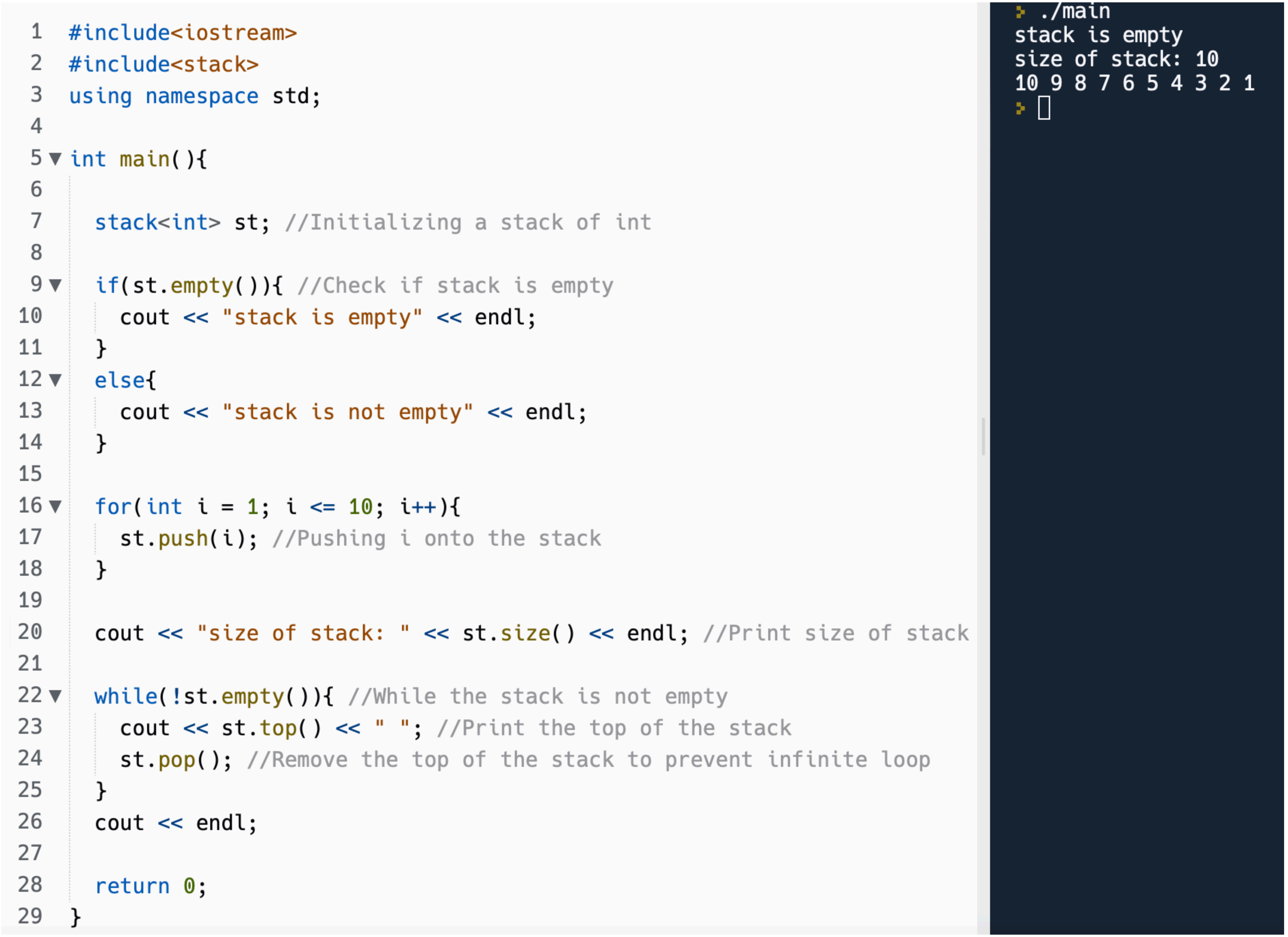
COSC 2436: Stacks

Basic stack functions

- **push()** - pushes a value onto the top of the stack
- **pop()** - removes the top of the stack (**pop()** will not return any value, it will only delete)
- **top()** - returns the value that is on the top of the stack
- **empty()** - returns **true** if stack is empty and returns **false** if stack is not empty
- **size()** - returns the size of the stack
- C++ website: **https://cplusplus.com/reference/stack/stack/**

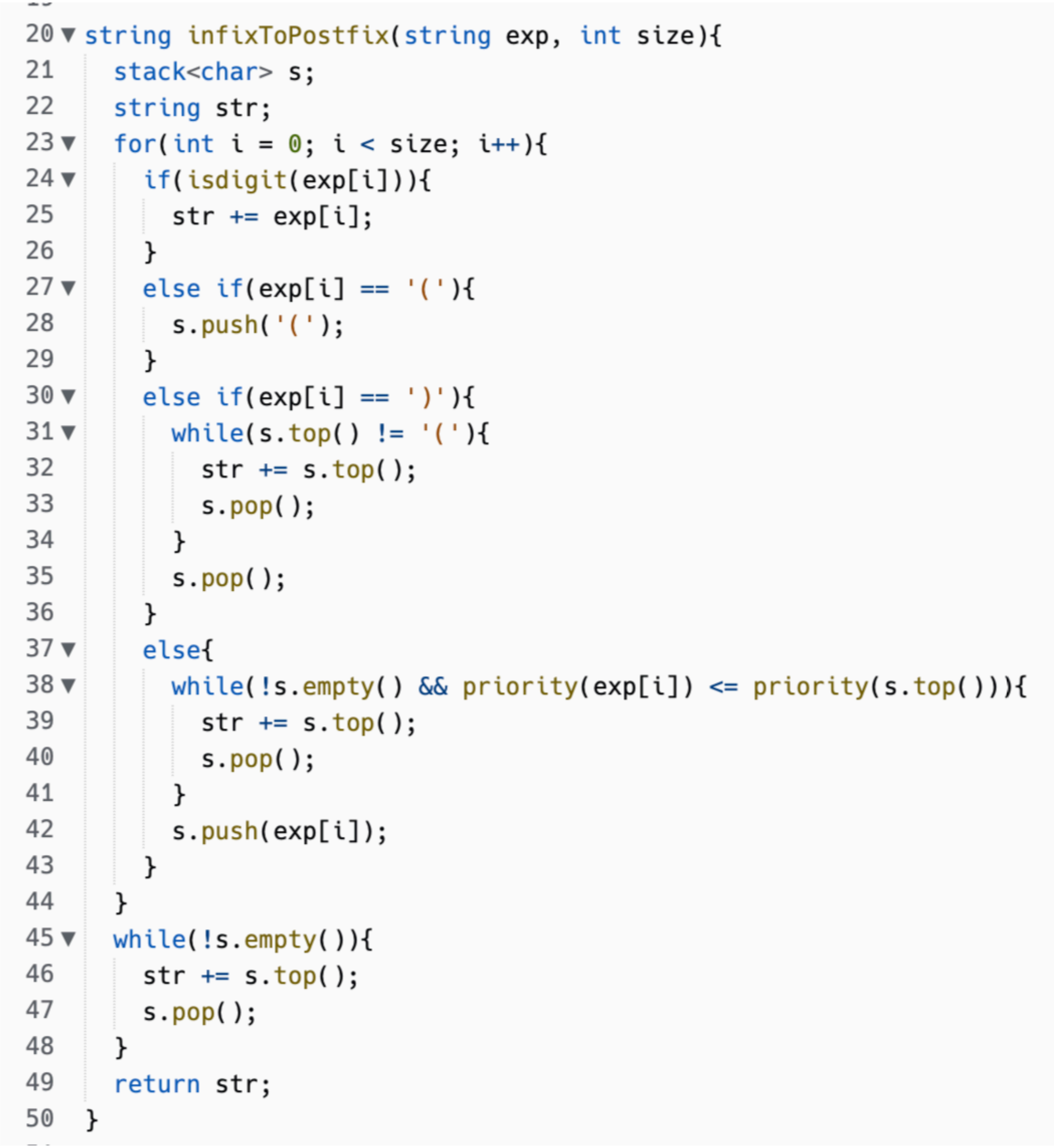
Simple Stack Program



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

int main(){

  stack<int> st; //Initializing a stack of int

  if(st.empty()){ //Check if stack is empty
    cout << "stack is empty" << endl;
  }
  else{
    cout << "stack is not empty" << endl;
  }

  for(int i = 1; i <= 10; i++){
    st.push(i); //Pushing i onto the stack
  }

  cout << "size of stack: " << st.size() << endl; //Print size of stack

  while(!st.empty()){ //While the stack is not empty
    cout << st.top() << " "; //Print the top of the stack
    st.pop(); //Remove the top of the stack to prevent infinite loop
  }
  cout << endl;

  return 0;
}
```

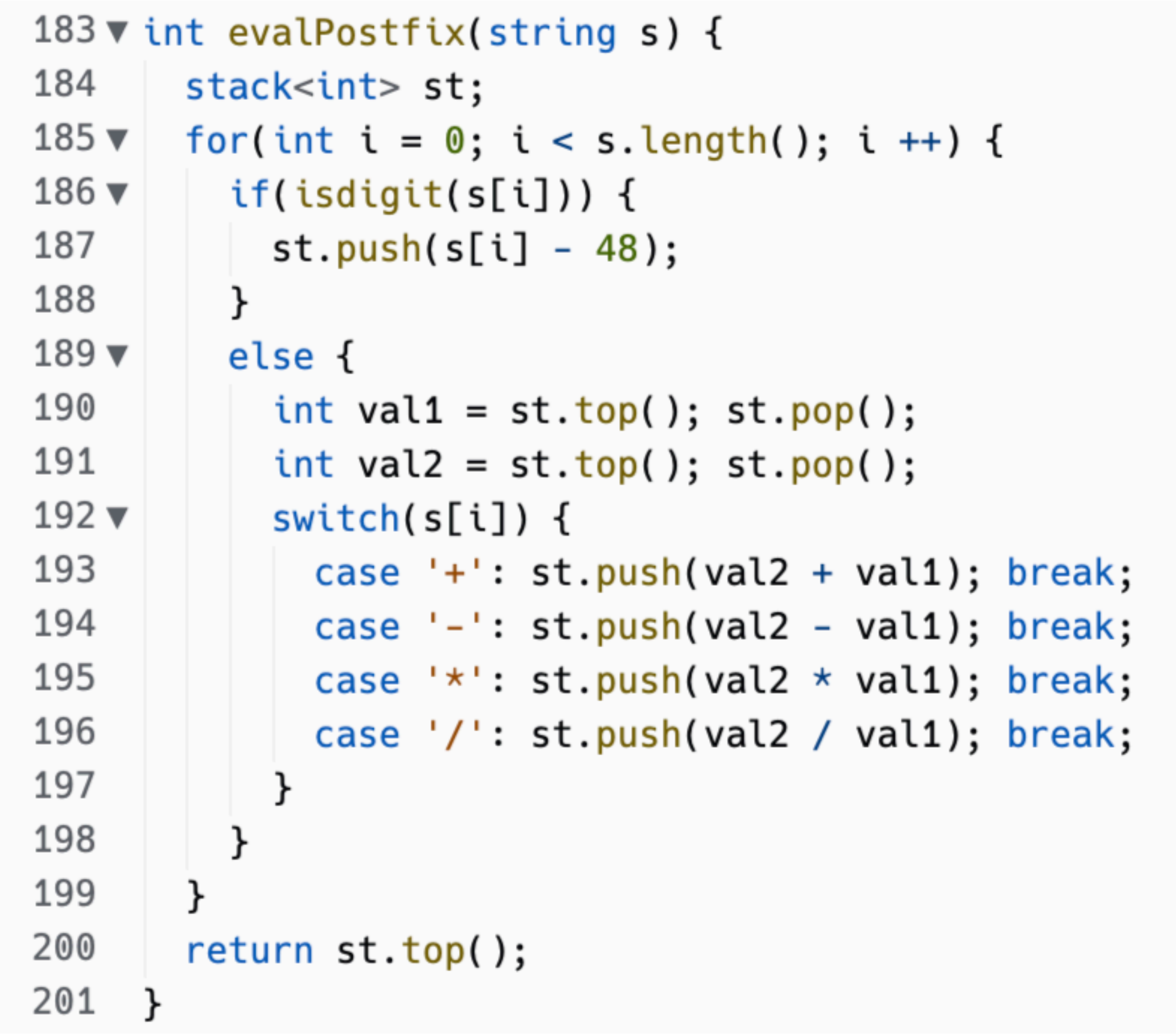
```
› ./main
stack is empty
size of stack: 10
10 9 8 7 6 5 4 3 2 1
› 
```

Infix to Postfix



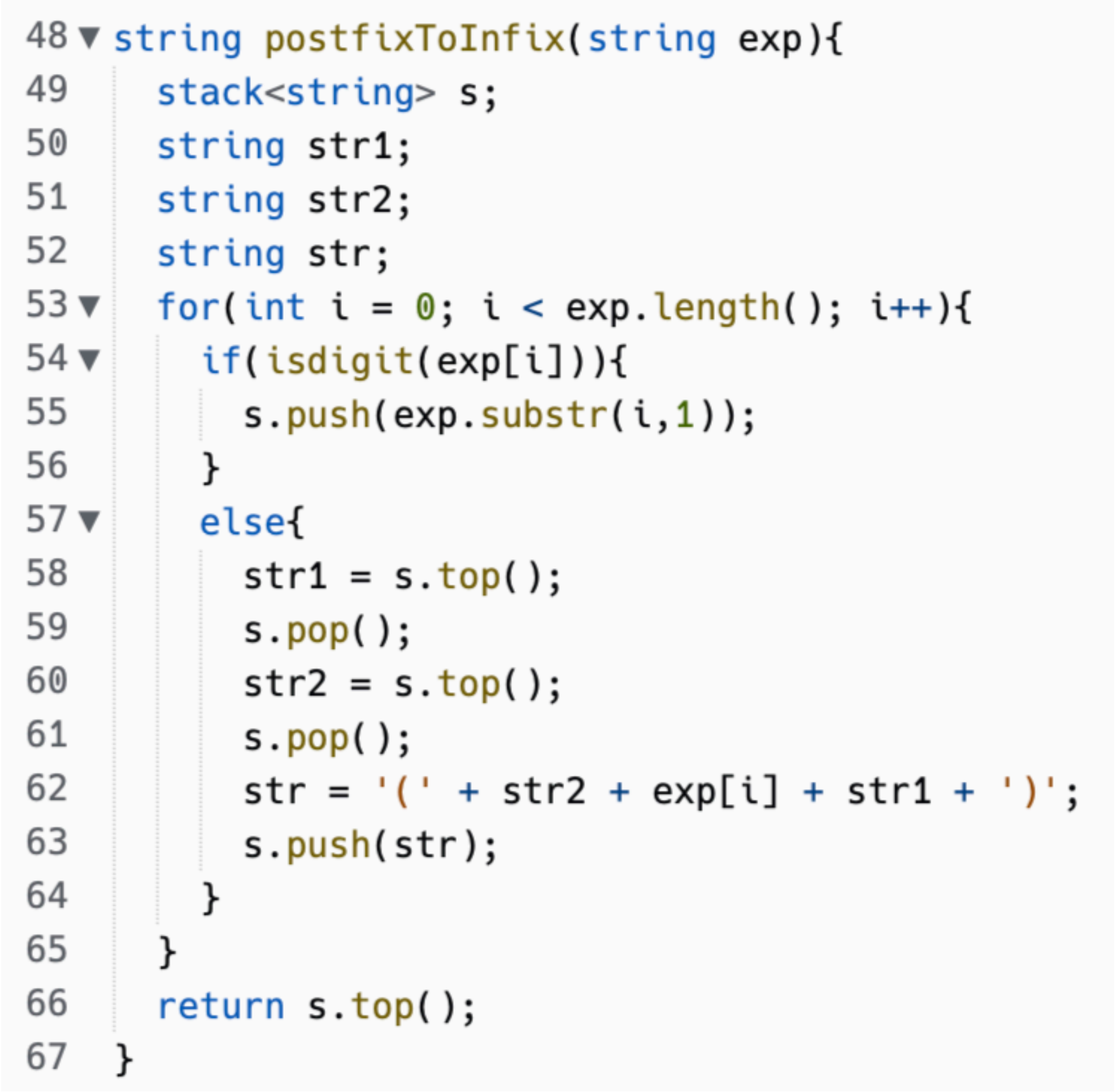
```cpp
string infixToPostfix(string exp, int size){
  stack<char> s;
  string str;
  for(int i = 0; i < size; i++){
    if(isdigit(exp[i])){
      str += exp[i];
    }
    else if(exp[i] == '('){
      s.push('(');
    }
    else if(exp[i] == ')'){
      while(s.top() != '('){
        str += s.top();
        s.pop();
      }
      s.pop();
    }
    else{
      while(!s.empty() && priority(exp[i]) <= priority(s.top())){
        str += s.top();
        s.pop();
      }
      s.push(exp[i]);
    }
  }
  while(!s.empty()){
    str += s.top();
    s.pop();
  }
  return str;
}
```

Evaluate Postfix



```cpp
int evalPostfix(string s) {
  stack<int> st;
  for(int i = 0; i < s.length(); i ++) {
    if(isdigit(s[i])) {
      st.push(s[i] - 48);
    }
    else {
      int val1 = st.top(); st.pop();
      int val2 = st.top(); st.pop();
      switch(s[i]) {
        case '+': st.push(val2 + val1); break;
        case '-': st.push(val2 - val1); break;
        case '*': st.push(val2 * val1); break;
        case '/': st.push(val2 / val1); break;
      }
    }
  }
  return st.top();
}
```

Postfix to Infix



```
string postfixToInfix(string exp){
  stack<string> s;
  string str1;
  string str2;
  string str;
  for(int i = 0; i < exp.length(); i++){
    if(isdigit(exp[i])){
      s.push(exp.substr(i,1));
    }
    else{
      str1 = s.top();
      s.pop();
      str2 = s.top();
      s.pop();
      str = '(' + str2 + exp[i] + str1 + ')';
      s.push(str);
    }
  }
  return s.top();
}
```

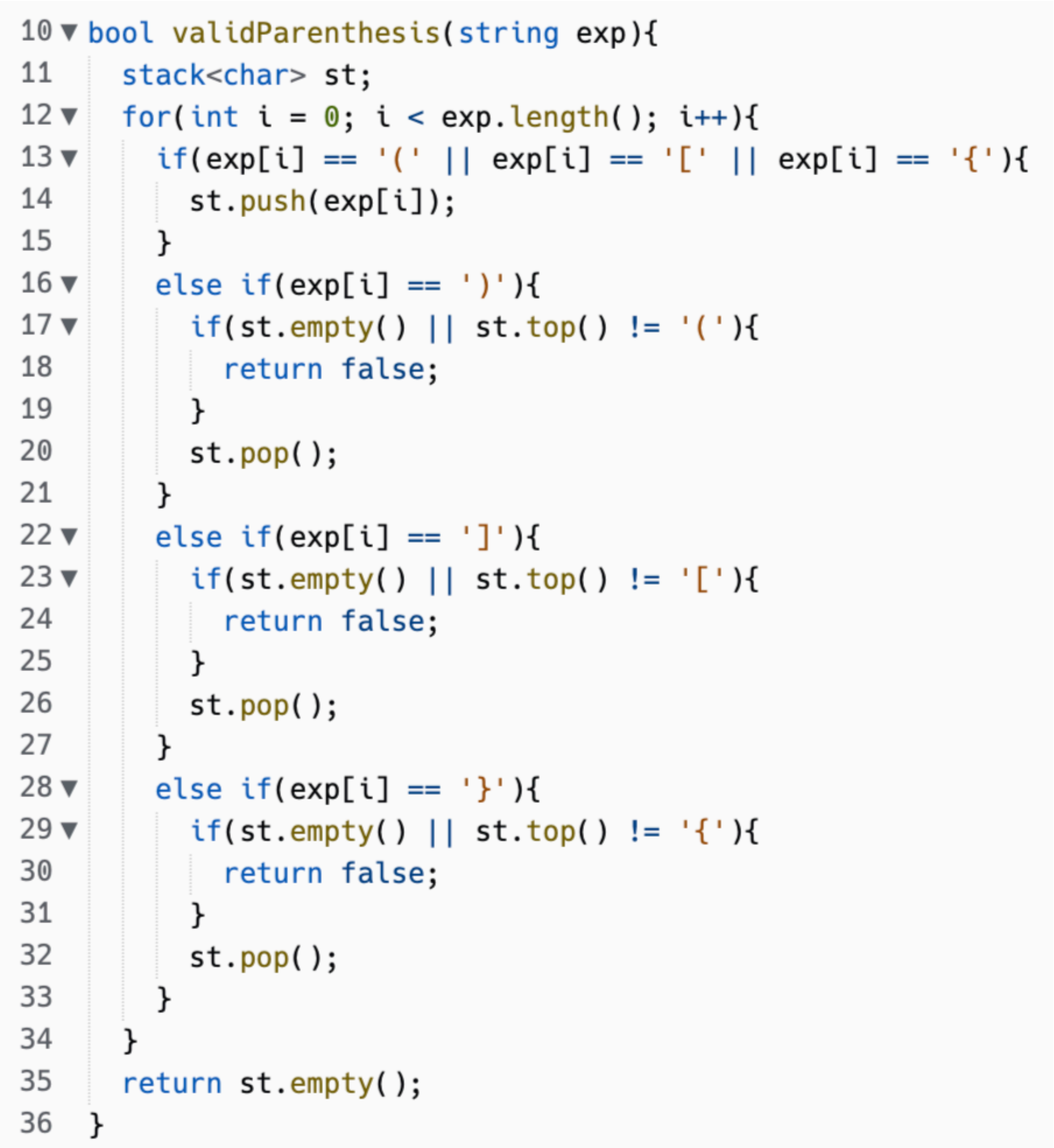
Valid Parenthesis

Given an expression, find out whether or not that expression contains valid parenthesis. An expression contains valid parenthesis if it has the proper parenthesis ( ), [ ], { } in the correct order.

validParenthesis("{ [ ] (( )) }") => true

validParenthesis("{ ( ]) } }") => false

Valid Parenthesis



```cpp
bool validParenthesis(string exp){
  stack<char> st;
  for(int i = 0; i < exp.length(); i++){
    if(exp[i] == '(' || exp[i] == '[' || exp[i] == '{'){
      st.push(exp[i]);
    }
    else if(exp[i] == ')'){
      if(st.empty() || st.top() != '('){
        return false;
      }
      st.pop();
    }
    else if(exp[i] == ']'){
      if(st.empty() || st.top() != '['){
        return false;
      }
      st.pop();
    }
    else if(exp[i] == '}'){
      if(st.empty() || st.top() != '{'){
        return false;
      }
      st.pop();
    }
  }
  return st.empty();
}
```

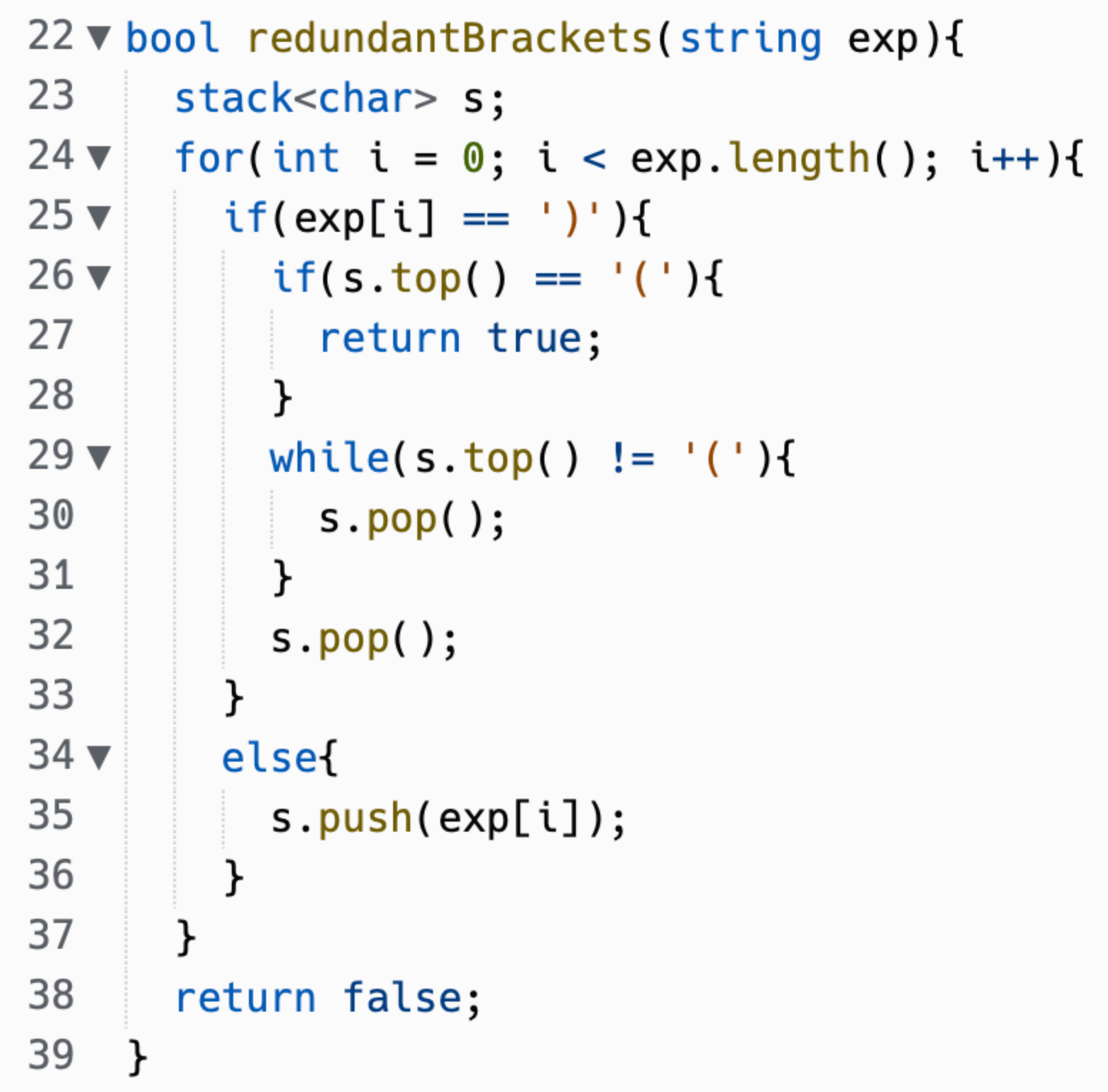
Redundant Brackets

Given an expression, find out whether or not that expression contains redundant brackets. The function should return **true** if the expression contains redundant brackets and **false** otherwise.

redundantBrackets(“((a+b))”) => true

redundantBrackets(“a*(b+c)”) => false

Redundant Brackets



```
bool redundantBrackets(string exp){
  stack<char> s;
  for(int i = 0; i < exp.length(); i++){
    if(exp[i] == ')'){
      if(s.top() == '('){
        return true;
      }
      while(s.top() != '('){
        s.pop();
      }
      s.pop();
    }
    else{
      s.push(exp[i]);
    }
  }
  return false;
}
```