Data Structures and Algorithms Lab Journal - Lab 2

Name: Mudassir aman

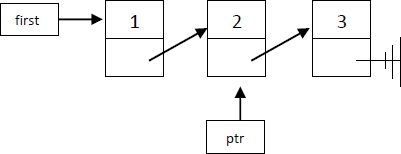
Enrollment #: 01-134232-102

Class/Section: 3D

**Objective**

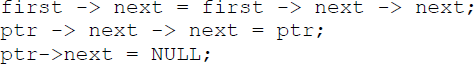
This lab session is aimed at introducing students to singly linked list. Students will be required to implement the Singly Linked List ADT. Students will also be required to create some non- member functions to manipulate a given linked list.

**Task 1: Given the following linked list, state output of the following statements.**



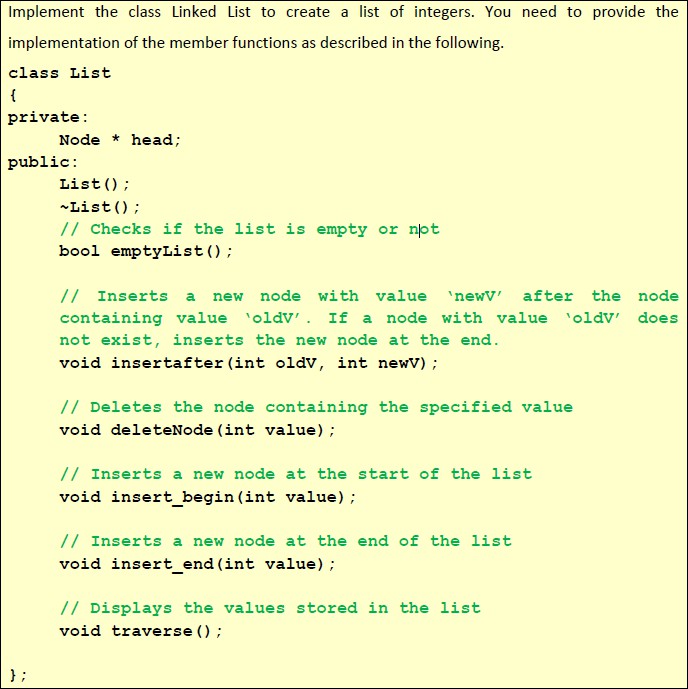
|  |  |
| --- | --- |
| **cout<< first->data;** | 1 |
| **cout<<first->next->next->data;** | 3 |
| **cout<<ptr->next->data;** | 2 |

**Task 2: Redraw the above list after the given instructions are executed.**



**Task 3: Exercises**

**Exercise 1 (Linked list implementation)**



## CODE:

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* next;

Node(int val) {

data = val; next = NULL;

}

};

class LinkedList {

private:

Node\* head;

public:

LinkedList() { head = NULL; }

void insert(int val) {

Node\* newnode = new Node(val);

if (head == nullptr) {

head = newnode;

}

else {

Node\* temp = head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newnode;

}

}

void insertbegin(int val) {

Node\* newnode = new Node(val);

newnode->next = head;

head = newnode;

}

bool insertafter(int oldv, int newv) {

Node\* currnode = head;

while (currnode != NULL && currnode->data != oldv) {

currnode = currnode->next;

}

if (currnode == NULL) {

cout << "Old value " << oldv << " not found in the list." << endl;

return false;

}

Node\* newnode = new Node(newv);

newnode->next = currnode->next;

currnode->next = newnode;

return true;

}

bool deletenode(int val) {

if (head == NULL) {

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

return false;

}

if (head->data == val) {

Node\* temp = head;

head = head->next;

delete temp;

return true;

}

Node\* currnode = head;

while (currnode->next != NULL && currnode->next->data != val) {

currnode = currnode->next;

}

if (currnode->next == NULL) {

cout << "Value " << val << " not found in the list." << endl;

return false;

}

Node\* temp = currnode->next;

currnode->next = temp->next;

delete temp;

return true;

}

bool empty() {

return head == NULL;

}

void display(){

Node\* temp = head;

while (temp != NULL) {

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

temp = temp->next;

}

cout << endl;

}

~LinkedList() {

while (head != NULL) {

deletenode(head->data);

}

}

};

int main() {

LinkedList list;

list.insert(10);

list.insert(20);

list.insert(30);

cout << "Initial List: ";

list.display();

list.insertbegin(5);

cout << "After inserting 5 at the beginning: ";

list.display();

list.insertafter(20, 25);

cout << "After inserting 25 after 20: ";

list.display();

list.deletenode(10);

cout << "After deleting node with value 10: ";

list.display();

cout << "Is the list empty? " << (list.empty() ? "Yes" : "No") << endl;

return 0;

}

## OUTPUT:



**Exercise 2 (Storing Records in a List)**

Modify the above list ADT to create a list of students, where each node of the list contains student ID and name. Alongwith, basic list functions, also write a non-member function that allows the user to search a student in a list by using his ID and then displays the name of the student, if found.

## CODE:

#include <iostream>

#include <string>

using namespace std;

class Student {

private:

int id;

string name;

Student\* next;

public:

Student(int id, string name) {

this->id = id;

this->name = name;

next = NULL;

}

int getId() {

return id;

}

string getName() {

return name;

}

Student\* getNext() {

return next;

}

void setNext(Student\* nextstudent) {

next = nextstudent;

}

};

class StudentList {

private:

Student\* head;

public:

StudentList() { head = NULL; }

void insert(int id, string name) {

Student\* newstudent = new Student(id, name);

if (head == NULL) {

head = newstudent;

}

else {

Student\* temp = head;

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

temp = temp->getNext();

}

temp->setNext(newstudent);

}

}

void searchById(int id) {

Student\* temp = head;

while (temp != NULL) {

if (temp->getId() == id) {

cout << "Student Name: " << temp->getName() << endl;

return;

}

temp = temp->getNext();

}

cout << "Student not found!" << endl;

}

void displaystudents() {

Student\* temp = head;

while (temp != NULL) {

cout << "ID: " << temp->getId() << ", Name: " << temp->getName() << endl;

temp = temp->getNext();

}

}

};

int main() {

StudentList List;

List.insert(197, "Ali");

List.insert(165, "Fahad");

List.insert(144, "Zain");

cout << "Student List:" << endl;

List.displaystudents();

int searchId;

cout << "Enter student ID to search: ";

cin >> searchId;

List.searchById(searchId);

return 0;

}

## OUTPUT:



**Exercise 3 (List Traversal based Utility Functions)**



Write a C++ program which allows users to create a list of integers and then provides them with choice to call following user-defined functions:

1. A function which accepts a pointer to the head of the list and returns the maximum value in the list.
2. A function that counts and returns the total number of nodes in the list.
3. A function that prints only the even-numbered (not even-valued) nodes of the list.
4. A function that splits the given list in half and swaps the two halves in the following manner.

Initial

After

\*Note: Create these as non-member functions.

## CODE:

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* next;

Node(int val) {

data=val;

next=NULL;

}

};

class LinkedList {

private:

Node\* head;

public:

LinkedList() {

head = NULL;

}

void insert(int val) {

Node\* newnode = new Node(val);

if (!head) {

head = newnode;

}

else {

Node\* temp = head;

while (temp->next) {

temp = temp->next;

}

temp->next = newnode;

}

}

void display() {

for (Node\* temp = head; temp; temp = temp->next) {

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

}

cout << endl;

}

int findMax() {

if (!head) return -1;

int maxval = head->data;

for (Node\* temp = head; temp; temp = temp->next) {

if (temp->data > maxval)

maxval = temp->data;

}

return maxval;

}

int countNodes() {

int count = 0;

for (Node\* temp = head; temp; temp = temp->next) {

count++;

}

return count;

}

void printevennodes() {

for (Node\* curr = head; curr; curr = curr->next) {

if (curr->data % 2 == 0) cout << curr->data << " ";

}

cout << endl;

}

void splitandswap() {

if (!head || !head->next) return;

Node\* slow = head, \* fast = head, \* prev = NULL;

while (fast && fast->next) {

prev = slow;

slow = slow->next;

fast = fast->next->next;

}

Node\* secondhalf = slow;

prev->next = NULL;

head = secondhalf;

while (secondhalf && secondhalf->next) {

secondhalf = secondhalf->next;

}

if (secondhalf) secondhalf->next = head;

}

~LinkedList() {

while (head) {

Node\* temp = head;

head = head->next;

delete temp;

}

}

};

int main() {

LinkedList list;

int choice, value;

while (true) {

cout << "\nMenu:\n";

cout << "1. Insert a value into the list\n";

cout << "2. Display the list\n";

cout << "3. Find the maximum value in the list\n";

cout << "4. Count total number of nodes\n";

cout << "5. Print data in even number nodes\n";

cout << "6. Split and swap the list\n";

cout << "7. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter value to insert: ";

cin >> value;

list.insert(value);

break;

case 2:

cout << "The list is: ";

list.display();

break;

case 3:

cout << "Maximum value in the list: " << list.findMax() << endl;

break;

case 4:

cout << "Total number of nodes: " << list.countNodes() << endl;

break;

case 5:

cout << "Even Number Nodes: ";

list.printevennodes();

break;

case 6:

list.splitandswap();

cout << "List after split and swap: ";

list.display();

break;

case 7:

return 0;

default:

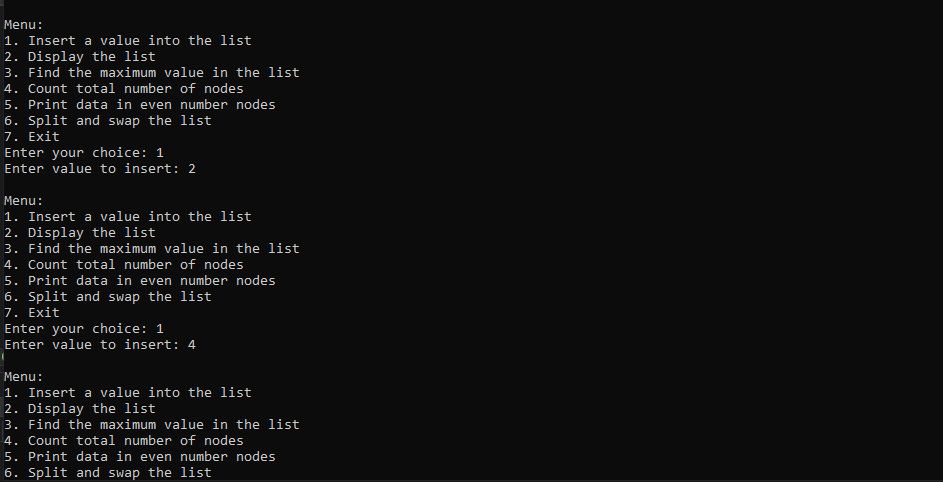
cout << "Invalid.\n";

}

}

}

## OUTPUT:



**Implement the given exercises and get them checked by your instructor.**

|  |  |  |
| --- | --- | --- |
| **S No.** | **Exercise** | **Checked By:** |
| 1. | Exercise 1 |  |
| 2. | Exercise 2 |  |
| 3. | Exercise 3a |  |
| 4. | Exercise 3b |  |
| 5. | Exercise 3c |  |
| 6. | Exercise 3d |  |

+++++++++++++++++++++++++