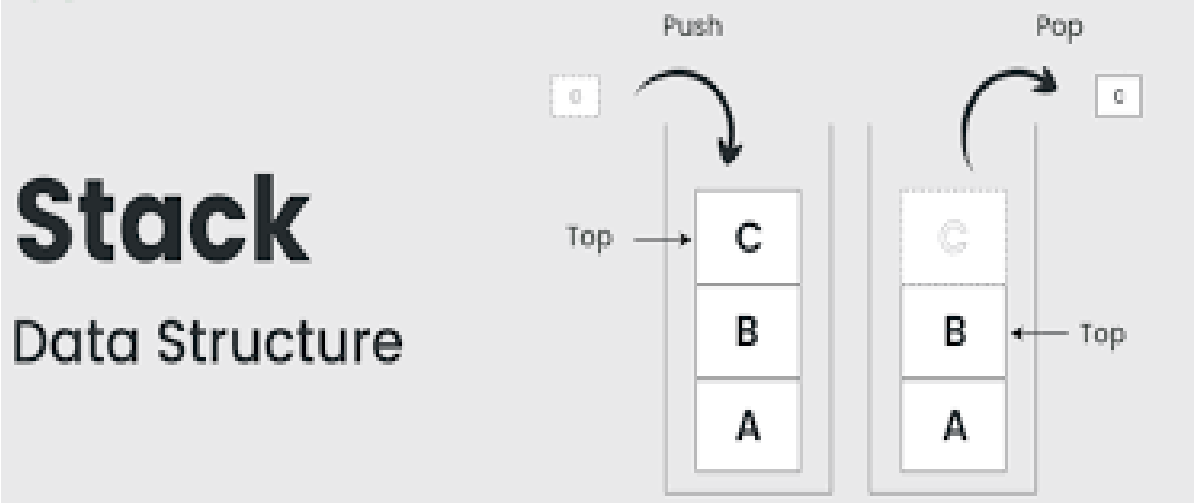
Cyber Security Engineering Department Lectures
Prepared by Assist. Prof. Imad Matti

هياكل البيانات

Stack Data Structure

Chapter 3



Prepared by Assist . Prof .
Imad Matti
AL-mamoon University College
Cyber Security Engineering Department

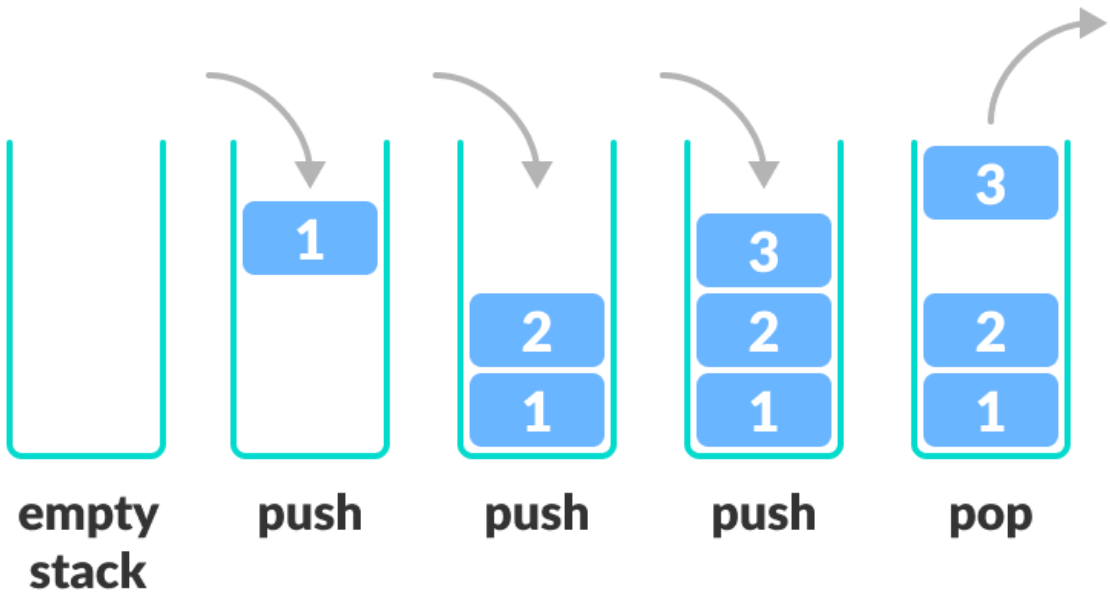
2024

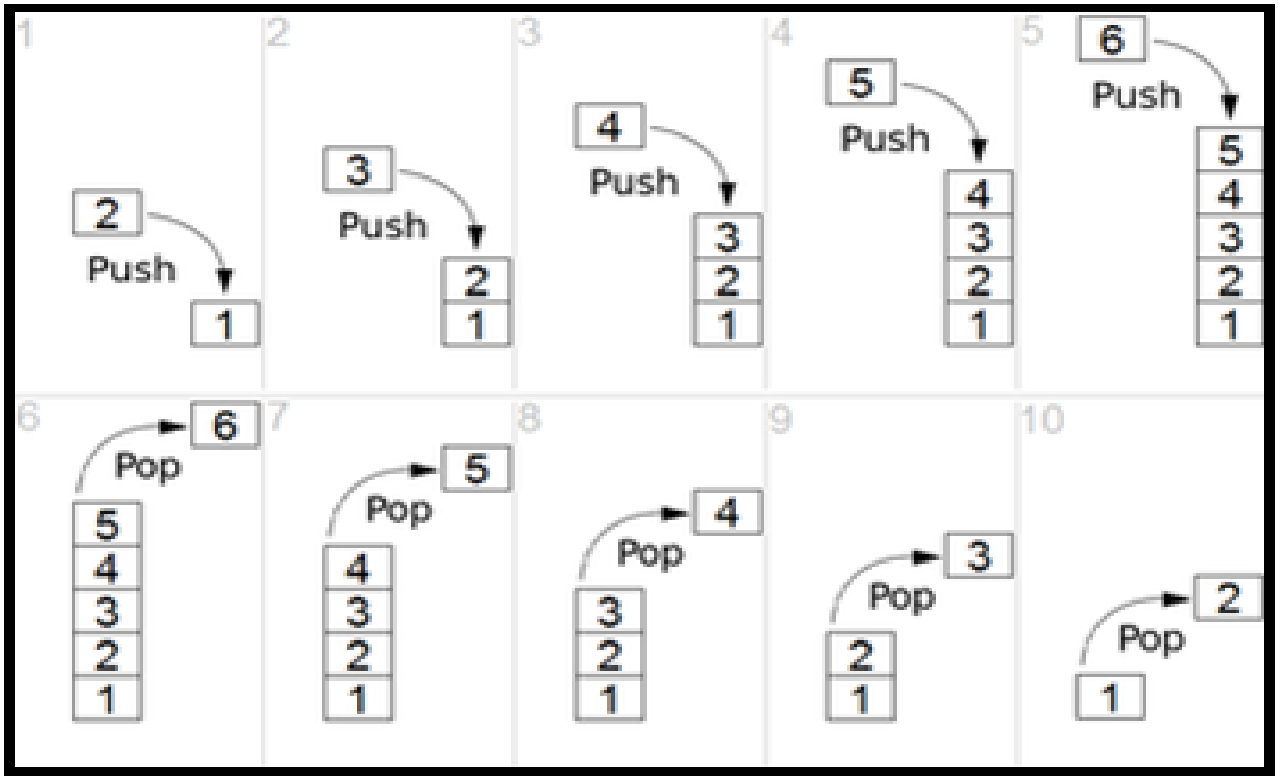
A **stack** is a linear data structure that follows the **Last In**, **First out** (**LIFO**) principle. This means that the last element added to the stack is the first one to be removed.

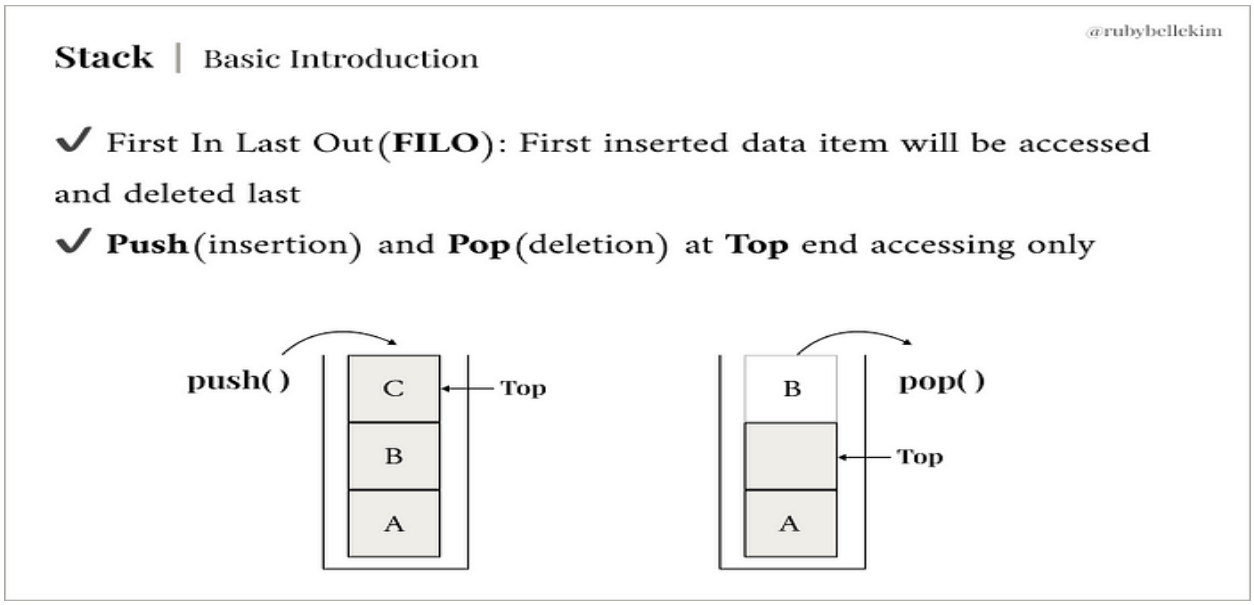
Operations on Stack:

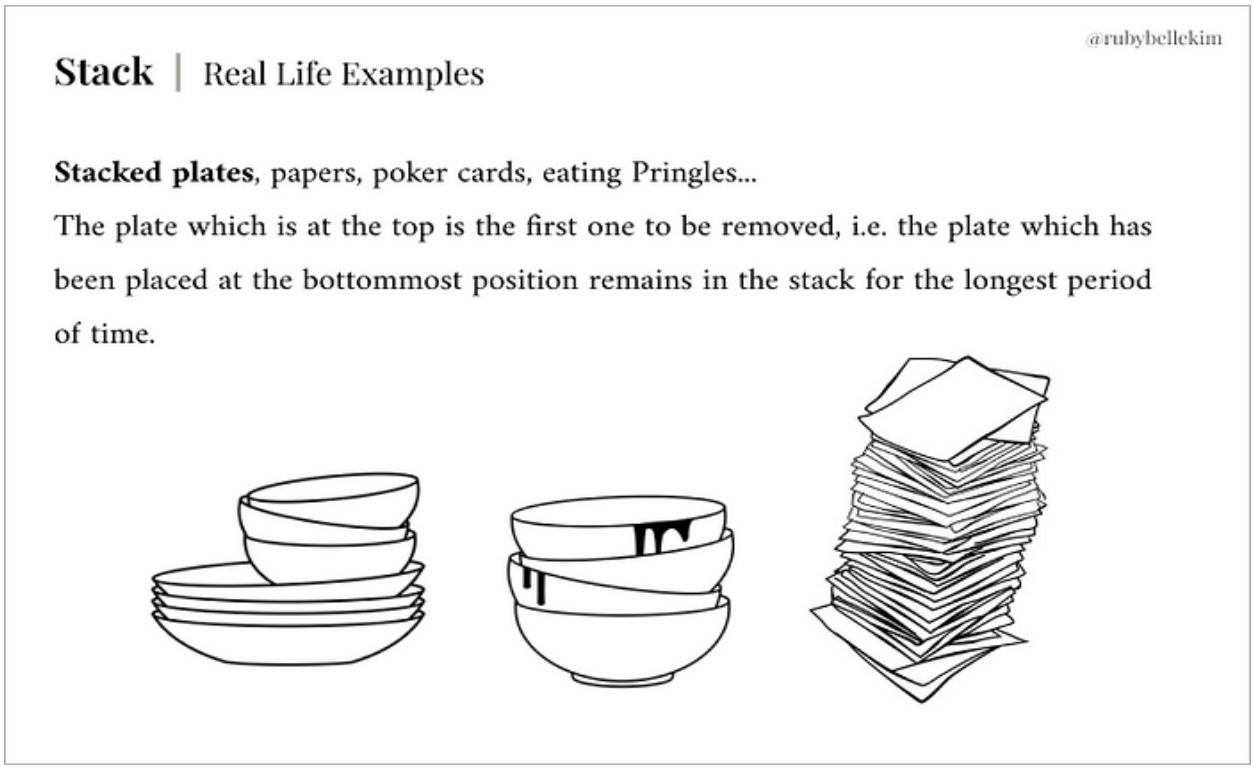
The primary operations associated with a stack are:

- **Push**: Adds an element to the top of the stack.
- **Pop**: Removes an element from top of the stack and returns the top element of the stack.
-
- **Peek (or Top)**: Returns the top element of the stack without removing it.
- **isEmpty**: Checks if the stack is empty.
- **isFull():** returns true if the stack is full else false.




1
2
Push
1
2
3
Push
2
1
3
4
Push
3
2
1
4
5
Push
4
3
2
1
5
6
Push
5
4
3
2
1
6
6
Pop
5
4
3
2
1
7
5
Pop
4
3
2
1
8
4
Pop
3
2
1
9
3
Pop
2
1
10
2
Pop
1


@rubybellekim
Stack | Basic Introduction
✓ First In Last Out (FILO): First inserted data item will be accessed and deleted last
✓ Push (insertion) and Pop (deletion) at Top end accessing only
push( )
C
B
A
Top
B
A
Top
pop( )

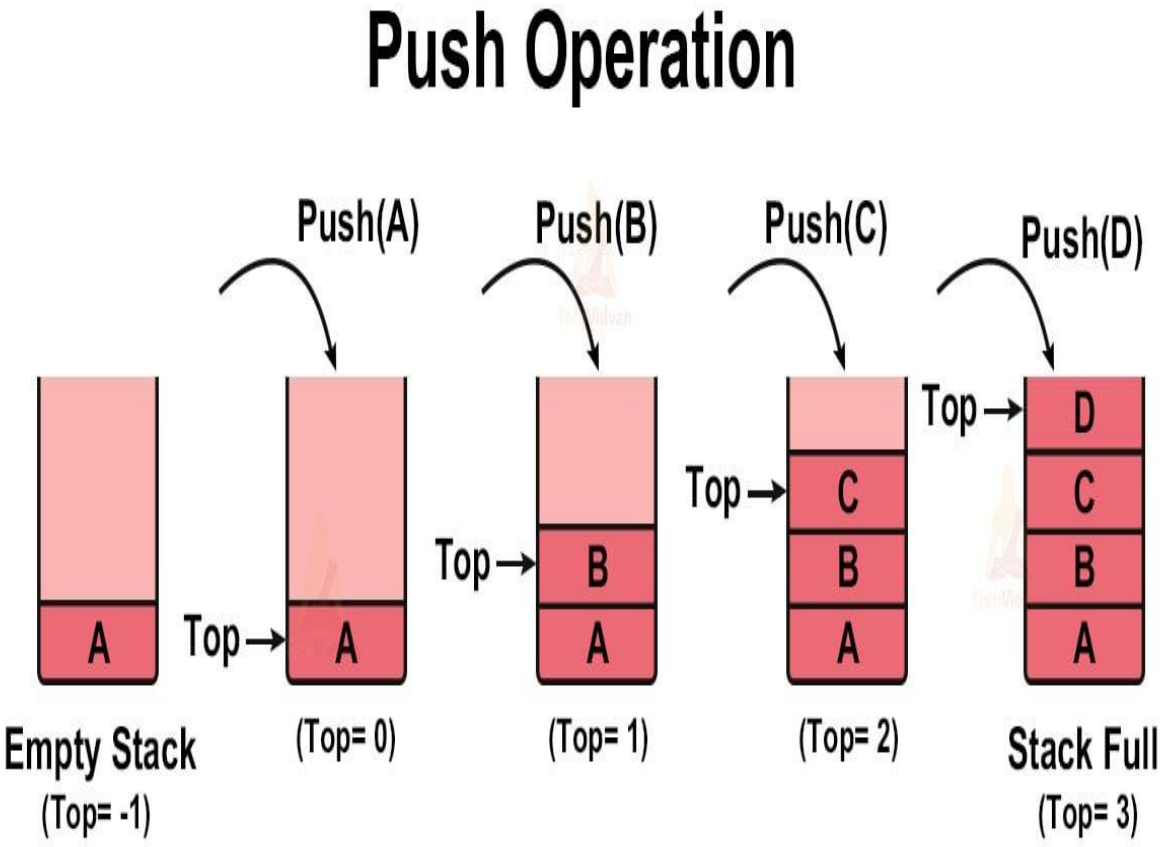


Stack Insertion: push()

The **push**() operation is one of the fundamental operations in a stack data structure. Pushing means inserting an element at the top of the stack. If the stack is full, then it is said to be an Overflow condition.

Algorithm for Push Operation:

- Before pushing the element to the stack, we check if the stack is **full.**
- If the stack is full **(top == capacity-1)**, then **Stack Overflows** and we cannot insert the element to the stack.
- Otherwise, we increment the value of top by 1 **(top = top + 1)** and the new value is inserted at **top position**.
- The elements can be pushed into the stack till we reach the **capacity** of the stack.

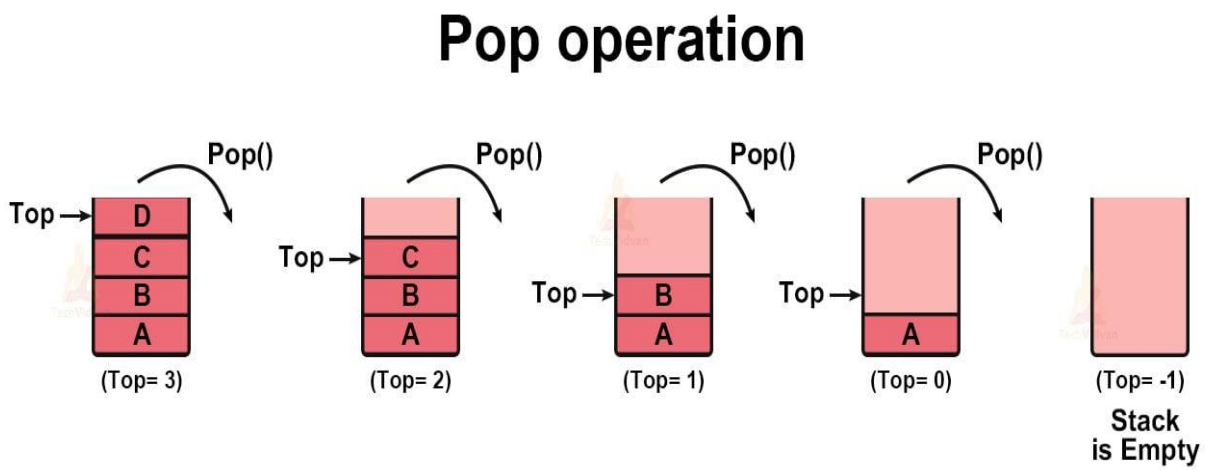


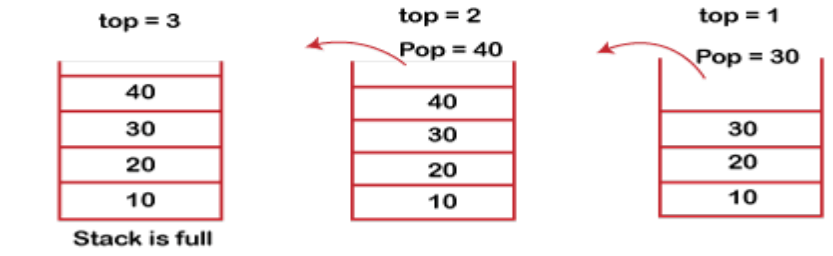
Stack Deletion: pop()

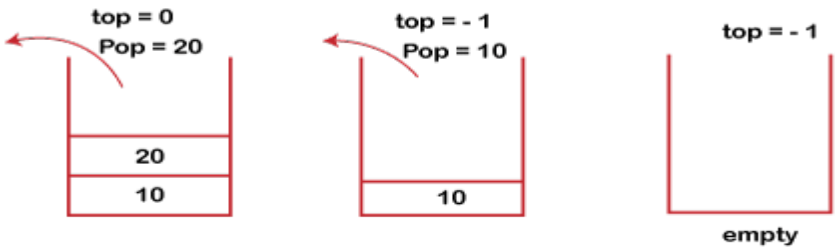
The pop() operation is used to remove the topmost element of the stack and return its value. The pop() operation modifies the state of the stack by removing the topmost element. The items are popped in the reversed order in which they are pushed. If the Stack is empty, it is a Underflow condition.

Algorithm for Pop Operation:

- Before popping the element from the stack, we check if the stack is **empty** .
- If the stack is empty (top == -1), then **Stack Underflows** and we cannot remove any element from the stack.
- Otherwise, we store the value at top, decrement the value of top by 1 **(top = top – 1)** and return the stored top value.





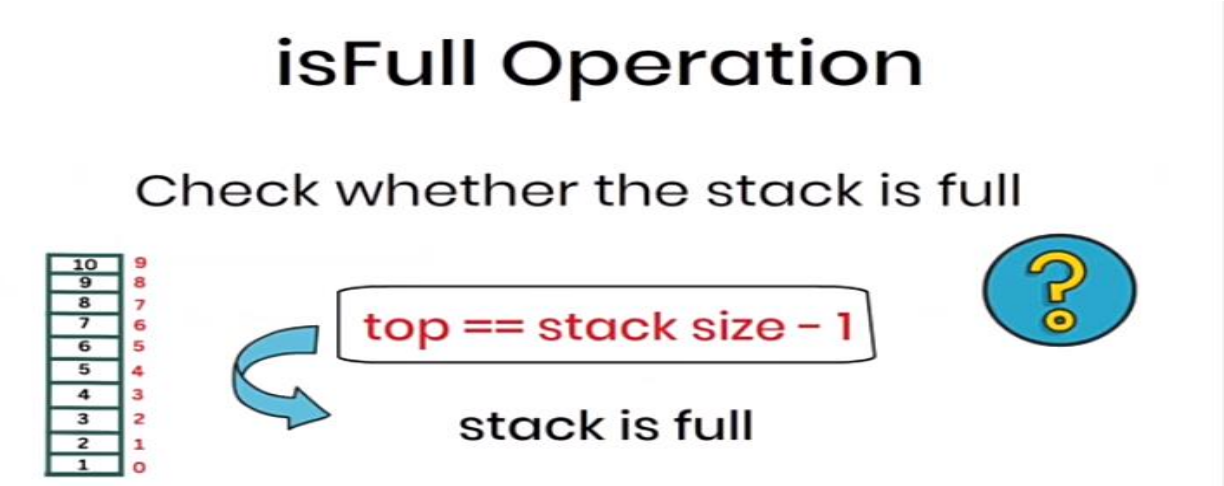


4. isFull()

The isFull() operation is used to determine if the stack is full or not. A stack is said to be full if it has reached its maximum capacity and there is no more space to add new elements to the stack.

Algorithm for isFull Operation:

- Check for the value of **top** in stack.
- If **(top == capacity-1),** then the stack is **full** so return **true**.
- Otherwise, the stack is not full so return **false**.

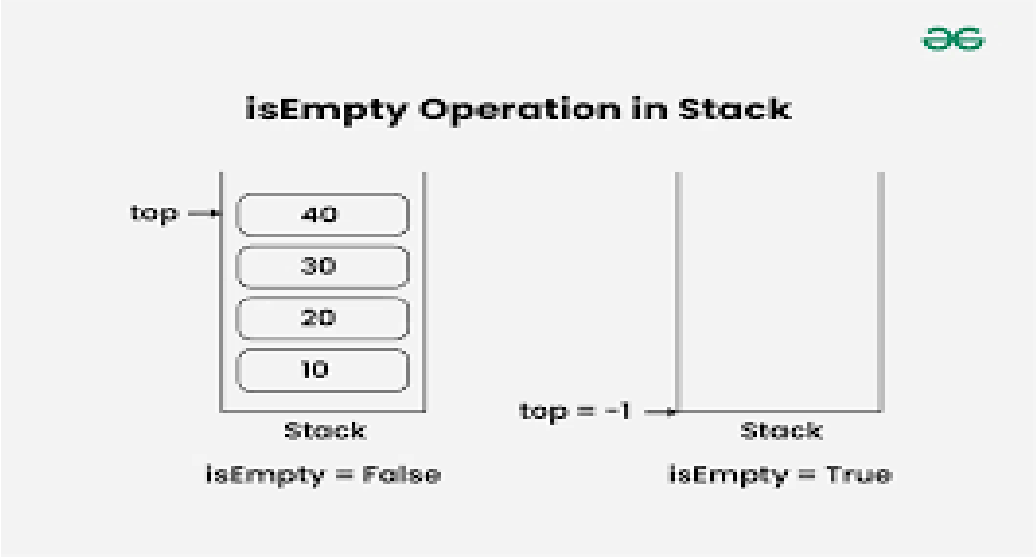


5. isEmpty()

The isEmpty() operation is used to check if the stack is empty or not. It returns a boolean value, true when the stack is empty, otherwise false.

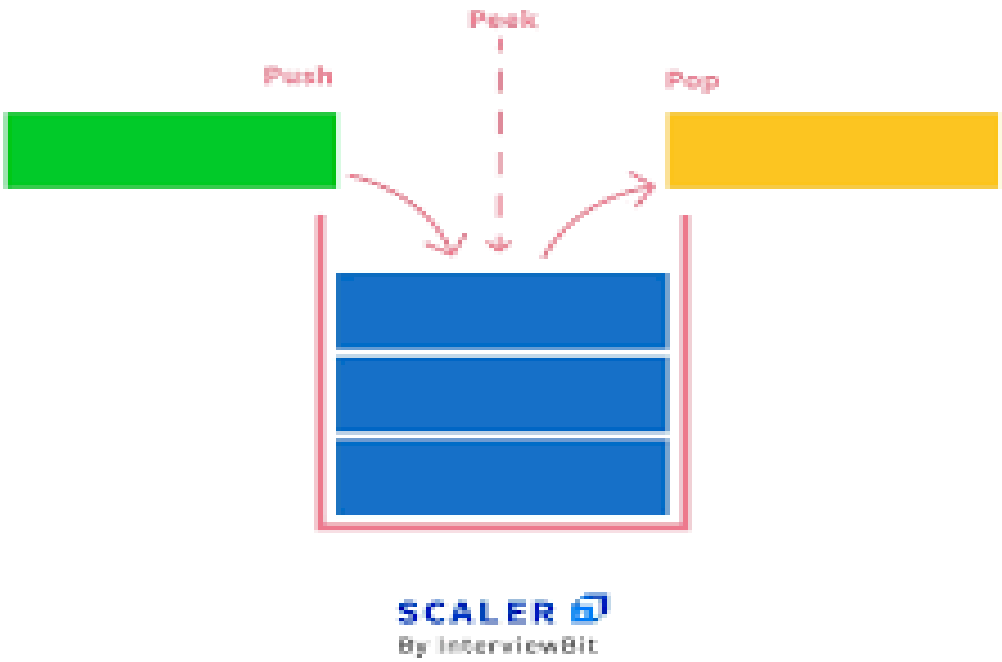
Algorithm for isEmpty Operation:

- Check for the value of **top** in stack.
- If **(top == -1)** , then the stack is **empty** so return **true** .
- Otherwise, the stack is not empty so return **false** .



3. peek()

The peek() operation returns the value of the topmost element of the stack without modifying the stack. This can be useful when you need to check the value of the top element before deciding whether to remove it or not.



Programs to demonstrate **push** and **pop** operation in stack in data structure in C

Program 1:

```
#include <stdio.h>

 int MAX SIZE = 8;

 int stack[8];

 int top = -1;

 /* Check if the stack is empty */

Int  isempty(){

 If (top == -1) return 1;

 else return 0;

 }

/* Check if the stack is full */
```

```
Int  isfull(){
 if (top == MAX SIZE)
 return 1;
 else return 0;
 }
/* Function to return the topmost element in the stack
*/
int peek(){
return stack[top];
 }
/* Function to delete from the stack */
int pop(){
int data;
 if (!isempty()) {
 data = stack[top];
 top = top - 1;
 return data;
 }   else  {
          printf("Could not retrieve data, Stack is empty.\n");
 }
}
```

```
/* Function to insert into the stack */
int push(int data){
 if (! isfull()) {
 top = top + 1;
 stack[top] = data;
 }  else  {
        printf("Could not insert data, Stack is full.\n");
 }
 }
/* Main function */
int main(){
push(44);
 push(10);
 push(62);
 push(123);
 push(15);
 printf("Element at top of the stack: %d\n" ,peek());
 printf("Elements: \n");
// print stack data
while(!isempty()) {
int data = pop();
```

```
 printf("%d\n",data);
 }

printf("Stack full: %s\n" , isfull()?"true":"false");
 printf("Stack empty: %s\n" ,
isempty()?"true":"false");
 return 0;
 }
```

Output

Element at top of the stack: 15
Elements:
15
123
62
10
44
Stack full: false
Stack empty: true

Program 2:

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #define SIZE 4
4. int top = -1, inp_array[SIZE];
5. void push();
6. void pop();
7. void show();
8. int main()
```

```
{
    int choice;
    while (1)
    {
    printf("\nPerform operations on the stack:");
    printf("\n1.Push the element\n2.Pop the element\n3.Show\n4.End");

        printf("\n\nEnter the choice: ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            push();
            break;
        case 2:
            pop();
            break;
        case 3:
            show();
            break;
        case 4:
            exit(0);
        default:
            printf("\nInvalid choice!!");
        }
    }
}
void push()
{
    int x;
    if (top == SIZE - 1)
    {
        printf("\nOverflow!!");
    }
    else
    {
        printf("\nEnter the element to be added onto the stack: ");
```

```
            scanf("%d", &x);
            top = top + 1;
            inp_array[top] = x;
        }
    }
    void pop()
    {
        if (top == -1)
        {
            printf("\nUnderflow!!");
        }
        else
        {
            printf("\nPopped element: %d", inp_array[top]);
            top = top - 1;
        }
    }
    void show()
    {
        if (top == -1)
        {
            printf("\nUnderflow!!");
        }
        else
        {
            printf("\nElements present in the stack: \n");
            for (int i = top; i >= 0; --i)
                printf("%d\n", inp_array[i]);
        }
    }
```

Output

10 pushed into stack
20 pushed into stack

```
30 pushed into stack
30 Popped from stack
Top element is : 20
Elements present in stack : 20 10
.....................................................
Process executed in 3.22 seconds
Press any key to continue.
```

Stack Home work and Lab work

Q1: let **S** be the **push** operation to add element into a stack, and let **U** be the pop operation to remove element from a stack. Suppose also we have a set of input element into a stack in the following sequence M, B, Y, N, R.
What is the output sequence of the characters (M, B, Y, N, R) after executing the following operations (with drawing):

a) S S U U S U S U S U
b) S S S U S U U S U U

Q2: let **S** be the push operation to add element into a stack, and let **U** be the pop operation to remove element from a stack. Suppose also we have a set of input element into a stack in the following sequence 1, 2, 3, 4, 5.
Which of the following outputs is correct according to stack operations?

a) 2 4 5 3 1
b) 4 2 3 1 5
c) 4 5 1 2 3
d) 4 3 5 2 1

Q3: let **S** stand for push stack operation, and **U** stand for stack pop operation. If the order of the stack input stream is **1 2 3 4 5**.

a) What is the output if we execute the following operations?
S S U U S S S U U U

b) Which of the following can be obtained as output stream?
i) 5 1 3 2 4
ii) 2 3 5 1 4
iii) 3 2 1 5 4

Q4: if the stack input stream is **A B C D E F** what is the sequence of operations to get the output **C B D E F A**?

Q5: write an algorithm to push three elements into a stack of size 10 characters.

Q6: write an algorithm to pop three elements from a stack of size 10 characters.

Q7: write an algorithm to check if a stack of size 10 characters is empty.

Q8: write an algorithm to check if a stack of size 10 characters is full.

Answer the following MCQ (multiple choice questions):

1. Process of inserting an element in stack is called _____
   a) Create
   b) Push
   c) Evaluation
   d) Pop
2. 2. Process of removing an element from stack is called _____
   a) Create
   b) Push
   c) Evaluation
   d) Pop
3. 3. In a stack, if a user tries to remove an element from an empty stack it is called

   a) Underflow
   b) Empty collection
   c) Overflow
   d) Garbage Collection

4. Pushing an element into stack already having five elements and stack size of 5, then stack becomes _____
   a) Overflow
   b) Crash
   c) Underflow
   d) User flow

5. **Consider the following operation performed on a stack of size 5.**

   Push(1); Pop(); Push(2); Push(3); Pop(); Push(4); Pop(); Pop(); Push(5);
   After the completion of all operation, the number of element present on stack are

   a) 1 b) 2 c) 3 d) 4

6. If the elements "A", "B", "C" and "D" are placed in a stack and are deleted one at a time, in what order will they be removed?

a) A B C D **b)** D C B A **c)** D C A B **d)** A B D C

7) What is the output of the below code?

```
1. #include <stdio.h>
2. int main()
3. {
4.    int arr[5]={10,20,30,40,50};
5.    printf("%d", arr[5]);
6.
7.     return 0;
8. }
```

a) Garbage value **b**) 10 **c**) 5 **d**) None of the above

8) When the user tries to delete the element from the empty stack then the condition is said to be a ____

1. Underflow
2. Garbage collection
3. Overflow
4. None of the above

9) If the size of the stack is 10 and we try to add the 11th element in the stack then the condition is known as___

1. Underflow
2. Garbage collection
3. Overflow
4. None of the above

10) Consider the following stack implemented using stack.

```
1. #define SIZE 11
2. struct STACK
3. {
4.    int arr[SIZE];
5.    int top= -1;
6. }
```

What would be the maximum value of the top that does not cause the overflow of the stack?

a) 8 **b**) 9 **c**) 11 **d**) 10

11. A pointer variable which contains the location at the top element of the stack is called …..
a) Top b) Last c) Final d) End

12. Choose correct output for the following sequence of operations.

push(5), push(8), pop, push(2), push(5), pop, pop, pop, push(1), pop

a) 8 5 2 5 1
b) 8 5 5 2 1
c) 8 2 5 5 1
d) 8 1 2 5 5

13. The following sequence of operations is performed on stack:
Push(1), push(2), pop, push(1), push(2), pop, pop, pop, push(2), pop.
The sequence of popped values is:
a) 2 2 1 1 2 **b**) 2 2 1 2 2 **c**) 2 1 2 2 1 **d**) 2 1 2 2 2

14. Which operation on a stack is used to retrieve the top element without removing it?
a) pop b) push c) peek d) size

15. the stack is based on the rule
a) first in first out **b**) last in first out **c**) both a and b **d**) first in first out

16. the push() operation is used
a) to insert an element **b**) to remove an element
c) to move an element **d**) all of the above

17. The pop() operation removes

a) the element lastly inserted **b**) first element of the stack
c) any element randomly **d**) none of the above

18. if push is performed on a stack holding elements equal to its capacity then such situation is called

a) stack overflow **b**) stack underflow **c**) illegal operation **d**) none of the above