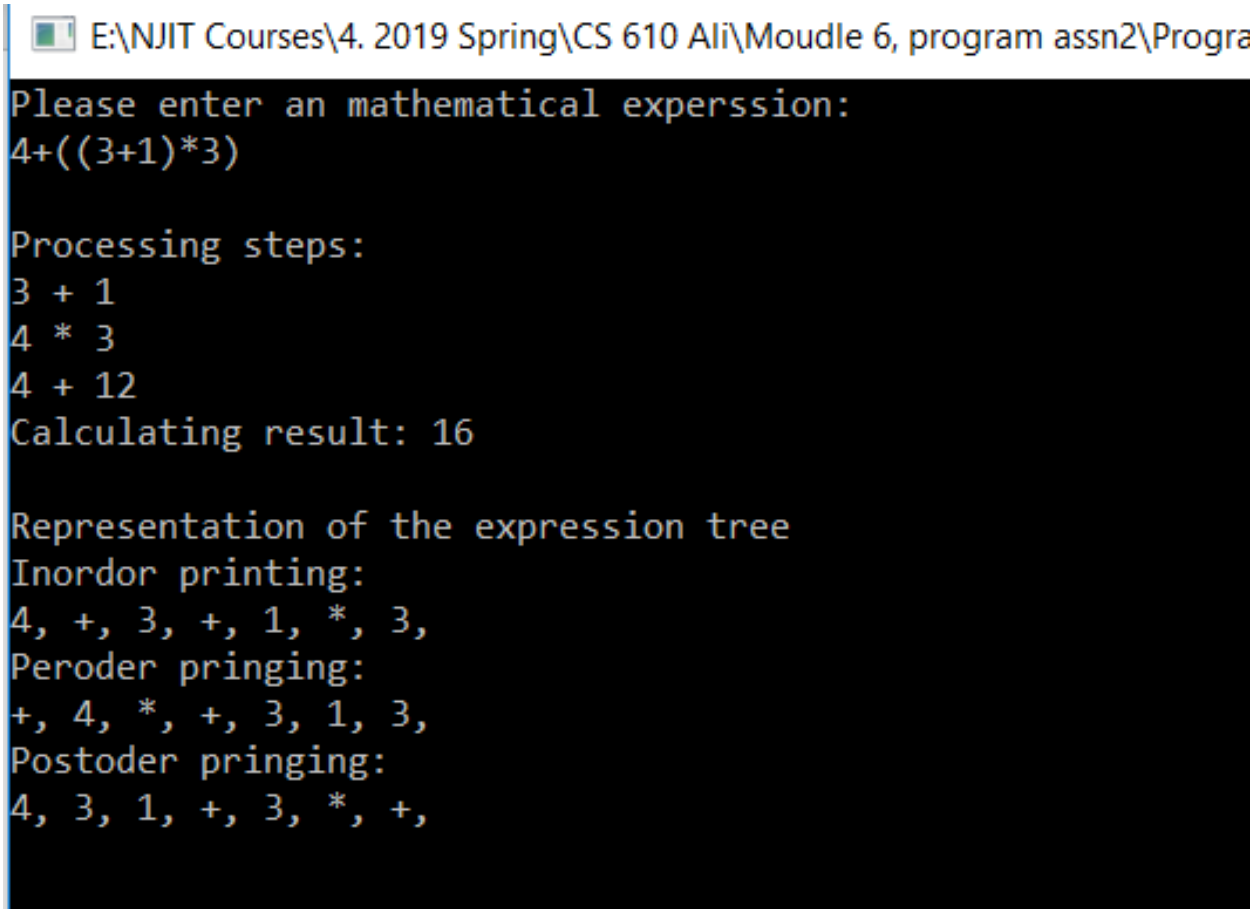
1. 4 + ((3 + 1) * 3)


E:\NJIT Courses\4. 2019 Spring\CS 610 Ali\Moudle 6, program assn2\Progra
Please enter an mathematical experssion:
4+((3+1)*3)
Processing steps:
3 + 1
4 * 3
4 + 12
Calculating result: 16
Representation of the expression tree
Inordor printing:
4, +, 3, +, 1, *, 3,
Peroder pringing:
+, 4, *, +, 3, 1, 3,
Postoder pringing:
4, 3, 1, +, 3, *, +,

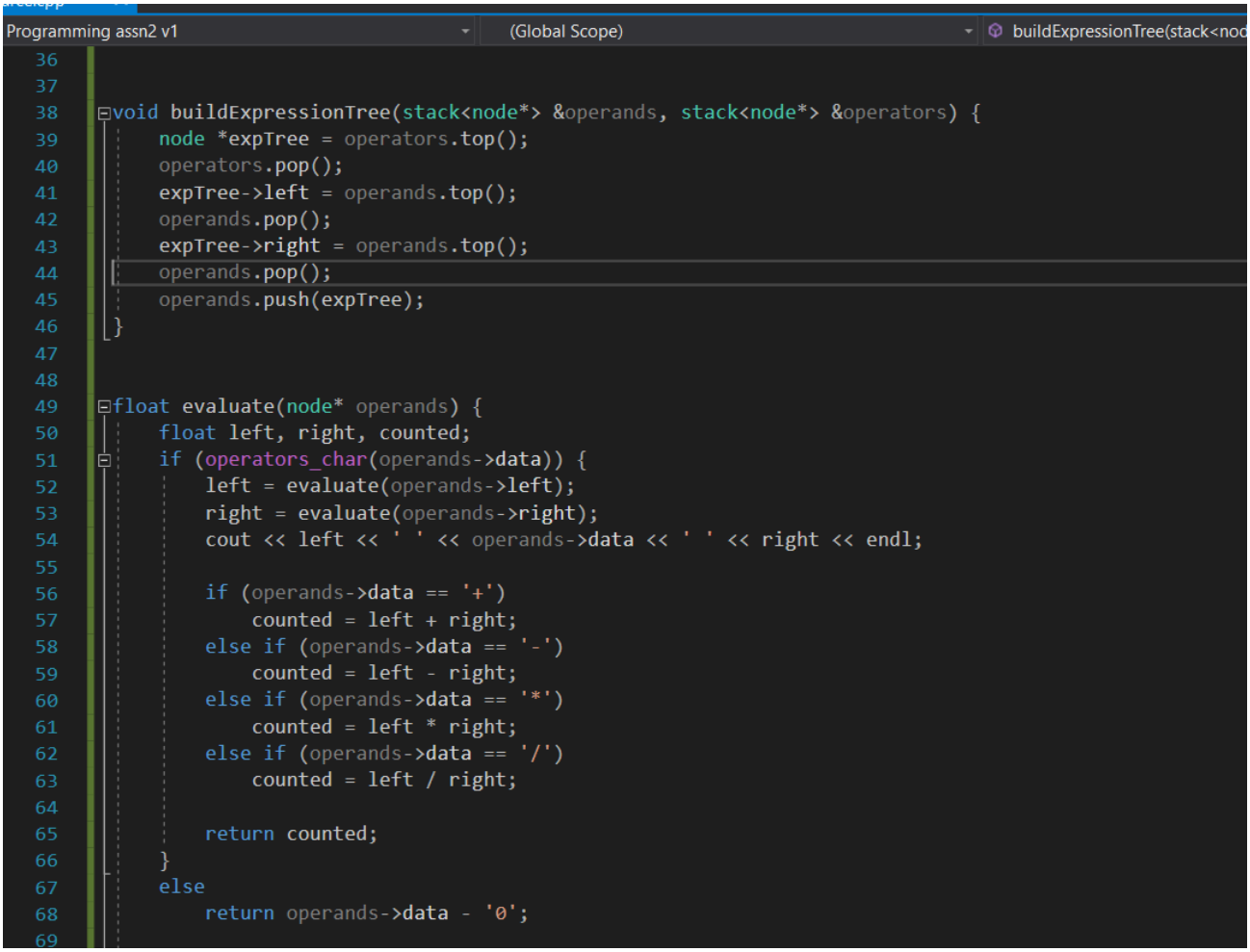
2. (((3 + 1) * 3) / (3 * (7 - 4)))- (((9 - 5) + 2）+6)

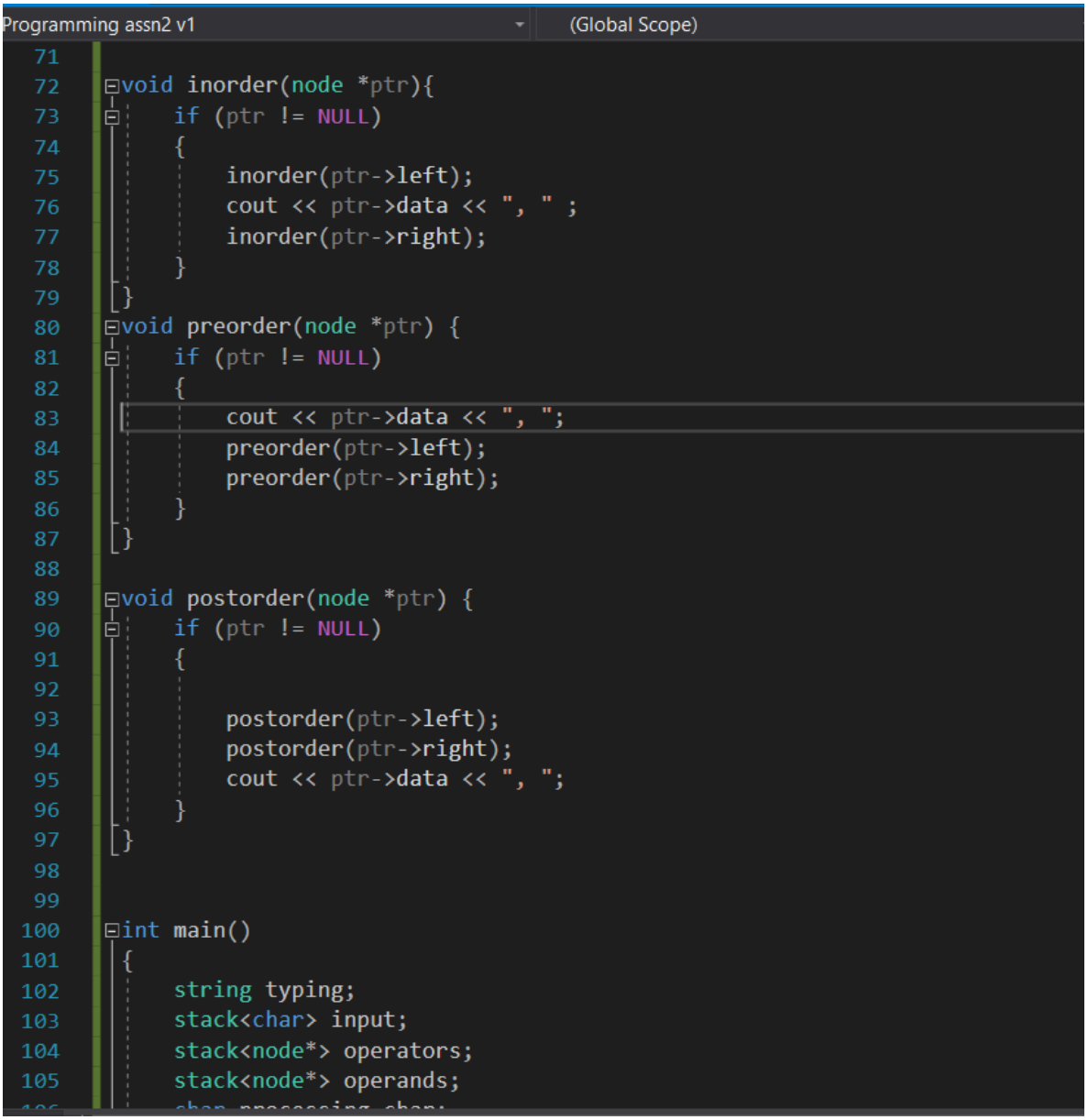
```
Please enter an mathematical experssion:
(((3+1)*3)/(3*(7-4)))-(((9-5)+2)+6)

Processing steps:
3 + 1
4 * 3
7 - 4
3 * 3
12 / 9
9 - 5
4 + 2
6 + 6
1.33333 - 12
Calculating result: -10.6667

Representation of the expression tree
Inordor printing:
3, +, 1, *, 3, /, 3, *, 7, -, 4, -, 9, -, 5, +, 2, +, 6,
Peroder pringing:
-, /, *, +, 3, 1, 3, *, 3, -, 7, 4, +, +, -, 9, 5, 2, 6,
Postoder pringing:
3, 1, +, 3, *, 3, 7, 4, -, *, /, 9, 5, -, 2, +, 6, +, -,
```

Source code:

ogramming assn2 v1 (Global Scope)

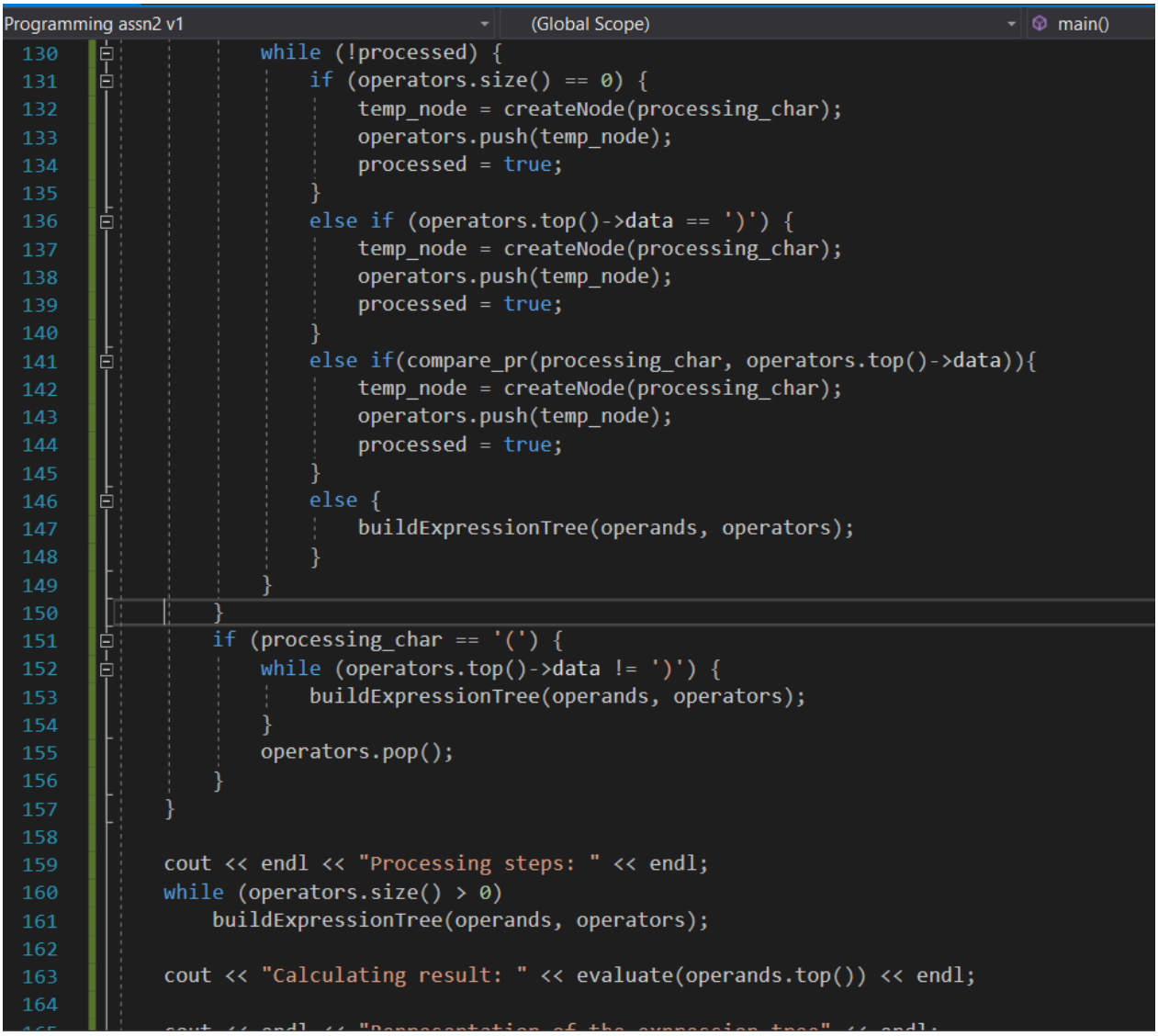
```cpp
#include <iostream>
#include <vector>
#include<string>
#include<stack>
using namespace std;

#define operators_char(_char) ((_char == '+') || (_char == '-') || (_char == '*') || (_char == '/'))


struct node {
    char data;
    struct node *left;
    struct node *right;
};

node* createNode(char value) {
    node *newNode;
    newNode = new node;
    newNode->data = value;
    newNode->left = NULL;
    newNode->right = NULL;

    return newNode;
}

bool compare_pr(char _char, char node_data) {
    if ((_char == '*' || _char == '/') && (node_data == '-' || node_data == '+'))
        return true;
    else if ((_char == '*' || _char == '/') && (node_data == '*' || node_data == '/'))
        return true;
    else if ((_char == '-' || _char == '+') && (node_data == '-' || node_data == '+'))
        return true;
    else
        return false;
}
```


Programming assn2 v1
(Global Scope)
buildExpressionTree(stack<nod
36
37
38 void buildExpressionTree(stack<node*> &operands, stack<node*> &operators) {
39 node *expTree = operators.top();
40 operators.pop();
41 expTree->left = operands.top();
42 operands.pop();
43 expTree->right = operands.top();
44 operands.pop();
45 operands.push(expTree);
46 }
47
48
49 float evaluate(node* operands) {
50 float left, right, counted;
51 if (operators_char(operands->data)) {
52 left = evaluate(operands->left);
53 right = evaluate(operands->right);
54 cout << left << ' ' << operands->data << ' ' << right << endl;
55
56 if (operands->data == '+')
57 counted = left + right;
58 else if (operands->data == '-')
59 counted = left - right;
60 else if (operands->data == '*')
61 counted = left * right;
62 else if (operands->data == '/')
63 counted = left / right;
64
65 return counted;
66 }
67 else
68 return operands->data - '0';
69


Programming assn2 v1
(Global Scope)
71
72 void inorder(node *ptr){
73 if (ptr != NULL)
74 {
75 inorder(ptr->left);
76 cout << ptr->data << ", " ;
77 inorder(ptr->right);
78 }
79 }
80 void preorder(node *ptr) {
81 if (ptr != NULL)
82 {
83 cout << ptr->data << ", ";
84 preorder(ptr->left);
85 preorder(ptr->right);
86 }
87 }
88
89 void postorder(node *ptr) {
90 if (ptr != NULL)
91 {
92
93 postorder(ptr->left);
94 postorder(ptr->right);
95 cout << ptr->data << ", ";
96 }
97 }
98
99
100 int main()
101 {
102 string typing;
103 stack<char> input;
104 stack<node*> operators;
105 stack<node*> operands;

Programming assn2 v1 (Global Scope) main()

```
 99
100  int main()
101  {
102      string typing;
103      stack<char> input;
104      stack<node*> operators;
105      stack<node*> operands;
106      char processing_char;
107      node *temp_node;
108
109      cout << "Please enter an mathematical experssion: "<<endl;
110      cin >> typing;
111
112      for (int i = 0; i < typing.length(); i++) {
113          input.push(typing[i]);
114      }
115
116      while (input.size() != 0) {
117          processing_char = input.top();
118          input.pop();
119
120          if (isdigit(processing_char)) {
121              temp_node = createNode(processing_char);
122              operands.push(temp_node);
123          }
124          if (processing_char == ')') {
125              temp_node = createNode(processing_char);
126              operators.push(temp_node);
127          }
128          if (operators_char(processing_char)) {
129              bool processed = false;
130              while (!processed) {
131                  if (operators.size() == 0) {
132                      temp_node = createNode(processing_char);
133                      operators.push(temp_node);
```


Programming assn2 v1
(Global Scope)
main()
while (!processed) {
if (operators.size() == 0) {
temp_node = createNode(processing_char);
operators.push(temp_node);
processed = true;
}
else if (operators.top()->data == ')') {
temp_node = createNode(processing_char);
operators.push(temp_node);
processed = true;
}
else if(compare_pr(processing_char, operators.top()->data)){
temp_node = createNode(processing_char);
operators.push(temp_node);
processed = true;
}
else {
buildExpressionTree(operands, operators);
}
}
}
if (processing_char == '(') {
while (operators.top()->data != ')') {
buildExpressionTree(operands, operators);
}
operators.pop();
}
}
cout << endl << "Processing steps: " << endl;
while (operators.size() > 0)
buildExpressionTree(operands, operators);
cout << "Calculating result: " << evaluate(operands.top()) << endl;

Programming assn2 v1 (Global Scope)

```
152             while (operators.top()->data != ')') {
153                 buildExpressionTree(operands, operators);
154             }
155             operators.pop();
156         }
157     }
158
159     cout << endl << "Processing steps: " << endl;
160     while (operators.size() > 0)
161         buildExpressionTree(operands, operators);
162
163     cout << "Calculating result: " << evaluate(operands.top()) << endl;
164
165     cout << endl << "Representation of the expression tree" << endl;
166     cout << "Inordor printing: " << endl;
167     inorder(operands.top());
168     cout << endl;
169
170     cout << "Peroder pringing: " << endl;
171     preorder(operands.top());
172     cout << endl;
173
174     cout << "Postoder pringing: " << endl;
175     postorder(operands.top());
176
177
178
179     return 0;
180 }
181
```