Q1. Implement the following stack operations using arrays:

1)Push 2)Pop 3)IsFull 4)isEmpty 5)Peek 6)stackTop

Create a stack structure and stack class for the above said implementation.
Ans1:

```
#include <iostream>
#include <stack>
using namespace std;
class Stack {
public:
 int size;
 int*a;
 int top1;
 Stack(int capacity) {
 this->size=capacity;
 a=new int[size];
 top1=-1;
 }
 void push(int num) {
 if(!isFull()){
 top1++;
 a[top1]=num;
 }
 else return;
 }
 int pop() {
 if(!isEmpty()){
 int x=a[top1];
 top1--;
 return x;
 }
 else return -1;
 }
 int top() {
 if(top1==-1) return -1;
 else return a[top1];
 }
 int isEmpty() {
 if(top1==-1) return 1;
 else return 0;
 }
 int isFull() {
```

```
 if(top1==size-1) return 1;
 else return 0;
 }
 int peek() {
 if (isEmpty()) {
 cout << "Stack is empty. Cannot peek." << endl;
 return -1; // Assuming -1 represents an invalid value.
 }

 return a[top1];
 }

 int stackTop() {
 return top1;
 }
};
int main() {
 int capacity;
 cout << "Enter the capacity of the stack: ";
 cin >> capacity;
 Stack myStack(capacity);
 // Pushing elements into the stack
 for (int i = 1; i <= capacity; i++) {
 myStack.push(i);
 cout << "Pushed: " << i << endl;
 }
 // Attempting to push when the stack is full
 myStack.push(99);
 // Checking if the stack is full
 if (myStack.isFull()) {
 cout << "Stack is full!" << endl;
 }
 cout << "Peek: " << myStack.peek() << endl;
 cout << "Stack Top: " << myStack.stackTop() <<endl;
 // Popping elements from the stack
 while (!myStack.isEmpty()) {
 cout << "Popped: " << myStack.pop() << endl;
 }
 // Attempting to pop when the stack is empty
 int poppedValue = myStack.pop(); // This should return -1 as the
stack is empty
 if (poppedValue == -1) {
 cout << "Stack is empty!" << endl;
 }
 return 0;
```

```
}
```

Output:
Enter the capacity of the stack: 10
Pushed: 1
Pushed: 2
Pushed: 3
Pushed: 4
Pushed: 5
Pushed: 6
Pushed: 7
Pushed: 8
Pushed: 9
Pushed: 10
Stack is full!
Peek: 10
Stack Top: 9
Popped: 10
Popped: 9
Popped: 8
Popped: 7
Popped: 6
Popped: 5
Popped: 4
Popped: 3
Popped: 2
Popped: 1
Stack is empty!

Q2. Implement the following stack operations using linked List: 1) Push 2) Pop 3) IsFull 4) isEmpty 5) Peek Create a stack structure and stack class for the above said implementation.

```
#include <iostream>
using namespace std;
struct Node {
 int data;
 Node* next;
 Node(int value) : data(value), next(nullptr) {}
};
// Stack class using linked list
class Stack {
```

```
private:
 Node* topNode;
public:
 Stack() : topNode(nullptr) {}
 // Push operation
 void push(int value) {
 Node* newNode = new Node(value);
 newNode->next = topNode;
 topNode = newNode;
 }
 // Pop operation
 int pop() {
 if (isEmpty()) {
 cout << "Stack underflow (empty)!" << endl;
 return -1; // Return -1 for an empty stack (considering the stack
contains only non-negative values)
 }
 int poppedValue = topNode->data;
 Node* temp = topNode;
 topNode = topNode->next;
 delete temp;
 return poppedValue;
 }
 // Peek operation
 int peek() {
 if (isEmpty()) {
 cout << "Stack is empty!" << endl;
 return -1; // Return -1 for an empty stack (considering the stack
contains only non-negative values)
 }
 return topNode->data;
 }
 // Check if the stack is empty
 bool isEmpty() {
 return topNode == nullptr;
 }
};
int main() {
 Stack myStack;
 // Pushing elements into the stack
 myStack.push(50);
 myStack.push(40);
 myStack.push(30);
 myStack.push(20);
 myStack.push(10);
```

```
 // Peeking at the top element
 cout << "Top element: " << myStack.peek() << endl;
 // Popping elements from the stack
 cout << "Popped: " << myStack.pop() << endl;
 cout << "Popped: " << myStack.pop() << endl;
 cout << "Popped: " << myStack.pop() << endl;
 cout << "Popped: " << myStack.pop() << endl;
 cout << "Popped: " << myStack.pop() << endl;
 // Checking if the stack is empty
 cout << "Is stack empty? " << (myStack.isEmpty() ? "Yes" : "No") <<
endl;
 // Popping from an empty stack
 cout << "Popped: " << myStack.pop() << endl;
 return 0;
}
```

Output:
1.
Top element: 10
Popped: 10
Popped: 20
Popped: 30
Popped: 40
Is stack empty? No
Popped: 50
2.
Top element: 10
Popped: 10
Popped: 20
Popped: 30
Popped: 40
Popped: 50
Is stack empty? Yes
Popped: Stack underflow (empty)!
-1

Q3: Write a function that accepts an array of integers A having a sequence of operations to be performedon stack along with values to be pushed into the stack, and size of the array N. The function should perform the given operations on a stack and return a pointer

on stack. The array element contains operation codes and values to be inserted in the stack.
PUSH (Code 1): Push the next value in array into the stack. If stack isfull, print Stack is full.
POP (Code 2): Removing an element from the stack. If stack is empty, print Stack is empty. Return (Code 3): Return. Input: 8 1 10 1 20 1 30 2 3 Where,First line represents the size of an array. Second line represents the elements of the array.

```
#include <iostream>
using namespace std;
template <typename T>
class Stack {
private:
 template <typename U>
 struct Node {
 U data;
 Node<U>* next;
 Node(U data) : data(data), next(nullptr) {}
 };
 Node<T>* head;
public:
 Stack() : head(nullptr) {}
 void push(T data) {
 Node<T>* newNode = new Node<T>(data);
 newNode->next = head;
 head = newNode;
 }
 T pop() {
 if (isEmpty()) {
 throw std::runtime_error("Stack is empty.");
 }
 T poppedElement = head->data;
 Node<T>* temp = head;
 head = head->next;
 delete temp;
 return poppedElement;
 }
 bool isEmpty() const {
 return head == nullptr;
 }
 ~Stack() {
 while (head != nullptr) {
 Node<T>* temp = head;
 head = head->next;
```

```
 delete temp;
 }
 }
};
template <typename T>
Stack<T> performOperations(const int operations[], int size) {
 Stack<T> resultStack;
 for (int i = 0; i < size; ++i) {
 int operationCode = operations[i];
 switch (operationCode) {
 case 1:
 if (i + 1 < size) {
 resultStack.push(static_cast<T>(operations[++i]));
 } else {
 cout << "Error: Incomplete PUSH operation." << endl;
 }
 break;
 case 2:
 if (!resultStack.isEmpty()) {
 resultStack.pop();
 } else {
 cout << "Stack is empty." << endl;
 }
 break;
 case 3:
 return resultStack;
 default:
 cout << "Error: Unknown operation code " << operationCode << "." <<
endl;
 break;
 }
 }
 return resultStack;
}
int main() {
 //input that provides
 int inputArray[] = {1, 10, 1, 20, 1, 30, 2, 3};
 Stack<int> resultStack = performOperations<int>(inputArray,
sizeof(inputArray) / sizeof(inputArray[0]));
 cout << "Final Stack: ";
 while (!resultStack.isEmpty()) {
 cout << resultStack.pop() << " ";
 }
 return 0;
}
```

Output:
Final Stack: 20 10