DAA Prac 02

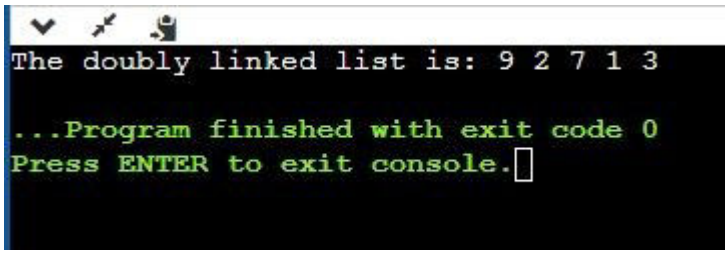
Name: Vijiyant Tanaji shejwalkar

Reg N0: 2020BIT057

1. 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;
}
```



2. enqueue_dequeue

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

// Define the default capacity of a queue
#define SIZE 1000
```

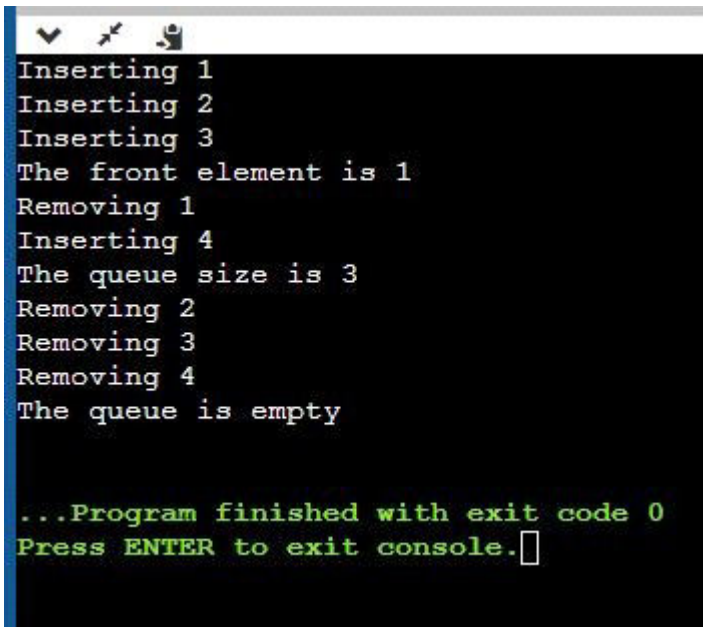
```
// A class to store a queue
class Queue
{
    int *arr;       // array to store queue elements
    int capacity;   // maximum capacity of the queue
    int front;      // front points to the front element in the queue (if any)
    int rear;       // rear points to the last element in the queue
    int count;      // current size of the queue

public:
    Queue(int size = SIZE);     // constructor
    ~Queue();                   // destructor

    int dequeue();
    void enqueue(int x);
    int peek();
    int size();
    bool isEmpty();
    bool isFull();
};

// Constructor to initialize a queue
Queue::Queue(int size)
{
    arr = new int[size];
    capacity = size;
```

```
    front = 0;
    rear = -1;
    count = 0;
}

// Destructor to free memory allocated to the queue
Queue::~Queue() {
    delete[] arr;
}

// Utility function to dequeue the front element
int Queue::dequeue()
{
    // check for queue underflow
    if (isEmpty())
    {
        cout << "Underflow\nProgram Terminated\n";
        exit(EXIT_FAILURE);
    }

    int x = arr[front];
    cout << "Removing " << x << endl;

    front = (front + 1) % capacity;
    count--;

    return x;
```

```
}

// Utility function to add an item to the queue
void Queue::enqueue(int item)
{
    // check for queue overflow
    if (isFull())
    {
        cout << "Overflow\nProgram Terminated\n";
        exit(EXIT_FAILURE);
    }

    cout << "Inserting " << item << endl;

    rear = (rear + 1) % capacity;
    arr[rear] = item;
    count++;
}

// Utility function to return the front element of the queue
int Queue::peek()
{
    if (isEmpty())
    {
        cout << "Underflow\nProgram Terminated\n";
        exit(EXIT_FAILURE);
    }
```

```
    return arr[front];
}

// Utility function to return the size of the queue
int Queue::size() {
    return count;
}

// Utility function to check if the queue is empty or not
bool Queue::isEmpty() {
    return (size() == 0);
}

// Utility function to check if the queue is full or not
bool Queue::isFull() {
    return (size() == capacity);
}

int main()
{
    // create a queue of capacity 5
    Queue q(5);

    q.enqueue(1);
    q.enqueue(2);
    q.enqueue(3);
```

```
    cout << "The front element is " << q.peek() << endl;
    q.dequeue();

    q.enqueue(4);

    cout << "The queue size is " << q.size() << endl;

    q.dequeue();
    q.dequeue();
    q.dequeue();

    if (q.isEmpty()) {
        cout << "The queue is empty\n";
    }
    else {
        cout << "The queue is not empty\n";
    }

    return 0;
}
```

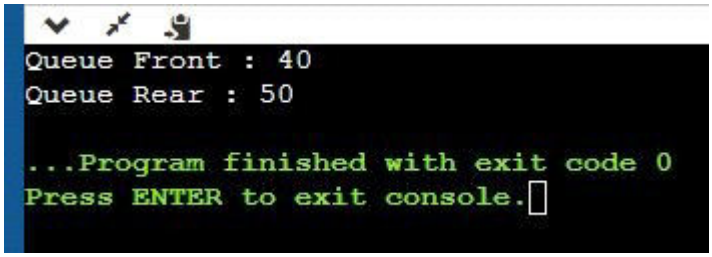


3. queue_linked_list

```
#include <bits/stdc++.h>
using namespace std;

struct QNode {
        int data;
        QNode* next;
        QNode(int d)
        {
                data = d;
                next = NULL;
        }
};
```

```
struct Queue {
        QNode *front, *rear;
        Queue() { front = rear = NULL; }

        void enQueue(int x)
        {


                QNode* temp = new QNode(x);


                if (rear == NULL) {
                        front = rear = temp;
                        return;
                }

                rear->next = temp;
                rear = temp;
        }

        void deQueue()
        {

                if (front == NULL)
                        return;
```

```
                QNode* temp = front;
                front = front->next;

                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);
}
```



4. stack_linked

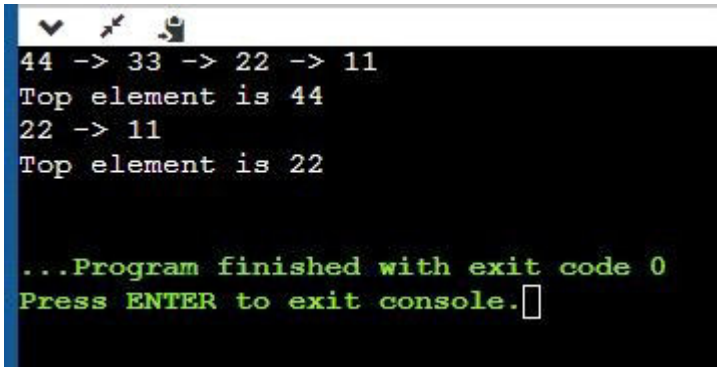
```
// C++ program to Implement a stack
// using singly linked list
#include <bits/stdc++.h>
using namespace std;

class Node {
public:
        int data;
        Node* link;

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

class Stack {
        Node* top;
```

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

        void push(int data)
        {


                Node* temp = new Node(data);

                if (!temp) {
                        cout << "\nStack Overflow";
                        exit(1);
                }

                temp->data = data;

                temp->link = top;

                top = temp;
        }

        bool isEmpty()
        {

                return top == NULL;
        }
```

```
int peek()
{

        if (!isEmpty())
                return top->data;
        else
                exit(1);
}

void pop()
{
        Node* temp;

        if (top == NULL) {
                cout << "\nStack Underflow" << endl;
                exit(1);
        }
        else {

                temp = top;

                top = top->link;

                free(temp);
        }
```

```
        }


        void display()
        {
                Node* temp;


                if (top == NULL) {
                        cout << "\nStack Underflow";
                        exit(1);
                }
                else {
                        temp = top;
                        while (temp != NULL) {


                                cout << temp->data;


                                temp = temp->link;
                                if (temp != NULL)
                                        cout << " -> ";
                        }
                }
        }
};
```

```
int main()
{
        Stack s;

        s.push(11);
        s.push(22);
        s.push(33);
        s.push(44);

        s.display();
        cout << "\nTop element is " << s.peek() << endl;


        s.pop();
        s.pop();


        s.display();


        cout << "\nTop element is " << s.peek() << endl;

        return 0;
}
```


44 -> 33 -> 22 -> 11
Top element is 44
22 -> 11
Top element is 22
...Program finished with exit code 0
Press ENTER to exit console.