Aryan Nanda **221070003**

Experiment 3 - Stack using 2 Queues

AIM : In this experiment we have used 2 queues to implement stack functionalities like push(), pop(), peek(), isEmpty(), isFull() and peek()

THEORY :
Stack : Stack is a linear data structure which follows the LIFO (Last in First Out) approach. This means that the element that was inserted last will be removed first. As a result only the top element of the stack is accessible at all times provided that the stack is not empty.
Queue : Queue is a linear data structure which follows the FIFO (First in First out) approach. Here the element inserted first will be removed first. In Queues, the element at the front is accessible to us.

Implementation of Stack using 2 Queues :
Since stacks follow the LIFO approach and queues follow the FIFO approach while implementing a stack using 2 queues at least 1 operation (either push() or pop()) will require O(n) time.

1. If push() operation takes O(n) time, this means that the first element of the queue is treated as the top of the stack.
   - Here we can directly use pop() operation in O(1) time since the front element of the queue is accessible to us.
   - For push() operation we will have to make use of a temporary queue where all the elements of the original queue (given that the original queue is not empty) have to be pushed in order. Once the temporary queue has all the elements of the original queue in order and the original queue is empty we push the new element in the original queue. After this, we push all the elements of the temporary queue back again to the original

queue in order. This ensures that the new element is the front element of the queue (which is treated as top of the stack).

2.On the other hand, if the pop() operation takes O(n) time, this means that the last element of the queue is treated as the top of the stack.

- Here we can directly use push() operation in O(1) time to push the new element at the back of the queue.
- For pop() operation we will have to make use of a temporary queue where all the elements of the original queue have to be pushed in order till only one element remains in the original queue (i.e the last element). Now the last element which is in the front of the original queue is popped .After this, we push all the elements of the temporary queue back again to the original queue in order. This ensures that the last element in the queue (which is the top of the stack) is popped.

ALGORITHM:

```
Algorithm: Implementing Stack using Queues
Class: queueUsingArray
    Constructor: queueUsingArray()
        ● Initialize a queue using an array with capacity 5.
        ● Set nextindex to 0, firstindex to -1, size to 0, and capacity to 5.
    Function: getSize()
        ● Returns the current size of the queue.
    Function: isEmpty()
        ● Returns true if the queue is empty, otherwise returns false.
    Function: enqueue(element)
        ● If the queue is at full capacity (size equals capacity), create a new array with
          double the capacity, copy existing elements, and update queue properties
          accordingly.
        ● Add the element at the nextindex position, considering circular behavior by using
          % capacity.
        ● Increment size.
        ● Adjust firstindex if initially -1.
    Function: dequeue()
        ● If the queue is empty, display "Queue is empty" and return 0.
```

- Retrieve the front element (data[firstindex]).
- Update firstindex and size.
- If size becomes 0, reset firstindex and nextindex.

Function: front()

- If the queue is empty, display "Queue is empty" and return 0.
- Return the element at firstindex.

Class: Stack

Member Variables:

- N: Integer to track the size of the stack.
- q1: Instance of the queue using an array.
- q2: Another instance of the queue using an array.

Constructor: Stack()

- Initialize N to 0.

Function: push(element)

- Enqueue element into q2.
- Increment N.
- Transfer elements from q1 to q2 and rearrange the queues to maintain stack properties.
- Swap q1 and q2.

Function: pop()

- Dequeue an element from q1.
- Decrement N.

Function: top()

- Return the front element of q1.

Function: size()

- Return the current size of the stack (N).

Main Function:

Create a Stack object s.

Push elements 1 to 10 onto the stack.

Display the size of the stack.

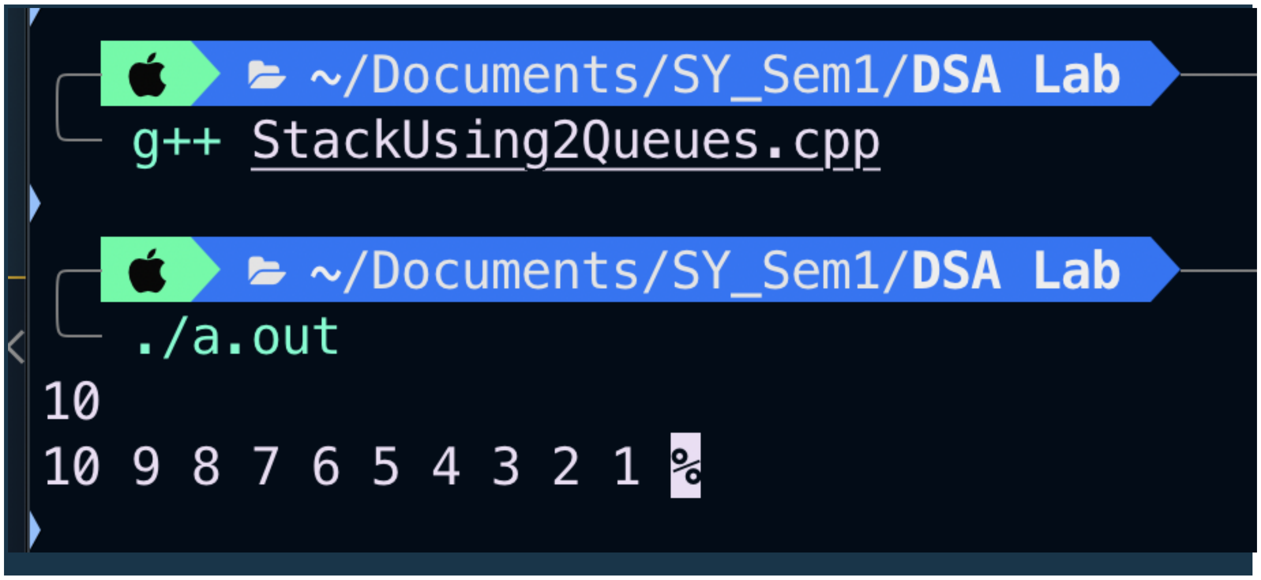
Pop all elements from the stack and display them.

CODE:

```
#include<iostream>
using namespace std;

class queueusingarray{
   int *data;
   int nextindex;
   int firstindex;
   int size;
   int capacity;
   public:
   queueusingarray(){
       data=new int[5];
       nextindex=0;
       firstindex=-1;
       size=0;
       capacity=5;
   }
   int getsize(){
       return size;
   }
   bool isempty(){
       return size==0;
   }
   void enqueue(int element){
       if(size==capacity){
           int* newdata=new int[2*capacity];
           int j=0;
           for(int i=firstindex;i<capacity;i++,j++){
               newdata[j]=data[i];
            }
           for(int i=0;i<firstindex;i++,j++){
               newdata[j]=data[i];
           }
           delete [] data;
           data=newdata;
           firstindex=0;
           nextindex=capacity;
           capacity=2*capacity;
       }
       data[nextindex]=element;
       nextindex=(nextindex+1)%capacity;
       size++;
       if(firstindex==-1){
           firstindex=0;
       }
   }
   int dequeue(){
       if(isempty()){
           cout<<"Queue is empty";
           return 0;
       }
       int ans=data[firstindex];
       firstindex=(firstindex+1)%capacity;
       size--;
       if(size==0){
           firstindex=-1;
           nextindex=0;
       }
       return ans;
   }
   int front(){
        if(isempty()){
            cout<<" Queue is empty";
           return 0;
       }
       return data[firstindex];
   }
};
class Stack{
   int N;
   queueusingarray q1;
   queueusingarray q2;
   public:
   Stack(){
       N = 0;
   }
   void push(int element){
       q2.enqueue(element);
       N++;
       while(!q1.isempty()){
           q2.enqueue(q1.front());
           q1.dequeue();
       }
       queueusingarray temp = q2;
       q2 = q1;
       q1 = temp;
   }
   void pop(){
       q1.dequeue();
       N--;
   }
   int top(){
       return q1.front();
   }
   int size(){
       return N;
   }
};
int main(){
   Stack s;
   s.push(1);
   s.push(2);
   s.push(3);
   s.push(4);
   s.push(5);
   s.push(6);
   s.push(7);
   s.push(8);
   s.push(9);
   s.push(10);
   cout<<s.size()<<endl;
   int size = s.size();
   for(int i=0;i<size;i++){
       cout<<s.top()<<" ";
       s.pop();
   }
}
```

EXAMPLE:



```
 ~/Documents/SY_Sem1/DSA Lab
g++ StackUsing2Queues.cpp

 ~/Documents/SY_Sem1/DSA Lab
./a.out
10
10 9 8 7 6 5 4 3 2 1 %
```

CONCLUSION : Thus from this experiment we learnt about the concept of stacks and queues and were able to implement stack using 2 queues.