Name: Ghulam Mustafa

Roll No: Fa-2022/BSCS/188

Section: E

ASSIGNMEMENT
DATA STRUCTURES & ALGORITHMS

Problem 1: Swap elements in two stacks using STL.

Pseudocode:	**Algorithm:**	**Code:**

Function swapStacks(stack1, stack2)
 	
   Create an empty stack tempStack
 	
   While stack1 is not empty
 	
      Push and Pop stack1.top() to/from tempStack
 	
   While stack2 is not empty
 	
     Push and Pop stack2.top()
-   to/from stack1
 	
   While tempStack is not empty
 	
      Push and Pop tempStack.top() to/from stack2
 	
End Function
 	
Create two empty stacks: Stack1 and Stack2
 	
Push elements into stacks
 	
Call swapStacks(Stack1, Stack2)
 	
Print Stack 1 , Stack 2 Using Loops
	1. Create an empty stack named tempStack.
	2. While stack1 is not empty, do the following:
	3. Push and Pop stack1.top() to/from tempStack.
	4. While stack2 is not empty, do the following:
	5. Push and Pop stack2.top() to/from stack1.
	6. While tempStack is not empty, do the following:
	7. Push and Pop tempStack.top() to/from stack2.
	8. Create two empty stacks: Stack1 and Stack2.
	9. Push elements 10, 20, 30 into Stack1.
	10. Push elements 40, 50, 60 into Stack2.
	11. Call swapStacks(Stack1, Stack2).
	12. Print "Stack1 after swap:".
	13. While Stack1 is not empty, do the following:
	14. Print and Pop Stack1.top().
	15. Print "Stack2 after swap:".
	16. While Stack2 is not empty, do Print and Pop Stack2.top().
		`#include <iostream>`
		`#include <stack>`
		`using namespace std;`
		`void swapStacks(stack<int> stack1, stack<int>& stack2) {`
		`   stack<int> tempStack;`
		`   while (!stack1.empty()) {`
		`      tempStack.push(stack1.top());`
		`      stack1.pop();`
		`   }`
		`   while (!stack2.empty()) {`
		`      stack1.push(stack2.top());`
		`      stack2.pop();`
		`   }`
		`   while (!tempStack.empty()) {`
		`      stack2.push(tempStack.top());`
		`      tempStack.pop();`
		`   }`
		`}`
		`int main() {`
		`   stack<int> Stack1, Stack2;`
		`   Stack1.push(10);`
		`   Stack1.push(20);`
		`   Stack1.push(30);`
		`   Stack2.push(40);`
		`   Stack2.push(50);`
		`   Stack2.push(60);`
		`   swapStacks(Stack1, Stack2);`
		
		`   cout << "Stack1 after swap: ";`
		`   while (!Stack1.empty()) {`
		`      cout << Stack1.top() << " ";`
		`      Stack1.pop();`
		`   }`
		`   cout << "\nStack2 after swap: ";`
		`   while (!Stack2.empty()) {`
		`      cout << Stack2.top() << " ";`
		`      Stack2.pop();`
		`   }`
		`   return 0;`
		`}`

Problem 2: **Check if the statement has valid brackets.**

Pseudocode:

```
Function isValidStatement(statement):

  Create an empty stack called 'brackets'For each character c in statement:

    If c is an opening bracket ('[', '{', or '('):

      Push c onto the 'brackets' stack

    Else if c is a closing bracket (']', '}', or ')'):

      If 'brackets' is empty:

        Return false (unmatched closing bracket)

      Pop the top element from 'brackets' and store it in top

      If c does not match the corresponding opening bracket for top:

        Return false (mismatched brackets)

  If 'brackets' is empty:

    Return true (valid statement)

  Else:

    Return false (unmatched opening bracket)

Input: A string statement containing brackets

Output: True if statement has properly matched brackets, False otherwise

  Initialize a string statement with the input string, e.g., "[{()}]"

  Call isValidStatement(statement)

If the result is true:
    Print "Valid statement"
  Else:
    Print "Invalid statement"
```

Algorithm:

1. Create an empty stack.
2. Iterate through the characters of the given statement.
3. If the current character is an opening bracket ([, {, or (), push it onto the stack.
4. If the current character is a closing bracket (], }, or)), pop the top element from the stack.
5. If the popped bracket does not match the current character, the statement is invalid.
6. After processing all characters, if the stack is empty, the statement is valid.

Code:

```
#include <iostream>
#include <stack>
#include <>
using namespace std;
bool isValidStatement(string statement)
{
  stack<char> brackets;
  for (int i = 0; i < statement.length(); i++) {
    char c = statement[i];
    if (c == '[' || c == '{' || c == '(') {
      brackets.push(c);
    } else if (c == ']' || c == '}' || c == ')')
{
      if (brackets.empty()) {
        return false;
      }
      char top = brackets.top();
      brackets.pop();
      if ((c == ']' && top != '[') || (c == '}' && top != '{') || (c == ')' && top != '(')) {
        return false;
      }
    }
  }
  return brackets.empty();
}

int main() {
  string statement = "[{()}]";
  if (isValidStatement(statement)) {
    cout << "Valid statement" << endl;
  } else {
    cout << "Invalid statement" << endl;
  }

  return 0;
}
```

<u>Problem 3:</u> Reverse the string using a stack.

Pseudocode:

```
Function reverseString(input: String) -> String
    Create an empty stack of characters called charStack

    For each character c in input
        Push c onto charStack

    Create an empty string called reversedString

    While charStack is not empty
        Append the top character from charStack to reversedString
        Pop the top character from charStack

    Return reversedString
End Function

Function main()
    Set input as "Pakistan"
    Call reverseString(input) and store the result in reversed
    Print "Reversed string: " followed by reversed
End Function

Call main()
```

Algorithm:

1. Create an empty stack.
2. Iterate through the characters of the input string.
3. Push each character onto the stack.
4. Pop characters from the stack and append them to a new string to reverse the original string.
5. Print the reversed string.

Code:

```cpp
#include <iostream>
#include <stack>

using namespace std;

string reverseString(string input) {
   stack<char> charStack;

  for (int i = 0; i < input.length(); i++){
      charStack.push(input[i]);
   }

   string reversedString;

   while (!charStack.empty()) {
      reversedString += charStack.top();
      charStack.pop();
   }

   return reversedString;
}

int main() {
   string input = "Pakistan";
   string reversed = reverseString(input);
   cout << "Reversed string: " << reversed << endl;

   return 0;
}
```