# EXPERIMENT-5

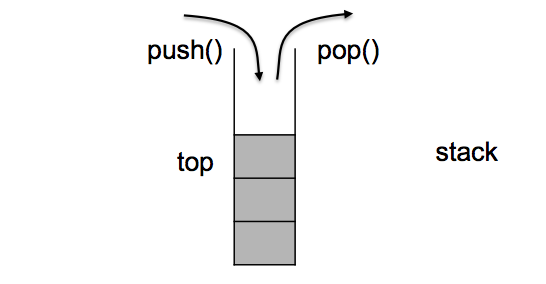
# AIM:

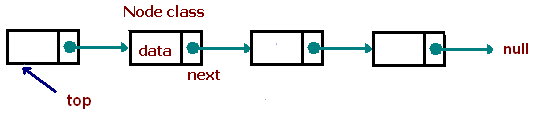
Develop a program to perform operations of a Stack using the implementation in Arrays and Linked Lists

# DESCRIPTION:

A stack is an ordered list with a restriction that the elements are inserted and removed according to the last-in first-out (LIFO) principle. A stack is a limited access data structure - elements can be added and removed from the stack only at one end of the list called the top, the other end remains deactivate. push adds an item to the top of the stack, pop removes the item from the top.

A linked stack is a linear list of elements commonly implemented as a singly linked list whose start pointer performs the role of the top pointer of the stack as shown in figure.





# ADVANTAGES :

* Simplicity: Stacks are simple to understand and implement.
* Efficient data management: Stacks provide efficient insertion and deletion operations.
* Function call management: Stacks are used to manage function calls and recursion.
* Undo/Redo operations: Stacks enable undo and redo functionality in applications.
* Expression evaluation: Stacks are used to evaluate expressions in various notations.
* Browser history: Stacks help browsers keep track of visited web pages.

# DISADVANTAGES:

* Fixed capacity: Stacks have a fixed capacity, which can lead to overflow if exceeded.
* Access limitations: Accessing elements in the middle of the stack is not efficient.
* No random access: Stacks only allow access to the topmost element.

# APPLICATIONS:

* Function calls and recursion: When a function is called, the current state of the program is pushed onto the stack. When the function returns, the state is popped from the stack to resume the previous function’s execution.
* Undo/Redo operations: Applications often use stacks to keep track of previous actions, enabling undo and redo functionality.
* Expression evaluation: Stack data structures are used to evaluate expressions in infix, postfix, and prefix notations. Operators and operands are pushed onto the stack, and operations are performed based on the stack’s top elements.
* Browser history: Web browsers use stacks to keep track of the web pages you visit1.

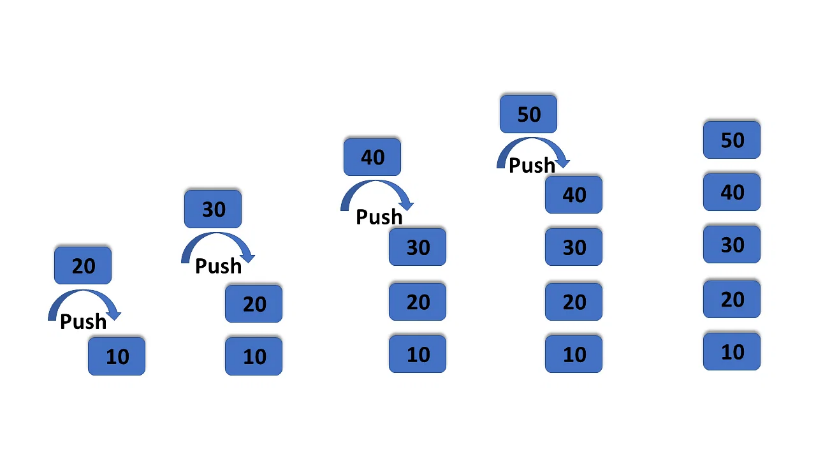
# PERFORMANCE:

|  |  |  |
| --- | --- | --- |
| Operation | Time Complexity | Space Complexity |
| push() | O(1) | O(1) |
| pop() | O(1) | O(1) |
| top() | O(1) | O(1) |
| size() | O(1) | O(1) |
| isEmpty() | O(1) | O(1) |
| isFull() | O(1) | O(1) |

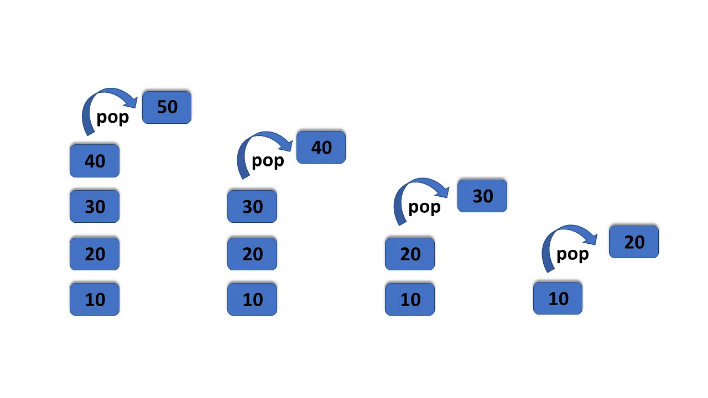
### 

### **Algorithm for stack operations (using array):**

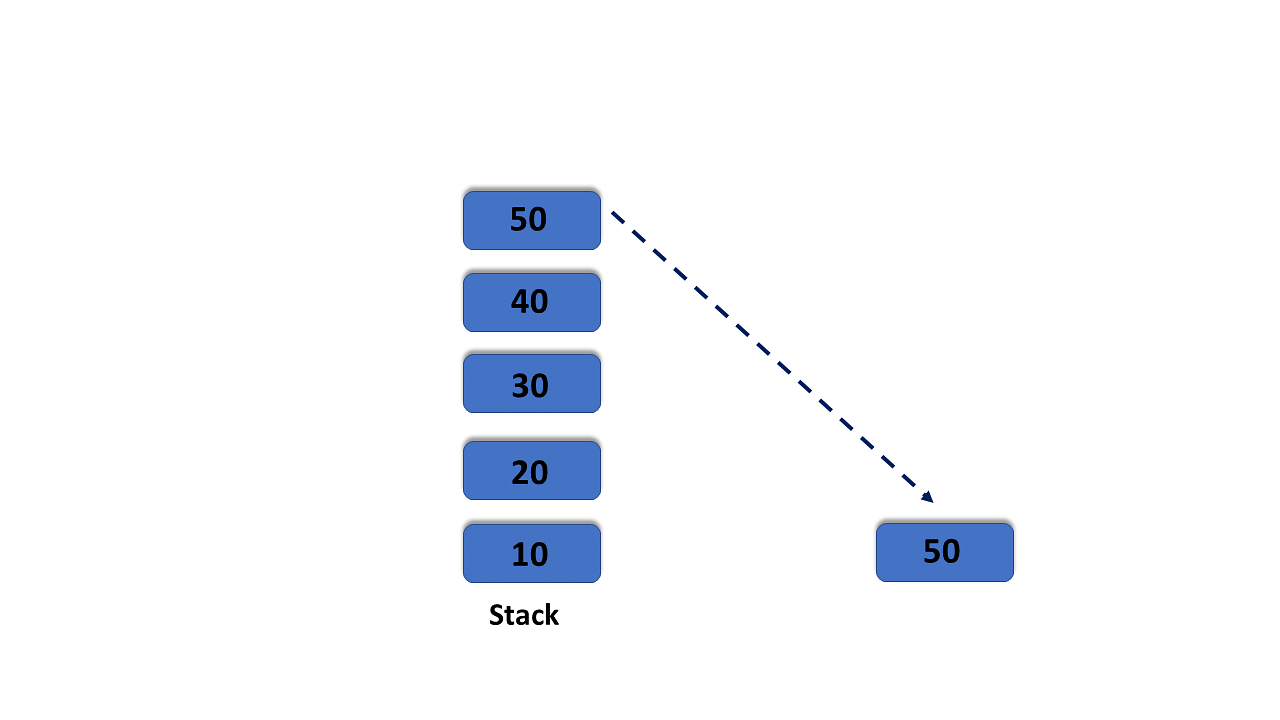
1. **Push Operation**:
   1. Check if the stack is full. If the top pointer is equal to (MAX-1), print an overflow message.
   2. If the stack is not full, increment the top variable.
   3. Store the value at the index pointed by the top variable.



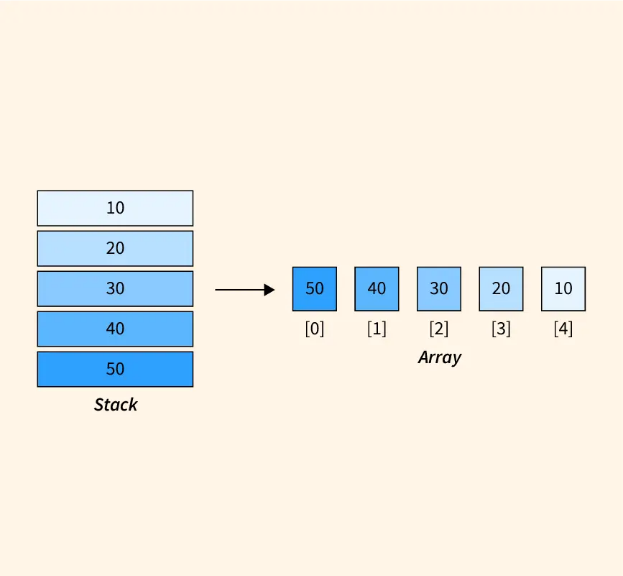
1. **Pop Operation**:
   1. Check if the stack is empty. If the top variable has a value less than 0, print an underflow condition.
   2. If the stack is not empty, decrement the top variable.



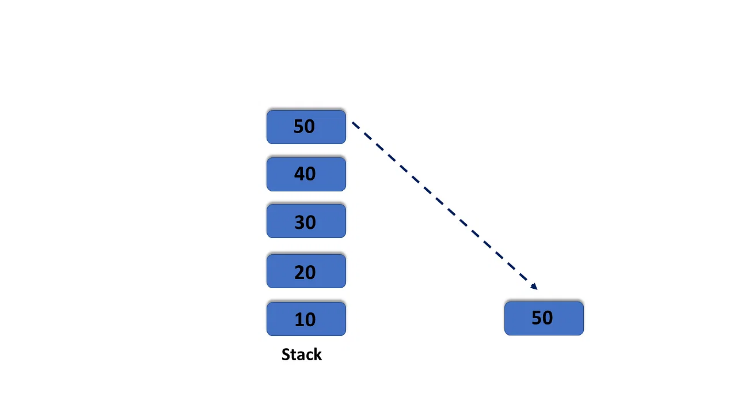
1. **Peek Operation**:
   1. Return the value of top element without decrementing top pointer.



1. **Display Operation**:
   1. Start from top and go till 0 (end of stack).
   2. Print values at all these positions during traversal.

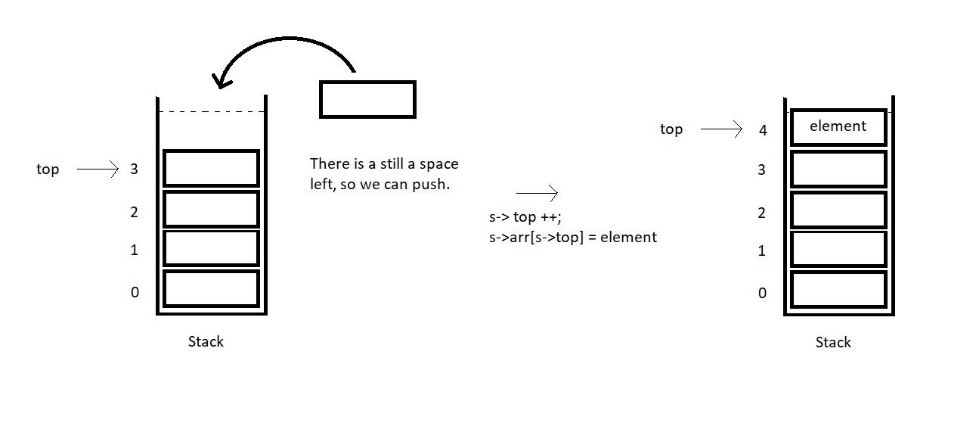


1. **Count Operation**:
   1. Return the count of elements present in the stack which can be easily tracked with a counter variable during push and pop operations.

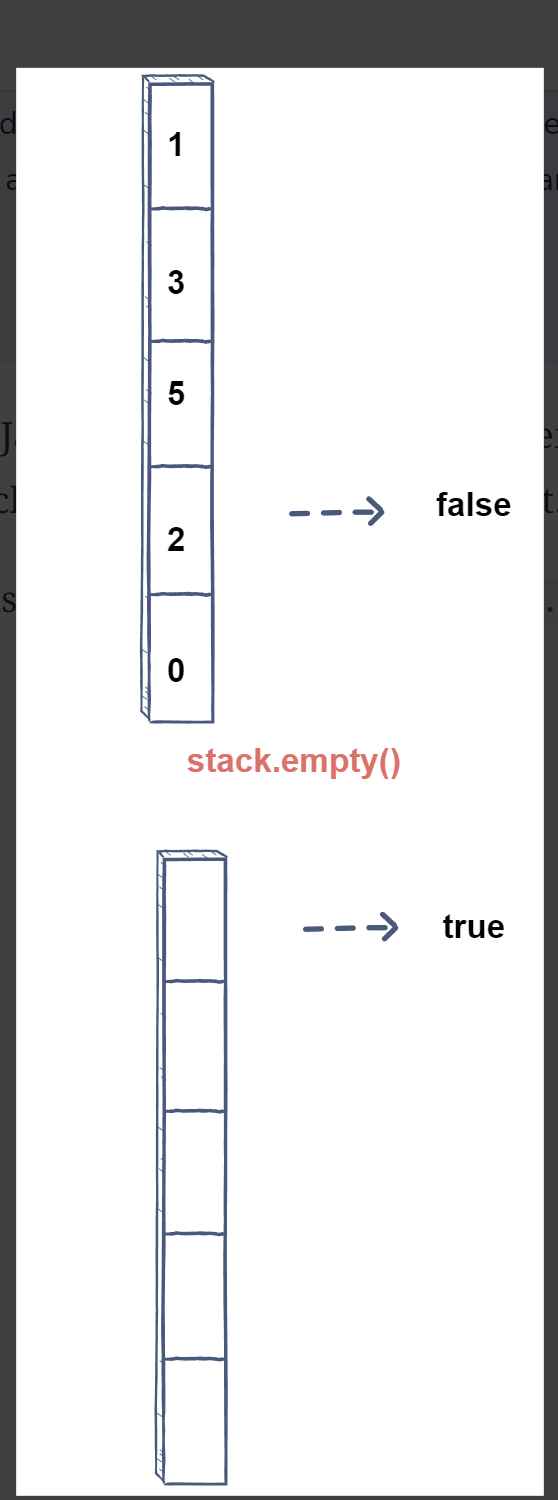


Count=5

1. **IsFull Operation**:
   1. Check if top equals to MAXSIZE-1 then return true which signifies that Stack is full.



1. **IsEmpty Operation**:
   1. Check if top equals to -1 then return true which signifies that Stack is empty.



### **Program:**

/\*Demonstration of stack operations using arrays\*/

#include<stdio.h>

#include<stdlib.h>

int isempty();

int isfull();

void push();

void pop();

void peek();

void display();

int count();

int n, top=-1;

int \*a;

void main()

{

printf("Enter the size of stack: ");

scanf("%d",&n);

a=(int\*)malloc(sizeof(int)\*n);

while(1)

{

printf("\n1. Push\n");

printf("2. Pop\n");

printf("3. Peek\n");

printf("4. isempty\n");

printf("5. isfull\n");

printf("6. Display\n");

printf("7. count\n");

int choice;

printf("\nEnter your choice: ");

scanf("%d",&choice);

switch(choice)

{

case 1: push();

break;

case 2: pop();

break;

case 3: peek();

break;

case 4: isempty();

break;

case 5: isfull();

break;

case 6: display();

break;

case 7: count();

break;

default: exit(0);

}

}

}

int isempty()

{

if(top == -1)

printf("\nThe list is empty\n");

else

printf("\nThe list is not empty\n");

}

int isfull()

{

if(top==n-1)

printf("\nThe list is full\n");

else

printf("\nThe list is not full\n");

}

void push()

{

int data;

printf("\nEnter the value to be added: ");

scanf("%d",&data);

if (top==n-1)

printf("\nStack is full\n");

else

{

top++;

a[top]=data;

}

}

void pop()

{

if (top==-1)

printf("\nStack is empty\n");

else{

printf("\nThe top most element is removed.\n");

top--;

}

}

void peek()

{

if(top==-1)

printf("\nStack is empty\n");

else

printf("\nThe top most element is %d\n",a[top]);

}

void display()

{

if (top==-1)

printf("\nThe stack is empty.\n");

else{

printf("\nThe stack have elements: \n");

for(int i=top;i>=0;i--)

printf("%d\n",a[i]);

}

}

int count()

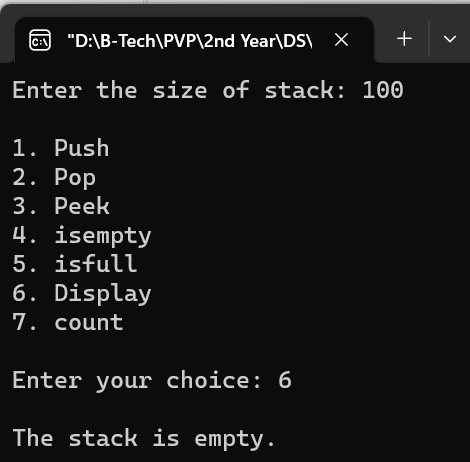
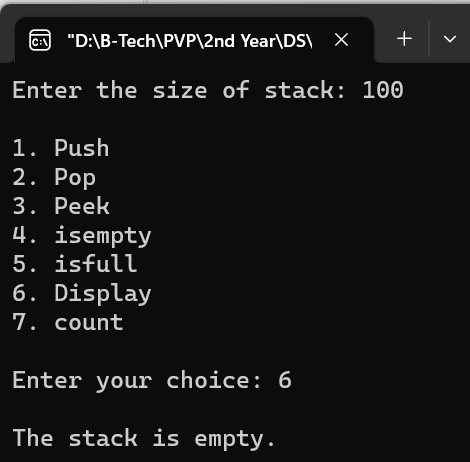
{

printf("\nThe number of elements: %d\n",top+1);

}

**Test case-1:** Initially Check if the stack is empty

Input:

Enter the size of stack: 100

1. Push

2. Pop

3. Peek

4. isempty

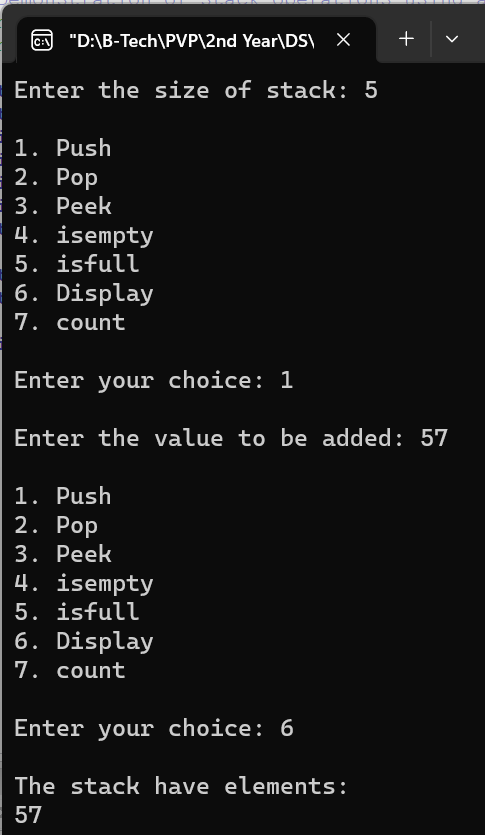
5. isfull

6. Display

7. count

Enter your choice: 6

Output:The stack is empty.

**Test case-2:** Insert one element into the stack as the first element and display stack

Input:

Enter the size of stack: 5

1. Push

2. Pop

3. Peek

4. isempty

5. isfull

6. Display

7. count

Enter your choice: 1

Enter the value to be added: 57

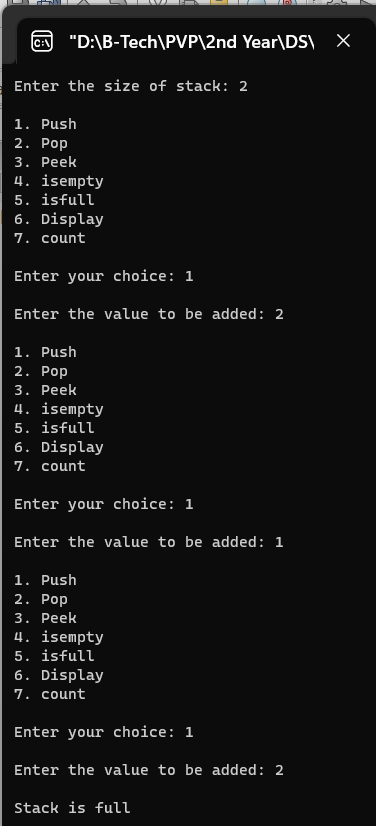
Enter your choice: 6

Output:

The stack have elements:

57

**Test case-3:** Continuously insert the elements into till stack is full and check for overflow condition

 Input:

Enter the size of stack: 2

1. Push

2. Pop

3. Peek

4. isempty

5. isfull

6. Display

7. count

Enter your choice: 1

Enter the value to be added: 2

Enter your choice: 1

Enter the value to be added: 1

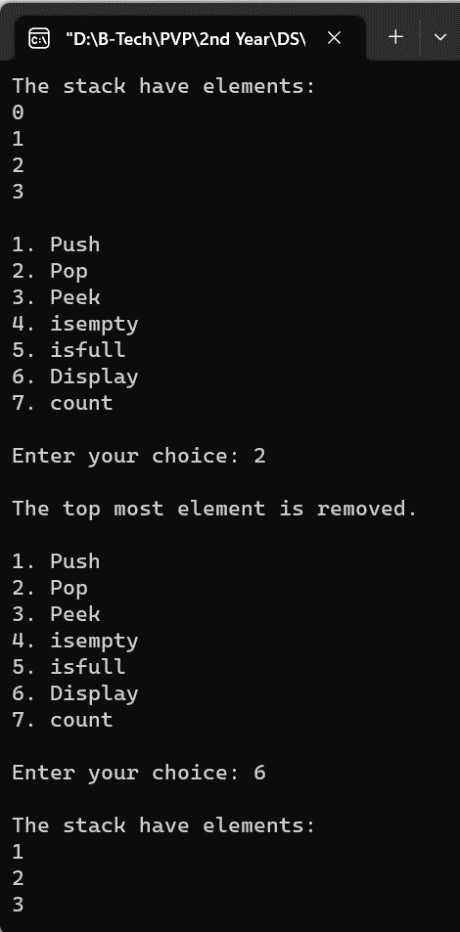
Enter your choice: 1

Enter the value to be added:2

Output:

Stack is full

**Test case-4:** Delete an element  from the stack and display

Input: 0 1 2 3

Enter the size of stack: 4

1. Push

2. Pop

3. Peek

4. isempty

5. isfull

6. Display

7. count

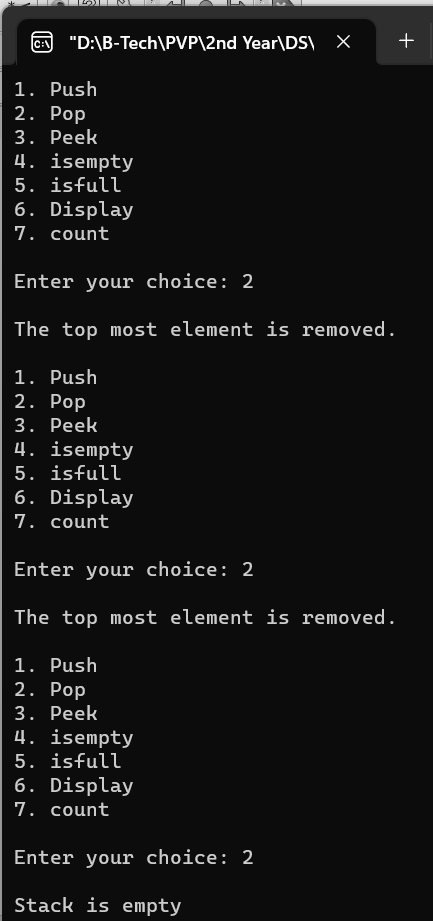
Enter your choice: 6

Enter your choice: 2

Enter your choice: 6

Output:

1 2 3

**Test case-5:** Delete all the elements from the stack and check for underflow condition

Input: 1 2

Enter the size of stack: 4

1. Push

2. Pop

3. Peek

4. isempty

5. isfull

6. Display

7. count

Enter your choice: 2

Enter your choice: 2

Enter your choice: 2

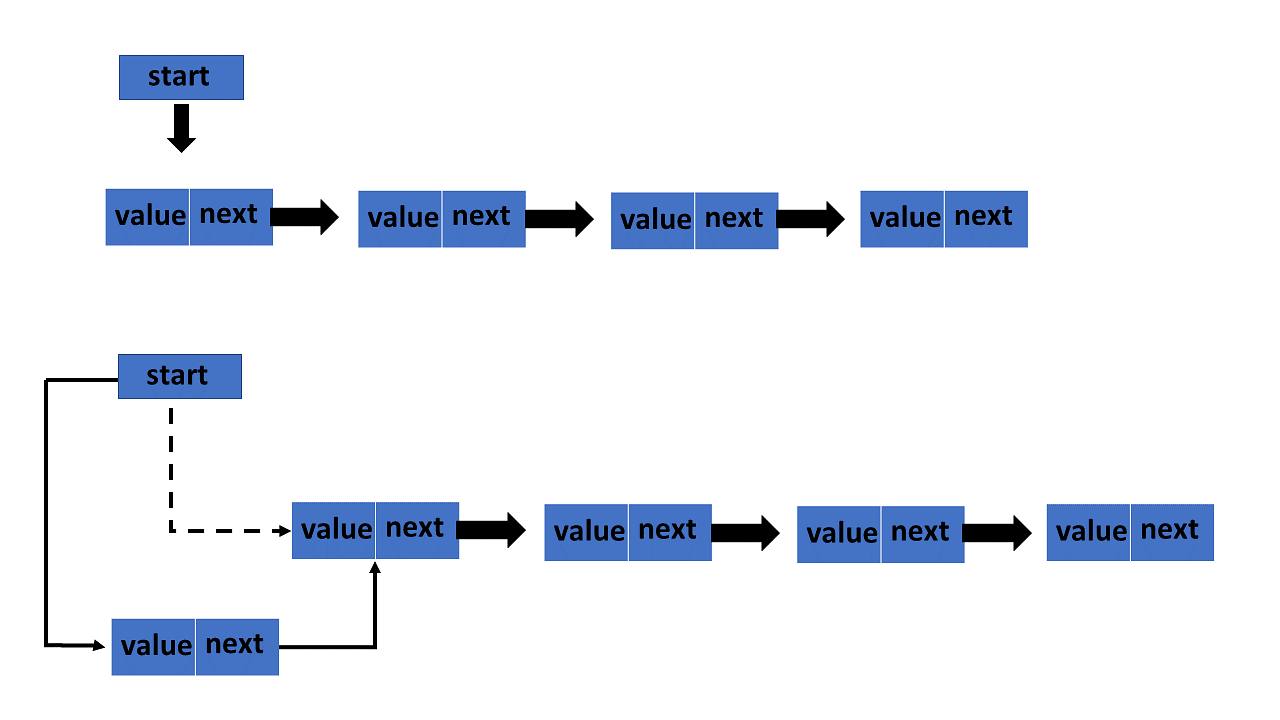
Output: Stack is empty

### **Linked list Implementation:**

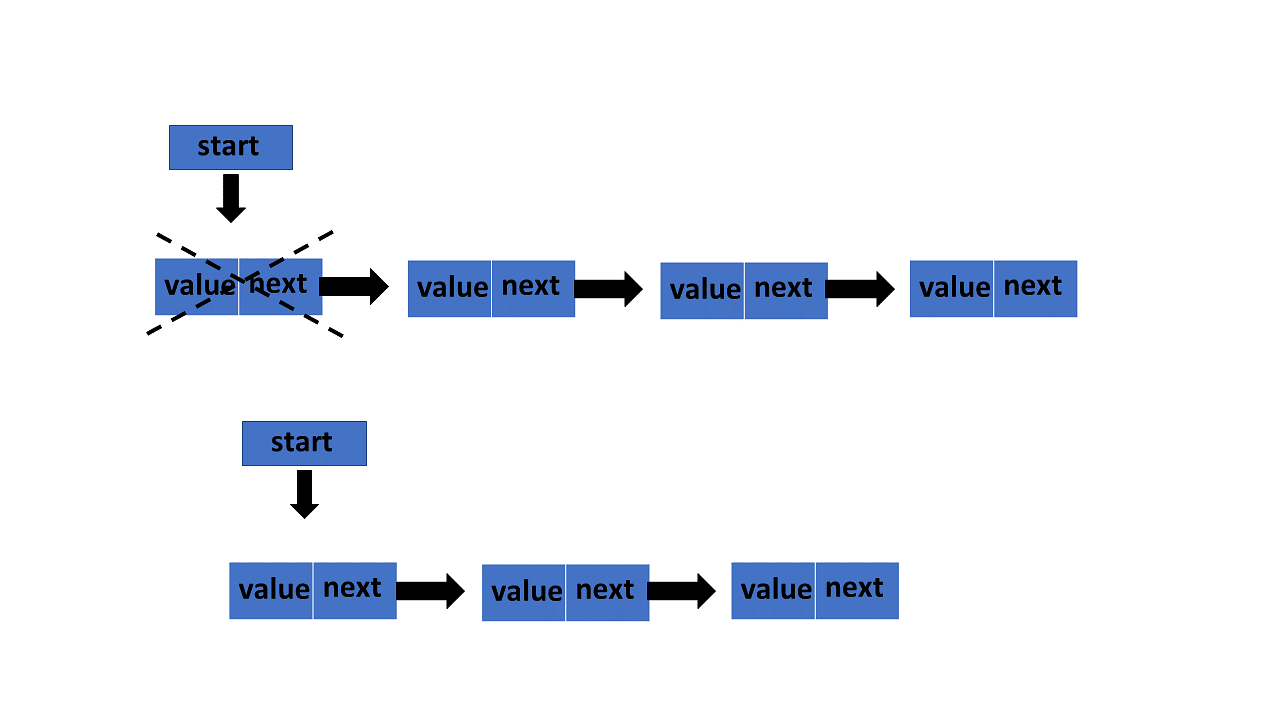
The other way of implementing stacks is by using Linked lists. Push operation is implemented by inserting element at the beginning of the list. Pop operation is implemented by deleting the node from the beginning (the header/top node).

### **Algorithm for stack operations (using LinkedList):**

1. **Push Operation**:
   1. Create a new node.
   2. If the list is empty, then the new node is pushed as the first node of the linked list. Assign a value to the data part of the node and assign null to the address part of the node.
   3. If there are some nodes in the list already, then add the new element at the beginning of the list (to not violate the property of the stack). Assign the address of the starting element to the address field of the new node and make the new node, the starting node of the list.



1. **Pop Operation**:
   1. Check whether there is any node present in the linked list or not, if not then return.
   2. Otherwise, make a pointer (let’s say temp) to point to the top node and move forward the top node by 1 step.
   3. Now free this temp node.



1. **Peek Operation**:
   1. Check if there is any node present or not, if not then return.
   2. Otherwise, return the value of top node of the linked list.

NULL

40

20000000

40

top

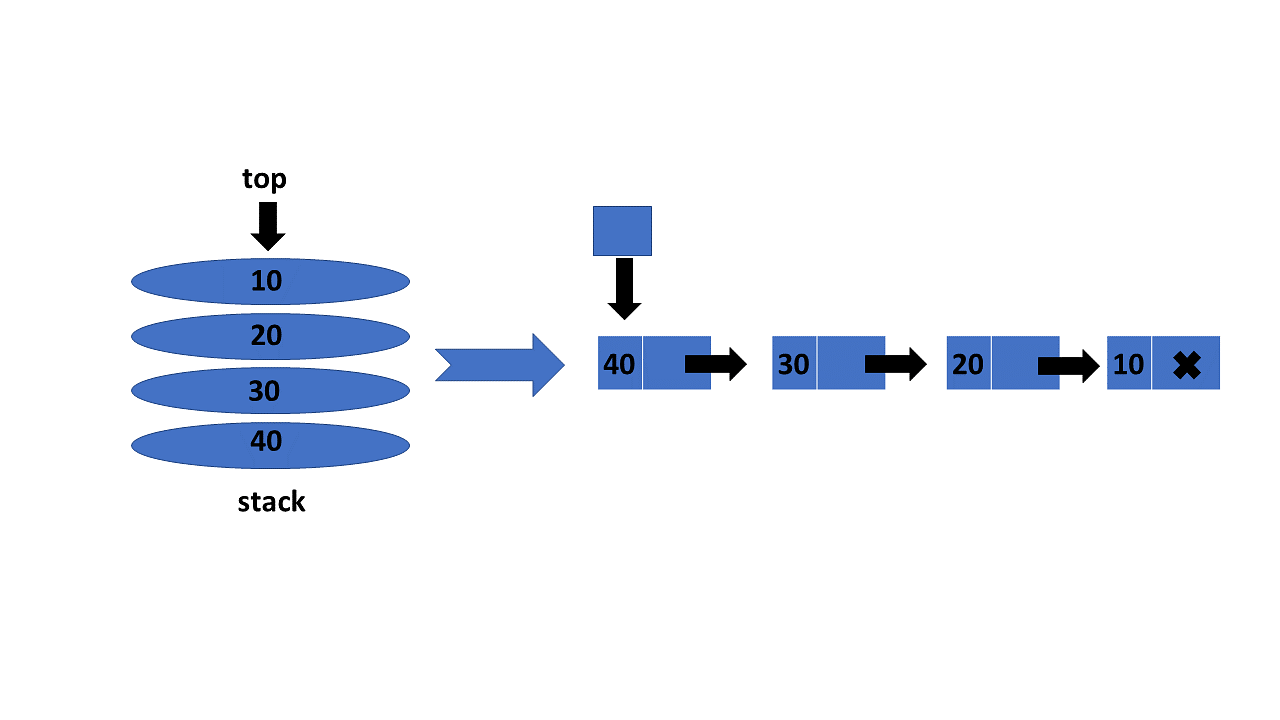
10000000

30

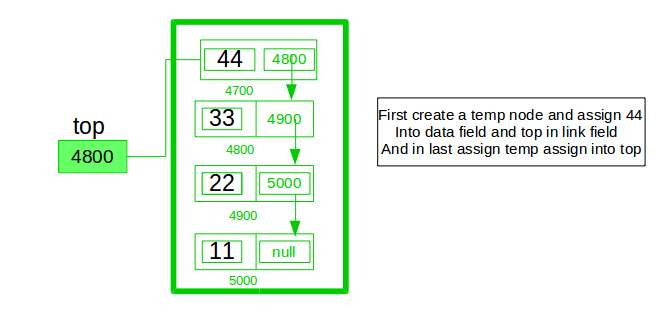
**output**

30

1. **Display Operation**:
   1. Take a temp node and initialize it with top pointer.
   2. Now start traversing temp till it encounters NULL.
   3. Simultaneously print the value of each temp node.

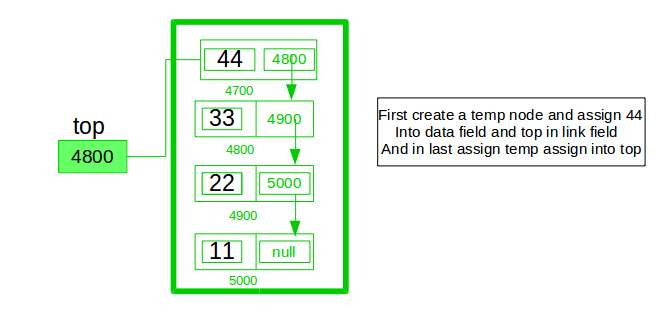
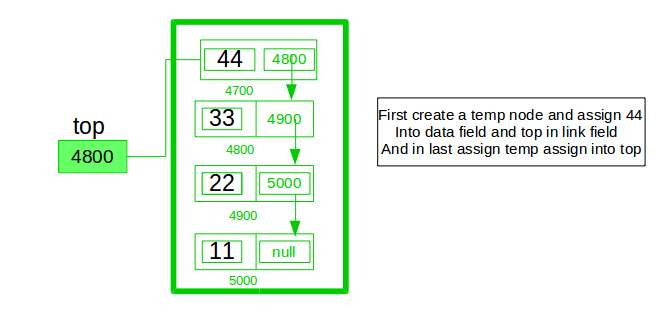


1. **Count Operation**:
   1. Initialize a counter variable to 0.
   2. Traverse through each element in the linked list and increment counter for each element until you reach NULL.



count=4

1. **IsEmpty Operation**:
   1. Check if top == NULL. If true, then stack is empty, otherwise it’s not.



NULL

Non Empty Stack Empty Stack

### **Program:**

/\*Demonstration of stack operations using linkedlists\*/

#include<stdio.h>

#include<stdlib.h>

typedef struct node{

int data;

struct node\* next;

}node;

int isempty();

void push();

void pop();

void peek();

void display();

int count();

node\* top;

int c=0;

void main(){

while(1){

printf("\n1. Push\n");

printf("2. Pop\n");

printf("3. Peek\n");

printf("4. isempty\n");

printf("5. Display\n");

printf("6. count\n");

int choice;

printf("\nEnter your choice: ");

scanf("%d",&choice);

switch(choice)

{

case 1: push();

break;

case 2: pop();

break;

case 3: peek();

break;

case 4: isempty();

break;

case 5: display();

break;

case 6: count();

break;

default: exit(0);

}

}

}

int isempty()

{

if(top == NULL)

printf("\nThe stack is empty\n");

else

printf("\nThe stack is not empty\n");

}

void push()

{

int value;

printf("\nEnter the value to be added: ");

scanf("%d",&value);

node\* temp;

temp = (node\*)malloc(sizeof(node\*));

temp->data = value;

temp->next = NULL;

if (top==NULL)

top = temp;

else{

temp->next=top;

top = temp;

}

c++;

}

void pop()

{

if (top==NULL)

{

printf("\nStack is empty\n");

}

else

{

node\* p = top;

printf("\nThe top most element is removed\n");

top = top->next;

free(p);

c--;

if (top == NULL)

{

printf("\nStack is now empty\n");

}

}

}

void peek()

{

if(top==NULL)

printf("\nStack is empty\n");

else

printf("\nThe top most element is %d\n",top->data);

}

void display()

{

if (top==NULL)

printf("\nThe stack is empty.\n");

else{

printf("\nThe elements in the stack are:\n");

node\* p;

p = top;

while(p->next!=NULL){

printf("%d\n",p->data);

p = p->next;

}

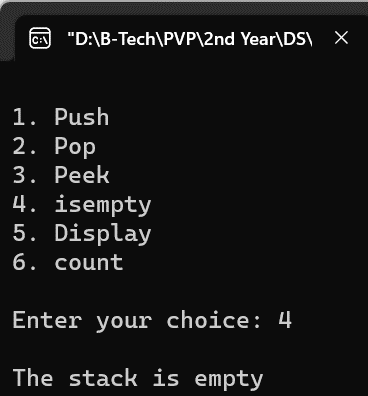
printf("%d\n",p->data);

}

}

int count(){

printf("\nThe number of elements: %d\n",c);

}

**Test case-1:** Initially Check if the stack is empty

Input:

1. Push

2. Pop

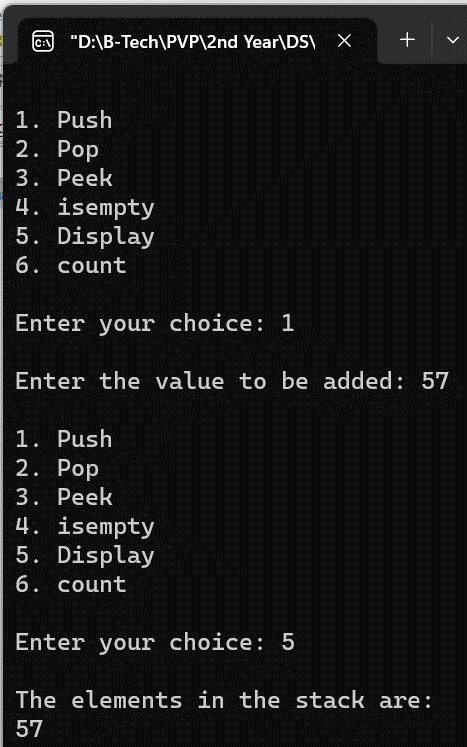
3. Peek

4. isempty

5. Display

6. count

Enter your choice: 4

 Output:

The stack is empty

**Test case-2:** Insert one element into the stack as the first element and display stack

Input:

1. Push

2. Pop

3. Peek

4. isempty

5. Display

6. count

Enter your choice: 1

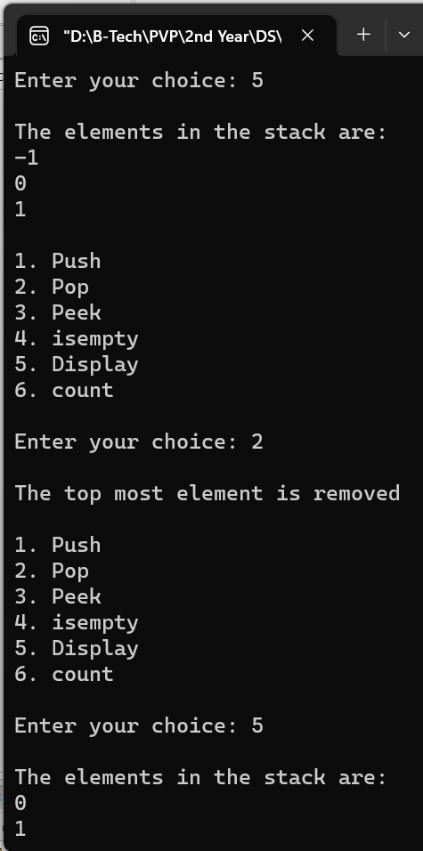
Enter the value to be added: 57

Enter your choice: 5

Output:The elements in the stack are:

57

**Test case-3:** Delete an element  from the stack and display

Input: -1 0 1

1. Push

2. Pop

3. Peek

4. isempty

5. Display

6. count

Enter your choice: 5

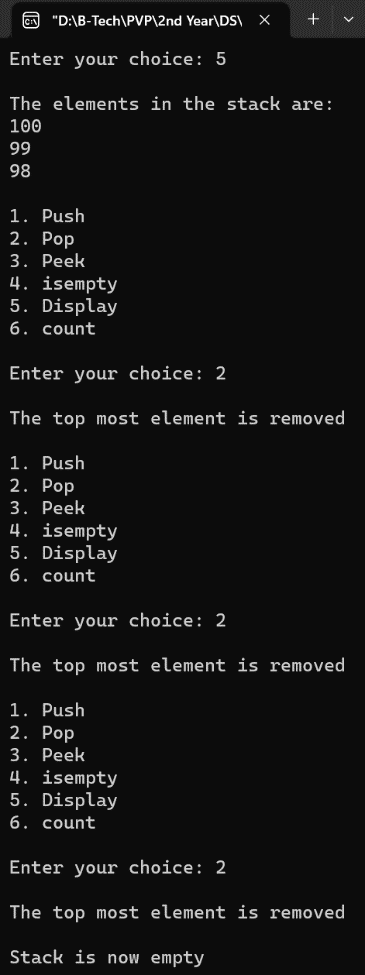
Enter your choice: 2

Enter your choice: 5

Output:0 1

**Test case-4:** Delete all the elements from the stack and check for underflow condition

Input: 100 98 99

Enter the size of stack: 4

1. Push

2. Pop

3. Peek

4. isempty

5. isfull

6. Display

7. count

Enter your choice: 5

Enter your choice: 2

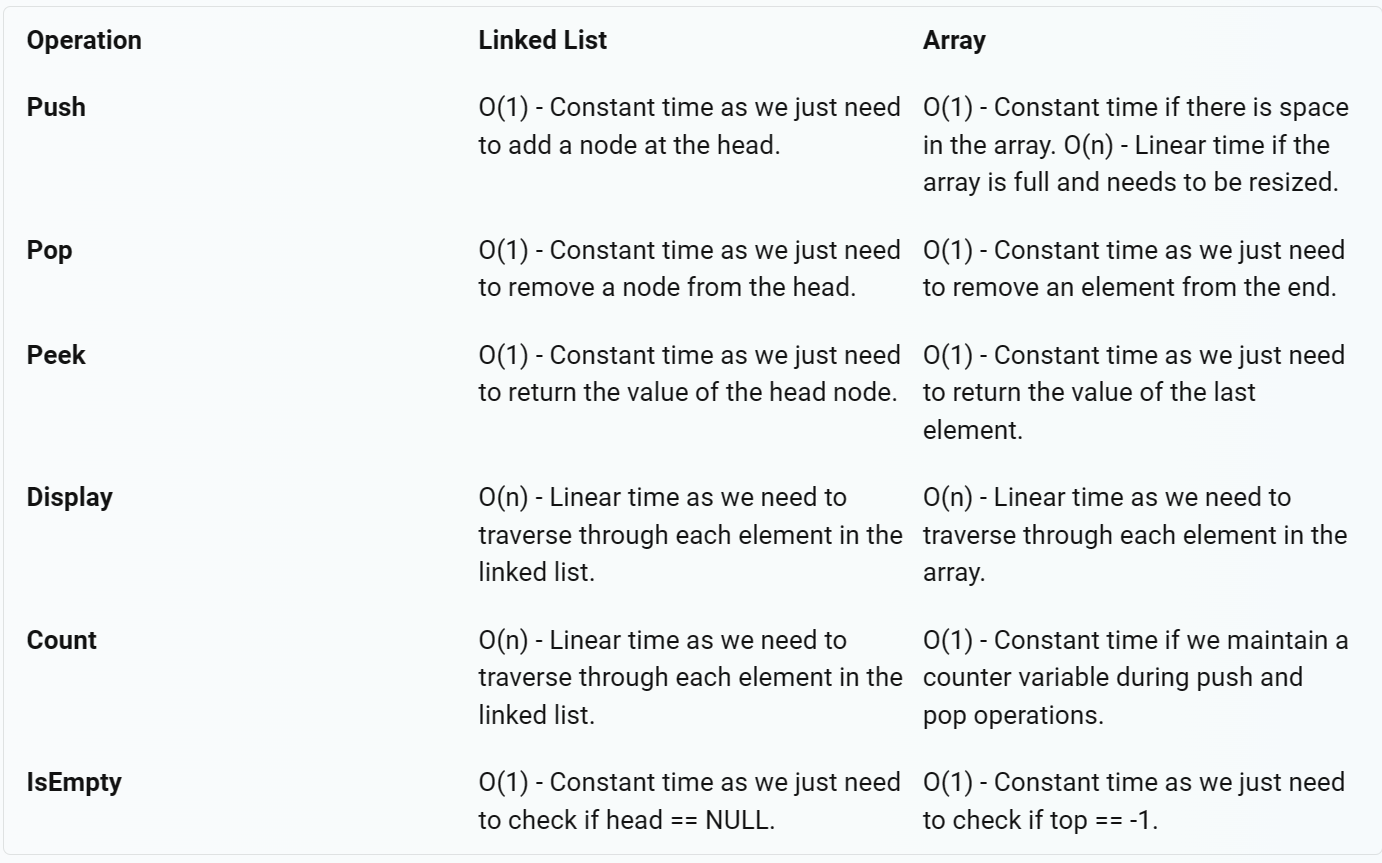
Enter your choice: 2

Enter your choice: 2

Output: The top most element is removed

Stack is now empty

**Comparison of stack operations using Linked Lists and arrays:**



**As for memory allocation:**

* Linked Lists are preferable when you don’t know how many items will be in the list. With arrays, you may need to re-declare and copy memory if the array grows too big.
* In a Linked List implementation, nodes can be added according to the user’s requirements, whereas in an Array, the size of the stack has to be predetermined.

### **Viva Questions:**

* 1. **What is a stack? Can you give some real-life examples?**
  2. A stack is a data structure that provides temporary storage of data in such a way that the element stored last will be retrieved first. This method is also called LIFO – Last In First Out. In real life, we can think of a stack as a stack of copies, stack of plates, stack of rotis etc.
  3. **What does ‘stack overflow’ refer to?**

1. Stack overflow refers to trying to add items to a full stack.
   1. **What does ‘stack underflow’ refer to?**
2. Stack underflow refers to trying to remove items from an empty stack.
   1. **What is the time complexity of pop() operation when the stack is implemented using an array?**
3. The time complexity for a pop operation in a stack implemented using an array is O(1).
   1. **Explain why stack is a recursive data structure?**
4. A stack is considered a recursive data structure because its definition is self-referential. A stack is either empty or it consists of a top and the rest, which is also a stack. So, the definition of a stack refers to a stack itself, making it recursive in nature. This recursive property of stacks makes them particularly useful in certain algorithms and problems that have recursive characteristics, such as navigating a file system’s directory structure or parsing expressions and syntax in programming languages.
   1. **Give some applications of stack.**
5. **Some of the applications of stack are,**
   * + When a program executes, stack is used to store the return address at time of function call. After the execution of the function is over, return address is popped from stack and control is returned back to the calling function.
     + Converting an infix expression to postfix operation and evaluating the postfix expression.
     + Reversing an array, converting decimal number into binary number etc.