**Write Linked List code with functions perform on it.**

**Code:**

#include <iostream>

using namespace std;

class node {

public:

int data;

node\* next;

node(int val);

node();

void insertLast(int val);

void insertFirst(int val);

void insertAt(int val, int pos);

void removeLast();

void removeFirst();

void removeAt(int pos);

void display();

void reverse();

void sort();

void search(int val);

};

node\* head;

int node\_count = 0;

node::node() {

data = 0;

next = NULL;

}

node::node(int val) {

data = val;

next = NULL;

}

void node::display() {

node\* trav = head;

if (trav == NULL) {

cout << "Stack is Empty" << endl;

}

else {

cout << "\nData: ";

while (trav != NULL) {

cout << trav->data << " ";

trav = trav->next;

}

cout << "\nTotal number of elements are " << node\_count+1 << endl;

}

}

void node::insertFirst(int val) {

node\* temp = new node(val);

temp->next = head;

head = temp;

node\_count++;

}

void node::removeFirst() {

if (head == NULL) {

cout << "Stack is Empty" << endl;

} else {

node\* temp = head;

head = head->next;

delete temp;

node\_count--;

}

}

void node::insertLast(int val) {

node\* temp = new node(val);

if (head == NULL) {

head = temp;

} else {

node\* trav = head;

while (trav->next != NULL) {

trav = trav->next;

}

trav->next = temp;

}

node\_count++;

}

void node::removeLast() {

if (head == NULL) {

cout << "Stack is Empty" << endl;

} else if (node\_count == 1) {

delete head;

head = NULL;

node\_count--;

} else {

node\* trav = head;

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

trav = trav->next;

}

delete trav->next;

trav->next = NULL;

node\_count--;

}

}

void node::insertAt(int val, int pos) {

if (pos < 1 || pos > node\_count + 1) {

cout << "Wrong Choice" << endl;

} else if (pos == 1) {

insertFirst(val);

} else if (pos == node\_count + 1) {

insertLast(val);

} else {

node\* temp = new node(val);

node\* trav = head;

int k = 1;

while (k < pos - 1) {

trav = trav->next;

k++;

}

temp->next = trav->next;

trav->next = temp;

node\_count++;

}

}

void node::removeAt(int pos) {

if (pos < 1 || pos > node\_count) {

cout << "Wrong Position" << endl;

} else if (pos == 1) {

removeFirst();

} else if (pos == node\_count) {

removeLast();

} else {

node\* trav = head;

int k = 1;

while (k < pos - 1) {

trav = trav->next;

k++;

}

node\* temp = trav->next;

trav->next = temp->next;

delete temp;

node\_count--;

}

}

void node::sort() {

node\* sort\_Head = head;

while (sort\_Head != NULL) {

node\* trav = sort\_Head->next;

while (trav != NULL) {

if (trav->data < sort\_Head->data) {

int temp = trav->data;

trav->data = sort\_Head->data;

sort\_Head->data = temp;

}

trav = trav->next;

}

sort\_Head = sort\_Head->next;

}

}

void node::reverse() {

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

return; // Nothing to reverse

}

node\* prev = NULL;

node\* current = head;

node\* nextNode;

while (current != NULL) {

nextNode = current->next;

current->next = prev;

prev = current;

current = nextNode;

}

head = prev;

}

//searching

void node::search(int data){

node\* trav = head;

bool flag = false;

while(trav != NULL && flag == false){

if(trav -> data == data){

flag = true;

break;

}

else{

trav = trav -> next;

}

}

if(flag == true){

cout << "Element Found!";

}

else{

cout << "Element Not Found!";

}

}

int main() {

head = new node(500);

cout << "Insert Last";

head->display();

head->insertLast(200);

cout << "Insert Last";

head->display();

head->insertLast(300);

cout << "Insert Last";

head->display();

head->insertLast(400);

cout << "Insert Last";

head->display();

head->insertFirst(700);

cout << "Insert First";

head->display();

head->removeFirst();

cout << "Remove First";

head->display();

head->insertAt(600, 1);

cout << "Insert at First";

head->display();

head->insertAt(100, 3);

cout << "Insert at First";

head->display();

head->removeFirst();

cout << "Remove Last";

head->display();

head->insertLast(500);

cout << "Insert Last";

head->display();

head->removeAt(4);

cout << "Remove at Position 4";

head->display();

head->reverse();

cout << "Reverse";

head->display();

head->sort();

cout << "Sorted";

head->display();

cout<<"\nSearching Element in the Linked list: ";

int n;

cin >> n;

head->search(n);

return 0;

}

**Output:**



