4. Stack Data Structure

What is Stack?

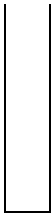
Stack is a data structure which is used to handle data in a last-in-first-out (LIFO) method. That is we can remove the most recently added element from the stack first.

Common operations of Stack are:

initializeStack() – initializes the stack as empty stack.
push()- adds an element on top of the stack.
pop()-removes and returns the top most element from the stack.
topElt()-returns the top element without removing it.
isEmpty() - returns true if the stack has no elements and false otherwise.
isFull() - returns true if the stack is full of elements and false otherwise.
displayStack() - displays all elements from top to bottom.

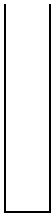
Graphical Representation of Stack Operation:

1. initializeStack()

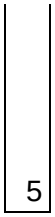


2. p =isEmpty()

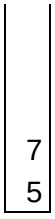
p = true



3. push(5)



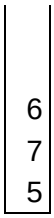
4. push(7)



5. push(6)

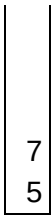
6. q = isEmpty(); r = isFull();

q = false; r = false



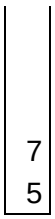
7. x = pop()

x = 6



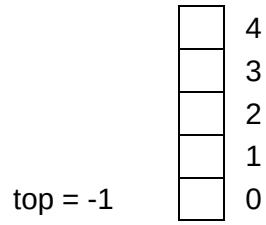
8. y = topElt()

y = 7



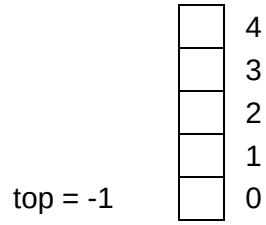
Static (Array based) Implementation of Stack Operations [Graphical Representation]:

1. initializeStack()

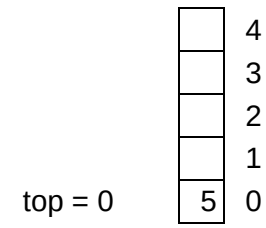


2. p =isEmpty()

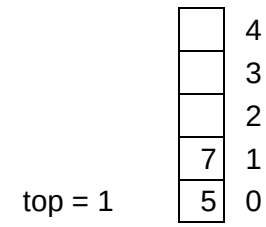
p = true



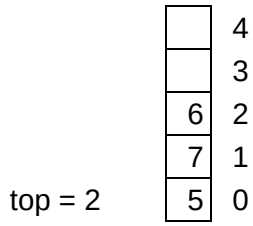
3. push(5)



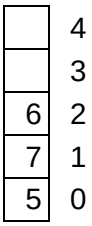
4. push(7)



5. push(6)

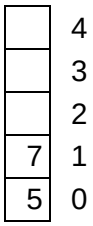


6. q = isEmpty(); r = isFull();



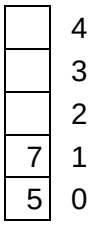
q = false; r = false top = 2

7. x = pop()



x = 6 top = 1

8. y = topElt()



y = 7 top = 1

Static (Array based) Implementation of Stack Operations [C++ Code]:

```
#include<iostream.h>
#include<conio.h>

const STK_SIZE=5;

class Stack
{
private:
   int top;
   int stk[STK_SIZE];

public:
   Stack();
   void initializeStack();
   void push(int);
   int pop();
   int topElt();
   int isEmpty();
   int isFull();
   void displayStack();
}
```

```
Stack::Stack()
{
  top=(-1);
}

void Stack::initializeStack()
{
  top=(-1);
}

void Stack::push(int elt)
{
  if (top < STK_SIZE-1) stk[++top]=elt;
}

int Stack::pop()
{
  if (top > -1)
    return stk[top--];
  else
    return 999; //Some invalid integer should be returned
}

int Stack::topElt()
{
  if (top > -1)
    return stk[top];
  else
    return 999; //Some invalid integer should be returned
}

int Stack::isEmpty()
{
  return (top == (-1));
}

int Stack::isFull()
{
  return (top == (STK_SIZE-1));
}

void Stack::displayStack()
{
  int i=top;
  while (i>-1)
  {
    cout<<stk[i]<<endl;
    i--;
  }
}

void main()
{
  clrscr();
  Stack s;
  s.push(5);
  s.push(7);
  s.push(6);
```

```
   int x=s.pop();
   s.push(9);
   s.displayStack();
}
```

Output:
9
7
5

Dynamic (Linked List based) Implementation of Stack Operations:
initializeStack() - top=NULL; //Similar to initialzeList() and it is better to use top instead of head.
push() - newNode->next=top; top=newNode; //Similar to insertAtFront()
pop() - x=top->data; top=top->next; return x;
topElt() - return top->data
isEmpty() - if (top==NULL) return 1 else return 0
isFull() - return 0; //Always return false
displayStack() - Similar to displayList()

Advantages of Stack:
Last-in-first-out access

Disadvantages of Stack:
Difficult to access other items