****

**NORTH SOUTH UNIVERSITY**

**DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING**

**GROUP MEMBER –**

1. **TASFIA NAUSHEEN 1711488042**
2. **NIAZI MEHRAB 1711248042**

**COURSE – CSE 225, SEC 5**

**SESSION - SUMMER 2020**

**SUBMITTED TO – Dr Ziaul Hossain (ZHO)**

**Answer to 1.a**

Like single linked list, double linked list is a collection of nodes with head and tail but only with two pointers where one pointer is for the previous node and another for the next node. (PTO)

Head

Node 4

Node 3

Node 2

Node 1

null

tail

null

N1

N4

N3

N2

In the figure above, each node is with two nodes where one is towards the previous node and another to the next node. Here the first node that is the head and the last node that is the tail, has their second pointer set to null. Since the double linked list has two pointers, it can navigate in the forward and backward directions which is the only thing the single linked list can’t do.

**Answer to 1.b**

In single linked list, each node can send their elements in one direction because they only have one pointer. On the other hand, in double linked list, each node can navigate their elements forward and backward because they have the advantage of two pointers. This is the way double linked list is different from single linked list.

**Answer to 1.c**

The advantages in terms of Order of Magnitude (Worst case running time) for any operation in double linked list than single linked list is that in double linked list, each node can navigate elements step by step both forward and backward direction which makes it to navigate faster than single linked list even in worst case runtime during any operations.

**Answer to 2**

There are total five files in our program. They are main.cpp, doublyLinkList.h, doublyLinkList.cpp, NodeType.h, NodeType.cpp. In .ccp mainly is the implementation of methods and .h file is the declaration of methods, variables. In main.cpp we create object and insert and delete some elements.

**//DoublyLinkList.h**

#ifndef DOUBLY\_LINK\_LIST

#define DOUBLY\_LINK\_LIST

#include "NodeType.h"

template<class ItemType>

class DoublyLinkList

{

public:

DoublyLinkList();

~DoublyLinkList();

bool isEmpty();

bool isFull();

void makeEmpty();

int getLength();

bool hasNext();

bool hasPrevious();

ItemType getNext();

ItemType getLast();

bool insert(ItemType item);

bool insertFirst(ItemType item);

bool insertLast(ItemType item);

bool deleteItem(ItemType item);

bool deleteFirst();

bool deleteLast();

void printContent();

private:

NodeType<ItemType>\* head;

NodeType<ItemType>\* tail;

int length;

NodeType<ItemType>\* iterator;

};

#endif //UNSORTED\_DOUBLY\_LLIST

**//NodeType.h**

#ifndef NODETYPE\_H

#define NODETYPE\_H

template<class ItemType>

class NodeType

{

public:

NodeType();

NodeType(ItemType data);

~NodeType();

ItemType getData();

NodeType<ItemType>\* getNext();

NodeType<ItemType>\* getLast();

void setData(ItemType data);

void setNext(NodeType\* next);

void setPrevious(NodeType\* previous);

private:

ItemType data;

NodeType<ItemType>\* next;

NodeType<ItemType>\* previous;

};

#endif //NODETYPE\_H

**//DoublyLinkList.cpp**

#include "DoublyLinkList.h"

#include <exception>

#include <iostream>

using namespace std;

template class DoublyLinkList<int>;

template class DoublyLinkList<char>;

template<class ItemType>

DoublyLinkList<ItemType>::DoublyLinkList()

{

head = nullptr;

tail = head;

length = 0;

iterator = nullptr;

}

template<class ItemType>

DoublyLinkList<ItemType>::~DoublyLinkList()

{

makeEmpty();

}

template<class ItemType>

bool DoublyLinkList<ItemType>::isEmpty()

{

return (length == 0);

}

template<class ItemType>

bool DoublyLinkList<ItemType>::isFull()

{

try

{

NodeType<ItemType>\* newNode = new NodeType<ItemType>(ItemType());

delete newNode;

}

catch (const exception& e)

{

cout << e.what() << endl;

return true;

}

return false;

}

template<class ItemType>

void DoublyLinkList<ItemType>::makeEmpty()

{

NodeType<ItemType> \* tempPtr;

while(head!= nullptr)

{

tempPtr = head;

head = head->getNext();

delete tempPtr;

}

length = 0;

iterator = 0;

}

template<class ItemType>

int DoublyLinkList<ItemType>::getLength()

{

return length;

}

template<class ItemType>

bool DoublyLinkList<ItemType>::hasNext()

{

//Check if iterator's has reached tail.

return (iterator != this->tail);

}

template<class ItemType>

bool DoublyLinkList<ItemType>::hasPrevious()

{

//Check if iterator's is at begining.

return (iterator != this->head);

}

template<class ItemType>

ItemType DoublyLinkList<ItemType>::getNext()

{

if (hasNext())

{

if (iterator == 0)

{

iterator = head;

}

else

{

iterator = iterator->getNext();

}

return (iterator->getData());

}

else

cout << "No valid GetNext calls can be made." << endl;

return 0;

}

template<class ItemType>

ItemType DoublyLinkList<ItemType>::getLast()

{

if (hasPrevious())

{

if (iterator == 0)

{

iterator = tail;

}

else

{

iterator = iterator->getLast();

}

return (iterator->getData());

}

cout << "No valid GetPrevious calls can be made." << endl;

return 0;

}

template<class ItemType>

bool DoublyLinkList<ItemType>::insert(ItemType item)

{

if (isFull())

{

printContent();

return false;

}

else

insertFirst(item);

return true;

}

template<class ItemType>

bool DoublyLinkList<ItemType>::insertFirst(ItemType item)

{

if (isFull())

{

printContent();

return false;

}

else

{

NodeType<ItemType>\*newNode = new NodeType<ItemType>;

newNode->setData(item);

if(head == 0)

{

newNode->setNext(nullptr);

newNode->setPrevious(nullptr);

head = newNode;

tail = newNode;

}

else

newNode->setNext(head);

head->setPrevious(newNode);

head= newNode;

}

length ++;

printContent();

return true;

}

template<class ItemType>

bool DoublyLinkList<ItemType>::insertLast(ItemType item)

{

if (isFull())

{

printContent();

return false;

}

else

{

NodeType<ItemType>\*newNode = new NodeType<ItemType>;

newNode->setData(item);

if(tail == 0)

{

newNode->setNext(nullptr);

newNode->setPrevious(nullptr);

head = newNode;

tail = newNode;

}

else

newNode->setNext(nullptr);

newNode->setPrevious(tail);

tail->setNext(newNode) ;

tail= newNode;

}

length ++;

printContent();

return true;

}

template<class ItemType>

bool DoublyLinkList<ItemType>::deleteItem(ItemType item)

{

bool found = false;

if (isEmpty())

{

printContent();

return false;

}

else

{

NodeType<ItemType>\*location = head;

NodeType<ItemType>\*templocation;

if(item==head->getData())

{

found= true;

deleteFirst();

}

else if (item == tail->getData())

{

found= true;

deleteLast();

}

else

{

while(!found)

{

if(item != (location->getNext()->getData()))

{

location = location->getNext();

templocation = location ->getNext();

}

else

found = true;

}

if(found)

{

location ->setNext((location->getNext()->getNext()));

(templocation->getNext())->setPrevious(location);

if(iterator==location)

{

iterator = iterator->getLast();

}

else

iterator = iterator;

}

else

cout << "Item is not in the List"<< endl;

delete templocation;

}

}

length--;

printContent();

return found;

}

template<class ItemType>

bool DoublyLinkList<ItemType>::deleteFirst()

{

if (isEmpty())

{

printContent();

return false;

}

else

{

if(iterator == head)

{

iterator = iterator->getNext();

}

else

{

iterator= iterator;

}

NodeType<ItemType>\*templocation=head;

head = head->getNext();

delete templocation;

length--;

}

printContent();

return true;

}

template<class ItemType>

bool DoublyLinkList<ItemType>::deleteLast()

{

if (isEmpty())

{

printContent();

return false;

}

else

{

if(iterator==tail)

{

iterator = iterator->getLast();

}

NodeType<ItemType>\*templocation=tail;

tail = tail->getLast();

tail->setNext(nullptr);

delete templocation;

}

length--;

printContent();

return true;

}

template<class ItemType>

void DoublyLinkList<ItemType>::printContent()

{

NodeType<ItemType>\* current = head;

while (current)

{

cout << current->getData() << " ";

current = current->getNext();

}

cout << endl;

}

//NodeType.cpp

#include "NodeType.h"

#include <iostream>

template class NodeType<int>;

template class NodeType<char>;

template<class ItemType>

NodeType<ItemType>::NodeType()

{

this->data = ItemType();

this->next = nullptr;

this->previous = nullptr;

}

template<class ItemType>

NodeType<ItemType>::NodeType(ItemType data)

{

this->data = data;

this->next = nullptr;

this->previous = nullptr;

}

template<class ItemType>

NodeType<ItemType>::~NodeType()

{

}

template<class ItemType>

void NodeType<ItemType>::setData(ItemType data)

{

this->data = data;

}

template<class ItemType>

ItemType NodeType<ItemType>::getData()

{

return this->data;

}

template<class ItemType>

void NodeType<ItemType>::setNext(NodeType \* next)

{

this->next = next;

}

template<class ItemType>

void NodeType<ItemType>::setPrevious(NodeType \* previous)

{

this->previous = previous;

}

template<class ItemType>

NodeType<ItemType>\* NodeType<ItemType>::getNext()

{

return this->next;

}

template<class ItemType>

NodeType<ItemType>\* NodeType<ItemType>::getLast()

{

return this->previous;

}

//Main.cpp

#include<iostream>

#include "DoublyLinkList.h"

using namespace std;

int main(void)

{

DoublyLinkList<int> list;

cout << "Current List Size :" << list.getLength() << endl;

list.insert(5);

list.insert(8);

list.insert(9);

list.insert(7);

list.insertFirst(1);

list.insertFirst(12);

list.insertLast(16);

list.insertFirst(122);

list.insertLast(89);

list.deleteItem(8);

list.deleteFirst();

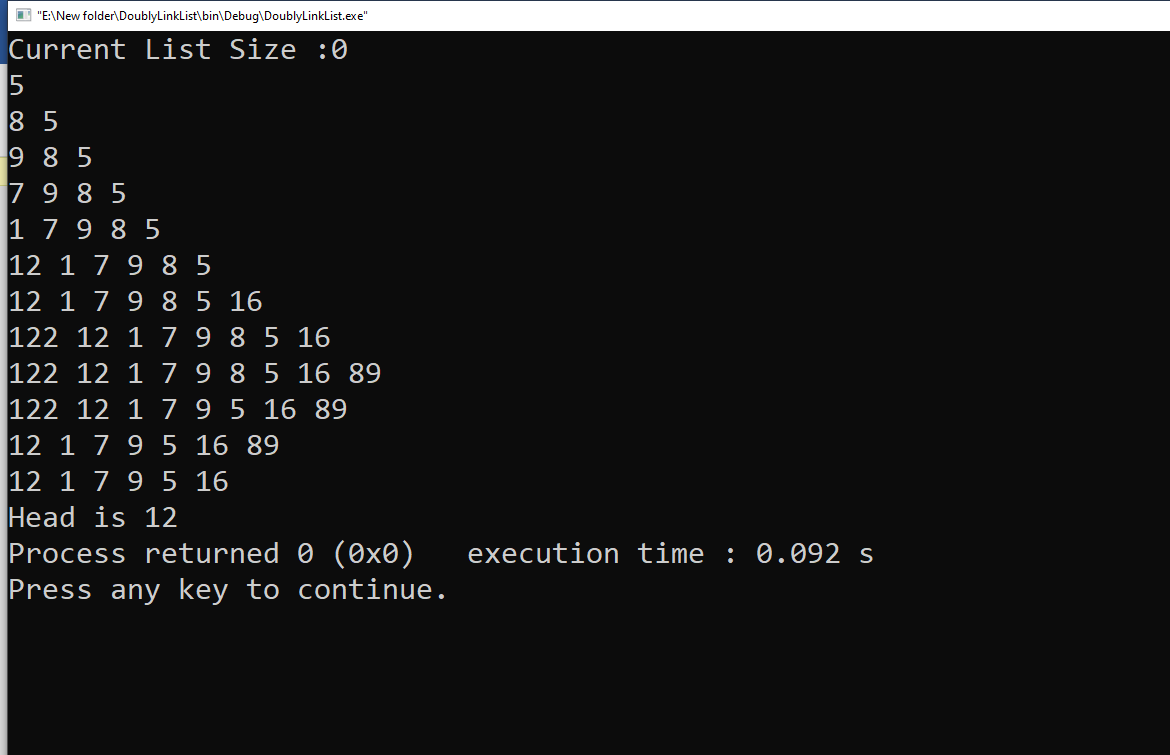
list.deleteLast();

cout<<"Head is "<<list.getNext();

return 0;

}

**//Output**

 **Ans to 3**

1. isEmpty() = O (1)
2. insertFirst = O (1)
3. InsertLast = O (1)
4. Insert = O (1)
5. GetNext = O (1)
6. GetLast = O (1)
7. DeleteFirst = O (1)
8. DeleteLast = O (1)
9. DeleteItem= O (N)