

Experiment 4.2

Student Name: Pranjal Kumar
Branch: CSE
Semester: 5th
Subject Name: DAA Lab

UID: 20BCS3504
Section/Group: 607 B
Date of Performance: 07/09/2022
Subject Code: 20CSP-312

1. Aim/Overview of the practical:

Code to push & pop and check Isempy, 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 –

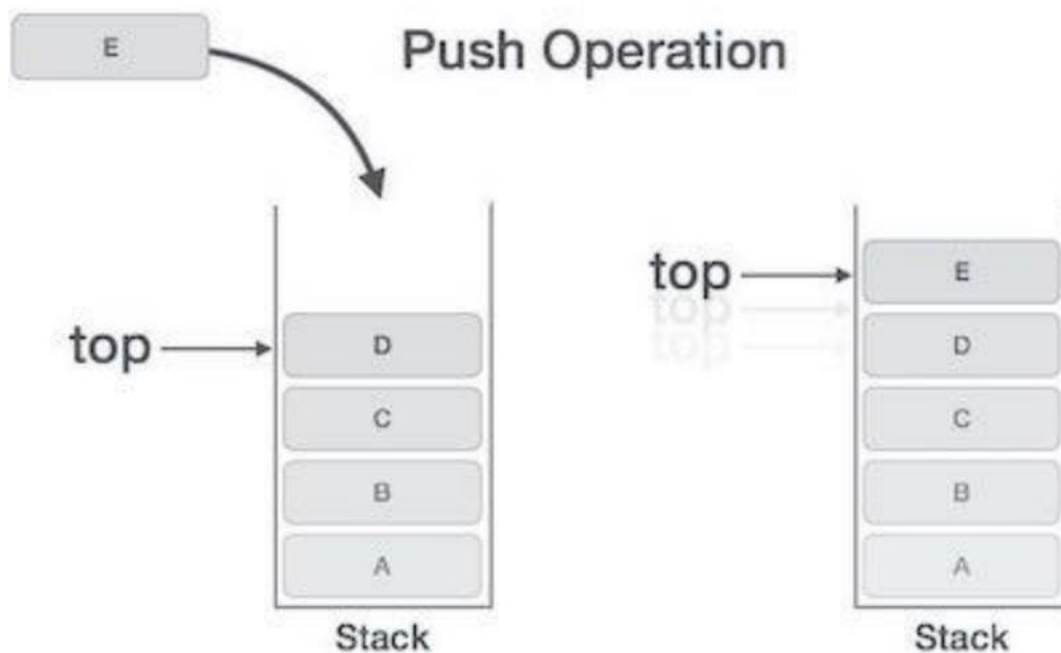
- **peek()** – get the top data element of the stack, without removing it.
- **isFull()** – check if stack is full.
- **isEmpty()** – check if stack is empty.

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

$\text{top} \leftarrow \text{top} + 1$

$\text{stack}[\text{top}] \leftarrow \text{data}$

end procedure

Pop Operation

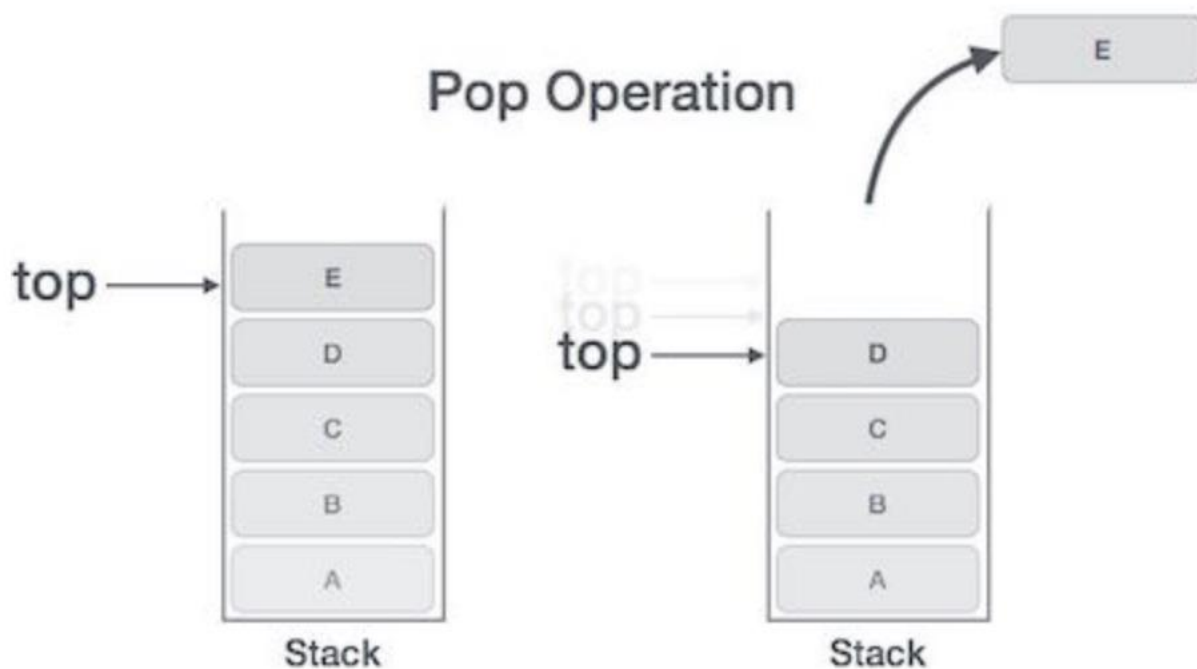
Step 1 – Checks if the stack is empty.

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 \leftarrow stack[top]

 top \leftarrow 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;
}
```



```
1  #include <iostream>
2  #include <cstdlib>
3  using namespace std;
4
5  // Define the default capacity of a stack
6  #define SIZE 10
7
8  // A class to represent a stack
9  template <class X>
10 class stack
11 {
12     X *arr;
13     int top;
14     int capacity;
15
16 public:
17     stack(int size = SIZE);           // constructor
18
19     void push(X);
20     X pop();
21     X peek();
22
23     int size();
24     bool isEmpty();
25     bool isFull();
26
27     // destructor
28     ~stack() {
29         delete[] arr;
30     }
31 };
32
33 // Constructor to initialize the stack
34 template <class X>
35 stack<X>::stack(int size)
36 {
37     arr = new X[size];
38     capacity = size;
39     top = -1;
40 }
41
42 // Function to add an element 'x' to the stack
43 template <class X>
44 void stack<X>::push(X x)
45 {
46     if (isFull())
47     {
48         cout << "Overflow\nProgram Terminated\n";
49         exit(EXIT_FAILURE);
50     }
```

```
51     cout << "Inserting " << x << endl;|
52     arr[++top] = x;
53 }
54
55 // Function to pop the top element from the stack
56 template <class X>
57 X stack<X>::pop()
58 {
59     // check for stack underflow
60     if (isEmpty())
61     {
62         cout << "Underflow\nProgram Terminated\n";
63         exit(EXIT_FAILURE);
64     }
65
66     cout << "Removing " << peek() << endl;
67
68     // decrease stack size by 1 and (optionally) return the popped element
69     return arr[top--];
70 }
71
72 // Function to return the top element of the stack
73 template <class X>
74 X stack<X>::peek()
75 {
76     if (!isEmpty()) {
77         return arr[top];
78     }
79     else {
80         exit(EXIT_FAILURE);
81     }
82 }
83 // Utility function to return the size of the stack
84 template <class X>
85 int stack<X>::size() {
86     return top + 1;
87 }
88
89 // Utility function to check if the stack is empty or not
90 template <class X>
91 bool stack<X>::isEmpty() {
92     return top == -1;           // or return size() == 0;
93 }
```

```
94 // Utility function to check if the stack is full or not
95 template <class X>
96 bool stack<X>::isFull() {
97     return top == capacity - 1;    // or return size() == capacity;
98 }
99
100 int main()
101 {
102     stack<string> pt(2);
103
104     pt.push("A");
105     pt.push("B");
106
107     pt.pop();
108     pt.pop();
109
110     pt.push("C");
111
112     // Prints the top of the stack
113     cout << "The top element is " << pt.peek() << endl;
114
115     // Returns the total number of elements present in the stack
116     cout << "The stack size is " << pt.size() << endl;
117
118     pt.pop();
119
120     // check if the stack is empty or not
121     if (pt.isEmpty()) {
122         cout << "The stack is empty\n";
123     }
124     else {
125         cout << "The stack is not empty\n";
126     }
127     return 0;
128 }
```

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.			