ASSIGNMENT #2

Question #1:

#include<iostream>

using namespace std;

class Node

{

public:

int data;

Node\* next;

Node(int d)

{

data = d;

next = NULL;

}

};

class linklist

{

private:

Node\* Head;

public:

linklist()

{

Head = NULL;

}

void insertatHead(int d) //inserting node at head

{

Node\* temp = new Node(d);

temp->next = Head;

Head = temp;

}

void insertattail(int da)

{

Node\* tail = new Node(da);

Head->next = tail;

tail = tail->next;

}

void insertatmid(int pos, int d)

{

//insert at start

if (pos == 1)

{

insertatHead(d);

return;

}

Node\* temp = Head;

int cnt = 1;

while (cnt < pos - 1)

{

temp = temp->next;

cnt++;

}

//inserting at last position

if (temp->next == NULL)

{

insertattail(d);

return;

}

//creating a Node for d

Node\* nodetoinsert = new Node(d);

nodetoinsert->next = temp->next;

temp->next = nodetoinsert;

}

void deleteNod(int pos)

{

if (pos == 1)

{

Node\*temp = Head;

Head = Head->next;

//memory free start Node

temp->next = NULL;

delete temp;

}

else

{

//deleting any middle node or last node

Node\* curr = Head;

Node\* prev = NULL;

int cnt = 1;

while (cnt < pos)

{

prev = curr;

curr = curr->next;

cnt++;

}

prev->next = curr->next;

prev->next = NULL;

delete curr;

}

}

void display()

{

Node\* temp = Head;

while (temp != NULL)

{

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

temp = temp->next;

}

cout << endl;

}

};

int main()

{

linklist obj;

obj.insertatHead(10);

obj.display();

obj.insertatHead(12);

obj.display();

obj.insertatHead(15);

obj.display();

obj.insertattail(9);

obj.display();

obj.insertattail(8);

obj.display();

obj.insertattail(7);

obj.display();

obj.insertatmid(2, 22);

obj.display();

obj.deleteNod(3);

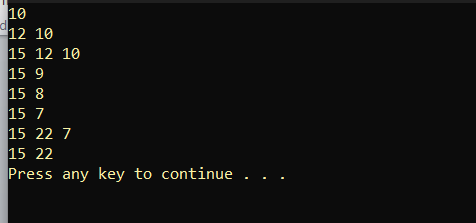
obj.display();

system("color E");

system("pause");

}

**Output:**

****

**Question #2:**

#include<iostream>

using namespace std;

class Node

{

public:

int data; //Doubly Link list

Node\* prev;

Node\* next;

Node(int d)

{

this->data = d;

this->prev = NULL;

this->next = NULL;

}

};

//traversing a link list

void print(Node\* Head)

{

Node\* temp = Head;

while (temp != NULL)

{

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

temp = temp->next;

}

cout << endl;

}

void insertatHead(Node\* &Head, int d)

{

if (Head == NULL)

{

Node\* temp = new Node(d);

Head = temp;

}

else

{

Node\* temp = new Node(d);

temp->next = Head;

Head->prev = temp;

Head = temp;

}

}

void insertattail(Node\* &tail, int d)

{

if (tail == NULL)

{

Node\* temp = new Node(d);

tail = temp;

}

else

{

Node\* temp = new Node(d);

tail->next = temp;

temp->prev = tail;

tail = temp;

}

}

void insertatpos(Node\* &tail, Node\* &Head, int pos, int d)

{

//insert at start

if (pos == 1)

{

insertatHead(Head, d);

return;

}

Node\* temp = Head;

int cnt = 1;

while (cnt < pos - 1)

{

temp = temp->next;

cnt++;

}

//insert at last position

if (temp->next == NULL)

{

insertattail(tail, d);

return;

}

//creating a node for d

Node\* nodetoinsert = new Node(d);

nodetoinsert->next = temp->next;

temp->next->prev = nodetoinsert;

temp->next = nodetoinsert;

nodetoinsert->prev = temp;

}

void deleteNod(int pos, Node\* &Head)

{

if (pos == 1)

{

Node\*temp = Head;

Head = Head->next;

temp->next->prev = NULL; //memory free start Node

Head = temp->next;

temp->next = NULL;

delete temp;

}

else

{

//deleting any middle node or last node

Node\* curr = Head;

Node\* prev = NULL;

int cnt = 1;

while (cnt < pos)

{

prev = curr;

curr = curr->next;

cnt++;

}

curr->prev = NULL;

prev->next = curr->next;

curr->next = NULL;

delete curr;

}

}

int main()

{

Node\* node1 = new Node(10);

Node\* Head = NULL;

Node\* tail = NULL;

print(Head);

insertatHead(Head, 12);

print(Head);

insertatHead(Head, 14);

print(Head);

insertatHead(Head, 16);

print(Head);

insertattail(tail, 8);

print(Head);

insertatpos(tail, Head, 1, 100);

print(Head);

insertatpos(tail, Head, 2, 200);

print(Head);

insertatpos(tail, Head, 3, 300);

print(Head);

deleteNod(4, Head);

print(Head);

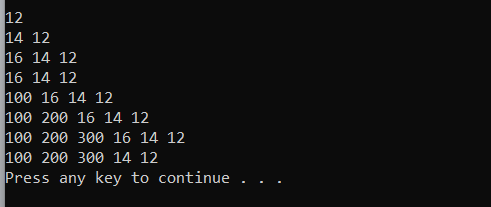
deleteNod(3, Head);

print(Head);

system("pause");

}

Output:



Task #3:

#include <iostream> // Merge sort for linked list

using namespace std;

// Node structure for a singly linked list

struct Node

{

int data;

Node\* next;

};

// Function to merge two sorted linked lists

Node\* Merge(Node\* head1, Node\* head2)

{

// If one of the linked lists is empty, return the other list

if (head1 == nullptr) return head2;

if (head2 == nullptr) return head1;

// Merge the two sorted lists recursively

if (head1->data < head2->data) {

head1->next = Merge(head1->next, head2);

return head1;

}

else {

head2->next = Merge(head1, head2->next);

return head2;

}

}

// Function to split a linked list into two halves using the slow-fast pointer approach

void Split(Node\* head, Node\*\* left, Node\*\* right) {

// If the linked list is empty or has only one node, return the same list as left and null as right

if (head == nullptr || head->next == nullptr) {

\*left = head;

\*right = nullptr;

return;

}

// Use the slow-fast pointer approach to find the middle node

Node\* slow = head;

Node\* fast = head->next;

while (fast != nullptr) {

fast = fast->next;

if (fast != nullptr) {

slow = slow->next;

fast = fast->next;

}

}

// Split the linked list into two halves at the middle node

\*left = head;

\*right = slow->next;

slow->next = nullptr;

}

// Function to perform Merge Sort on a linked list

void MergeSort(Node\*\* head) {

// If the linked list is empty or has only one node, return

if (\*head == nullptr || (\*head)->next == nullptr) {

return;

}

// Split the linked list into two halves

Node\* left;

Node\* right;

Split(\*head, &left, &right);

// Recursively apply Merge Sort on the two halves

MergeSort(&left);

MergeSort(&right);

// Merge the two sorted halves into a single sorted list

\*head = Merge(left, right);

}

// Function to insert a new node at the beginning of a linked list

void Insert(Node\*\* head, int data) {

Node\* newNode = new Node;

newNode->data = data;

newNode->next = \*head;

\*head = newNode;

}

// Function to print the nodes of a linked list

void PrintList(Node\* head)

{

while (head != nullptr)

{

cout << head->data << " ";

head = head->next;

}

cout << endl;

}

int main()

{

// Create a linked list and insert some nodes

Node\* head = nullptr;

Insert(&head, 5);

Insert(&head, 3);

Insert(&head, 8);

Insert(&head, 1);

Insert(&head, 9);

Insert(&head, 2);

Insert(&head, 7);

// Print the unsorted linked list

cout << "Before sorting:" << endl;

PrintList(head);

// Sort the linked list using Merge Sort

MergeSort(&head);

// Print the sorted linked list

cout << "After sorting:" << endl;

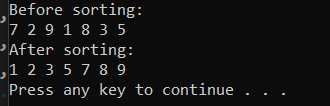
PrintList(head);

system("pause");

return 0;

}

Output:



Task #4:

#include<iostream>

#include<cstdlib>

using namespace std;

class Node {

public:

int data;

Node\* next;

Node(int data) {

this->data = data;

next = NULL;

}

};

class CircularLinkedList {

public:

Node\* head;

CircularLinkedList() {

head = NULL;

}

// function to add a new node to the list

void add\_node(int data) {

Node\* new\_node = new Node(data);

if (head == NULL) {

head = new\_node;

new\_node->next = head;

}

else {

Node\* curr\_node = head;

while (curr\_node->next != head) {

curr\_node = curr\_node->next;

}

curr\_node->next = new\_node;

new\_node->next = head;

}

}

// function to print the contents of the list

void print\_list() {

if (head == NULL) {

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

}

else {

Node\* curr\_node = head;

do {

cout << curr\_node->data << " ";

curr\_node = curr\_node->next;

} while (curr\_node != head);

cout << endl;

}

}

// function to shift the nodes of the list

void shift(int n) {

if (head == NULL) {

return;

}

if (n > 0) {

// right shift

for (int i = 0; i < n; i++) {

head = head->next;

}

}

else if (n < 0) {

// left shift

n = abs(n);

for (int i = 0; i < n; i++) {

head = head->next;

}

}

}

};

int main() {

CircularLinkedList list;

int choice, data, n;

do {

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

cout << "1. Add a new node" << endl;

cout << "2. Print the list" << endl;

cout << "3. Shift the list" << endl;

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

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter the data for the new node: ";

cin >> data;

list.add\_node(data);

break;

case 2:

list.print\_list();

break;

case 3:

cout << "Enter the number of shifts: ";

cin >> n;

list.shift(n);

break;

case 4:

cout << "Exiting..." << endl;

break;

default:

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

}

cout << endl;

} while (choice != 4);

system("pause");

return 0;

}

