**TASK 01**

#include <iostream>

// Define the Node structure

struct Node {

int data;

Node\* next;

Node(int val) : data(val), next(0) {} // Use 0 instead of nullptr or NULL

};

// Define the LinkedList class

class LinkedList {

private:

Node\* head;

public:

LinkedList() : head(0) {} // Use 0 instead of nullptr or NULL

// Function to add a new node at the end of the linked list

void addNode(int data) {

Node\* newNode = new Node(data);

if (!head) {

head = newNode;

} else {

Node\* current = head;

while (current->next) {

current = current->next;

}

current->next = newNode;

}

}

// Function to print the linked list along with addresses

void printListWithAddresses() {

Node\* current = head;

while (current) {

std::cout << "Node address: " << current << " | Data: " << current->data << " | Next Node address: " << current->next << std::endl;

current = current->next;

}

std::cout << std::endl;

}

};

int main() {

// Create a linked list and add elements

LinkedList myList;

myList.addNode(1);

myList.addNode(2);

myList.addNode(20);

myList.addNode(30);

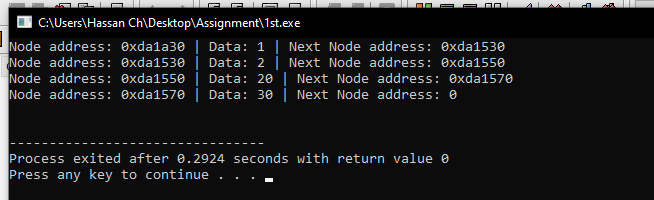
// Print the linked list with addresses

myList.printListWithAddresses();

return 0;

}

**OUTPUT:**

****

**TASK02:**

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* next;

Node(int value) {

data = value;

next = NULL;

}

};

class DoublyNode : public Node {

public:

Node\* prev;

DoublyNode(int value) : Node(value) {

prev = NULL;

}

};

class CircularNode : public Node {

public:

CircularNode(int value) : Node(value) {}

};

class LinkedList {

protected:

Node\* head;

public:

LinkedList() {

head = NULL;

}

// Function to add a node at the end of the list

void insertAtEnd(int value) {

Node\* newNode = new Node(value);

if (head == NULL) {

head = newNode;

} else {

Node\* current = head;

while (current->next != NULL) {

current = current->next;

}

current->next = newNode;

}

}

// Function to add a node at the beginning of the list

void insertAtStart(int value) {

Node\* newNode = new Node(value);

newNode->next = head;

head = newNode;

}

// Function to add a node at a specific index

void insertAtIndex(int value, int index) {

if (index < 0) {

cout << "Invalid index. Cannot insert at a negative index." << endl;

return;

}

Node\* newNode = new Node(value);

if (index == 0) {

newNode->next = head;

head = newNode;

} else {

Node\* current = head;

int currentIndex = 0;

while (current != NULL && currentIndex < index - 1) {

current = current->next;

currentIndex++;

}

if (current == NULL) {

cout << "Invalid index. Cannot insert at the specified index." << endl;

return;

}

newNode->next = current->next;

current->next = newNode;

}

}

// Function to delete a node at a specific index

void deleteAtIndex(int index) {

if (index < 0) {

cout << "Invalid index. Cannot delete at a negative index." << endl;

return;

}

if (head == NULL) {

cout << "List is empty. Cannot delete from an empty list." << endl;

return;

}

if (index == 0) {

Node\* temp = head;

head = head->next;

delete temp;

} else {

Node\* current = head;

int currentIndex = 0;

while (current->next != NULL && currentIndex < index - 1) {

current = current->next;

currentIndex++;

}

if (current->next == NULL) {

cout << "Invalid index. Cannot delete at the specified index." << endl;

return;

}

Node\* temp = current->next;

current->next = current->next->next;

delete temp;

}

}

// Function to print the entire linked list

void printList() {

Node\* current = head;

while (current != NULL) {

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

current = current->next;

}

cout << "nullptr" << endl;

}

};

class DoublyLinkedList : public LinkedList {

public:

DoublyLinkedList() : LinkedList() {}

// Function to add a node at the end of the doubly linked list

void insertAtEnd(int value) {

DoublyNode\* newNode = new DoublyNode(value);

if (head == NULL) {

head = newNode;

} else {

Node\* current = head;

while (current->next != NULL) {

current = current->next;

}

current->next = newNode;

newNode->prev = current;

}

}

};

class CircularLinkedList : public LinkedList {

public:

CircularLinkedList() : LinkedList() {}

// Function to add a node at the end of the circular linked list

void insertAtEnd(int value) {

Node\* newNode = new CircularNode(value);

if (head == NULL) {

head = newNode;

newNode->next = newNode; // Point to itself for circularity

} else {

Node\* current = head;

while (current->next != head) {

current = current->next;

}

current->next = newNode;

newNode->next = head; // Make it circular

}

}

// Function to print the entire circular linked list

void printList() {

if (head == NULL) {

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

return;

}

Node\* current = head;

do {

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

current = current->next;

} while (current != head);

cout << "nullptr" << endl;

}

};

int main() {

LinkedList myList;

DoublyLinkedList myDoublyList;

CircularLinkedList myCircularList;

while (true) {

int choice;

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

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

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

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

cout << "4. Singly Linked List: Delete at index" << endl;

cout << "5. Singly Linked List: Print list" << endl;

cout << "6. Doubly Linked List: Insert at end" << endl;

cout << "7. Doubly Linked List: Print list" << endl;

cout << "8. Circular Linked List: Insert at end" << endl;

cout << "9. Circular Linked List: Print list" << endl;

cout << "10. Exit" << endl;

cout << "Enter your choice: ";

cin >> choice;

int value, index;

switch (choice) {

case 1:

cout << "Enter value to insert at end: ";

cin >> value;

myList.insertAtEnd(value);

break;

case 2:

cout << "Enter value to insert at start: ";

cin >> value;

myList.insertAtStart(value);

break;

case 3:

cout << "Enter value to insert: ";

cin >> value;

cout << "Enter index to insert at: ";

cin >> index;

myList.insertAtIndex(value, index);

break;

case 4:

cout << "Enter index to delete: ";

cin >> index;

myList.deleteAtIndex(index);

break;

case 5:

cout << "Singly Linked List: ";

myList.printList();

break;

case 6:

cout << "Enter value to insert at end: ";

cin >> value;

myDoublyList.insertAtEnd(value);

break;

case 7:

cout << "Doubly Linked List: ";

myDoublyList.printList();

break;

case 8:

cout << "Enter value to insert at end: ";

cin >> value;

myCircularList.insertAtEnd(value);

break;

case 9:

cout << "Circular Linked List: ";

myCircularList.printList();

break; // Add this break statement to exit the switch case

case 10:

return 0;

default:

cout << "Invalid choice. Please try again." << endl;

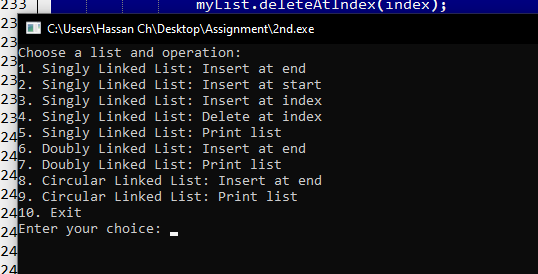
}

}

return 0;

}

**OUTPUT:**

****