|  |  |
| --- | --- |
| download | COMSATS University Islamabad, Vehari Campus Department of Computer Science |

**Class: BCS-SP22-4A Submission Deadline: 9 Oct 2023**

**Subject: Data Structures and Algorithms-Lab Instructor: Yasmeen Jana Max Marks: 20**

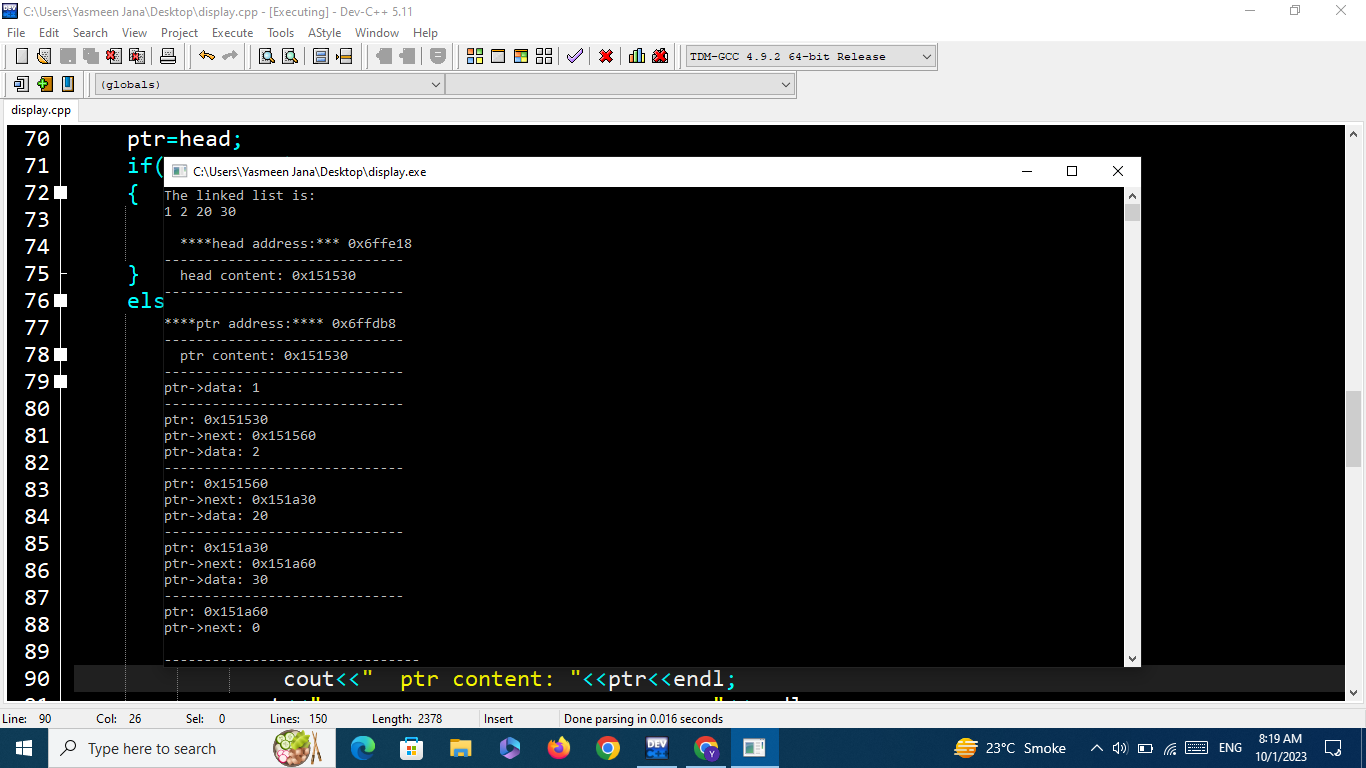
**Name: Ahmad Hussain Reg. No: SP22-BCS-008**

**Email:** [**yasmeenjana@cuivehari.edu.pk**](mailto:yasmeenjana@cuivehari.edu.pk)

**You can ask queries related to Lab Activities on the above email.**

**Activity 1:**

Create a function to display linked list output as below:



**Activity 2:**

Write a program that will implement single, doubly, and circular linked link list operations by showing a menu to the user.

The menu should be:

**Which linked list you want:**

1: Single

2: Double

3: Circular

After the option is chosen by the user:

**Which operation you want to perform:**

1: Insertion

2: Deletion

3: Display

4: Reverse

4: Seek

5: Exit

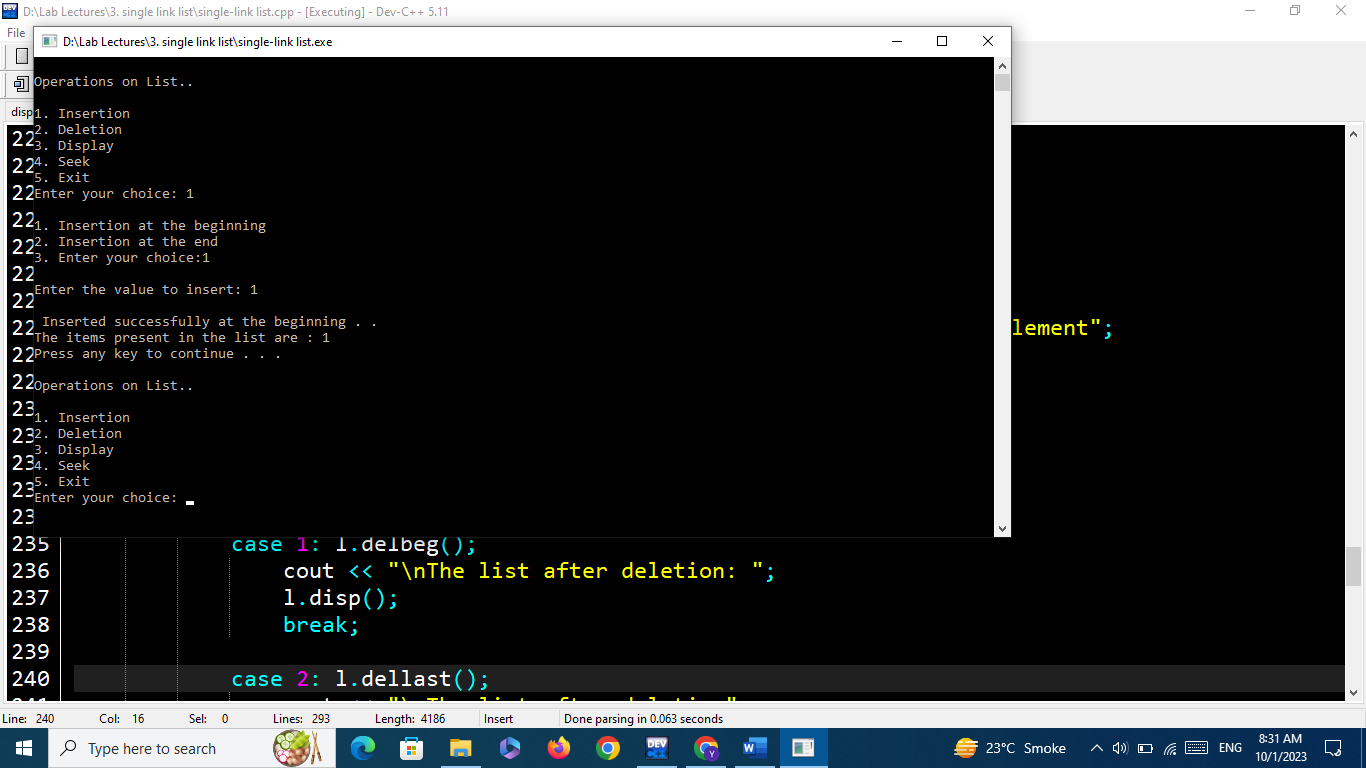
**Let's suppose, the user has chosen the insertion option then the following menu should be displayed:**

1: insertion at beginning

2: insertion at end

3: insertion at the specific data node

A sample output screenshot is below:



**You can get help from the below link:**

[**https://github.com/programming-debug/Data-Structure-Lab/blob/main/Lab3/single-link%20list.cpp**](https://github.com/programming-debug/Data-Structure-Lab/blob/main/Lab3/single-link%20list.cpp)

In this Word file, you should place the code and its output screenshot.

After completing the activities, Upload the final pdf and cpp code files to the “**DSA\_Lab”** repository.

**Activity #1**

**Code :**

#include<iostream>

using namespace std;

//creating the node

struct Node {

int data;

Node \* next; //it is Node type pointer as it points the next node.

};

class Linklist {

private: Node \* head;

public: Linklist() {

head = NULL;

}

//insert at the end

void insertAtEnd(int val) {

Node \* newNode = new Node;

newNode -> data = val;

newNode -> next = NULL;

// check if the node linked list is empty or not

if (head == NULL) {

head = newNode;

} else {

Node \* currentNode = head;

while (currentNode -> next != NULL) {

currentNode = currentNode -> next;

}

currentNode -> next = newNode;

}

}

// Function to display the contents of the list

void display() {

cout<<"The linked list is:\n";

Node \* ptr = head;

while (ptr != NULL) {

cout << ptr -> data << " ";

ptr = ptr -> next;

}

ptr = head;

cout<<endl <<endl

<<" \*\*\*Head Address:\*\*\* "<<&head<<endl

<<"--------------------------------------"<<endl

<<" head content: "<<head->next<<endl

<<"--------------------------------------"<<endl

<<" \*\*\*ptr address:\*\*\* " <<&ptr<<endl

<<"--------------------------------------"<<endl

<<" ptr content: "<<ptr->next<<endl

<<"--------------------------------------"<<endl

<<"ptr -> data: " << ptr -> data <<endl

<<"--------------------------------------"<<endl

;

ptr = head->next;

// for printing the data

while (ptr != NULL) {

cout<<"ptr: " << ptr <<endl

<<"ptr -> next: " << ptr -> next <<endl

<<"ptr -> data: " << ptr -> data <<endl

<<"--------------------------------------"<<endl

;

ptr = ptr -> next;

}

}

};

int main() {

Linklist list;

list.insertAtEnd(1);

list.insertAtEnd(2);

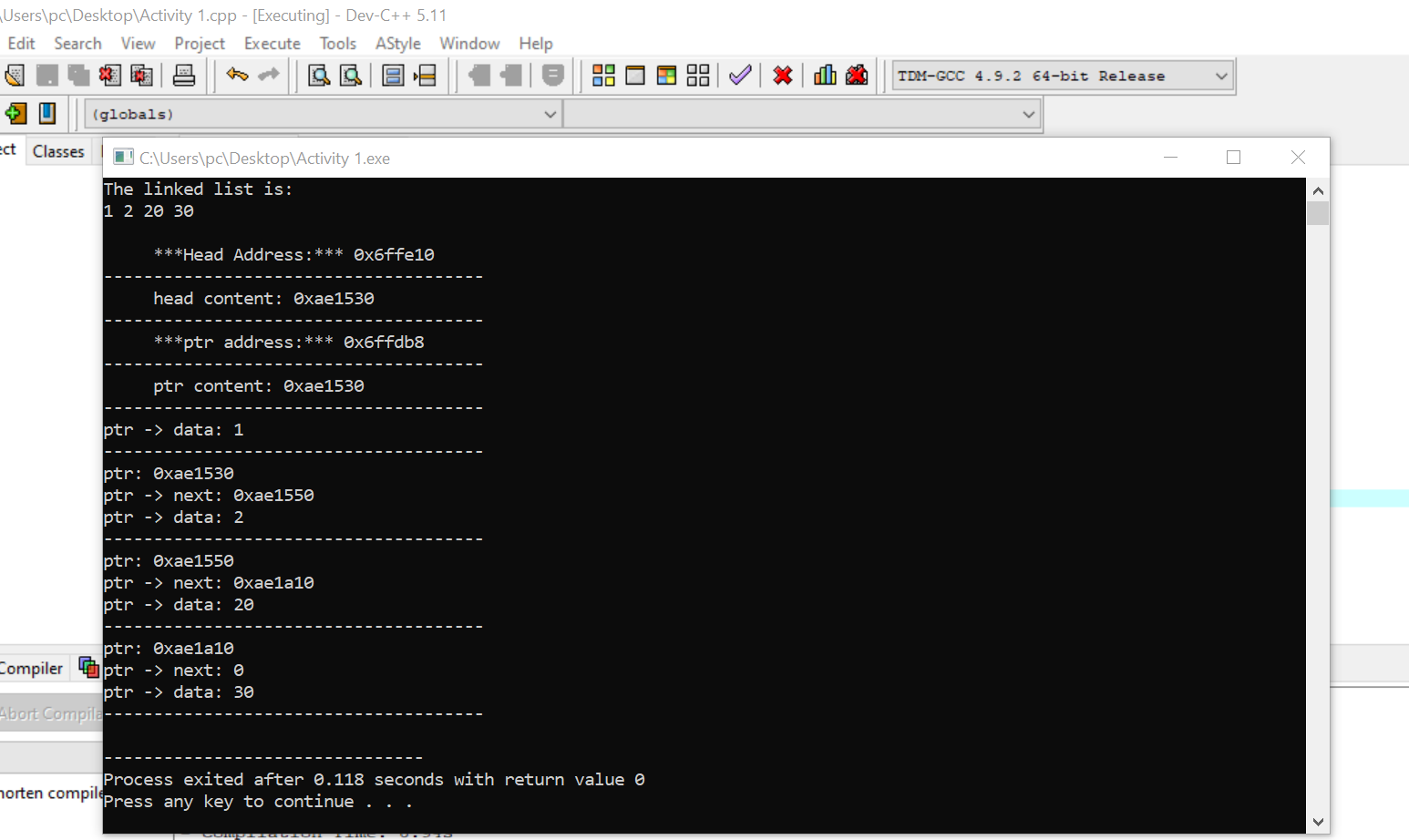
list.insertAtEnd(20);

list.insertAtEnd(30);

list.display();

}

**Screenshorts :**



**Activity # 2**

**Code :**

#include<iostream>

using namespace std;

class Node {

public:

int data;

Node\* next;

Node(int data) {

this->data = data;

this->next = NULL;

}

};

class SinglyLinkedList {

public:

Node\* head;

Node\* tail;

SinglyLinkedList() {

head = tail = NULL;

}

void insertAtHead(int d) {

Node\* temp = new Node(d);

temp->next = head;

head = temp;

}

void insertAtTail(int d) {

Node\* temp = new Node(d);

if (tail == NULL) {

head = tail = temp;

} else {

tail->next = temp;

tail = temp;

}

}

void print() {

Node\* temp = head;

while (temp != NULL) {

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

temp = temp->next;

}

cout << endl;

}

void insertAtPosition(int position, int d) {

if (position == 1) {

insertAtHead(d);

return;

}

Node\* temp = head;

int cnt = 1;

while (cnt < position - 1 && temp != NULL) {

temp = temp->next;

cnt++;

}

if (temp == NULL) {

insertAtTail(d);

} else {

Node\* nodeToInsert = new Node(d);

nodeToInsert->next = temp->next;

temp->next = nodeToInsert;

}

}

void deleteNode(int position) {

if (position == 1) {

if (head == NULL) return;

Node\* temp = head;

head = head->next;

temp->next = NULL;

delete temp;

} else {

Node\* curr = head;

Node\* prev = NULL;

int cnt = 1;

while (cnt < position && curr != NULL) {

prev = curr;

curr = curr->next;

cnt++;

}

if (curr != NULL) {

prev->next = curr->next;

curr->next = NULL;

delete curr;

}

}

}

void reverse() {

Node\* current = head;

Node \*prev = NULL, \*next = NULL;

while (current != NULL) {

next = current->next;

current->next = prev;

prev = current;

current = next;

}

head = prev;

}

};

class Node2 {

public:

int data;

Node2\* prev;

Node2\* next;

Node2(int d) {

this->data = d;

this->prev = NULL;

this->next = NULL;

}

};

class DoubleLinkedList {

public:

Node2\* head;

Node2\* tail;

DoubleLinkedList(Node2\* node) {

this->head = node;

this->tail = node;

}

void print() {

Node2\* temp = head;

while (temp != NULL) {

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

temp = temp->next;

}

cout << endl;

}

void insertAtHead(int d) {

Node2\* temp = new Node2(d);

temp->next = head;

head->prev = temp;

head = temp;

}

void insertAtTail(int d) {

Node2\* temp = new Node2(d);

tail->next = temp;

temp->prev = tail;

tail = temp;

}

void deleteNode(int position) {

if (position == 1) {

Node2\* temp = head;

temp->next->prev = NULL;

head = temp->next;

temp->next = NULL;

delete temp;

}

else {

Node2\* curr = head;

Node2\* prev = NULL;

int cnt = 1;

while (cnt < position) {

prev = curr;

curr = curr->next;

cnt++;

}

curr->prev = NULL;

prev->next = curr->next;

curr->next = NULL;

delete curr;

}

}

void reverse() {

Node2\* current = head;

Node2\* temp = NULL;

while (current != NULL) {

temp = current->prev;

current->prev = current->next;

current->next = temp;

current = current->prev;

}

if (temp != NULL) {

head = temp->prev;

}

}

void insertAtPosition(int position, int d) {

// Insert at start

if (position == 1) {

insertAtHead(d);

return;

}

Node2\* temp = head;

int cnt = 1;

while (cnt < position - 1) {

temp = temp->next;

cnt++;

}

// Insert at end

if (temp->next == NULL) {

insertAtTail(d);

return;

}

// Creating a node for d

Node2\* nodeToInsert = new Node2(d);

nodeToInsert->next = temp->next;

temp->next->prev = nodeToInsert;

temp->next = nodeToInsert;

nodeToInsert->prev = temp;

}

};

class Node3 {

public:

int data;

Node3\* next;

// constructor

Node3(int d) {

this->data = d;

this->next = NULL;

}

~Node3() {

int value = this->data;

if (this->next != NULL) {

next->next = NULL; // Set next to NULL before deleting to avoid potential issues

delete next;

next = NULL;

}

cout << " Memory is freed for node with data " << value << endl;

}

};

class CircularLinkedList {

private:

Node3\* tail;

public:

// Constructor

CircularLinkedList() : tail(NULL) {}

// Destructor

~CircularLinkedList() {

if (tail != NULL) {

delete tail;

tail = NULL;

}

}

// Function to insert a node

void insertNode(int element, int d) {

// Empty list

if (tail == NULL) {

Node3\* newNode = new Node3(d);

tail = newNode;

newNode->next = newNode;

}

else {

// Non-empty list

// Assuming that the element is present in the list

Node3\* curr = tail;

while (curr->data != element) {

curr = curr->next;

}

Node3\* temp = new Node3(d);

temp->next = curr->next;

curr->next = temp;

}

}

// Function to print the circular linked list

void print() {

Node3\* temp = tail;

// Empty list

if (tail == NULL) {

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

return;

}

do {

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

temp = temp->next;

} while (temp != tail);

cout << endl;

}

// Function to delete a node

void deleteNode(int value) {

// Empty list

if (tail == NULL) {

cout << " List is empty, please check again" << endl;

return;

}

else {

// Non-empty

// Assuming that "value" is present in the Linked List

Node3\* prev = tail;

Node3\* curr = prev->next;

while (curr->data != value) {

prev = curr;

curr = curr->next;

}

prev->next = curr->next;

// 1 Node Linked List

if (curr == prev) {

tail = NULL;

}

// >=2 Node linked list

else if (tail == curr) {

tail = prev;

}

curr->next = NULL;

delete curr;

}

}

// Function to reverse the circular linked list

void reverse() {

if (tail == NULL || tail->next == tail) {

// No or only one element in the list, no change needed

return;

}

Node3\* prev = NULL;

Node3\* current = tail->next;

Node3\* nextNode;

do {

nextNode = current->next;

current->next = prev;

prev = current;

current = nextNode;

} while (current != tail->next);

tail->next = prev; // Update the next of the tail to the new head of the reversed list

}

};

int main() {

SinglyLinkedList list;

Node2\* node1 = new Node2(10);

DoubleLinkedList doubleLinkedList(node1);

CircularLinkedList circularList;

while (true) {

int choice;

cout << "which link list did you want :" << endl;

cout << "1. Singly Linked List: " << endl;

cout << "2. Doubly Linked List: " << endl;

cout << "3. Circular Linked List:" << endl;

cout << "4. Exit" << endl;

cout << "Enter your choice: ";

cin >> choice;

if(choice==1) {

list.insertAtTail(20);

list.insertAtTail(30);

list.insertAtHead(10);

int n=0;

cout << "Choose a list and operation:" << endl;

cout << "1. Singly Linked List: Insert at head" << endl;

cout << "2. Singly Linked List: Insert at tail" << endl;

cout << "3. Singly Linked List: Insert at position" << endl;

cout << "4. Singly Linked List: reverse " << endl;

cout << "5. Singly Linked List: Delete a node " << endl;

cout << "6. Singly Linked List: Print list" << endl;

cin >> n ;

if(n==1){

int d;

cout<<"enter a value"<<endl;

cin>> d;

list.insertAtHead(d);

list.print();

}

else if(n==2){

int d;

cout<<"enter a value"<<endl;

cin>> d;

list.insertAtTail(d);

list.print();

}

else if(n==3){

int p;

int d;

cout<<"enter a position"<<endl;

cin>> p;

cout<<"enter a data"<<endl;

cin>>d;

list.insertAtPosition(p, d);

list.print();

}

else if(n==4){

list.reverse();

cout << "Reversed Linked List: ";

list.print();

}

else if(n==5){

int p;

cout<<"enter a position"<<endl;

cin>> p;

list.deleteNode(p);

list.print();

}

else if(n==6){

list.print();

}

else{

cout<<"invalid choice"<<endl;

}

cout<<" "<<endl;

}

else if (choice==2)

{

doubleLinkedList.insertAtHead(11);

doubleLinkedList.insertAtHead(12);

doubleLinkedList.insertAtTail(25);

int n=0;

cout << "Choose a list and operation:" << endl;

cout << "1.Double Linked List: Insert at head" << endl;

cout << "2.Double Linked List: Insert at tail" << endl;

cout << "3.Double Linked List: Insert at position" << endl;

cout << "4.Double Linked List: reverse " << endl;

cout << "5.Double Linked List: Delete a node " << endl;

cout << "6.Double Linked List: Print list" << endl;

cin >> n ;

if(n==1){

int d;

cout<<"enter a value"<<endl;

cin>> d;

doubleLinkedList.insertAtHead(d);

doubleLinkedList.print();

}

else if(n==2){

int d;

cout<<"enter a value"<<endl;

cin>> d;

doubleLinkedList.insertAtTail(d);

doubleLinkedList.print();

}

else if(n==3){

// Insert at position 2 (index starts from 1)

int p;

int d;

cout<<"enter a position"<<endl;

cin>> p;

cout<<"enter a data"<<endl;

cin>>d;

doubleLinkedList.insertAtPosition(p, d);

doubleLinkedList.print();

}

else if(n==4){

// Reverse the doubly linked list

doubleLinkedList.reverse();

cout << "Reversed Linked List: ";

doubleLinkedList.print();

}

else if(n==5){

// Assuming you want to delete the node at position 2 (index starts from 1)

int p;

cout<<"enter a position"<<endl;

cin>> p;

doubleLinkedList.deleteNode(p);

doubleLinkedList.print();

}

else if(n==6){

doubleLinkedList.print();

}

else{

cout<<"invalid choice"<<endl;

}

cout<<" "<<endl;

}

else if(choice==3){

circularList.insertNode(NULL, 3);

circularList.insertNode(3, 5);

circularList.insertNode(5, 7);

int n=0;

cout << "Choose a list and operation:" << endl;

cout << "1.Circular Linked List: Insert a node " << endl;

cout << "2.Circular Linked List: delete a node " << endl;

cout << "3.Circular Linked List: Print list" << endl;

cout << "4.CircularLinked List: reverse " << endl;

cin >> n ;

if(n==1){

int e;

int d;

cout<<"enter a element"<<endl;

cin>> e;

cout<<"enter a value"<<endl;

cin>> d;

circularList.insertNode(e, d);

circularList.print();

}

else if(n==2){

int d;

cout<<"enter a value you want delete "<<endl;

cin>> d;

circularList.deleteNode(d);

circularList.print();

}

else if(n==3){

circularList.print();

}

else if(n==4){

circularList.reverse();

cout << "Reversed Linked List: ";

circularList.print();

}

else{

cout<<"invalid choice"<<endl;

}

cout<<" "<<endl;

}

else if(choice==4){

return 0;

}

else{

cout<<"invalid choice"<<endl;

}

cout<<" "<<endl;

}

}

**Sreenshorts :**

