Name – Bhakti Bapurao Patil

Reg_No – 2020BIT064

Practical No – 2

1 Queue using linked list

```
#include <iostream>
using namespace std;
struct node {
  int data;
  struct node *next;
};
struct node* front = NULL;
struct node* rear = NULL;
struct node* temp;
void Insert() {
  int val;
  cout<<"Insert the element in queue : "<<endl;
  cin>>val;
  if (rear == NULL) {
    rear = (struct node *)malloc(sizeof(struct node));
```

```
    rear->next = NULL;
    rear->data = val;
    front = rear;
  } else {
    temp=(struct node *)malloc(sizeof(struct node));
    rear->next = temp;
    temp->data = val;
    temp->next = NULL;
    rear = temp;
  }
}
void Delete() {
  temp = front;
  if (front == NULL) {
    cout<<"Underflow"<<endl;
    return;
  }
  else
  if (temp->next != NULL) {
    temp = temp->next;
    cout<<"Element deleted from queue is : "<<front->data<<endl;
    free(front);
    front = temp;
  } else {
    cout<<"Element deleted from queue is : "<<front->data<<endl;
```

```
    free(front);
    front = NULL;
    rear = NULL;
  }
}
void Display() {
  temp = front;
  if ((front == NULL) && (rear == NULL)) {
    cout<<"Queue is empty"<<endl;
    return;
  }
  cout<<"Queue elements are: ";
  while (temp != NULL) {
    cout<<temp->data<<" ";
    temp = temp->next;
  }
  cout<<endl;
}
int main() {
  int ch;
  cout<<"1) Insert element to queue"<<endl;
  cout<<"2) Delete element from queue"<<endl;
  cout<<"3) Display all the elements of queue"<<endl;
  cout<<"4) Exit"<<endl;
  do {
```

```
    cout<<"Enter your choice : "<<endl;
    cin>>ch;
    switch (ch) {
      case 1: Insert();
      break;
      case 2: Delete();
      break;
      case 3: Display();
      break;
      case 4: cout<<"Exit"<<endl;
      break;
      default: cout<<"Invalid choice"<<endl;
    }
  } while(ch!=4);
  return 0;
}
```

```
/tmp/Nm1bitD9I4.o
1) Insert element to queue
2) Delete element from queue
3) Display all the elements of queue
4) Exit
Enter your choice :
2
Underflow
Enter your choice :
3
Queue is empty
Enter your choice :
4
Exit
```

2 Stack using linked list

```
// C++ program to Implement a stack
// using singly linked list
#include <bits/stdc++.h>
using namespace std;

// creating a linked list;
class Node {
public:
        int data;
        Node* link;

        // Constructor
```

```
	Node(int n)
	{
		this->data = n;
		this->link = NULL;
	}
};

class Stack {
	Node* top;

public:
	Stack() { top = NULL; }

	void push(int data)
	{

		// Create new node temp and allocate memory in
heap
		Node* temp = new Node(data);

		// Check if stack (heap) is full.
		// Then inserting an element would
		// lead to stack overflow
		if (!temp) {
			cout << "\nStack Overflow";
```

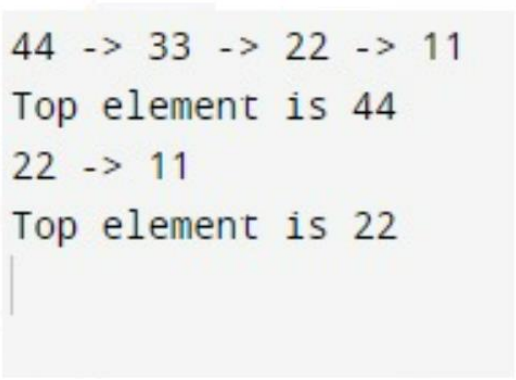
```
                exit(1);
        }

        // Initialize data into temp data field
        temp->data = data;

        // Put top pointer reference into temp link
        temp->link = top;

        // Make temp as top of Stack
        top = temp;
}

// Utility function to check if
// the stack is empty or not
bool isEmpty()
{
        // If top is NULL it means that
        // there are no elements are in stack
        return top == NULL;
}

// Utility function to return top element in a stack
int peek()
{
```

```
	// If stack is not empty , return the top element
	if (!isEmpty())
		return top->data;
	else
		exit(1);
}

// Function to remove
// a key from given queue q
void pop()
{
	Node* temp;

	// Check for stack underflow
	if (top == NULL) {
		cout << "\nStack Underflow" << endl;
		exit(1);
	}
	else {

		// Assign top to temp
		temp = top;

		// Assign second node to top
		top = top->link;
```

```
                        // This will automatically destroy
                        // the link between first node and second
node

                        // Release memory of top node
                        // i.e delete the node
                        free(temp);
                }
        }

        // Function to print all the
        // elements of the stack
        void display()
        {
                Node* temp;

                // Check for stack underflow
                if (top == NULL) {
                        cout << "\nStack Underflow";
                        exit(1);
                }
                else {
                        temp = top;
                        while (temp != NULL) {
```

```
                                        // Print node data
                                        cout << temp->data;

                                        // Assign temp link to temp
                                        temp = temp->link;
                                        if (temp != NULL)
                                                cout << " -> ";
                                }
                        }
                }
};

// Driven Program
int main()
{
        // Creating a stack
        Stack s;

        // Push the elements of stack
        s.push(11);
        s.push(22);
        s.push(33);
        s.push(44);
```

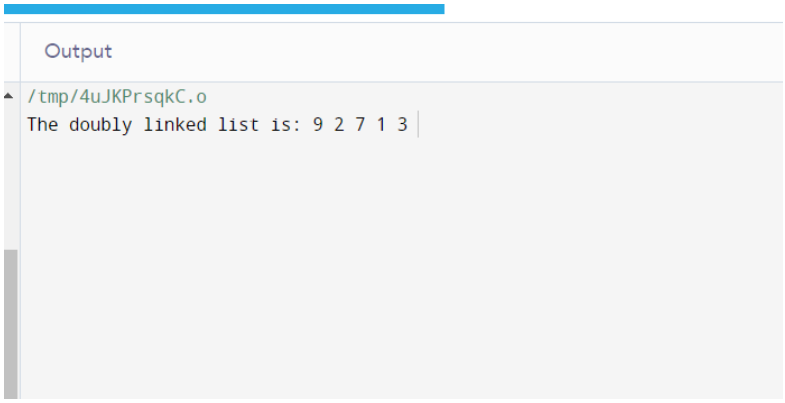
```
        // Display stack elements
        s.display();

        // Print top element of stack
        cout << "\nTop element is " << s.peek() << endl;

        // Delete top elements of stack
        s.pop();
        s.pop();

        // Display stack elements
        s.display();

        // Print top element of stack
        cout << "\nTop element is " << s.peek() << endl;

        return 0;
}
```



3) Doubly linked list

```
#include <iostream>
using namespace std;
struct Node {
  int data;
  struct Node *prev;
  struct Node *next;
};
struct Node* head = NULL;
void insert(int newdata) {
  struct Node* newnode = (struct Node*) malloc(sizeof(struct
Node));
  newnode->data = newdata;
  newnode->prev = NULL;
  newnode->next = head;
```

```
  if(head != NULL)
  head->prev = newnode ;
  head = newnode;
}
void display() {
  struct Node* ptr;
  ptr = head;
  while(ptr != NULL) {
    cout<< ptr->data <<" ";
    ptr = ptr->next;
  }
}
int main() {
  insert(3);
  insert(1);
  insert(7);
  insert(2);
  insert(9);
  cout<<"The doubly linked list is: ";
  display();
  return 0;
}
```



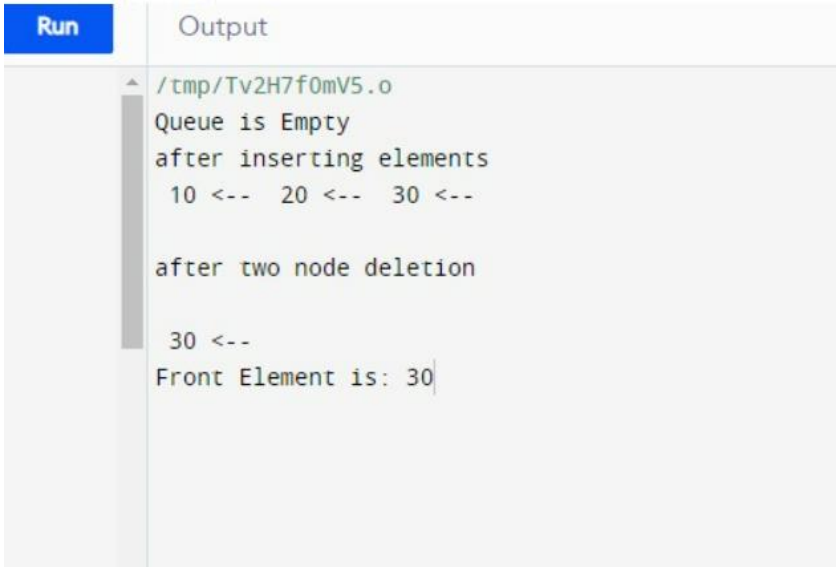
4) Enqueue

```
#include <bits/stdc++.h>
using namespace std;
struct Q {
  int f, r, capacity;
  int* q;
  Q(int c) {
    f = r= 0;
    capacity = c;
    q = new int;
  }
  ~Q() { delete[] q; }
  void Enqueue(int d) {
    if (capacity == r) {
```

```
    printf("\nQueue is full\n");
    return;
  } else {
    q[r] = d;
    r++;
  }
  return;
}
void Dequeue() {
  if (f == r) {
    printf("\nQueue is empty\n");
    return;
  } else {
    for (int i = 0; i < r - 1; i++) {
      q[i] = q[i + 1];
    }
    r--; //update rear
  }
  return;
}
void Display()  {
  int i;
  if (f == r) {
    printf("\nQueue is Empty\n");
    return;
```

```
    }
    for (i = f; i < r; i++) {
      printf(" %d <-- ", q[i]);
    }
    return;
  }
  void Front() {
    if (f == r) {
      printf("\nQueue is Empty\n");
      return;
    }
    printf("\nFront Element is: %d", q[f]);
    return;
  }
};
int main(void) {
  Q qu(3);
  qu.Display();
  cout<<"after inserting elements"<<endl;
  qu.Enqueue(10);
  qu.Enqueue(20);
  qu.Enqueue(30);
  qu.Display();
  qu.Dequeue();
  qu.Dequeue();
```

```
  printf("\n\nafter two node deletion\n\n");
  qu.Display();
  qu.Front();
  return 0;
}
```



5 Dequeue

```
#include <iostream>
using namespace std;
#define MAX 100
class Deque {
        int arr[MAX];
        int front;
```

```
        int rear;
        int size;
public:
        Deque(int size)
        {
                front = -1;
                rear = 0;
                this->size = size;
        }
        void insertfront(int key);
        void insertrear(int key);
        void deletefront();
        void deleterear();
        bool isFull();
        bool isEmpty();
        int getFront();
        int getRear();
};
bool Deque::isFull()
{
        return ((front == 0 && rear == size - 1)
                        || front == rear + 1);
}
bool Deque::isEmpty() { return (front == -1); }
void Deque::insertfront(int key)
```

```
{
        if (isFull()) {
                cout << "Overflow\n" << endl;
                return;
        }
        if (front == -1) {
                front = 0;
                rear = 0;
        }
        else if (front == 0)
                front = size - 1;
        else
                front = front - 1;
        arr[front] = key;
}
void Deque ::insertrear(int key)
{
        if (isFull()) {
                cout << " Overflow\n " << endl;
                return;
        }
        if (front == -1) {
                front = 0;
                rear = 0;
        }
```

```
        else if (rear == size - 1)
                rear = 0;
        else
                rear = rear + 1;
        arr[rear] = key;
}
void Deque ::deletefront()
{
        if (isEmpty()) {
                cout << "Queue Underflow\n" << endl;
                return;
        }
        if (front == rear) {
                front = -1;
                rear = -1;
        }
        else
                if (front == size - 1)
                front = 0;

        else
                front = front + 1;
}
void Deque::deleterear()
{
```

```
        if (isEmpty()) {
                cout << " Underflow\n" << endl;
                return;
        }
        if (front == rear) {
                front = -1;
                rear = -1;
        }
        else if (rear == 0)
                rear = size - 1;
        else
                rear = rear - 1;
}
int Deque::getFront()
{
        if (isEmpty()) {
                cout << " Underflow\n" << endl;
                return -1;
        }
        return arr[front];
}
int Deque::getRear()
{
        if (isEmpty() || rear < 0) {
                cout << " Underflow\n" << endl;
```

```
            return -1;
        }
        return arr[rear];
}
int main()
{
        Deque dq(5);
        cout << "Insert element at rear end : 5 \n";
        dq.insertrear(5);

        cout << "insert element at rear end : 10 \n";
        dq.insertrear(10);

        cout << "get rear element "
            << " " << dq.getRear() << endl;

        dq.deleterear();
        cout << "After delete rear element new rear"
            << " become " << dq.getRear() << endl;

        cout << "inserting element at front end \n";
        dq.insertfront(15);

        cout << "get front element "
            << " " << dq.getFront() << endl;
```

```
        dq.deletefront();


        cout << "After delete front element new "
             << "front become " << dq.getFront() << endl;
        return 0;
}
```

```
/tmp/4NYESgualv.o
Insert element at rear end : 5
insert element at rear end : 10
get rear element  10
After delete rear element new rear become 5
inserting element at front end
get front element  15
After delete front element new front become 5
```