**Name: Saba Zulfiqar**

**Roll NO :SU92-BSSEM-S24-107**

**Section : 3A**

**Lab 1:**

**Task: Create a program that declares an integer variable and a pointer to it. Modify the value of the variable using**  **the pointer and display both the variable and pointer values.**

#include<iostream>

using namespace std;

int main(){

int num = 10 ;

int \*ptr = &num;

cout<<"initial val of num = "<< num <<endl;

cout<<"pointer point to address = "<< ptr <<endl;

cout<<"val at ptr = "<< \*ptr <<endl;

\*ptr = 20;

cout<< "After using pointer : "<<endl;

cout<<"initial val of num = "<< num <<endl;

cout<<"pointer still point to address = "<< ptr <<endl;

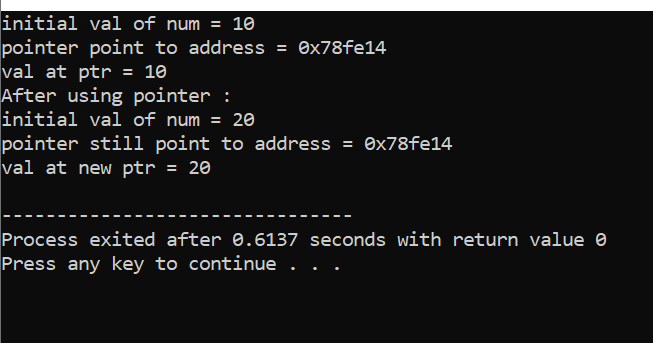
cout<<"val at new ptr = "<< \*ptr <<endl;

}

**EXPLANATION:**

This C++ program shows how pointers work by linking a variable to its memory address. First, we declare an integer num and set it to 10. Then, we create a pointer ptr that stores the address of num. We print the initial values to see what’s happening. Next, we change the value of num using \*ptr, setting it to 20. Since ptr directly points to num, modifying \*ptr updates num as well. Finally, we print everything again to confirm that the change took effect. This shows how pointers allow us to manipulate variables indirectly through memory addresses.

**Output:**



**Lab 2 :**

#include <iostream>

using namespace std;

int findMax(int arr[], int n) {

if (n == 0) {

cout << "Array is empty!" << endl;

return -1;

}

int maxVal = arr[0];

for (int i = 1; i < n; i++) {

if (arr[i] > maxVal) {

maxVal = arr[i];

}

}

return maxVal;

}

int main() {

int arr[] = {10, 45, 3, 89, 29, 56};

int n = sizeof(arr) / sizeof(arr[0]);

cout << "The largest number in the array is: " << findMax(arr, n) << endl;

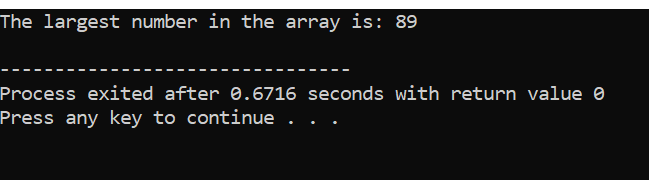
return 0;

}

**EXPLANATION:**

This C++ program finds the largest number in an array. First, it defines a function findMax that takes an array and its size as inputs. It initializes maxVal with the first element and then loops through the rest of the array, updating maxVal whenever a larger number is found. If the array is empty, it prints a message and returns -1. In the main function, an array is defined, and its size is calculated. Finally, the program calls findMax and prints the largest number found in the array.

**Output:**



**Lab 3 :**

**Singly Linked List (Insert at End, Insert at Start)**

**Task: Implement a singly linked list with functions to insert a node at the start and at the end.**

**Display the list after each insertion.**

#include <iostream>

using namespace std;

class Node{

public:

int data;

Node\* next;

Node(int d){

data = d;

next = NULL;

}

};

class LinkedList {

public:

Node\* head;

LinkedList() {

head = NULL;

}

void insertATstart(int d){

Node\* newNode = new Node(d);

newNode->next = head;

head = newNode;

display();

}

void insertATend(int d){

Node\* newNode = new Node(d);

if(head==NULL){

head = newNode;

}

else{

Node\* temp = head;

while(temp->next!=NULL){

temp= temp->next;

}

temp->next = newNode;

}

display();

}

void display(){

Node\* temp = head;

while (temp!=NULL){

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

temp= temp->next;

}

cout << "NULL" << endl;

}

};

int main(){

LinkedList list;

cout<<"Insert at start"<<endl;

list.insertATstart(10);

list.insertATstart(20);

list.insertATstart(30);

cout<<"Insert at end"<<endl;

list.insertATend(40);

list.insertATend(50);

list.insertATend(60);

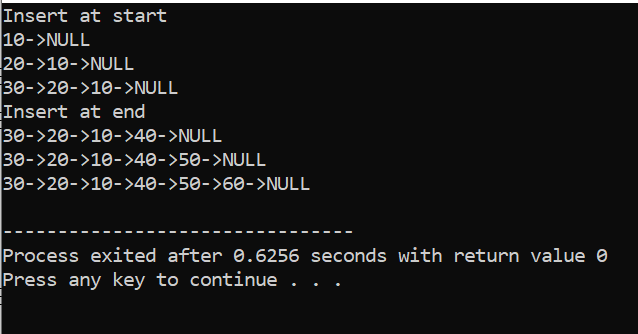
return 0;

}

**EXPLANATION:**

This program implements a **singly linked list** with insertion at the start and end. The Node class represents a list node with data and a pointer next. The LinkedList class manages the list, allowing insertion at the beginning (insertATstart) and at the end (insertATend). After each insertion, the display function prints the list. In main(), elements are added at both ends, demonstrating how nodes are dynamically linked. The output shows the updated list after every insertion.

**Output:**



**Lab 4 :**

**Singly Linked List (Insert at Specific Location)**

**Task: Write a function to insert a node at a specific position in a singly linked list,**

**ensuring valid position handling.**

#include <iostream>

using namespace std;

class Node{

public:

int data;

Node\* next;

Node(int d) {

data = d;

next = NULL;

}

};

class LinkedList{

public:

Node\* head;

LinkedList() {

head = NULL;

}

void insertATpos(int pos , int d){

Node\* newNode = new Node(d);

if(pos<0){

cout<<"invalid position"<<endl;

}

else if (pos==1){

newNode->next = head;

head = newNode;

return;

}

Node\* temp = head;

for(int i=1 ;i<pos-1; i++){

if(temp==NULL){

cout<<"INVALID POSITION"<<endl;

return;

}

temp = temp->next;

}

newNode -> next = temp -> next;

temp -> next = newNode;

}

void display() {

Node\* temp = head;

while (temp != NULL) {

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

temp = temp->next;

}

cout << "NULL" << endl;

}

};

int main(){

LinkedList list;

list.insertATpos(1,10);

list.insertATpos(2,20);

list.insertATpos(3,30);

list.display();

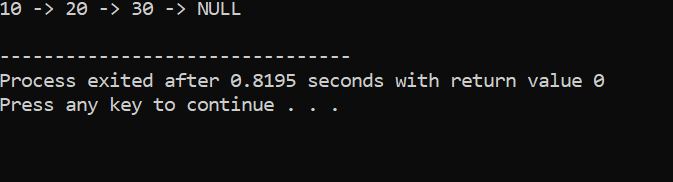
return 0;

}

**EXPLANATION:**

This program implements a **singly linked list** with insertion at a specific position. The Node class represents each node with data and a next pointer. The LinkedList class manages the list and provides the insertATpos function, which inserts a new node at the given position. If the position is invalid, an error message is displayed. The display function prints the list. In main(), nodes are inserted at positions **1, 2, and 3**, and the final list is displayed.

**Output:**



**Lab 5 :**

**Task: Implement functions to display the first node, last node, Nth node, and centre node of a singly linked list.**

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* next;

Node(int d) {

data = d;

next = NULL;

}

};

class LinkedList {

public:

Node\* head;

LinkedList() {

head = NULL;

}

void insertAtEnd(int d){

Node\* newNode = new Node(d);

if(head == NULL){

head = newNode;

return;

}

Node\* temp= head;

while(temp->next!=NULL){

temp = temp->next;

}

temp->next = newNode;

}

void display(){

Node\* temp = head;

while(temp != NULL){

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

temp = temp->next;

}

cout<<"NULL"<<endl;

}

void displayFirstNode(){

if (head != NULL){

cout<<"First node:"<<head ->data<<endl;

}

else

cout<<"list is empty" <<endl;

}

void displaylastNode(){

if (head == NULL){

cout<<"list is empty" <<endl;

return;

}

Node\* temp= head;

while(temp->next!=NULL){

temp = temp->next;

}

cout<<"last node:"<<temp ->data<<endl;

}

void displayNthNode(int n) {

Node\* temp = head;

int count = 1;

while (temp != NULL && count < n) {

temp = temp->next;

count++;

}

if (temp != NULL)

cout << "Node at position " << n << ": " << temp->data << endl;

else

cout << "Position " << n << " is out of bounds!" << endl;

}

void displayCenterNode() {

if (head == NULL) {

cout << "List is empty!" << endl;

return;

}

Node\* slow = head;

Node\* fast = head;

while (fast != NULL && fast->next != NULL) {

slow = slow->next;

fast = fast->next->next;

}

cout << "Center Node: " << slow->data << endl;

}

};

int main() {

LinkedList list;

list.insertAtEnd(10);

list.insertAtEnd(20);

list.insertAtEnd(30);

list.insertAtEnd(40);

list.insertAtEnd(50);

list.display();

list.displayFirstNode();

list.displaylastNode();

list.displayNthNode(3);

list.displayCenterNode();

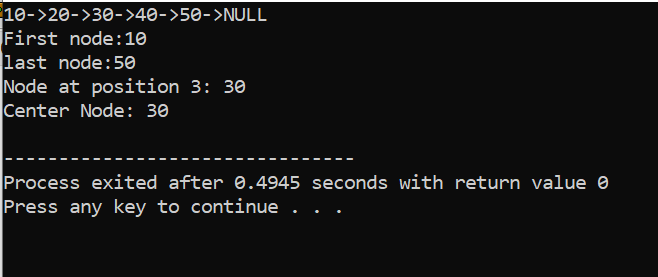
return 0;

}

**EXPLANATION:**

This program implements a **singly linked list** with functions to display specific nodes. The Node class represents each node with data and a next pointer. The LinkedList class allows inserting nodes at the end and provides functions to display the **first node, last node, Nth node, and center node**. The center node is found using the **slow and fast pointer approach**. In main(), nodes are inserted, and these functions are called to print specific nodes, demonstrating how linked lists are traversed and accessed efficiently.

**Output:**



**Lab 6 :**

**Singly Linked List (Delete Nodes)**

**Task: Implement functions to delete the first node, last node, Nth node, and center node of a singly linked list.**

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* next;

Node(int d) {

data = d;

next = NULL;

}

};

class LinkedList {

public:

Node\* head;

LinkedList() {

head = NULL;

}

void insert\_at\_end(int d) {

Node\* newNode = new Node(d);

if (head == NULL) {

head = newNode;

return;

}

Node\* temp = head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

}

void display() {

Node\* temp = head;

while (temp != NULL) {

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

temp = temp->next;

}

cout << "NULL" << endl;

}

void deletefirstnode(){

if (head == NULL){

cout<<"list is empty . cannot del"<<endl;

return;

}

Node\* temp = head;

head = head->next;

delete temp;

cout<<"first node delete"<<endl;

}

void deletelastnode(){

if (head == NULL){

cout<<"list is empty . cannot del"<<endl;

return;

}

if (head->next == NULL) {

delete head;

head = NULL;

cout << "Last node deleted." << endl;

return;

}

Node\* temp = head;

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

temp = temp->next;

}

delete temp->next;

temp->next = NULL;

cout<<"last node deleted"<<endl;

}

void deleteNthnode(int n) {

if (head == NULL) {

cout << "List is empty! Cannot delete." << endl;

return;

}

if (n == 1) {

deletefirstnode();

return;

}

Node\* temp = head;

for (int i = 1; temp != NULL && i < n - 1; i++) {

temp = temp->next;

}

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

cout << "Invalid position!" << endl;

return;

}

Node\* nodeToDelete = temp->next;

temp->next = temp->next->next;

delete nodeToDelete;

cout << "Node at position " << n << " deleted." << endl;

}

void deleteCenternode() {

if (head == NULL) {

cout << "List is empty! Cannot delete." << endl;

return;

}

if (head->next == NULL) {

deletefirstnode();

return;

}

Node\* slow = head;

Node\* fast = head;

Node\* prev = NULL;

while (fast != NULL && fast->next != NULL) {

prev = slow;

slow = slow->next;

fast = fast->next->next;

}

prev->next = slow->next;

delete slow;

cout << "Center node deleted." << endl;

}

};

int main() {

LinkedList list;

list.insert\_at\_end(10);

list.insert\_at\_end(20);

list.insert\_at\_end(30);

list.insert\_at\_end(40);

list.insert\_at\_end(50);

cout << "Original List: " << endl;

list.display();

list.deletefirstnode();

cout << "After deleting first node: ";

list.display();

list.deletelastnode();

cout << "After deleting last node: ";

list.display();

list.deleteNthnode(2);

cout << "After deleting 2nd node: ";

list.display();

list.deleteCenternode();

cout << "After deleting center node: ";

list.display();

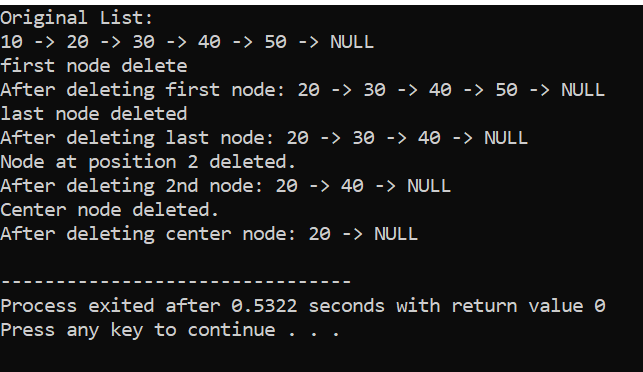
return 0;

}

**EXPLANATION:**

This program manages a **singly linked list** with functions to **delete the first node, last node, Nth node, and center node**. The LinkedList class handles node insertions and deletions. The deletefirstnode function removes the head node, while deletelastnode finds and deletes the last node. The deleteNthnode function deletes a node at a given position, and deleteCenternode removes the middle node using the **slow and fast pointer approach**. In main(), nodes are inserted, and various deletion functions are called, with the list displayed after each operation.

**Output:**



**Lab 7 :**

**Doubly Linked List (Insert & Display Nodes)**

**Task: Implement functions to insert node at first, last, Nth location, and centre of a doubly linked list. And display in order and display in reverse order.**

#include<iostream>

using namespace std;

class Node{

public:

int data;

Node\* prev;

Node\* next;

Node(int d){

data= d;

next=prev=NULL;

}

};

class DoublyList{

public:

Node\* head;

Node\* tail;

public:

DoublyList(){

head=tail=NULL;

}

void insertAtstart(int d){

Node\* newNode= new Node(d);

if(head==NULL){

head=tail=newNode;

}

else{

newNode->next = head;

head->prev= newNode;

head = newNode;

}

}

void insertAtend(int d){

Node\* newNode= new Node(d);

if(head==NULL){

head=tail=newNode;

}

else{

newNode->prev= tail;

tail->next= newNode;

tail = newNode;

}

}

void deleteAtstart(){

if(head == NULL){

cout<<"list is empty"<<endl;

return;

}

Node\*temp = head;

head = head->next;

if (head!=NULL){

head->prev = NULL;

}

else{

tail = NULL;

}

delete temp;

cout<<"first node delete:"<<endl;

}

void deleteAtend(){

if(tail == NULL){

cout<<"list is empty"<<endl;

return;

}

Node\*temp = tail;

tail= tail->prev;

if (tail!=NULL) {

tail->next = NULL;

}

else{

head = NULL;

}

delete temp;

cout<<"last node delete:"<<endl;

}

void displayForward() {

Node\* temp = head;

cout<<"Forward : ";

while (temp != NULL) {

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

temp = temp->next;

}

cout <<"NULL"<< endl;

}

void displayReverse() {

Node\* temp = tail;

cout<<"Reverse : ";

while (temp != NULL) {

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

temp = temp->prev;

}

cout << "NULL"<< endl;

}

};

int main(){

DoublyList dll;

dll.insertAtstart(6);

dll.insertAtstart(5);

dll.insertAtstart(4);

dll.displayForward();

dll.displayReverse();

dll.insertAtend(7);

dll.insertAtend(8);

dll.displayForward();

dll.displayReverse();

dll.deleteAtstart();

dll.displayForward();

dll.displayReverse();

dll.deleteAtend();

dll.displayForward();

dll.displayReverse();

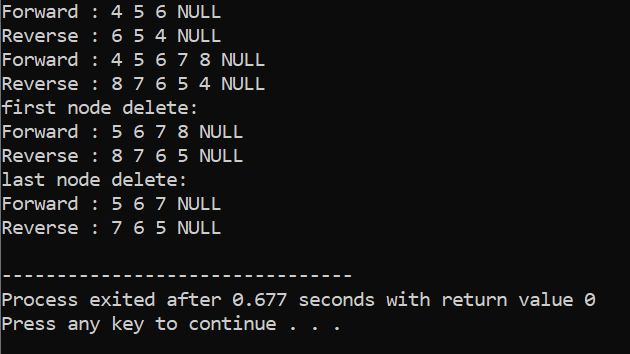
return 0;

}

**EXPLANATION:**

This program implements a **doubly linked list** with functions to **insert and delete nodes** at the start and end. The Node class represents each node with data, prev, and next pointers. The DoublyList class manages the list, allowing insertions (insertAtstart, insertAtend) and deletions (deleteAtstart, deleteAtend). The list can be displayed **forwards (**displayForward**) and in reverse (**displayReverse**)**. In main(), nodes are inserted at both ends, displayed in both directions, then deleted from both ends, with the updated list shown each time.

**Output:**



**Lab 8 :**

**Merge two LinkedLists**

**1. Create 2 Singly LinkedLists and Merge them and display them.**

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* next;

Node(int d) {

data = d;

next = nullptr;

}

};

//

class singlyLinkedList {

public:

Node\* head;

singlyLinkedList() {

head = NULL;

}

void insert\_at\_end(int d) {

Node\* newNode = new Node(d);

if (head == NULL) {

head = newNode;

return;

}

//

Node\* temp = head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

}

//

void display() {

Node\* temp = head;

while (temp != NULL) {

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

temp = temp->next;

}

cout << "NULL" << endl;

}

void merge(singlyLinkedList& list2) {

if (head == NULL){

head = list2.head;

return ;

}

Node\* temp = head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = list2.head;

}

};

int main(){

singlyLinkedList list1 , list2;

list1.insert\_at\_end(1);

list1.insert\_at\_end(3);

list1.insert\_at\_end(5);

list2.insert\_at\_end(2);

list2.insert\_at\_end(4);

list2.insert\_at\_end(6);

cout << "First Singly Linked List: ";

list1.display();

cout << "Second Singly Linked List: ";

list2.display();

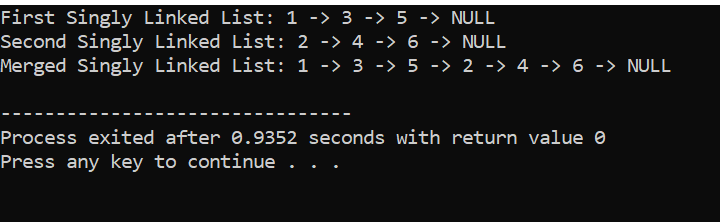
list1.merge(list2);

cout << "Merged Singly Linked List: ";

list1.display();

return 0;

}



**2. Create 2 Double LinkedLists and Merge them and display them.**

#include<iostream>

using namespace std;

class Node{

public:

int data;

Node\* prev;

Node\* next;

Node(int d){

data= d;

next=prev=NULL;

}

};

class DoublyLinkedList{

public:

Node\* head;

Node\* tail;

public:

DoublyLinkedList(){

head=tail=NULL;

}

void insertAtEnd(int d){

Node\* newNode= new Node(d);

if(head==NULL){

head=tail=newNode;

return;

}

newNode->prev= tail;

tail->next= newNode;

tail = newNode;

}

void displayForward() {

Node\* temp = head;

cout<<"Forward : ";

while (temp != NULL) {

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

temp = temp->next;

}

cout <<"NULL"<< endl;

}

void displayReverse() {

Node\* temp = tail;

cout<<"Reverse : ";

while (temp != NULL) {

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

temp = temp->prev;

}

cout << "NULL"<< endl;

}

void merge(DoublyLinkedList& list2) {

if (head == NULL) {

head = list2.head;

tail = list2.tail;

return;

}

if (list2.head == NULL) {

return;

}

tail->next = list2.head;

list2.head->prev = tail;

tail = list2.tail;

}

};

int main() {

DoublyLinkedList list1, list2;

list1.insertAtEnd(1);

list1.insertAtEnd(3);

list1.insertAtEnd(5);

list2.insertAtEnd(2);

list2.insertAtEnd(4);

list2.insertAtEnd(6);

cout << "First Doubly Linked List:" << endl;

list1.displayForward();

list1.displayReverse();

cout << "Second Doubly Linked List:" << endl;

list2.displayForward();

list2.displayReverse();

list1.merge(list2);

cout << "Merged Doubly Linked List:" << endl;

list1.displayForward();

list1.displayReverse();

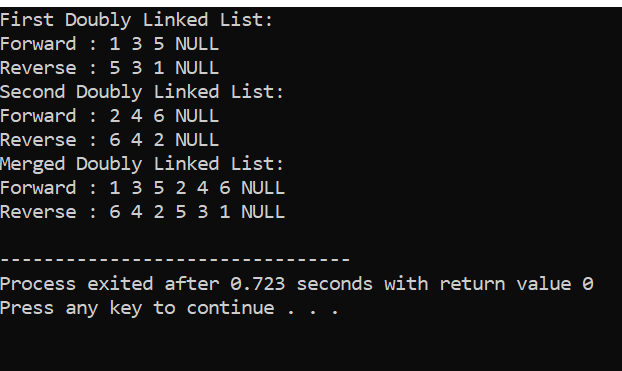
return 0;

}

**EXPLANATION:**

This program **merges two linked lists** (singly and doubly) and displays them. The singlyLinkedList class allows inserting nodes at the end and merging two singly linked lists by linking the last node of the first list to the head of the second. Similarly, the DoublyLinkedList class merges two doubly linked lists while maintaining the prev and next pointers. The main() function creates two linked lists, inserts nodes, displays them, merges them, and then displays the merged list in both **forward and reverse order** for the doubly linked list.

**Output:**



**Lab 9 :**

**Circular Linked List (Insert & Display Nodes)**

**Task: Implement functions to insert node at first, last, Nth location, and centre of a circular linked list. And display in order and display in reverse order.**

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* next;

Node\* prev;

Node(int d) {

data = d;

next = prev = NULL;

}

};

class CircularDoublyLinkedList {

public:

Node\* head;

CircularDoublyLinkedList() {

head = NULL;

}

void insertAtFirst(int d) {

Node\* newNode = new Node(d);

if (head == NULL) {

head = newNode;

head->next = head;

head->prev = head;

return;

}

Node\* tail = head->prev;

newNode->next = head;

newNode->prev = tail;

tail->next = newNode;

head->prev = newNode;

head = newNode;

}

void insertAtLast(int d) {

Node\* newNode = new Node(d);

if (head == NULL) {

insertAtFirst(d);

return;

}

Node\* tail = head->prev;

tail->next = newNode;

newNode->prev = tail;

newNode->next = head;

head->prev = newNode;

}

void insertAtNth(int d, int pos) {

if (pos <= 1) {

insertAtFirst(d);

return;

}

Node\* newNode = new Node(d);

Node\* temp = head;

int count = 1;

while (count < pos - 1 && temp->next != head) {

temp = temp->next;

count++;

}

newNode->next = temp->next;

newNode->prev = temp;

temp->next->prev = newNode;

temp->next = newNode;

}

void insertAtCenter(int d) {

if (head == NULL) {

insertAtFirst(d);

return;

}

Node\* slow = head;

Node\* fast = head;

while (fast->next != head && fast->next->next != head) {

slow = slow->next;

fast = fast->next->next;

}

insertAtNth(d, 2);

}

void displayForward() {

if (head == NULL) {

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

return;

}

Node\* temp = head;

cout << "Forward: ";

do {

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

temp = temp->next;

} while (temp != head);

cout << endl;

}

void displayReverse() {

if (head == NULL) {

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

return;

}

Node\* temp = head->prev;

cout << "Reverse: ";

do {

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

temp = temp->prev;

} while (temp != head->prev);

cout << endl;

}

};

int main() {

CircularDoublyLinkedList list;

list.insertAtFirst(10);

list.insertAtFirst(5);

list.insertAtLast(15);

list.insertAtLast(20);

list.insertAtNth(12, 3);

list.insertAtCenter(8);

cout << "Displaying Circular Linked List:" << endl;

list.displayForward();

list.displayReverse();

return 0;

}

**EXPLANATION:**

This program implements a **circular doubly linked list** where nodes are connected in a loop, with the last node pointing back to the first. It includes functions to **insert a node** at the **beginning, end, Nth position, and center** of the list. The displayForward() and displayReverse() functions print the list in both directions. In main(), nodes are inserted at various positions, and the list is displayed in both forward and reverse order.

**Output:**

