
147     } else {
148         return false;
149     }
150 }
```

```

150     } else if (wait_semicolon && token == ";") {
151         return true;
152     } else {
153         return false;
154     }
155 }
156
157 // Verify that there are no unclosed parentheses or brackets
158 if (num_Parenthesis_Open
159     != 0 || num_Brackets_Open != 0) {
160     return false;
161 }
162
163 // Verify that you have not ended up waiting for an operand.
164 if (wait_operand) {
165     return false;
166 }
167
168 return true;
169 }

```

Implementation

2. Code execution

2.1. Input

```

1  int main() {
2
3      std::string codigo1 = "if(a == 4){cout << a;}";
4      std::cout << (Check_If_Syntax(codigo1) ? "Code 1: correct" : "Code 1: incorrect") << std::endl
5      ;
6      std::string codigo2 = "if(a == 4;){cout << a;}";
7      std::cout << (Check_If_Syntax(codigo2) ? "Code 2: correct" : "Code 2: incorrect") << std::endl
8      ;
9      std::string codigo3 = "if(a == b){cout << a;}";
10     std::cout << (Check_If_Syntax(codigo3) ? "Code 3: correct" : "Code 3: Incorrect") << std::endl
11     ;
12     std::string codigo4 = "if(a > b){cout << a;}";
13     std::cout << (Check_If_Syntax(codigo4) ? "Code 4: correct" : "Code 4: Incorrect") << std::endl
14     ;
15     std::string codigo5 = "if(a > b && a < c){cout << c;}";
16     std::cout << (Check_If_Syntax(codigo5) ? "Code 5: correct" : "Code 5: Incorrect") << std::endl
17     ;
18     std::string codigo6 = "if(a > b && a a < c || b == a){cout << c;}";
19     std::cout << (Check_If_Syntax(codigo6) ? "Code 6: correct" : "Code 6: Incorrect") << std::endl
20     ;
21     std::string codigo7 = "if(a != b < c ; 1){cout << c;}";
22     std::cout << (Check_If_Syntax(codigo7) ? "Code 7: correct" : "Code 7: Incorrect") << std::endl
23     ;
24     std::string codigo8 = "if(a != b < c) {cout << c}";
25     std::cout << (Check_If_Syntax(codigo8) ? "Code 8: correct" : "Code 8: Incorrect") << std::endl
26     ;
27     std::string codigo9 = "if(a and b){cout << 2;}";
28     std::cout << (Check_If_Syntax(codigo9) ? "Code 9: correct" : "Code 9: Incorrect") << std::endl
29     ;
30 }

```

```
21     std::string codigo10 = "if(a or b){cout << 20;}";
22     std::cout << (Check_If_Syntax(codigo10) ? "Code 10: correct" : "Code 10: Incorrect") << std::
endl;
23     return 0;
24 }
```

2.2. Output

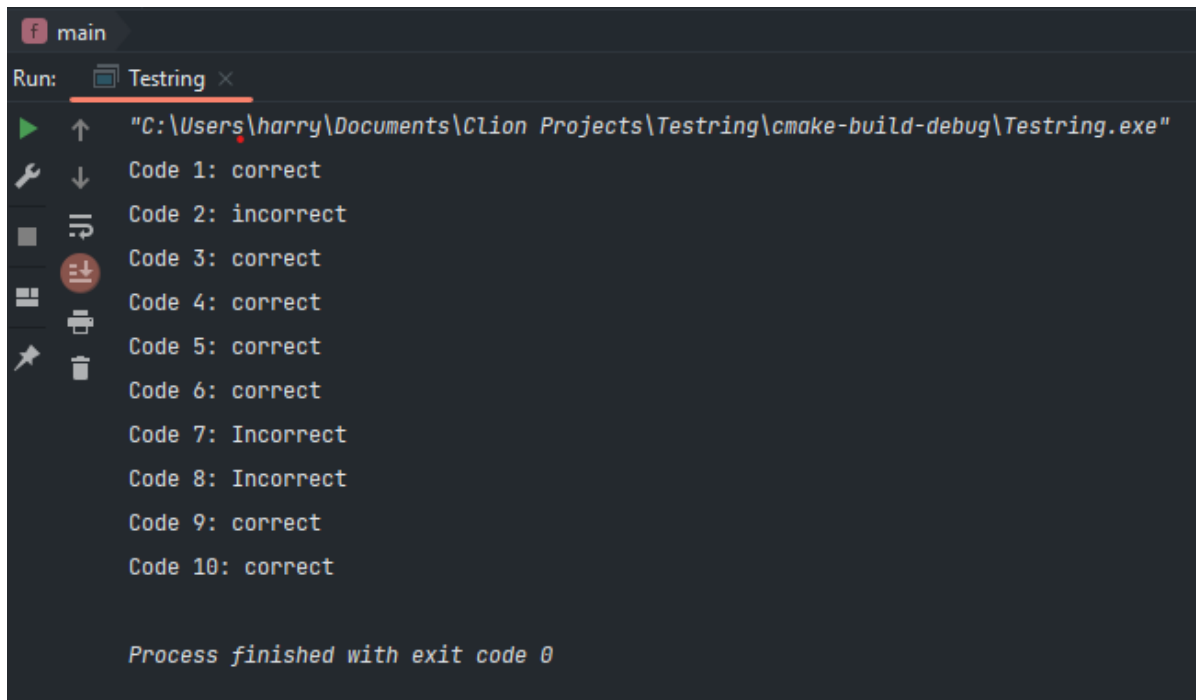


Figura 1: Code Execution