**Analysis:-**

The time complexity of the entrie code is O(n). The methods used are entirely efficient.They are faster. The code has been done using the template coding style. There was a lot of difficulty while coding with different parts. The templates were showing multiple errors when I tried executing the code in multiple parts. Headers and Classes for different implementations. So, I chose to make it two parts one for Array implementation, it has array header and arraycpp file. Similarly, there is the LinkedList implementation with listheader and listcpp. This removed the conflict I had initially. So, I am submitting this code for evaluation.

**Array Implementation**

insertFront – O(1).

InsertBack – O(1).

EraseFront – O(1).

EraseBack – O(1).

IsFull – O(1).

IsEmpty - O(1).

Front - O(1).

Rear - O(1).

Display – O(n).

push - O(1).

Pop – O(1).

Deque - O(1).

Enqueue – O((1).

**LinkedList Implementation**

insertFront – O(1).

InsertBack – O(1).

EraseFront – O(1).

EraseBack – O(1).

IsFull – O(1).

IsEmpty - O(1).

Front - O(1).

Rear - O(1).

Display – O(n).

push - O(1).

Pop – O(1).

Deque - O(1).

Enqueue – O((1).

The total time complexity if excluding the display function is O(1).

DequeADTArray.h

*#ifndef DEQUE\_DEQUEADTARRAY\_H*

*#define DEQUE\_DEQUEADTARRAY\_H*

*#include<iostream>*

*#define MAX 100*

*template <typename E>*

*class DequeADTArray {*

*int arr[MAX];*

*int front;*

*int rear;*

*int size;*

*public:*

*DequeADTArray(int i); //constructor*

*~DequeADTArray(); //destructor*

*void insertFront(const E& e);*

*void insertBack(const E& e);*

*const E& eraseFront(); // throw(DequeEmpty); //note that the use of exceptions is optional*

*const E& eraseBack(); // throw(DequeEmpty); //note that the use of exceptions is optional*

*const E& frontOfDeque() const; // throw(DequeEmpty); //note that the use of exceptions is optional*

*const E& backOfDeque() const; // throw(DequeEmpty); //note that the use of exceptions is optional*

*bool isFull() const;*

*bool isEmpty() const;*

*int sizeOfDeque() const;*

*void Display();*

*//Stack Functions*

*void pop();*

*void push(const E &element);*

*//Queue Functions*

*void enqueue(const E &e);*

*void deque();*

*DequeADTArray();*

*};*

*#endif //DEQUE\_DEQUEADTARRAY\_H*

DequeADTArray.cpp

*#include "DequeADTArray.h"*

*using namespace std;*

*template<typename E>*

*DequeADTArray<E>::DequeADTArray(int size) {*

*front = -1;*

*rear = 0;*

*this->size = size;*

*}*

*template<typename E>*

*DequeADTArray<E>::~DequeADTArray() {*

*}*

*template<typename E>*

*void DequeADTArray<E>::insertFront(const E &e) {*

*if(isFull()){*

*cout<<"Deque is Full"<<endl;*

*return;*

*}*

*//queue is empty initially*

*if(front==-1)*

*{*

*front = 0;*

*rear = 0;*

*}*

*else if (front==0)*

*{*

*front = size-1;*

*}*

*else*

*{*

*front = front-1;*

*}*

*arr[front] = e;*

*size++;*

*}*

*template<typename E>*

*void DequeADTArray<E>::insertBack(const E &e) {*

*if(isFull())*

*{*

*cout<<"Deque is Full"<<endl;*

*return;*

*}*

*//if queue is empty initially*

*if(front==-1){*

*front = 0;*

*rear=0;*

*} else if(rear == size-1) {*

*rear=0;*

*} else{*

*rear = rear+1;*

*}*

*arr[rear] = e;*

*size++;*

*}*

*template<typename E>*

*const E &DequeADTArray<E>::eraseFront() {*

*if(isEmpty())*

*{*

*cout<<"Deque is Empty"<<endl;*

*}*

*const E& temp = arr[front];*

*if(front ==rear)*

*{*

*front = -1;*

*rear = -1;*

*} else {*

*if(front == size-1)*

*{*

*front=0;*

*}*

*else{*

*front = front+1;*

*}*

*}*

*size--;*

*return temp;*

*}*

*template<typename E>*

*const E &DequeADTArray<E>::eraseBack() {*

*if(isEmpty())*

*{*

*cout<<"Deque is Empty"<<endl;*

*}*

*E temp = arr[rear];*

*if(front == rear){*

*front = -1;*

*rear = -1;*

*}*

*else if(rear == 0)*

*{*

*rear = size-1;*

*}*

*else{*

*rear = rear-1;*

*}*

*size--;*

*return temp;*

*}*

*template<typename E>*

*bool DequeADTArray<E>::isFull() const {*

*return ((front==0&&rear==size-1)||front==rear+1);*

*}*

*template<typename E>*

*bool DequeADTArray<E>::isEmpty() const {*

*return (front==-1);*

*}*

*template<typename E>*

*int DequeADTArray<E>::sizeOfDeque() const {*

*return this->size;*

*}*

*template<typename E>*

*const E &DequeADTArray<E>::frontOfDeque() const {*

*if(isEmpty())*

*{*

*cout<<"Deque is Empty"<<endl;*

*return -1;*

*}*

*return arr[front];*

*}*

*template<typename E>*

*const E &DequeADTArray<E>::backOfDeque() const {*

*if(isEmpty()||rear<0)*

*{*

*cout<<"Deque is Empty"<<endl;*

*return -1;*

*}*

*return arr[rear];*

*}*

*template <typename E>*

*void DequeADTArray<E>::Display(){*

*if(!isEmpty())*

*{*

*int size = sizeOfDeque();*

*for(int i=0;i<size;i++)*

*{*

*cout<<arr[i]<<" ";*

*}*

*cout<<endl;*

*}*

*cout<<"Deque is Empty"<<endl;*

*}*

*// push to push element at top of stack*

*// using insert at last function of deque*

*template<typename E>*

*void DequeADTArray<E>:: push(const E &element)*

*{*

*insertBack(element);*

*}*

*// pop to remove element at top of stack*

*// using remove at last function of deque*

*template<typename E>*

*void DequeADTArray<E>::pop()*

*{*

*eraseBack();*

*}*

*template<typename E>*

*void DequeADTArray<E>::deque() {*

*eraseFront();*

*}*

*template<typename E>*

*void DequeADTArray<E>::enqueue(const E &e) {*

*insertBack(e);*

*}*

*template<typename E>*

*DequeADTArray<E>::DequeADTArray() {*

*front = rear = NULL;*

*size=0;*

*}*

*int main(){*

*// template<typename E>, I have choosen Integer type data.*

*DequeADTArray<int> dq;*

*cout << "Insert element '5' at rear end\n";*

*dq.insertBack(5);*

*cout << "Insert element '10' at rear end\n";*

*dq.insertBack(10);*

*cout << "Rear end element: "*

*<< dq.backOfDeque() << endl;*

*dq.eraseBack();*

*cout << "After deleting rear element new rear"*

*<< " is: " << dq.backOfDeque() << endl;*

*cout << "Inserting element '15' at front end \n";*

*dq.insertFront(15);*

*cout << "Front end element: "*

*<< dq.frontOfDeque() << endl;*

*cout << "Number of elements in Deque: "*

*<< dq.sizeOfDeque() << endl;*

*dq.eraseFront();*

*cout << "After deleting front element new "*

*<< "front is: " << dq.frontOfDeque() << endl;*

*cout<<"#####################################################"<<endl;*

*cout<<"Implementing the Stack using LinkedList and Deque"<<endl;*

*DequeADTArray<int> stack;*

*// push 7 and 8 at top of stack*

*stack.push(7);*

*stack.push(8);*

*cout << "Stack: ";*

*stack.Display();*

*// pop an element*

*stack.pop();*

*cout << "Stack: ";*

*stack.Display();*

*cout<<"Successfully implemented Stack "<<endl;*

*cout<<"#####################################################"<<endl;*

*// object of Queue*

*cout<<"Implementing the Queue using LinkedList and Deque"<<endl;*

*DequeADTArray<int> que;*

*// insert 12 and 13 in queue*

*que.enqueue(12);*

*que.enqueue(13);*

*que.enqueue(16);*

*que.enqueue(84);*

*cout << "Queue: ";*

*que.Display();*

*// delete an element from queue*

*que.deque();*

*cout << "Queue: ";*

*que.Display();*

*cout<<"Successfully implemented Queue "<<endl;*

*cout<<"#####################################################"<<endl;*

*return 0;*

*}*

**DequeADTList.h**

*#ifndef DEQUE\_DEQUEADTLIST\_H*

*#define DEQUE\_DEQUEADTLIST\_H*

*#include<istream>*

*template <typename E>*

*class DequeADTList{*

*private:*

*struct dequeNode{*

*int data;*

*dequeNode \*prev, \*next;*

*// Function to get a new node*

*static dequeNode\* getnode(int data)*

*{*

*dequeNode\* newNode = (dequeNode\*)malloc(sizeof(dequeNode));*

*newNode->data = data;*

*newNode->prev = newNode->next = NULL;*

*return newNode;*

*}*

*};*

*dequeNode\* front;*

*dequeNode\* rear;*

*int Size;*

*public:*

*DequeADTList(); //constructor*

*~DequeADTList(); //destructor*

*void insertFront(const E& e);*

*void insertBack(const E& e);*

*const E& eraseFront(); // throw(DequeEmpty); //note that the use of exceptions is optional*

*const E& eraseBack(); // throw(DequeEmpty); //note that the use of exceptions is optional*

*const E &getfront() const; // throw(DequeEmpty); //note that the use of exceptions is optional*

*const E& getback() const; // throw(DequeEmpty); //note that the use of exceptions is optional*

*int size () const;*

*void Display();*

*bool isEmpty() const;*

*//Stack Functions*

*void pop();*

*void push(const E &element);*

*//Queue Functions*

*void enqueue(const E &e);*

*void deque();*

*};*

*#endif //DEQUE\_DEQUEADTLIST\_H*

**DequeADTList.cpp**

*#include<iostream>*

*#include "DequeADTList.h"*

*using namespace std;*

*template<typename E>*

*DequeADTList<E>::DequeADTList() {*

*front = rear = nullptr;*

*Size = 0;*

*}*

*template<typename E>*

*DequeADTList<E>::~DequeADTList(){*

*rear = NULL;*

*while (front != NULL)*

*{*

*dequeNode\* temp = front;*

*front = front->next;*

*free(temp);*

*}*

*Size = 0;*

*}*

*template<typename E>*

*void DequeADTList<E>::insertFront(const E &data) {*

*dequeNode\* newNode = dequeNode::getnode(data);*

*// If true then new element cannot be added*

*// and it is an 'Overflow' condition*

*if (newNode == NULL)*

*cout << "OverFlow\n";*

*else*

*{*

*// If deque is empty*

*if (front == NULL)*

*rear = front = newNode;*

*// Inserts node at the front end*

*else*

*{*

*newNode->next = front;*

*front->prev = newNode;*

*front = newNode;*

*}*

*// Increments count of elements by 1*

*Size++;*

*}*

*}*

*template<typename E>*

*void DequeADTList<E>::insertBack(const E &data) {*

*dequeNode\* newNode = dequeNode::getnode(data);*

*// If true then new element cannot be added*

*// and it is an 'Overflow' condition*

*if (newNode == NULL)*

*cout << "OverFlow\n";*

*else*

*{*

*// If deque is empty*

*if (rear == NULL)*

*front = rear = newNode;*

*// Inserts node at the rear end*

*else*

*{*

*newNode->prev = rear;*

*rear->next = newNode;*

*rear = newNode;*

*}*

*Size++;*

*}*

*}*

*template<typename E>*

*const E &DequeADTList<E>::eraseFront() {*

*// If deque is empty then*

*// 'Underflow' condition*

*if (isEmpty())*

*cout << "UnderFlow\n";*

*// Deletes the node from the front end and makes*

*// the adjustment in the links*

*else*

*{*

*dequeNode\* temp = front;*

*front = front->next;*

*// If only one element was present*

*if (front == NULL)*

*rear = NULL;*

*else*

*front->prev = NULL;*

*free(temp);*

*// Decrements count of elements by 1*

*Size--;*

*}*

*}*

*template<typename E>*

*const E &DequeADTList<E>::eraseBack() {*

*// If deque is empty then*

*// 'Underflow' condition*

*if (isEmpty())*

*cout << "UnderFlow\n";*

*// Deletes the node from the rear end and makes*

*// the adjustment in the links*

*else*

*{*

*dequeNode\* temp = rear;*

*rear = rear->prev;*

*// If only one element was present*

*if (rear == NULL)*

*front = NULL;*

*else*

*rear->next = NULL;*

*free(temp);*

*// Decrements count of elements by 1*

*Size--;*

*}*

*}*

*template<typename E>*

*const E &DequeADTList<E>::getfront() const {*

*// If deque is empty, then returns*

*// garbage value*

*if (isEmpty())*

*return -1;*

*return front->data;*

*}*

*template<typename E>*

*const E &DequeADTList<E>::getback() const {*

*// If deque is empty, then returns*

*// garbage value*

*if (isEmpty())*

*return -1;*

*return rear->data;*

*}*

*template<typename E>*

*int DequeADTList<E>::size() const {*

*return Size;*

*}*

*template<typename E>*

*void DequeADTList<E>::Display() {*

*// if list is not empty*

*if (!isEmpty()) {*

*dequeNode\* temp = front;*

*while (temp != NULL) {*

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

*temp = temp->next;*

*}*

*cout << endl;*

*return;*

*}*

*cout << "List is Empty" << endl;*

*}*

*template<typename E>*

*bool DequeADTList<E>::isEmpty() const {*

*return (front == NULL);*

*}*

*// push to push element at top of stack*

*// using insert at last function of deque*

*template<typename E>*

*void DequeADTList<E>:: push(const E &element)*

*{*

*insertBack(element);*

*}*

*// pop to remove element at top of stack*

*// using remove at last function of deque*

*template<typename E>*

*void DequeADTList<E>::pop()*

*{*

*eraseBack();*

*}*

*template<typename E>*

*void DequeADTList<E>::deque() {*

*eraseFront();*

*}*

*template<typename E>*

*void DequeADTList<E>::enqueue(const E &e) {*

*insertBack(e);*

*}*

*int main(){*

*// template<typename E>, I have chosen Integer type data.*

*DequeADTList<int> dq;*

*cout << "Insert element '5' at rear end\n";*

*dq.insertBack(5);*

*cout << "Insert element '10' at rear end\n";*

*dq.insertBack(10);*

*cout << "Rear end element: "*

*<< dq.getback() << endl;*

*dq.eraseBack();*

*cout << "After deleting rear element new rear"*

*<< " is: " << dq.getback() << endl;*

*cout << "Inserting element '15' at front end \n";*

*dq.insertFront(15);*

*cout << "Front end element: "*

*<< dq.getfront() << endl;*

*cout << "Number of elements in Deque: "*

*<< dq.size() << endl;*

*dq.eraseFront();*

*cout << "After deleting front element new "*

*<< "front is: " << dq.getfront() << endl;*

*cout<<"#####################################################"<<endl;*

*cout<<"Implementing the Stack using LinkedList and Deque"<<endl;*

*DequeADTList<int> stack;*

*// push 7 and 8 at top of stack*

*stack.push(7);*

*stack.push(8);*

*cout << "Stack: ";*

*stack.Display();*

*// pop an element*

*stack.pop();*

*cout << "Stack: ";*

*stack.Display();*

*cout<<"Successfully implemented Stack "<<endl;*

*cout<<"#####################################################"<<endl;*

*// object of Queue*

*cout<<"Implementing the Queue using LinkedList and Deque"<<endl;*

*DequeADTList<int> que;*

*// insert 12 and 13 in queue*

*que.enqueue(12);*

*que.enqueue(13);*

*que.enqueue(16);*

*que.enqueue(84);*

*cout << "Queue: ";*

*que.Display();*

*// delete an element from queue*

*que.deque();*

*cout << "Queue: ";*

*que.Display();*

*cout<<"Successfully implemented Queue "<<endl;*

*cout<<"#####################################################"<<endl;*

*return 0;*

*}*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*THE\*\*\*END\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*