**TASK:**

**Create a simple C++ class named Stack that stores integers. It should provide the fundamental stack**

**operations: Push, Pop, Peek, IsEmpty, and Size. Consider how your implementation will handle**

**potential edge cases.**

**1. Design and implement a comprehensive set of test cases to verify the correctness and**

**robustness of all the stack operations. Consider various input scenarios and edge cases.**

**Explore different assertion types provided by Google Test to effectively validate the behavior**

**of your stack.**

**2. Consider how features like test fixtures and parameterized testing could enhance the**

**organization and efficiency of your test suite. Implement these features where you deem them**

**beneficial.**

**In your lab report, document your testing strategy. Explain the different types of test cases you**

**designed and why they are important. Describe how you handled any challenges or design decisions**

**you made during the testing process. If you used test fixtures or parameterized testing, explain your**

**reasoning and how they improved your testing.**

**Stack.h:**

**#pragma once**

**#include <vector>**

**#include <stdexcept>**

**class Stack {**

**private:**

**std::vector<int> data;**

**public:**

**void Push(int value);**

**int Pop();**

**int Peek() const;**

**bool IsEmpty() const;**

**int Size() const;**

**};**

**TEST.CPP:**

**#include <gtest/gtest.h>**

**#include "stack.cpp" // include implementation**

**class StackTest : public ::testing::Test {**

**protected:**

**Stack s;**

**};**

**TEST\_F(StackTest, PushAndPeek) {**

**s.Push(10);**

**EXPECT\_EQ(s.Peek(), 10);**

**EXPECT\_EQ(s.Size(), 1);**

**}**

**TEST\_F(StackTest, PopRemovesTopElement) {**

**s.Push(5);**

**s.Push(15);**

**EXPECT\_EQ(s.Pop(), 15);**

**EXPECT\_EQ(s.Peek(), 5);**

**EXPECT\_EQ(s.Size(), 1);**

**}**

**TEST\_F(StackTest, IsEmptyInitially) {**

**EXPECT\_TRUE(s.IsEmpty());**

**EXPECT\_EQ(s.Size(), 0);**

**}**

**TEST\_F(StackTest, PopThrowsOnEmpty) {**

**EXPECT\_THROW(s.Pop(), std::out\_of\_range);**

**}**

**TEST\_F(StackTest, PeekThrowsOnEmpty) {**

**EXPECT\_THROW(s.Peek(), std::out\_of\_range);**

**}**

**class StackParamTest : public ::testing::TestWithParam<std::vector<int>> {};**

**TEST\_P(StackParamTest, PushMultipleElements) {**

**Stack s;**

**std::vector<int> values = GetParam();**

**for (int v : values) s.Push(v);**

**EXPECT\_EQ(s.Size(), values.size());**

**if (!values.empty())**

**EXPECT\_EQ(s.Peek(), values.back());**

**}**

**INSTANTIATE\_TEST\_SUITE\_P(**

**MultipleInputTests,**

**StackParamTest,**

**::testing::Values(**

**std::vector<int>{},**

**std::vector<int>{1},**

**std::vector<int>{3, 6, 9},**

**std::vector<int>{-1, -5, 0, 20}**

**)**

**);**

**stack.cpp:**

**#include "stack.h"**

**void Stack::Push(int value) {**

**data.push\_back(value);**

**}**

**int Stack::Pop() {**

**if (IsEmpty()) throw std::out\_of\_range("Stack is empty");**

**int val = data.back();**

**data.pop\_back();**

**return val;**

**}**

**int Stack::Peek() const {**

**if (IsEmpty()) throw std::out\_of\_range("Stack is empty");**

**return data.back();**

**}**

**bool Stack::IsEmpty() const {**

**return data.empty();**

**}**

**int Stack::Size() const {**

**return data.size();**

**}**

**stack.cpp**

**#include "stack.h"**

**void Stack::Push(int value) {**

**data.push\_back(value);**

**}**

**int Stack::Pop() {**

**if (IsEmpty()) throw std::out\_of\_range("Stack is empty");**

**int val = data.back();**

**data.pop\_back();**

**return val;**

**}**

**int Stack::Peek() const {**

**if (IsEmpty()) throw std::out\_of\_range("Stack is empty");**

**return data.back();**

**}**

**bool Stack::IsEmpty() const {**

**return data.empty();**

**}**

**int Stack::Size() const {**

**return data.size();**

**}**

