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

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

**Subject: Data Structures and Algorithms-Lab Instructor: Yasmeen Jana Max Marks: 20 Reg. No: SP22-BCS-022**

**Q#1**

#include <iostream>

using namespace std;

// Node structure for a singly linked list

struct Node {

int data;

Node\* next;

};

// Function to display the linked list with memory addresses and content

void displayLinkedList(Node\* head) {

Node\* ptr = head;

cout << "The linked list is: ";

while (ptr !=NULL) {

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

ptr = ptr->next;

}

cout << endl;

ptr = head;

cout << "\*\*\*\*head address:\*\*\* " << &head << endl;

cout << "head content: " << head << endl;

while (ptr != NULL) {

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

cout << "ptr content: " << ptr << endl;

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

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

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

ptr = ptr->next;

}

}

int main() {

// Create a sample linked list

Node\* head = NULL;

Node\* second = NULL;

Node\* third = NULL;

Node\* fourth = NULL;

head = new Node;

second = new Node;

third = new Node;

fourth = new Node;

head->data = 1;

head->next = second;

second->data = 2;

second->next = third;

third->data = 20;

third->next = fourth;

fourth->data = 30;

fourth->next = NULL;

// Display the linked list using the function

displayLinkedList(head);

// Clean up memory (free the allocated nodes)

delete head;

delete second;

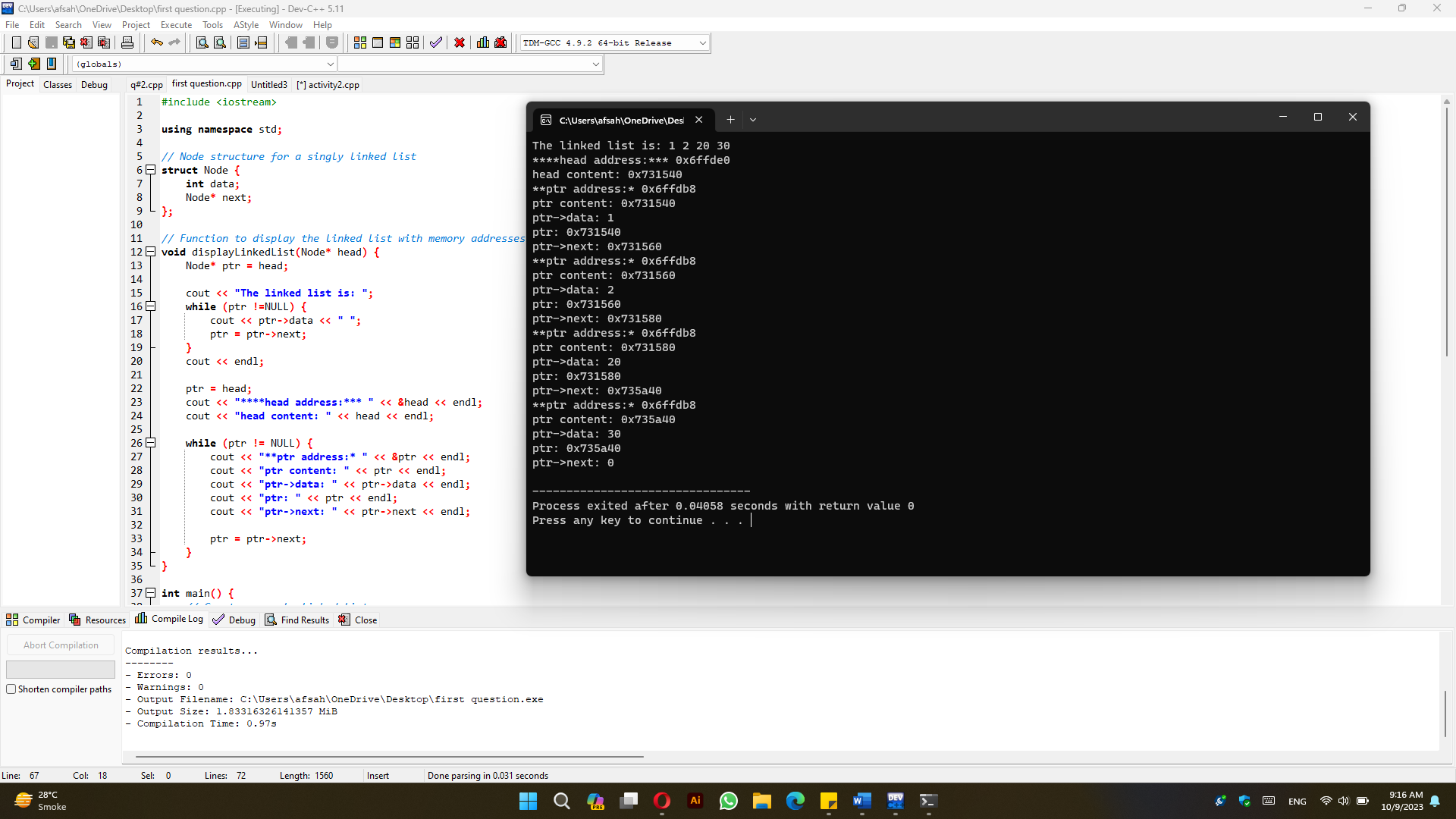
delete third;

delete fourth;

return 0;

}

**OUTPUT:**



**Q#2**

#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 Singly {

private: Node \* start;

public: Singly() {

start = 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 (start == NULL) {

start = newNode;

} else {

Node \* currentNode = start;

while (currentNode -> next != NULL) {

currentNode = currentNode -> next;

}

currentNode -> next = newNode;

}

}

//delete at the end

void deleteAtEnd() {

Node \* temp1 = start, \* temp2;

while (temp1 -> next != NULL) {

temp2 = temp1;

temp1 = temp1 -> next;

}

temp2 -> next = NULL;

delete temp1;

}

// function to insert at start

void insertAtStart(int val) {

// creating the node

Node \* newNode = new Node;

newNode -> data = val;

newNode -> next = NULL;

newNode -> next = start;

start = newNode;

}

// function to delete at start

void deleteAtStart() {

Node \* temp = new Node;

temp = start;

start = temp -> next;

delete temp;

}

// insert at given index

void insertAtDataNode(int val, int data) {

Node \* newNode = new Node;

newNode -> data = val;

newNode -> next = NULL;

Node \*current =start;

if(current == NULL){

cout<<"List is empty\n";

return;

}

else if(current->data == data){

start = newNode;

newNode->next= current;

}

else{

current = start;

while(data !=current->next->data){

if(current->next->next==NULL){

cout<<"No data found\n";

return;

}

current = current->next;

}

newNode->next=current->next;

current->next=newNode;

}

}

// delete at specific index

void deleteDataIndex(int data) {

Node \*currentNode = start,\*temp;

if(currentNode==NULL){

cout<<"Empty list";

return;

}

else if(currentNode->data == data){

start = currentNode->next;

delete currentNode;

currentNode =NULL;

}

else{

while(currentNode->next->data != data){

if(currentNode->next->next==NULL){

cout<<"No data found";

return;

}

currentNode = currentNode->next;

}

temp = currentNode->next;

currentNode->next=currentNode->next->next;

delete temp;

temp = NULL;

}

}

void reverse()

{

// Initialize current, previous and next pointers

Node\* current = start;

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

while (current != NULL) {

// Store next

next = current->next;

// Reverse current node's pointer

current->next = prev;

// Move pointers one position astart.

prev = current;

current = next;

}

start = prev;

}

// Function to display the contents of the list

void display() {

Node \* current = start;

while (current != NULL) {

cout << current -> data << " ";

current = current -> next;

}

cout << endl;

}

};

struct NPNode{

int data;

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

NPNode \* prev;

};

class Doubly {

private: NPNode \* start;

public: Doubly() {

start = NULL;

}

//insert at the end

void insertAtEnd(int val) {

NPNode \* newNode = new NPNode;

newNode -> data = val;

newNode -> next = NULL;

newNode -> prev =NULL;

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

if (start == NULL) {

start = newNode;

} else {

NPNode \* currentNode = start;

while (currentNode -> next != NULL) {

currentNode = currentNode -> next;

}

currentNode -> next = newNode;

newNode->prev = currentNode;

}

}

//delete at the end

void deleteAtEnd() {

NPNode \* currentNode = start;

if(currentNode==NULL){

cout<<"List is empty";

return;

}

else if(currentNode->next==NULL){

delete currentNode;

currentNode=NULL;

}

else{

while (currentNode->next->next !=NULL){

currentNode = currentNode->next;

}

delete currentNode->next;

currentNode->next =NULL;

}

}

// function to insert at start

void insertAtStart(int val) {

// creating the node

NPNode \* newNode = new NPNode;

newNode -> data = val;

newNode -> next = NULL;

newNode ->prev = NULL;

newNode -> next = start;

start->prev=newNode;

start = newNode;

}

// function to delete at start

void deleteAtStart() {

NPNode \* temp = new NPNode;

temp = start;

start = temp -> next;

delete temp;

}

// insert at given index

void insertAtDataNode(int val, int data) {

NPNode \* newNode = new NPNode;

newNode -> data = val;

newNode -> next = NULL;

newNode ->prev = NULL;

NPNode \*current =start;

if(current == NULL){

cout<<"List is empty\n";

return;

}

else if(current->data == data){

start = newNode;

newNode->next= current;

current->prev = newNode;

}

else{

current = start;

while(data !=current->next->data){

if(current->next->next==NULL){

cout<<"No data found\n";

return;

}

current = current->next;

}

newNode->next=current->next;

current->next->prev =newNode;

current->next=newNode;

newNode->prev = current;

}

}

// delete at specific index

void deleteDataIndex(int data) {

NPNode \*currentNode = start,\*temp;

if(currentNode==NULL){

cout<<"Empty list";

return;

}

else if(currentNode->data == data){

start = currentNode->next;

currentNode->next->prev = NULL;

delete currentNode;

currentNode =NULL;

}

else{

while(currentNode->next->data != data){

if(currentNode->next->next==NULL){

cout<<"No data found";

return;

}

currentNode = currentNode->next;

}

temp = currentNode->next;

// checking if the current node's next is the last element or not

if(currentNode->next->next!=NULL){

delete temp;

temp = NULL;

}

else{

currentNode->next->prev = currentNode;

currentNode->next=currentNode->next->next;

delete temp;

temp = NULL;

}

}

}

void reverse(){

NPNode\* temp = NULL;

NPNode\* current = start;

/\* swap next and prev for all nodes of

doubly linked list \*/

while (current != NULL) {

temp = current->prev;

current->prev = current->next;

current->next = temp;

current = current->prev;

}

/\* Before changing head, check for the cases like empty

list and list with only one node \*/

if (temp != NULL)

start = temp->prev;

}

// Function to display the contents of the list

void display() {

NPNode \* current = start;

while (current != NULL) {

cout << current -> data << " ";

current = current -> next;

}

cout << endl;

}

};

class Circular {

private: Node \* start;

public: Circular() {

start = NULL;

}

//insert at the end

void insertAtEnd(int val) {

Node \* newNode = new Node;

newNode -> data = val;

newNode -> next = start;

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

if (start == NULL) {

start = newNode;

newNode->next=start; // For a circular list, the only node should point to itself.

} else {

Node \* currentNode = start;

while (currentNode -> next != start) {

currentNode = currentNode -> next;

}

currentNode -> next = newNode;

}

}

//delete at the end

void deleteAtEnd() {

Node \* temp1 = start, \* temp2;

while (temp1 -> next != start) {

temp2 = temp1;

temp1 = temp1 -> next;

}

temp2 -> next = start;

delete temp1;

}

// function to insert at start

void insertAtStart(int val) {

// creating the node

Node \* newNode = new Node;

newNode -> data = val;

newNode -> next = start;

start = newNode;

}

// function to delete at start

void deleteAtStart() {

Node \* temp = new Node;

temp = start;

start = temp -> next;

Node \*currentNode = temp;

while(currentNode->next != temp){

currentNode=currentNode->next;

}

currentNode->next=start;

delete temp;

}

// insert at given index

void insertAtDataNode(int val, int data) {

Node \* newNode = new Node;

newNode -> data = val;

Node \*current =start, \*temp = start;

if(current == NULL){

cout<<"List is empty\n";

return;

}

else if(current->data == data){

start = newNode;

newNode->next= current;

}

else{

current = start;

while(data !=current->next->data){

if(current->next->next==start){

cout<<"No data found\n";

return;

}

current = current->next;

}

newNode->next=current->next;

current->next=newNode;

}

current = temp;

while(current->next != temp){

current=current->next;

}

current->next=start;

}

// delete at specific index

void deleteDataIndex(int data) {

Node \*currentNode = start,\*temp,\*p=start;

if(currentNode==NULL){

cout<<"Empty list";

return;

}

else if(currentNode->data == data){

start = currentNode->next;

// change the last node start to new start value

Node \* currentNode = p;

while (currentNode -> next != p) {

currentNode = currentNode -> next;

}

currentNode->next=start;

delete currentNode;

currentNode =NULL;

}

else{

while(currentNode->next->data != data){

if(currentNode->next->next==start){

cout<<"No data found";

return;

}

currentNode = currentNode->next;

}

temp = currentNode->next;

currentNode->next=currentNode->next->next;

delete temp;

temp = NULL;

}

}

void reverse()

{

// if list is empty

if (start == NULL)

return;

// reverse procedure same as reversing a

// singly linked list

Node\* prev = NULL;

Node\* current = start;

Node\* next;

do {

next = current->next;

current->next = prev;

prev = current;

current = next;

} while (current != (start));

// adjusting the links so as to make the

// last node point to the first node

(start)->next = prev;

start = prev;

}

// Function to display the contents of the list

void display() {

Node \* current = start;

while (current->next != start) {

cout << current -> data << " ";

current = current -> next;

}

cout<<current->data<<endl;

cout << endl;

}

};

void SinglyList(Singly list){

}

int main() {

Singly slist;

Doubly dlist;

Circular clist;

int num;

cout << "Which list do you want?\n1 Singly\n2 Doubly\n3 Circular\n";

cin >> num;

switch (num) {

case 1:

start:

int opNum;

cout << "Which operation do you want to perform:"

<< "\n1: Insertion"

<< "\n2: Deletion"

<< "\n3: Display"

<< "\n4: Reverse"

<< "\n5: Seek"

<< "\n6: Exit\n";

cin >> opNum;

switch (opNum) {

case 1:

int funNum, val, pos;

cout << "Where to Delate:\n1: Insertion at beginning\n2: Insertion at end\n3: Insertion at a specific data node\n";

cin >> funNum;

cout << "Enter value: ";

cin >> val;

switch (funNum) {

case 1:

slist.insertAtStart(val);

goto start;

break;

case 2:

slist.insertAtEnd(val);

goto start;

break;

case 3:

cout << "Data Node: ";

cin >> pos;

slist.insertAtDataNode(val, pos);

goto start;

break;

default:

break;

}

break;

case 2:

int deleteFunNum;

cout << "Where to Delate:\n1: Deletion at beginning\n2: Deletion at end\n3: Deletion at a specific data node\n";

cin >> deleteFunNum;

switch (deleteFunNum) {

case 1:

slist.deleteAtStart();

goto start;

break;

case 2:

slist.deleteAtEnd();

goto start;

break;

case 3:

cout << "Data Node: ";

cin >> pos;

slist.deleteDataIndex(pos);

goto start;

break;

default:

break;

}

break;

case 3:

slist.display();

goto start;

break;

case 4:

slist.reverse();

goto start;

break;

default:

break;

}

break;

//case 2 for doubly

case 2:

Dstart:

cout << "Which operation do you want to perform:"

<< "\n1: Insertion"

<< "\n2: Deletion"

<< "\n3: Display"

<< "\n4: Reverse"

<< "\n5: Seek"

<< "\n6: Exit\n";

cin >> opNum;

switch (opNum) {

case 1:

int funNum, val, pos;

cout << "Where to Delate:\n1: Insertion at beginning\n2: Insertion at end\n3: Insertion at a specific data node\n";

cin >> funNum;

cout << "Enter value: ";

cin >> val;

switch (funNum) {

case 1:

dlist.insertAtStart(val);

goto Dstart;

break;

case 2:

dlist.insertAtEnd(val);

goto Dstart;

break;

case 3:

cout << "Data Node: ";

cin >> pos;

dlist.insertAtDataNode(val, pos);

goto Dstart;

break;

default:

break;

}

break;

case 2:

int deleteFunNum;

cout << "Where to Delate:\n1: Deletion at beginning\n2: Deletion at end\n3: Deletion at a specific data node\n";

cin >> deleteFunNum;

switch (deleteFunNum) {

case 1:

dlist.deleteAtStart();

goto Dstart;

break;

case 2:

dlist.deleteAtEnd();

goto Dstart;

break;

case 3:

cout << "Data Node: ";

cin >> pos;

dlist.deleteDataIndex(pos);

goto Dstart;

break;

default:

break;

}

break;

case 3:

dlist.display();

goto Dstart;

break;

case 4:

dlist.reverse();

goto Dstart;

break;

default:

break;

}

break;

// case 3 for circular

case 3:

Cstart:

cout << "Which operation do you want to perform:"

<< "\n1: Insertion"

<< "\n2: Deletion"

<< "\n3: Display"

<< "\n4: Reverse"

<< "\n5: Seek"

<< "\n6: Exit\n";

cin >> opNum;

switch (opNum) {

case 1:

int funNum, val, pos;

cout << "Where to Delate:\n1: Insertion at beginning\n2: Insertion at end\n3: Insertion at a specific data node\n";

cin >> funNum;

cout << "Enter value: ";

cin >> val;

switch (funNum) {

case 1:

clist.insertAtStart(val);

goto Cstart;

break;

case 2:

clist.insertAtEnd(val);

goto Cstart;

break;

case 3:

cout << "Data Node: ";

cin >> pos;

clist.insertAtDataNode(val, pos);

goto Cstart;

break;

default:

break;

}

break;

case 2:

int deleteFunNum;

cout << "Where to Delate:\n1: Deletion at beginning\n2: Deletion at end\n3: Deletion at a specific data node\n";

cin >> deleteFunNum;

switch (deleteFunNum) {

case 1:

clist.deleteAtStart();

goto Cstart;

break;

case 2:

clist.deleteAtEnd();

goto Cstart;

break;

case 3:

cout << "Data Node: ";

cin >> pos;

clist.deleteDataIndex(pos);

goto Cstart;

break;

default:

break;

}

break;

case 3:

clist.display();

goto Cstart;

break;

case 4:

clist.reverse();

goto Cstart;

break;

default:

break;

}

break;

default:

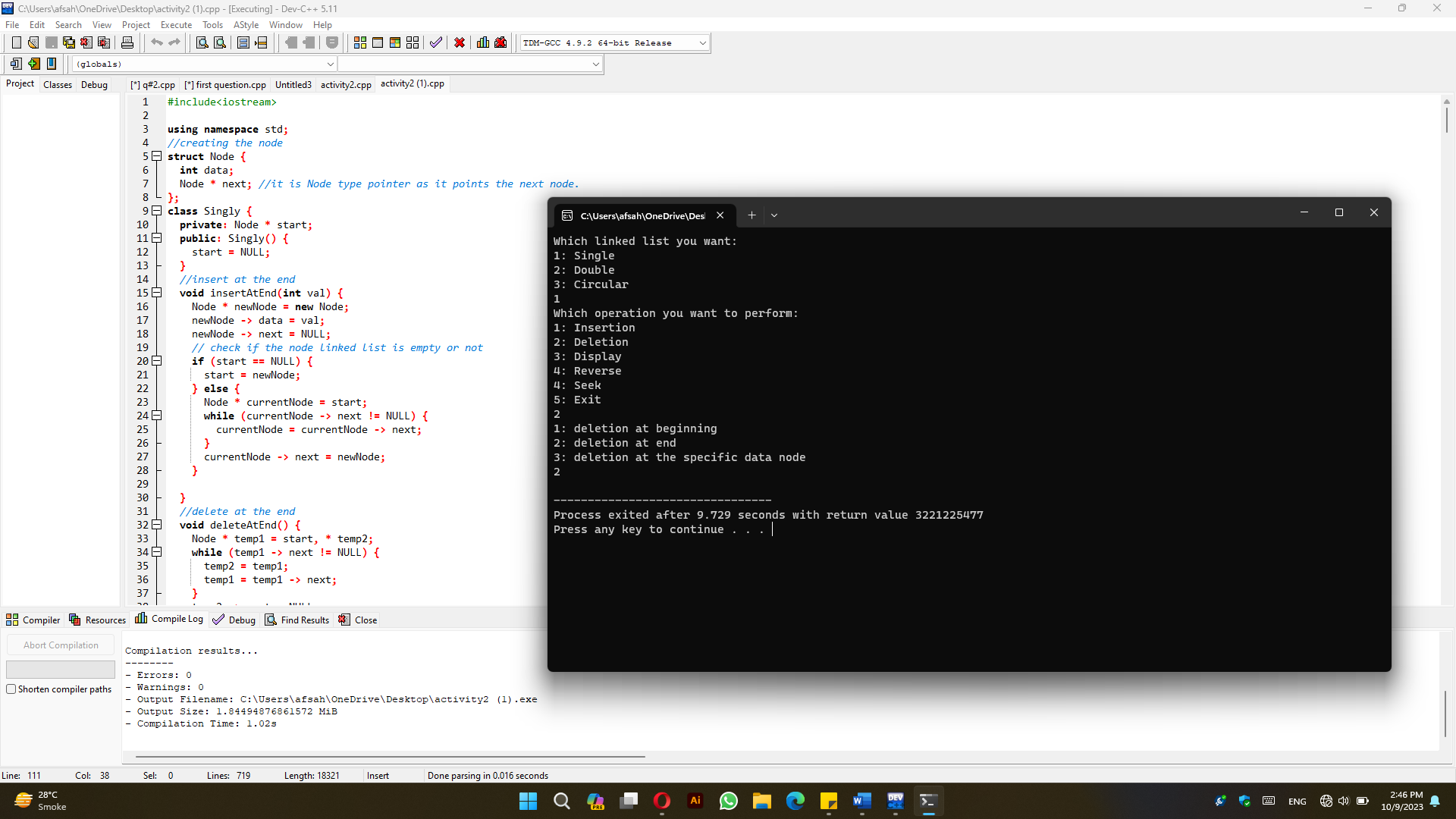
break;

}

return 0;

}

**OUTPUT:**

****