# Experiment 4.2

| | |
|---|---|
| **Student Name:** Pranjal Kumar | **UID:** 20BCS3504 |
| **Branch:** CSE | **Section/Group:** 607 B |
| **Semester:** 5th | **Date of Performance:** 07/09/2022 |
| **Subject Name:** DAA Lab | **Subject Code:** 20CSP-312 |

## 1. Aim/Overview of the practical:

Code to push & pop and check Isempty, Isfull,and Return top element in stacks using templates

## 2.Task to be done/ Which logistics used:

Templates in C++ : A template is a simple and yet very powerful tool in C++. The simple idea is to pass data type as a parameter so that we don't need to write the same code for different data types. For example, a software company may need sort() for different data types. Rather than writing and maintaining the multiple codes, we can write one sort() and pass data type as a parameter. C++ adds two new keywords to support templates: 'template' and 'typename'. The second keyword can always be replaced by keyword 'class'.

How templates work?
Templates are expanded at compiler time. This is like macros. The difference is, compiler does type checking before template expansion. The idea is simple, source code contains only function/class, but compiled code may contain multiple copies of same function/class.

## 3. Operation Perform:-

Basic Operations Stack operations may involve initializing the stack, using it and then de-initializing it. Apart from these basic stuffs, a stack is used for the following two primary operations −

- **push()** − Pushing (storing) an element on the stack.
- **pop()** − Removing (accessing) an element from the stack.

When data is PUSHed onto stack. To use a stack efficiently, we need to check the status of stack as well. For the same purpose, the following functionality is added to stacks −

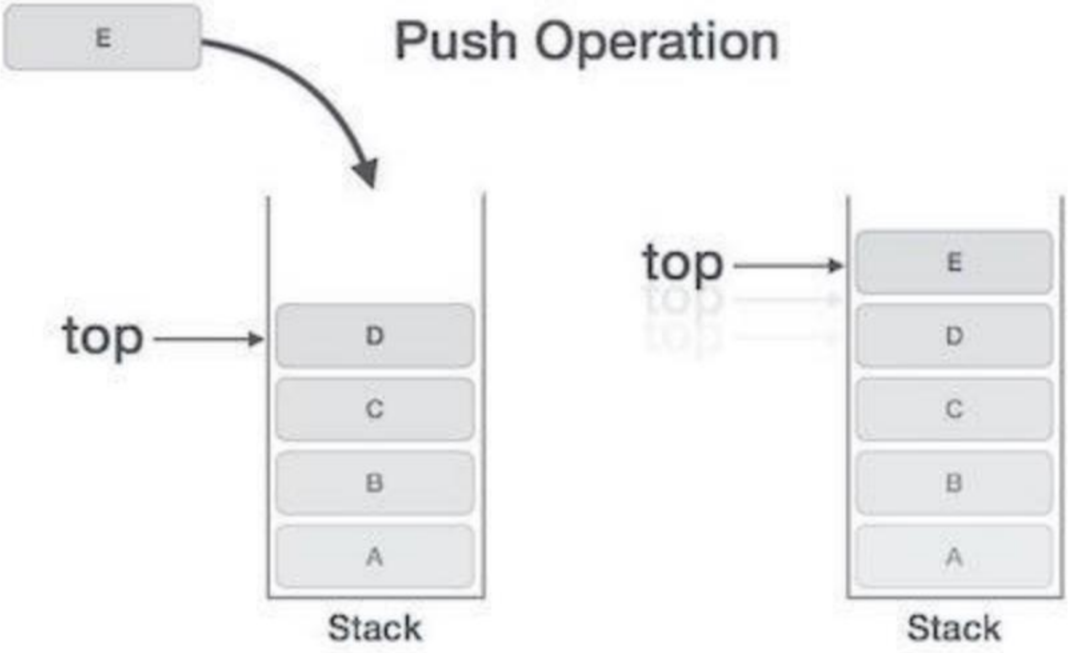
- **peek()** − get the top data element of the stack, without removing it.
- **isFull()** − check if stack is full.
- **isEmpty()** − check if stack is empty.

At all times, we maintain a pointer to the last PUSHed data on the stack. As this pointer always represents the top of the stack, hence named top. The top pointer provides top value of the stack without actually removing it.

## 4. Algorithm/Flowchart (For programming based labs):

**Push Operation:**

- Step 1 – Checks if the stack is full.
- Step 2 – If the stack is full, produces an error and exit.
- Step 3 – If the stack is not full, increments top to point next empty space.
- Step 4 – Adds data element to the stack location, where top is pointing.
- Step 5 – Returns success.



## Algorithm for PUSH Operation

```
begin procedure push: stack, data

  if stack is full

    return null

  endif

  top ← top + 1

  stack[top] ← data

end procedure
```

## Pop Operation

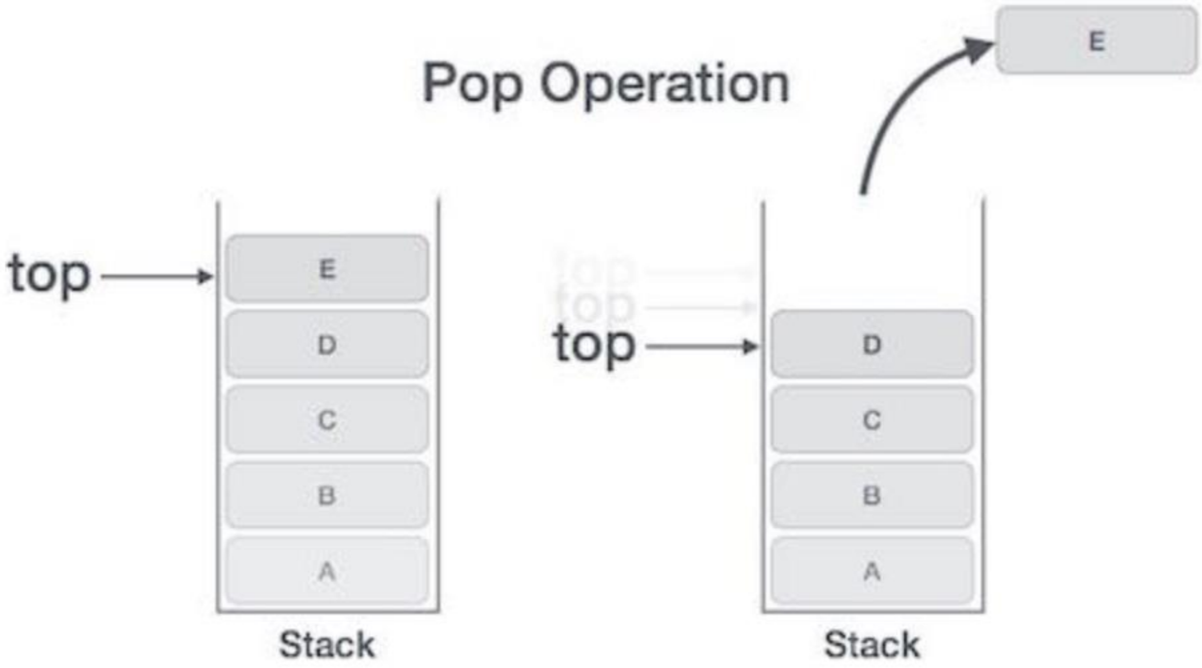
**Step 1** – Checks if the stack is empty.

**Step 2** – If the stack is empty, produces an error and exit.

**Step 3** – If the stack is not empty, accesses the data element at which **top** is pointing.

**Step 4** – Decreases the value of top by 1.

**Step 5** – Returns success.



## Algorithm for Pop Operation

```
begin procedure pop: stack

  if stack is empty

    return null

  endif

  data ← stack[top]

  top ← top - 1

  return data

end procedure
```

## peek()

Algorithm of peek() function –

```
begin procedure peek
   return stack[top]
end procedure
```

## Isfull()

Algorithm of isfull() function –

```
begin procedure isfull

   if top equals to MAXSIZE
      return true
   else
      return false
   endif
end procedure
```

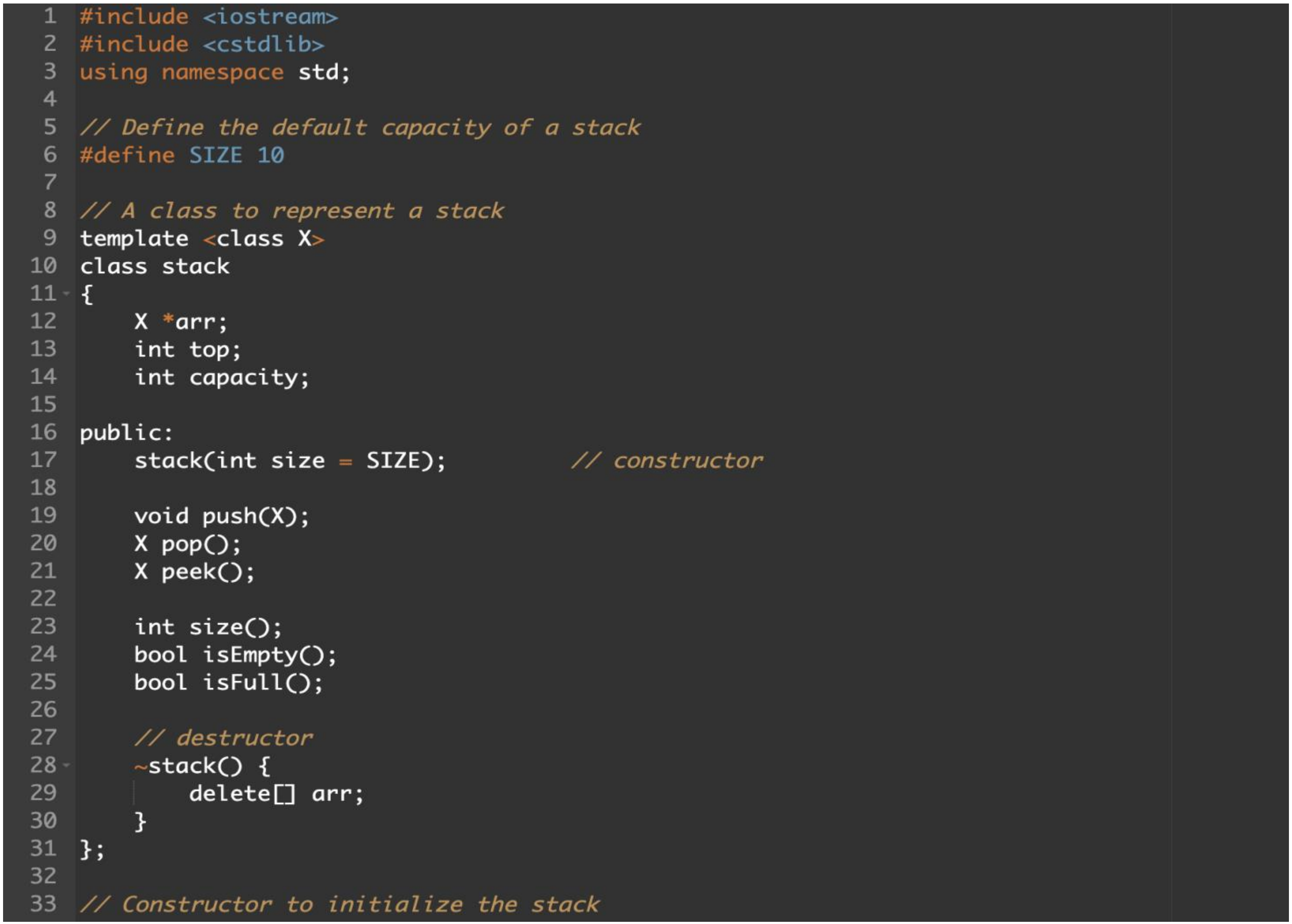
## isempty()

Algorithm of isempty() function –

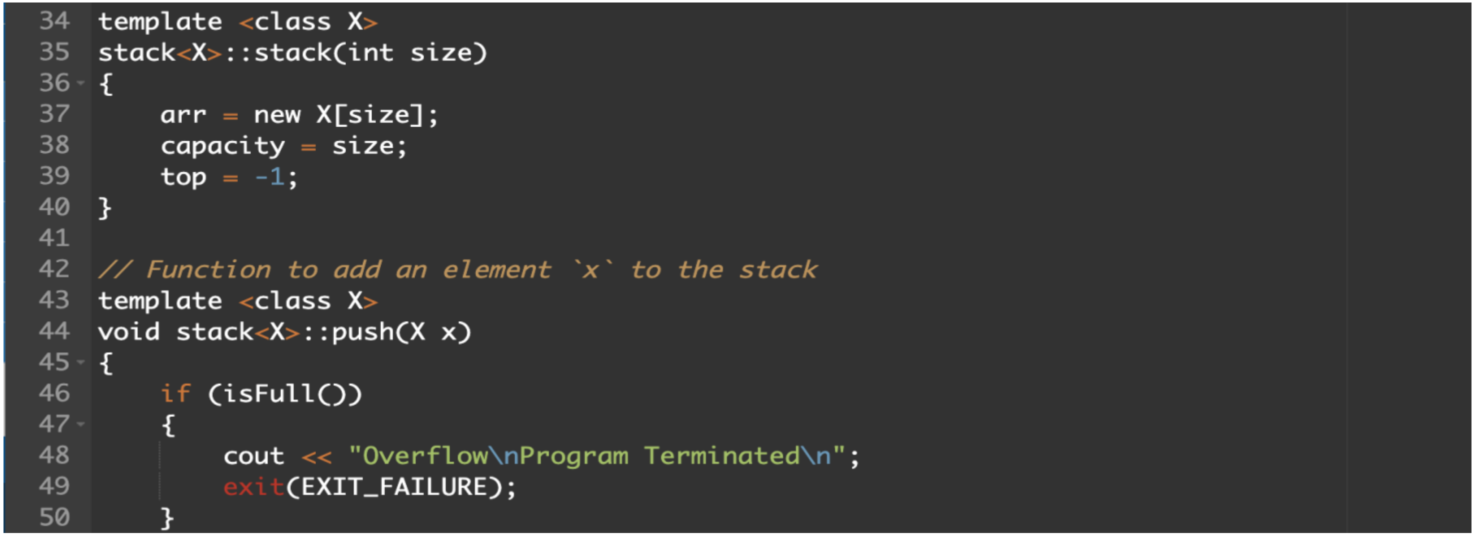
```
begin procedure isempty
   if top less than 1
      return true
   else
      return false
   endif
end procedure
```

## 5. Steps for experiment/practical/Code:-

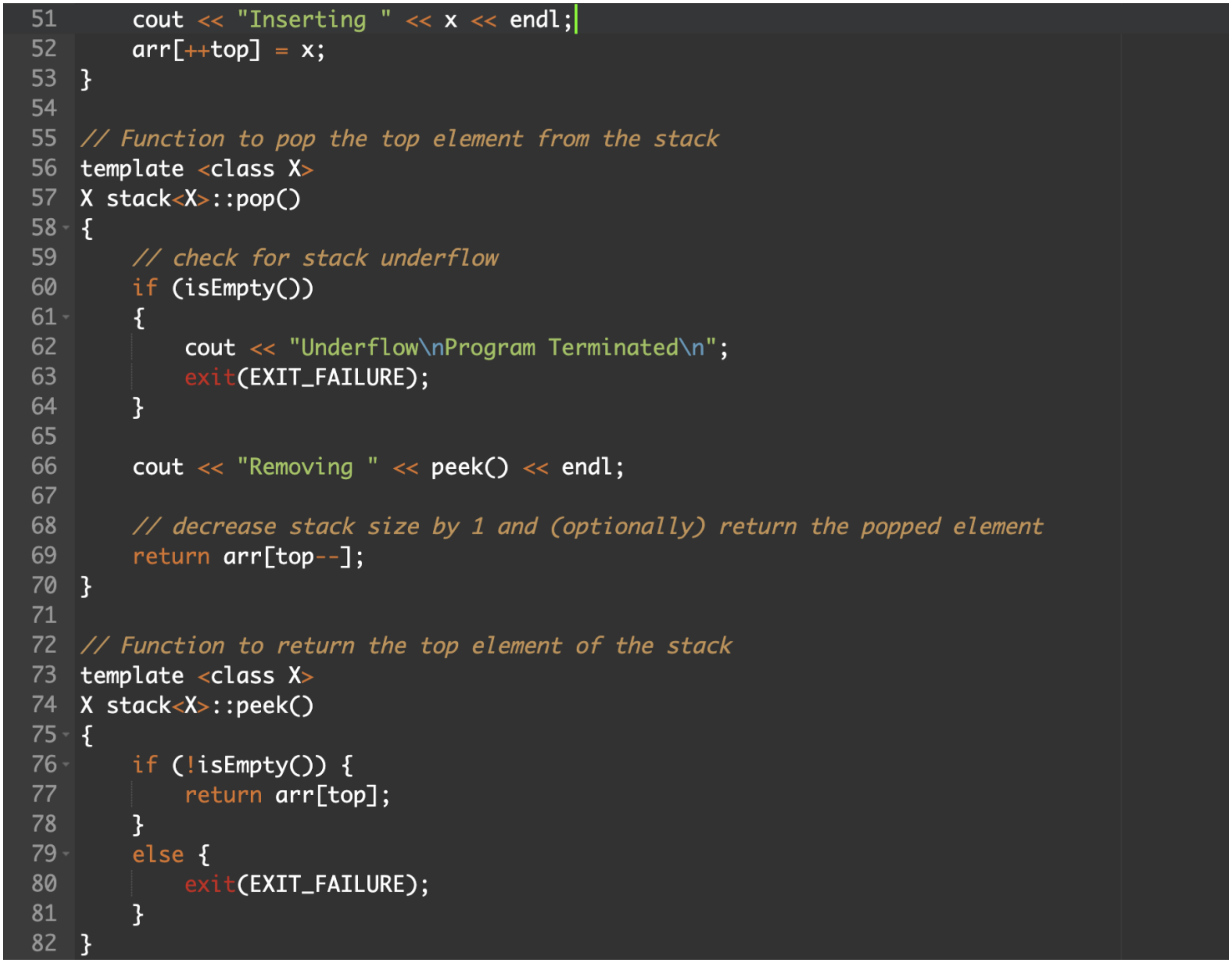
```
#include <iostream>
#include <cstdlib>
using namespace std;

// Define the default capacity of a stack
#define SIZE 10

// A class to represent a stack
template <class X>
class stack
{
    X *arr;
    int top;
    int capacity;

public:
    stack(int size = SIZE);         // constructor

    void push(X);
    X pop();
    X peek();

    int size();
    bool isEmpty();
    bool isFull();

    // destructor
    ~stack() {
        delete[] arr;
    }
};

// Constructor to initialize the stack
template <class X>
stack<X>::stack(int size)
{
    arr = new X[size];
    capacity = size;
    top = -1;
}

// Function to add an element `x` to the stack
template <class X>
void stack<X>::push(X x)
{
    if (isFull())
    {
        cout << "Overflow\nProgram Terminated\n";
        exit(EXIT_FAILURE);
    }

    cout << "Inserting " << x << endl;
    arr[++top] = x;
}

// Function to pop the top element from the stack
template <class X>
X stack<X>::pop()
{
    // check for stack underflow
    if (isEmpty())
    {
        cout << "Underflow\nProgram Terminated\n";
        exit(EXIT_FAILURE);
    }

    cout << "Removing " << peek() << endl;

    // decrease stack size by 1 and (optionally) return the popped element
    return arr[top--];
}

// Function to return the top element of the stack
template <class X>
X stack<X>::peek()
{
    if (!isEmpty()) {
        return arr[top];
    }
    else {
        exit(EXIT_FAILURE);
    }
}

// Utility function to return the size of the stack
template <class X>
int stack<X>::size() {
    return top + 1;
}

// Utility function to check if the stack is empty or not
template <class X>
bool stack<X>::isEmpty() {
    return top == -1;               // or return size() == 0;
}

// Utility function to check if the stack is full or not
template <class X>
bool stack<X>::isFull() {
    return top == capacity - 1;     // or return size() == capacity;
}

int main()
{
    stack<string> pt(2);

    pt.push("A");
    pt.push("B");

    pt.pop();
    pt.pop();

    pt.push("C");

    // Prints the top of the stack
    cout << "The top element is " << pt.peek() << endl;

    // Returns the total number of elements present in the stack
    cout << "The stack size is " << pt.size() << endl;

    pt.pop();

    // check if the stack is empty or not
    if (pt.isEmpty()) {
        cout << "The stack is empty\n";
    }
    else {
        cout << "The stack is not empty\n";
    }

    return 0;
}
```



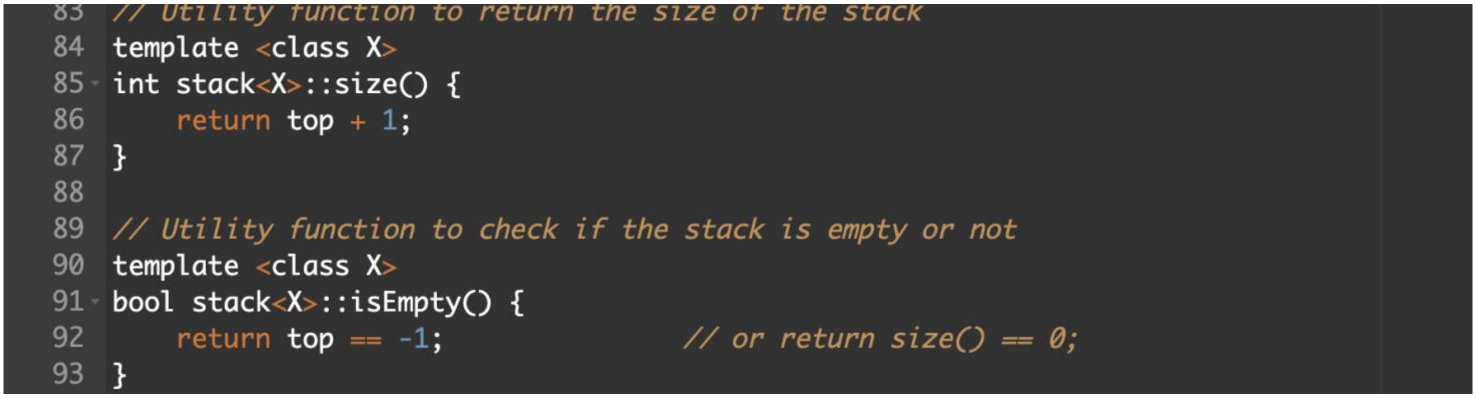
```
#include <iostream>
#include <cstdlib>
using namespace std;

// Define the default capacity of a stack
#define SIZE 10

// A class to represent a stack
template <class X>
class stack
{
    X *arr;
    int top;
    int capacity;

public:
    stack(int size = SIZE);         // constructor

    void push(X);
    X pop();
    X peek();

    int size();
    bool isEmpty();
    bool isFull();

    // destructor
    ~stack() {
        delete[] arr;
    }
};

// Constructor to initialize the stack
```



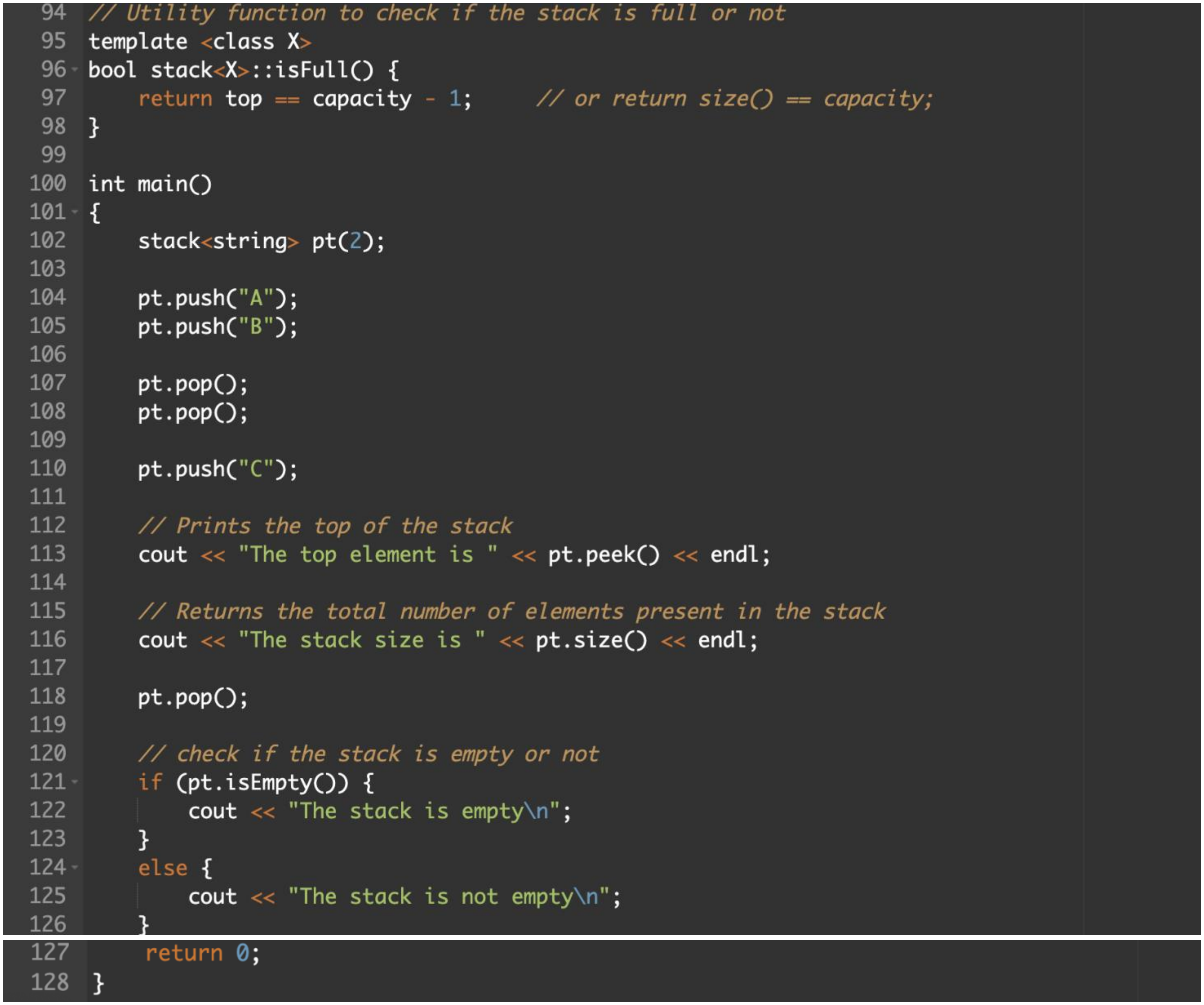
```
template <class X>
stack<X>::stack(int size)
{
    arr = new X[size];
    capacity = size;
    top = -1;
}

// Function to add an element `x` to the stack
template <class X>
void stack<X>::push(X x)
{
    if (isFull())
    {
        cout << "Overflow\nProgram Terminated\n";
        exit(EXIT_FAILURE);
    }
```



```
    cout << "Inserting " << x << endl;
    arr[++top] = x;
}

// Function to pop the top element from the stack
template <class X>
X stack<X>::pop()
{
    // check for stack underflow
    if (isEmpty())
    {
        cout << "Underflow\nProgram Terminated\n";
        exit(EXIT_FAILURE);
    }

    cout << "Removing " << peek() << endl;

    // decrease stack size by 1 and (optionally) return the popped element
    return arr[top--];
}

// Function to return the top element of the stack
template <class X>
X stack<X>::peek()
{
    if (!isEmpty()) {
        return arr[top];
    }
    else {
        exit(EXIT_FAILURE);
    }
}
```



```
// Utility function to return the size of the stack
template <class X>
int stack<X>::size() {
    return top + 1;
}

// Utility function to check if the stack is empty or not
template <class X>
bool stack<X>::isEmpty() {
    return top == -1;                // or return size() == 0;
}
```



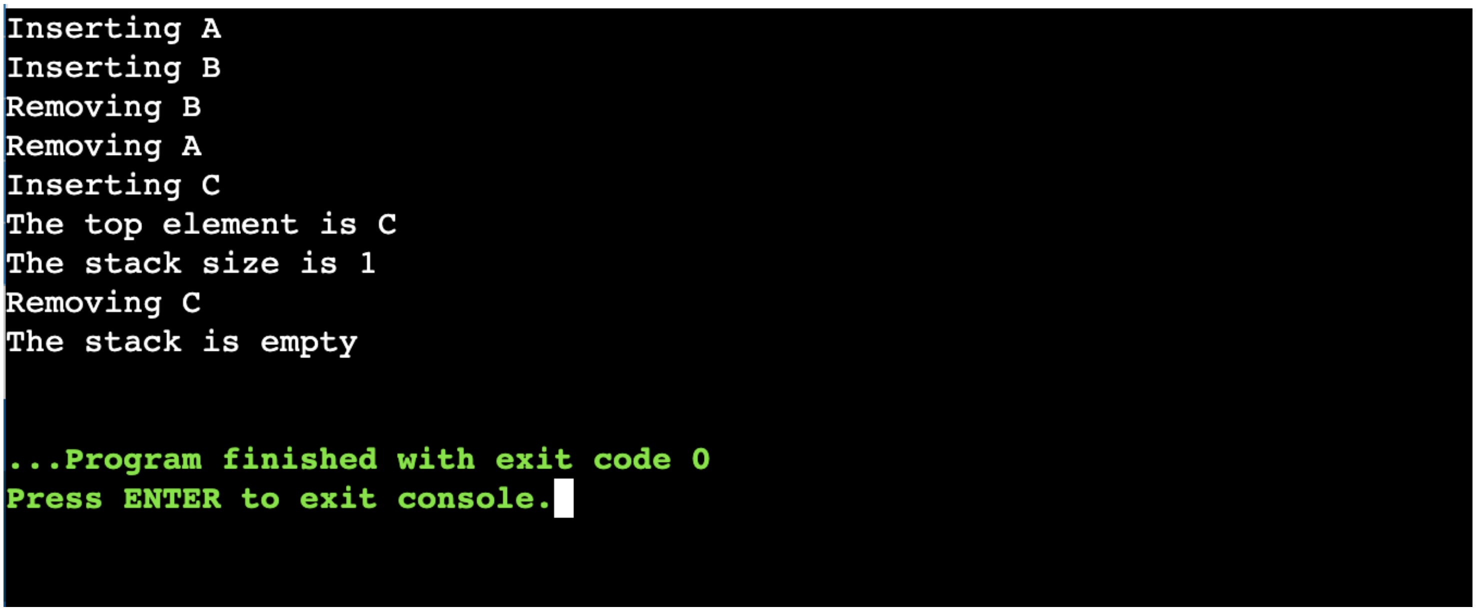
```
// Utility function to check if the stack is full or not
template <class X>
bool stack<X>::isFull() {
    return top == capacity - 1;     // or return size() == capacity;
}

int main()
{
    stack<string> pt(2);

    pt.push("A");
    pt.push("B");

    pt.pop();
    pt.pop();

    pt.push("C");

    // Prints the top of the stack
    cout << "The top element is " << pt.peek() << endl;

    // Returns the total number of elements present in the stack
    cout << "The stack size is " << pt.size() << endl;

    pt.pop();

    // check if the stack is empty or not
    if (pt.isEmpty()) {
        cout << "The stack is empty\n";
    }
    else {
        cout << "The stack is not empty\n";
    }
    return 0;
}
```

## 6. Observations/Discussions/ Complexity Analysis:

### Time Complexity

| Operations | Complexity |
|---|---|
| push() | O(1) |
| pop() | O(1) |
| isEmpty() | O(1) |
| size() | O(1) |

## 7. Result/Output/Writing Summary:-



```
Inserting A
Inserting B
Removing B
Removing A
Inserting C
The top element is C
The stack size is 1
Removing C
The stack is empty


...Program finished with exit code 0
Press ENTER to exit console.
```

## Learning outcomes (What I have learnt):

1. Stack.
2. operation.
3. Complexity .

**Evaluation Grid (To be created as per the SOP and Assessment guidelines by the faculty):**

| Sr. No. | Parameters | Marks Obtained | Maximum Marks |
|---|---|---|---|
| 1. | | | |
| 2. | | | |
| 3. | | | |
| | | | |