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.

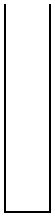
Undo sequence in a text editor and the chain of method calls in a programming language are examples for applications of stack.

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.

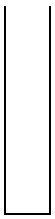
Graphical Representation of <u>Stack Operation</u>:

1. initializeStack()

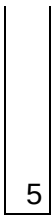


2. p =isEmpty()

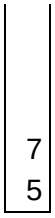
p = true



3. push(5)

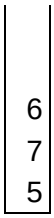


4. push(7)



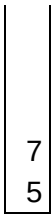
5. push(6)

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



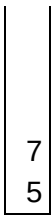
q = false; r = false

7. x = pop()



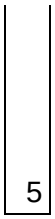
x = 6

8. y = topElt()



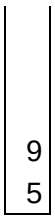
y = 7

9. z = pop()



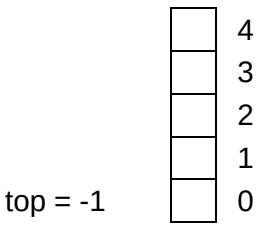
z = 7

10. push(9)

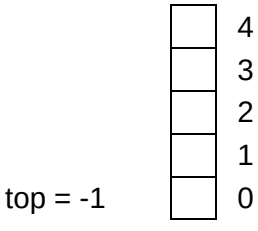


Static (Array based) Implementation of Stack Operations [Graphical Representation]:

1. initializeStack()

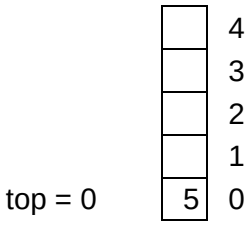


2. p =isEmpty()

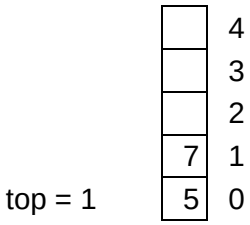


p = true

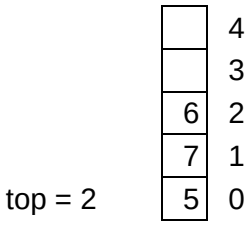
3. push(5)



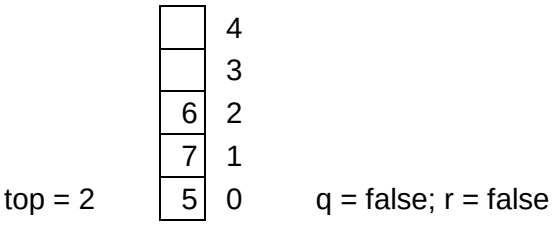
4. push(7)



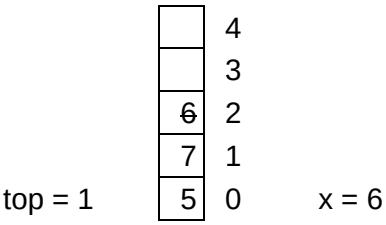
5. push(6)



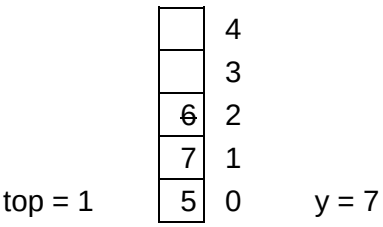
6. q = isEmpty(); r = isFull();



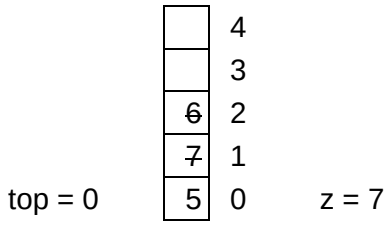
7. x = pop()



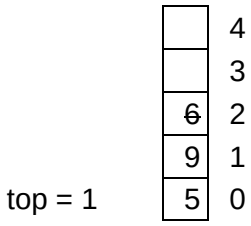
8. y = topElt()



9. z = pop()



10. push(9)



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.initializeStack();
  int p=s.isEmpty();
  s.push(5);
  s.push(7);
  s.push(6);
  int x=s.pop();
  int y=s.topElt();
  int z=s.pop();
  s.push(9);
  cout<<"p="<<p<<"\t"<<"x="<<x<<"\t" <<"y="<<y<<"\t" <<"z="<<z<<"\n";
  cout<<"Current stack elements:"<<endl;
  s.displayStack();
}
```

Output:
p=1 x=6 y=7 z=7
Current stack elements:
9
5

Dynamic (Linked List based) Implementation of Stack Operations:

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

struct node
{
  int data;
  node *next;
};

class Stack
{
private:
  node *top;
public:
  Stack();
  void initializeStack();
  void push(int);
  int pop();
  int topElt();
  int isEmpty();
  int isFull();
  void displayStack();
};

Stack::Stack()
{
  top=NULL;
}

void Stack::initializeStack()
{
  top=NULL;
}
void Stack::push(int elt)
{
  node *newNode;
  newNode=new node;
  newNode->data = elt;
  newNode->next=top;
  top=newNode;
}

int Stack::pop()
{
  if(top != NULL)
  {
    node *temp=top;
    top=top->next;
    return temp->data;
  }
  else
    return 999;
}

int Stack::topElt()
{
```

```
  if(top != NULL)
    return top->data;
  else
    return 999;
}

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

int Stack::isFull()
{
  return 0; //Always false.
}

void Stack::displayStack()
{
  node *temp=top;
  while(temp != NULL)
  {
   cout<<temp->data<<"\t";
   temp=temp->next;
  }
  cout<<endl;
}

void main()
{
  Stack stk; char ch; int n;

  clrscr();
  do
  {
    cout<<"Stack options:\nP for Push\nO for Pop\nD for Display\nQ for Quit\n";
    cout<<"Enter your choice: "; cin>>ch;
    switch(ch)
    {
      case 'P':
        cout<<"Enter a number to add: "; cin>>n;
        stk.push(n);
        break;
      case 'O':
        cout<<"The number "<<stk.pop()<<" is removed from the stack.";
        break;
      case 'D':
        cout<<"The stack elements are ";
         stk.displayStack();
        break;
      }
    } while(ch!='Q');
}
```

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

Disadvantages of Stack:
Difficult to access other items