# Assignment 3

Name :- Shreyas Anil Dimbar

PRN :- 245100140

Batch :- S6

* Q1-Write a Menu Driven program to perform following operations on singly linked list.
  + Insert new element at Beginning, End and middle position.
  + Delete element from Beginning, End and middle position.
  + Display Linked List.

Ans :- #include <iostream>

using namespace std;

class Node {

public:

int val;

Node \*next;

Node(int val) {

this->val = val;

this->next = NULL;

}

};

class LinkedList {

private:

Node\* head;

public:

LinkedList() {

head = NULL;

}

void insertAtBeginning(int val) {

Node\* newNode = new Node(val);

newNode->next = head;

head = newNode;

cout << "Okay, " << val << " is now at the front of the list." << endl;

}

void insertAtEnd(int val) {

Node\* newNode = new Node(val);

if (head == NULL) {

head = newNode;

cout << "Got it, " << val << " has been added to the end." << endl;

return;

}

Node\* temp = head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

cout << "Got it, " << val << " has been added to the end." << endl;

}

void insertAtMiddle(int val, int pos) {

if (pos <= 1) {

insertAtBeginning(val);

return;

}

Node\* newNode = new Node(val);

Node\* temp = head;

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

temp = temp->next;

}

if (temp == NULL) {

cout << "That position doesn't exist, so I'll add it to the end for you." << endl;

insertAtEnd(val);

} else {

newNode->next = temp->next;

temp->next = newNode;

cout << "Done! " << val << " is now at position " << pos << "." << endl;

}

}

void deleteFromBeginning() {

if (head == NULL) {

cout << "Whoops, the list is empty! Nothing to delete." << endl;

return;

}

Node\* temp = head;

head = head->next;

cout << temp->val << " has been removed from the front." << endl;

delete temp;

}

void deleteFromEnd() {

if (head == NULL) {

cout << "Whoops, the list is empty! Nothing to delete." << endl;

return;

}

if (head->next == NULL) {

cout << head->val << " has been removed from the end." << endl;

delete head;

head = NULL;

return;

}

Node\* temp = head;

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

temp = temp->next;

}

Node\* nodeToDelete = temp->next;

cout << nodeToDelete->val << " has been removed from the end." << endl;

temp->next = NULL;

delete nodeToDelete;

}

void deleteFromMiddle(int pos) {

if (head == NULL) {

cout << "Whoops, the list is empty! Nothing to delete." << endl;

return;

}

if (pos <= 1) {

deleteFromBeginning();

return;

}

Node\* temp = head;

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

temp = temp->next;

}

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

cout << "Sorry, that position isn't valid. Can't delete." << endl;

return;

}

Node\* nodeToDelete = temp->next;

temp->next = nodeToDelete->next;

cout << "Removed " << nodeToDelete->val << " from position " << pos << "." << endl;

delete nodeToDelete;

}

void display() {

if (head == NULL) {

cout << "The list is currently empty." << endl;

return;

}

cout << "Here's your list: ";

Node\* temp = head;

while (temp != NULL) {

cout << temp->val << " -> ";

temp = temp->next;

}

cout << "NULL" << endl;

}

};

int main() {

LinkedList list;

int choice, value, position;

while (true) {

cout << "\n--- Linked List Menu ---" << endl;

cout << "1. Add to Beginning" << endl;

cout << "2. Add to End" << endl;

cout << "3. Add at a Specific Spot" << endl;

cout << "4. Remove from Beginning" << endl;

cout << "5. Remove from End" << endl;

cout << "6. Remove from a Specific Spot" << endl;

cout << "7. Show the List" << endl;

cout << "8. Exit" << endl;

cout << "--------------------------" << endl;

cout << "What would you like to do? ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter a number to add: ";

cin >> value;

list.insertAtBeginning(value);

break;

case 2:

cout << "Enter a number to add: ";

cin >> value;

list.insertAtEnd(value);

break;

case 3:

cout << "Enter a number to add: ";

cin >> value;

cout << "What position should it be in? ";

cin >> position;

list.insertAtMiddle(value, position);

break;

case 4:

list.deleteFromBeginning();

break;

case 5:

list.deleteFromEnd();

break;

case 6:

cout << "Enter position to remove: ";

cin >> position;

list.deleteFromMiddle(position);

break;

case 7:

list.display();

break;

case 8:

cout << " shutting down" << endl;

return 0;

default:

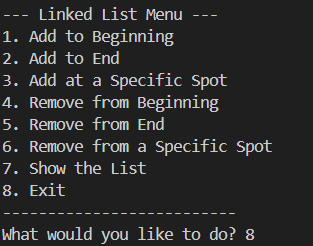
cout << " that's not a valid option. " << endl;

}

}

return 0;

}

Output :- 

* Q2 - Write a program to delete nodes at even position in a Circular Linked List.

Ans :- #include <iostream>

using namespace std;

class Node {

public:

    int data;

    Node\* next;

    Node(int data) {

        this->data = data;

        this->next = NULL;

    }

};

class CircularLinkedList {

public:

    Node\* head;

    CircularLinkedList() {

        head = NULL;

    }

    void push(int data) {

        Node\* newNode = new Node(data);

        if (head == NULL) {

            head = newNode;

            newNode->next = head;

            return;

        }

        Node\* temp = head;

        while (temp->next != head) {

            temp = temp->next;

        }

        temp->next = newNode;

        newNode->next = head;

    }

    void display() {

        if (head == NULL) {

            cout << "The list is currently empty." << endl;

            return;

        }

        Node\* temp = head;

        cout << "Here's your circular list: ";

        do {

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

            temp = temp->next;

        } while (temp != head);

        cout << "(head: " << head->data << ")" << endl;

    }

    void deleteEvenNodes() {

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

            cout << "List is too short, no even positions to delete." << endl;

            return;

        }

        Node\* prev = head;

        Node\* current = head->next;

        while (true) {

            prev->next = current->next;

            cout << "Deleting node with value: " << current->data << endl;

            delete current;

            if (prev->next == head) {

                break;

            }

            prev = prev->next;

            if (prev->next == head) {

                break;

            }

            current = prev->next;

        }

    }

};

int main() {

    CircularLinkedList cll;

    cll.push(1);

    cll.push(2);

    cll.push(3);

    cll.push(4);

    cll.push(5);

    cll.push(6);

    cll.push(7);

    cout << "--- Original List ---" << endl;

    cll.display();

    cout << "\n--- Deleting Even Position Nodes ---" << endl;

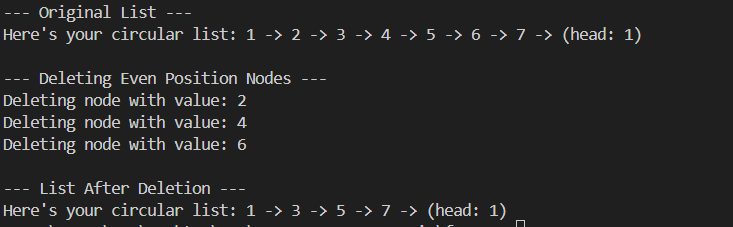
    cll.deleteEvenNodes();

    cout << "\n--- List After Deletion ---" << endl;

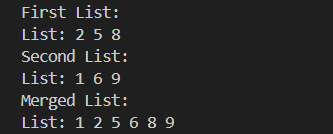
    cll.display();

    return 0;

}

Output :- 

* Q3 - Write a program to Merge two sorted circular Linked List.
* #include <iostream>
* using namespace std;
* class Node {
* public:
* int data;
* Node\* next;
* Node(int data) {
* this->data = data;
* this->next = NULL;
* }
* };
* class CircularLinkedList {
* public:
* Node\* head;
* CircularLinkedList() {
* head = NULL;
* }
* void sortedInsert(int data) {
* Node\* newNode = new Node(data);
* if (head == NULL) {
* head = newNode;
* newNode->next = head;
* return;
* }
* if (head->data >= newNode->data) {
* Node\* temp = head;
* while (temp->next != head) {
* temp = temp->next;
* }
* temp->next = newNode;
* newNode->next = head;
* head = newNode;
* } else {
* Node\* temp = head;
* while (temp->next != head && temp->next->data < newNode->data) {
* temp = temp->next;
* }
* newNode->next = temp->next;
* temp->next = newNode;
* }
* }
* void display() {
* if (head == NULL) {
* cout << "List is empty." << endl;
* return;
* }
* Node\* temp = head;
* cout << "List: ";
* do {
* cout << temp->data << " ";
* temp = temp->next;
* } while (temp != head);
* cout << endl;
* }
* };
* Node\* sortedMerge(Node\* a, Node\* b) {
* if (a == NULL) return b;
* if (b == NULL) return a;
* Node\* result = NULL;
* if (a->data <= b->data) {
* result = a;
* result->next = sortedMerge(a->next, b);
* } else {
* result = b;
* result->next = sortedMerge(a, b->next);
* }
* return result;
* }
* void mergeLists(CircularLinkedList &list1, CircularLinkedList &list2) {
* if (list1.head == NULL) {
* list1.head = list2.head;
* list2.head = NULL;
* return;
* }
* if (list2.head == NULL) {
* return;
* }
* Node\* head1 = list1.head;
* Node\* head2 = list2.head;
* Node\* last1 = head1;
* while (last1->next != head1) {
* last1 = last1->next;
* }
* Node\* last2 = head2;
* while (last2->next != head2) {
* last2 = last2->next;
* }
* last1->next = NULL;
* last2->next = NULL;
* list1.head = sortedMerge(head1, head2);
* list2.head = NULL;
* Node\* new\_last = list1.head;
* while (new\_last->next != NULL) {
* new\_last = new\_last->next;
* }
* new\_last->next = list1.head;
* }
* int main() {
* CircularLinkedList list1;
* list1.sortedInsert(2);
* list1.sortedInsert(5);
* list1.sortedInsert(8);
* CircularLinkedList list2;
* list2.sortedInsert(1);
* list2.sortedInsert(6);
* list2.sortedInsert(9);
* cout << "First List:" << endl;
* list1.display();
* cout << "Second List:" << endl;
* list2.display();
* mergeLists(list1, list2);
* cout << "Merged List:" << endl;
* list1.display();
* return 0;
* }

Output :- 

* Q4 - Write a program to Search an element in a Linked List using recursive function.

Code :- #include <iostream>

using namespace std;

class Node {

public:

    int data;

    Node\* next;

    Node(int data) {

        this->data = data;

        this->next = NULL;

    }

};

class LinkedList {

public:

    Node\* head;

    LinkedList() {

        head = NULL;

    }

    void push(int data) {

        Node\* newNode = new Node(data);

        newNode->next = head;

        head = newNode;

    }

    bool search(int key) {

        return recursiveSearch(head, key);

    }

private:

    bool recursiveSearch(Node\* current, int key) {

        if (current == NULL) {

            return false;

        }

        if (current->data == key) {

            return true;

        }

        return recursiveSearch(current->next, key);

    }

};

int main() {

    LinkedList list;

    list.push(10);

    list.push(20);

    list.push(30);

    list.push(40);

    list.push(50);

    int keyToFind1 = 30;

    if (list.search(keyToFind1)) {

        cout << keyToFind1 << " found." << endl;

    } else {

        cout << keyToFind1 << " not found." << endl;

    }

    int keyToFind2 = 99;

    if (list.search(keyToFind2)) {

        cout << keyToFind2 << " found." << endl;

    } else {

        cout << keyToFind2 << " not found." << endl;

    }

    return 0;

}

Output - 