DATA STRUCTURES LABORATORY MANUAL – ICE 2144

III SEMESTER B. TECH

EXPT. 8 STACKS AND QUEUES

STACKS:

➔ Operations on a Stack

- The following operations are performed on the stack...
  1. Push (To insert an element on to the stack)
  2. Pop (To delete an element from the stack)
  3. Display (To display elements of the stack)

→ Stack data structure can be implemented in two ways.

1. Using Arrays
2. Using Linked List

1) Write a program that implement stack (its operations) using (i)Arrays (ii)Linked list (Pointers).

(i) Program:

```
#include<iostream>
#include<cstdlib>
using namespace std;
#define SIZE 10
    void push(int);
    void pop();
    void display();
    int stack[SIZE], top = -1;
    int main()
    {
            int value, choice;
            while(1)
            {
                    cout<<"\n\n***** MENU *****\n";
                    cout<<"1. Push\n2. Pop\n3. Display\n4. Exit";
                    cout<<"\nEnter your choice: ";
                    cin>>choice;
                    switch(choice)
                    {
                            case 1:
                            cout<<"Enter the value to be insert: ";
                            cin>>value;
                            push(value);
                            break;
                            case 2: pop();
                            break;
                            case 3: display();
                            break;
                            case 4: exit(0);
                            default: cout<<"\nWrong selection!!! Try again!!!";
                    }
    }
}
void push(int value)
{
    if(top == SIZE-1)
        cout<<"\nStack is Full!!! Insertion is not possible!!!";
    else
     {
        top++;
        stack[top] = value;
        cout<<"\nInsertion success!!!";
     }
}
void pop()
{
    if(top == -1)
        cout<<"\nStack is Empty!!! Deletion is not possible!!!";
    else
    {
        cout<<"\nDeleted :"<<stack[top];
        top--;
    }
}
void display()
{
    if(top == -1)
        cout<<"\nStack is Empty!!!";
    else
    {
        int i;
        cout<<"\nStack elements are:\n";
        for(i=top; i>=0; i--)
            cout<<stack[i];
    }
}
```

(ii) Program:

```
#include<iostream>
#include<cstdlib>
using namespace std;
class Node
{
   public:
        int data;
        Node *next;
}
```

```
  *top = NULL;
  void push(int);
  void pop();
  void display();
  int main()
  {
      int choice, value;
      cout<<"\n:: Stack using Linked List ::\n";
      while(1)
      {
            cout<<"\n****** MENU ******\n";
            cout<<"1. Push\n2. Pop\n3. Display\n4. Exit\n";
            cout<<"Enter your choice: ";
            cin>>choice;
            switch(choice)
            {
                  case 1:
                  cout<<"Enter the value to be insert: ";
                  cin>>value;
                  push(value);
                  break;
                  case 2: pop();
                  break;
                  case 3: display(); break;
                  case 4: exit(0);
                  default: cout<<"\nWrong selection!!! Please try again!!!\n";
            }
      }
}
      void push(int value)
      {
            Node *newNode;
            newNode = new Node();
            newNode→data = value;
            if(top == NULL)
            newNode→next = NULL;
            else
            newNode→next = top;
            top = newNode;
            cout<<"\nInsertion is Success!!!\n";
      }
      void pop()
      {
            if(top == NULL)
                  cout<<"\nStack is Empty!!!\n";
            else
```

```
        {
            Node *temp = top;
            cout<<"\nDeleted element:", temp→data;
            top = temp→next;
            delete(temp);
        }
    }
    void display()
    {
        if(top == NULL)
            cout<<"\nStack is Empty!!!\n";
        else
        {
            Node *temp = top;
            while(temp→next != NULL)
            {
                cout<<temp→data;
                temp = temp→next;
            }
            cout<<temp→data;
        }
}
```

Queue:

2) Write a program that implement stack (its operations) using (i)Arrays (ii)Linked list (Pointers).

(i) Arrays:

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

// A structure to represent a queue
class Queue
{
    public:
        int front, rear, size;
        unsigned capacity;
        int* array;
};

// function to create a queue of given capacity. It initializes size of queue as 0
Queue *createQueue(unsigned capacity)
{
    Queue* queue = new Queue();
    Queue→capacity = capacity;
    Queue→front = queue→size = 0;
    // enqueue
    Queue→rear = capacity - 1;
    Queue→array = new int[queue→capacity];
    return queue;
}

// Queue is full when size becomes equal to the capacity
int isFull(Queue* queue)
{
    return (queue→size == queue→capacity);
}

// Queue is empty when size is 0
int isEmpty(Queue* queue)
{
    return (queue→size == 0);
}

// Function to add an item to the queue. It changes rear and size
void enqueue(Queue* queue, int item)
{
    if (isFull(queue))
        return;
    queue→rear = (queue→rear + 1)
                        % queue→capacity;
    Queue→array[queue→rear] = item;
    Queue→size = queue→size + 1;
    cout << item << " enqueued to queue\n";
}

// Function to remove an item from queue. It changes front and size
int dequeue(Queue* queue)
{
    if (isEmpty(queue))
        return 0;
    int item = queue→array[queue→front];
    queue→front = (queue→front + 1)
                        % queue->capacity;
    queue→size = queue→size - 1;
    return item;
}

// Function to get front of queue
int front(Queue* queue)
{
	if (isEmpty(queue))
		return 0;
	return queue→array[queue→front];
}

// Function to get rear of queue
int rear(Queue* queue)
{
	if (isEmpty(queue))
		return 0;
	return queue→array[queue→rear];
}

int main()
{
	Queue* queue = createQueue(1000);

	enqueue(queue, 10);
	enqueue(queue, 20);
	enqueue(queue, 30);
	enqueue(queue, 40);

	cout << dequeue(queue)
		<< " dequeued from queue\n";

	cout << "Front item is "
		<< front(queue) << endl;
	cout << "Rear item is "
		<< rear(queue) << endl;

	return 0;
}
```

(ii) Linked list:

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

class QNode
{
	public:
		int data;
		QNode* next;
		QNode(int d)
		{
			data = d;
```

```
            next = NULL;
        }
};

class Queue
{
    public:
        QNode *front, *rear;
        Queue()
        {
            front = rear = NULL;
        }

    void enQueue(int x)
    {

        // Create a new LL node
        QNode* temp = new QNode(x);

        // If queue is empty, then new node is front and rear both
        if (rear == NULL)
        {
            front = rear = temp;
            return;
        }

        // Add the new node at the end of queue and change rear
        rear->next = temp;
        rear = temp;
    }

    // Function to remove a key from given queue q
    void deQueue()
    {
        // If queue is empty, return NULL.
        if (front == NULL)
            return;

        // Store previous front and move front one node ahead
        QNode* temp = front;
        front = front->next;

        // If front becomes NULL, then change rear also as NULL
        if (front == NULL)
            rear = NULL;
            delete (temp);
        }
};

int main()
{

        Queue q;
        q.enQueue(10);
        q.enQueue(20);
        q.deQueue();
        q.deQueue();
        q.enQueue(30);
        q.enQueue(40);
        q.enQueue(50);
        q.deQueue();
        cout << "Queue Front : " << ((q.front != NULL) ? (q.front)->data : -1)<< endl;
        cout << "Queue Rear : " << ((q.rear != NULL) ? (q.rear)->data : -1);
}
```

Exercise:

1) Write a program to implement stack using queue data structure.
2) Write a program to implement queue using stack data structure.
3) Write a program to implement both a) insertion at the front and b) deletion at the rear for a deque (double ended queue) data structure.