Practical-2

Name : Jadhav Harshvardhan Vijaysinh

Reg. No. : 2020BIT065.

1 . Stack using linked list :

#include<iostream>

using namespace std;

//Node class

class Node{

    public:

    int data;

    Node \*next;

};

//Stack class

class Stack{

    private:

    Node \*head;

    public:

    Stack(){head = NULL;}

    void push(int);

    int pop();

    void display();

};

//Push operation to insert elements in stack

void Stack::push(int value){

    Node \*temp = new Node;

    temp->data = value;

    temp->next = head;

    head = temp;

}

//Pop operation to delete elements from stack

int Stack::pop(){

    if(head == NULL){

        cout<<"Underflow"<<endl;

        return -1;

    }

    int x = head->data;

    Node \*temp = head;

    head = head->next;

    delete temp;

    return x;

}

//Display the elements in stack

void Stack::display(){

    Node \*temp = head;

    while(temp != NULL){

        cout<<temp->data<<" ";

        temp = temp->next;

    }

}

int main(){

    Stack s;

    s.push(1);

    s.push(2);

    s.push(3);

    s.push(4);

    s.display();

    cout<<endl;

    cout<<"Popped element: "<<s.pop()<<endl;

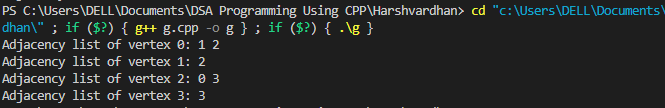
    cout<<"Popped element: "<<s.pop()<<endl;

    s.display();

    return 0;

}

Output :



2 . Queue using linked list :

#include <iostream>

using namespace std;

struct node

{

    int data;

    node \*next;

};

class Queue

{

private:

    node \*front;

    node \*rear;

public:

    Queue()

    {

        front = NULL;

        rear = NULL;

    }

    void enqueue(int value)

    {

        node \*temp = new node;

        temp->data = value;

        temp->next = NULL;

        if (rear == NULL)

        {

            front = rear = temp;

            return;

        }

        rear->next = temp;

        rear = temp;

    }

    void dequeue()

    {

        node \*temp = front;

        if (front == NULL)

        {

            cout << "Queue is Empty\n";

            return;

        }

        if (front == rear)

        {

            front = rear = NULL;

        }

        else

        {

            front = front->next;

        }

        delete temp;

    }

    int peek()

    {

        if (front == NULL)

        {

            cout << "Queue is Empty\n";

            return -1;

        }

        return front->data;

    }

    void display()

    {

        node \*temp = front;

        while (temp != NULL)

        {

            cout << temp->data << " ";

            temp = temp->next;

        }

        cout << endl;

    }

};

int main()

{

    Queue q;

    q.enqueue(20);

    q.enqueue(30);

    q.enqueue(60);

    cout << "Elements in Queue: ";

    q.display();

    cout << "Top Element: " << q.peek() << endl;

    q.dequeue();

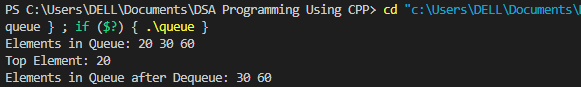
    cout << "Elements in Queue after Dequeue: ";

    q.display();

    return 0;

}

Output :



3 . Doubly linked list :

#include <iostream>

using namespace std;

struct node

{

    int data;

    node \*prev;

    node \*next;

};

class DoublyLinkedList

{

private:

    node \*head;

    node \*tail;

public:

    DoublyLinkedList()

    {

        head = NULL;

        tail = NULL;

    }

    void insert\_head(int value)

    {

        node \*temp = new node;

        temp->data = value;

        temp->prev = NULL;

        temp->next = head;

        if (head != NULL)

        {

            head->prev = temp;

        }

        head = temp;

        if (tail == NULL)

        {

            tail = head;

        }

    }

    void insert\_tail(int value)

    {

        node \*temp = new node;

        temp->data = value;

        temp->next = NULL;

        temp->prev = tail;

        if (tail != NULL)

        {

            tail->next = temp;

        }

        tail = temp;

        if (head == NULL)

        {

            head = tail;

        }

    }

    void delete\_node(int value)

    {

        node \*temp = head;

        while (temp != NULL)

        {

            if (temp->data == value)

            {

                if (temp->prev != NULL)

                {

                    temp->prev->next = temp->next;

                }

                if (temp->next != NULL)

                {

                    temp->next->prev = temp->prev;

                }

                if (temp == head)

                {

                    head = temp->next;

                }

                if (temp == tail)

                {

                    tail = temp->prev;

                }

                delete temp;

                break;

            }

            temp = temp->next;

        }

    }

    void display\_forward()

    {

        node \*temp = head;

        while (temp != NULL)

        {

            cout << temp->data << " ";

            temp = temp->next;

        }

        cout << endl;

    }

    void display\_backward()

    {

        node \*temp = tail;

        while (temp != NULL)

        {

            cout << temp->data << " ";

            temp = temp->prev;

        }

        cout << endl;

    }

};

int main()

{

    DoublyLinkedList dll;

    dll.insert\_head(11);

    dll.insert\_head(21);

    dll.insert\_head(31);

    dll.insert\_tail(42);

    dll.insert\_tail(51);

    cout << "Elements in Forward Direction: ";

    dll.display\_forward();

    cout << "Elements in Backward Direction: ";

    dll.display\_backward();

    dll.delete\_node(31);

    dll.delete\_node(51);

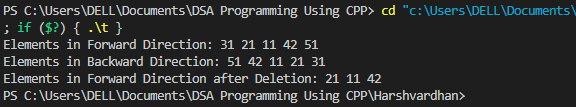
    cout << "Elements in Forward Direction after Deletion: ";

    dll.display\_forward();

    return 0;

}

Output :



4 . Enqueue and Dequeue implementation :

#include<iostream>

using namespace std;

class Node{

public:

   int data;

   Node\* next;

   Node(int x){

     data = x;

     next = NULL;

  }

};

void dequeue (Node\* head, Node\* second){

   cout<<"Implementation of dequeue :"<<endl;

   head->next = second;

   delete(head);

   while(second!=NULL){

     cout<<second->data<<" ";

     second=second->next;

   }

}

void enqueue (Node\* head, int element){

  Node\* temp = new Node (element);

  Node\* p = head;

  cout<<"Implementation of enqueue :"<<endl;

  if(p==NULL){

    temp->next=NULL;

    head=temp;

    while(head!=NULL){

       cout<<head->data<<" ";

       head-head->next;

    }

}

while(p->next!=NULL){

    p = p->next;

}

p->next = temp;

temp->next =NULL;

while(head!=NULL){

   cout<<head->data<<" ";

   head= head->next;

   }

   cout<<endl;

}

int main(){

   Node\* head = new Node(101);

   Node\* second = new Node (201);

   Node\* third = new Node (301);

   Node\* fourth = new Node (401);

   head->next = second;

   second->next = third;

   third->next = fourth;

   fourth->next = NULL;

   enqueue(head, 501);

   dequeue(head, second);

}

Output :

