# Department of Computing

## **Name : Mahum Samar**

## **CMS ID : 290647**

**CS250: Data Structure and Algorithms**

## **Class: BSCS 9B**

# Lab 2: Singly Linked Lists

# Course Instructor: Dr. Yasir Faheem

## **Code for all the classes and the methods**

#include <iostream>

using namespace std;

class ListNode{

public:

int data;

ListNode \*next;

};

class SinglyLinkedList{

public:

ListNode \*headNode; // special variable which stores address of the head node.

ListNode \*lastNode; // special variable which stores address of the last node.

ListNode \*preLoc; //to be used by Search(value) method to store address of logical predecessor of value in a list.

ListNode \*loc; //to be used by Search(value) method to store address of the node containing the searched value in a list. If it is not found it contains NULL.

//constructor for creating the empty list

SinglyLinkedList(){

headNode=NULL; //headNode to point to the node at the start of the list

preLoc=NULL; //preLoc fro the predecessor node of the loc

loc=NULL; //loc to assign tho the node which is required to be found

}

//method for checking whether the singly linked list is empty or not.

bool isEmpty()

{

return headNode == NULL;

}

//method for inserting the value at the headNode

void insertAtFront(int value)

{

ListNode \*newNode = new ListNode();

newNode -> data = value;

if(isEmpty())

{

//executes if the list is empty

headNode = newNode;

lastNode = newNode;

}

else

{

//executes if the list is not empty

newNode -> next = headNode;

headNode = newNode;

}

printList(); //prints the list after insertion at front

}

//method for inserting the value at the lastNode

void insertAtEnd(int value)

{

ListNode \*newNode = new ListNode();

newNode -> data = value;

if(isEmpty())

{

//executes if the list is empty

headNode = newNode;

lastNode = newNode;

}

else

{

//executes if the list is not empty

lastNode -> next = newNode;

lastNode = newNode;

}

printList(); //prints the list after insertion at end

}

//method for printing the list on the screen

void printList()

{

//method to print the list by using the temporary pointer.

ListNode \*tempNode = headNode;

if(!isEmpty())

{

while(tempNode != NULL)

{

cout << tempNode -> data << endl;

tempNode = tempNode -> next;

}

}

}

//method for searching the node which contains the value entered by the user.

void searchNode(int value)

{

preLoc = NULL;

loc = headNode;

if (isEmpty())

{

//returns if the list is empty.

return;

}

while(loc != NULL && loc -> data < value )

{

//this loop executes until loc is not null and logical position of the value is also not found.

preLoc = loc;

loc = loc -> next;

}

if(loc != NULL && loc -> data != value)

{

//if the value is not found the loc is assigned NULL.

loc = NULL;

}

}

//method to insert the value in the list at its logical position

void insertSorted(int value)

{

//for this method we have assumptions that duplication is not allowed and the list is already sorted.

searchNode(value);

if(loc != NULL)

{

//if the value already exists the value is not inserted because

cout << "Value already exists." << endl;

return;

}

//if value is to exerted

if(preLoc == NULL)

{

//if the logical position of the value is the first position in the list.

insertAtFront(value);

}

else

if(preLoc != NULL && loc == lastNode)

{

//if the logical position of the value is the end of the end.

insertAtEnd(value);

}

else

{

//if the logical position of the value is in the middle of the list.

ListNode \*newNode = new ListNode();

newNode -> data = value;

newNode -> next = preLoc -> next;

preLoc -> next = newNode;

printList(); //displays the list on the screen

}

}

Delete (int value)

{

//this method deletes the specific value at a time if it exists.

if (isEmpty())

{

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

}

searchNode(value);

if (loc != NULL)

{

//if the value to be deleted is the headNode

if(preLoc == NULL && loc == headNode)

{

preLoc = headNode;

headNode = loc -> next;

delete preLoc;

}

else if (preLoc != NULL)

{

// if the value to be deleted is in the middle of the list.

preLoc -> next = loc -> next;

if(loc -> next == NULL)

{

//if the value is in the end of the list.

lastNode = preLoc;

}

delete loc;

}

}

printList();

}

void DestroyLinkedList()

{

//this method destroys the list both logically and physically.

if(!isEmpty())

{

ListNode \*temp = headNode;

while(temp!= NULL)

{

headNode = headNode -> next;

delete temp;

temp = headNode;

}

cout << "Calling print function." << endl;

printList();

}

else

{

cout << "The list is already Empty." << endl;

}

}

};

Main function for:

## **Bool IsEmpty() function**

### Code:

**int main()**

{

//creating the object of the SinglyLinkedList class

SinglyLinkedList \*singlyLinkedList = new SinglyLinkedList();

//checking if the list is empty or not

if (singlyLinkedList -> isEmpty())

{

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

}

else

{

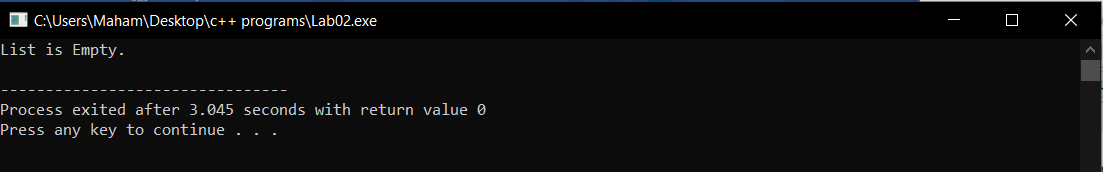
cout << "List is not Empty." << endl;

}

}

}

### Output:



## **Inserting a value at the Front of a list**

### Code:

**int main()**

{

//creating the object of the SinglyLinkedList class

SinglyLinkedList \*singlyLinkedList = new SinglyLinkedList();

cout << "Insertion at front." << endl;

cout << "Insertion of 4." << endl;

singlyLinkedList -> insertAtFront(4);

cout << "Insertion of 3." << endl;

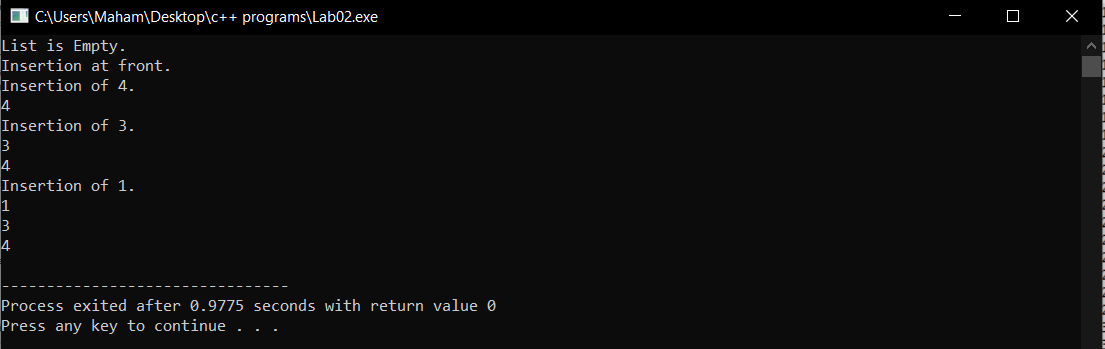
singlyLinkedList -> insertAtFront(3);

cout << "Insertion of 1." << endl;

singlyLinkedList -> insertAtFront(1);

}

### Output:



## **Inserting a value at the tail end of a list**

### Code:

**int main()**

{

//creating the object of the SinglyLinkedList class

SinglyLinkedList \*singlyLinkedList = new SinglyLinkedList();

cout << "Insertion at front." << endl;

cout << "Insertion of 4." << endl;

singlyLinkedList -> insertAtFront(4);

cout << "Insertion of 3." << endl;

singlyLinkedList -> insertAtFront(3);

cout << "Insertion of 1." << endl;

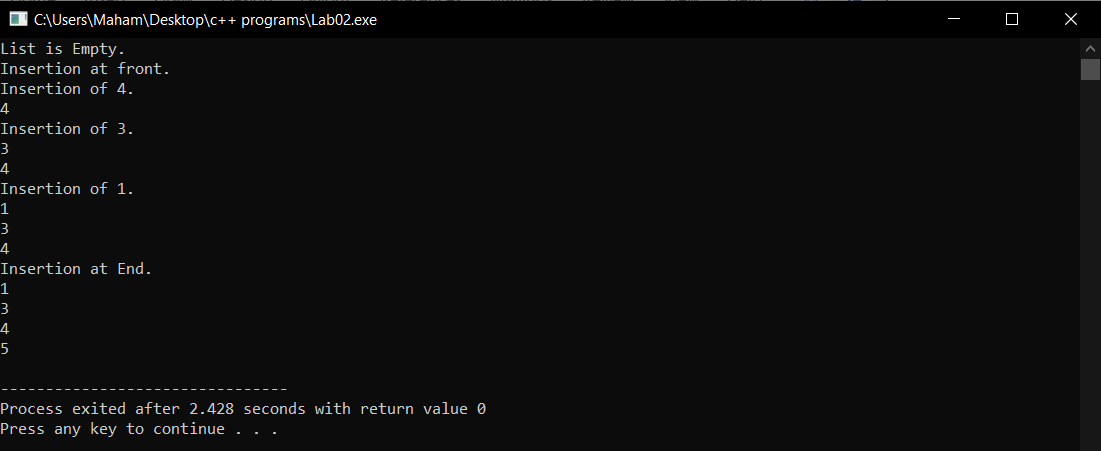
singlyLinkedList -> insertAtFront(1);

cout << "Insertion at End." << endl;

singlyLinkedList -> insertAtEnd(5);

}

### Output:



## **Insertion in a Sorted List**

### Code:

**int main()**

{

//creating the object of the SinglyLinkedList class

SinglyLinkedList \*singlyLinkedList = new SinglyLinkedList();

//checking if the list is empty or not

if (singlyLinkedList -> isEmpty())

{

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

}

else

{

cout << "List is not Empty." << endl;

}

cout << "Insertion at front." << endl;

cout << "Insertion of 4." << endl;

singlyLinkedList -> insertAtFront(4);

cout << "Insertion of 3." << endl;

singlyLinkedList -> insertAtFront(3);

cout << "Insertion of 1." << endl;

singlyLinkedList -> insertAtFront(1);

cout << "Insertion in sorted list ." << endl;

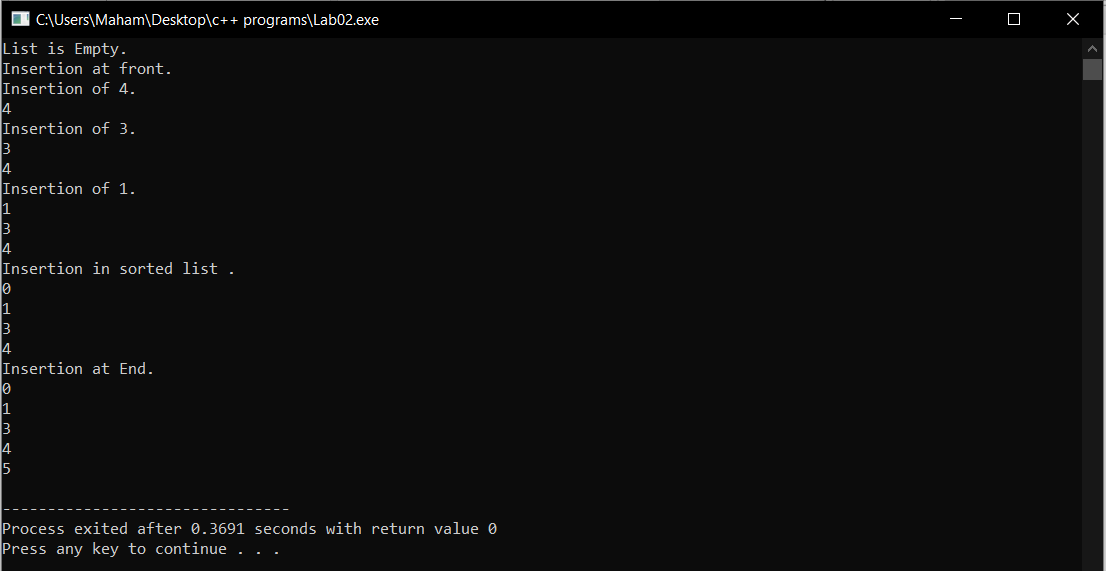
singlyLinkedList -> insertSorted(0);

cout << "Insertion at End." << endl;

singlyLinkedList -> insertAtEnd(5);

}

### Output:



## **Delete a Value**

### Code:

**int main()**

{

//creating the object of the SinglyLinkedList class

SinglyLinkedList \*singlyLinkedList = new SinglyLinkedList();

//checking if the list is empty or not

if (singlyLinkedList -> isEmpty())

{

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

}

else

{

cout << "List is not Empty." << endl;

}

cout << "Insertion at front." << endl;

cout << "Insertion of 4." << endl;

singlyLinkedList -> insertAtFront(4);

cout << "Insertion of 3." << endl;

singlyLinkedList -> insertAtFront(3);

cout << "Insertion of 1." << endl;

singlyLinkedList -> insertAtFront(1);

cout << "Insertion in sorted list ." << endl;

singlyLinkedList -> insertSorted(0);

cout << "Insertion at End." << endl;

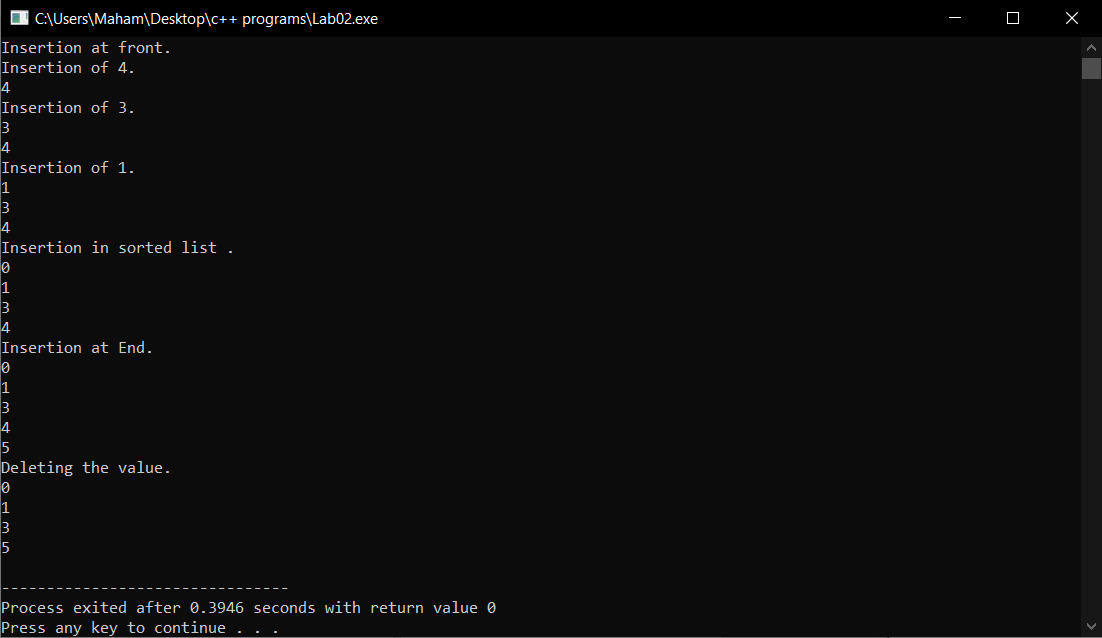
singlyLinkedList -> insertAtEnd(5);

cout << "Deleting the value." << endl;

singlyLinkedList -> Delete(4);

}

### Output:



## **Destroy a Linked List**

### Code:

**int main()**

{

//creating the object of the SinglyLinkedList class

SinglyLinkedList \*singlyLinkedList = new SinglyLinkedList();

//checking if the list is empty or not

if (singlyLinkedList -> isEmpty())

{

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

}

else

{

cout << "List is not Empty." << endl;

}

cout << "Insertion at front." << endl;

cout << "Insertion of 4." << endl;

singlyLinkedList -> insertAtFront(4);

cout << "Insertion of 3." << endl;

singlyLinkedList -> insertAtFront(3);

cout << "Insertion of 1." << endl;

singlyLinkedList -> insertAtFront(1);

cout << "Insertion in sorted list ." << endl;

singlyLinkedList -> insertSorted(0);

cout << "Insertion at End." << endl;

singlyLinkedList -> insertAtEnd(5);

cout << "Deleting the value." << endl;

singlyLinkedList -> Delete(4);

cout << "Destroying the entire list." << endl;

singlyLinkedList -> DestroyLinkedList();

cout << "Printing the entire list but nothing printed.." << endl;

singlyLinkedList -> printList();

}

### Output:

