**Problem#01**

**//Inserting elements at position and then adding display function etc**

#include<iostream>

//creating struct node for linked list

struct Node {

public:

int data;

Node\* next;

Node(int dat = 0) {

data = dat;

}

};

//creating a linked list

class List {

Node\* head;

public:

List(Node\* h = NULL) {

head = h;

}

//methode declaration

Node\* createNode(int val);

bool insert\_pos(int val, int index);

void display();

bool isempty();

};

//funtion to create node

Node\* List::createNode(int val) {

Node\* n = new Node(val);

return n;

}

//to check is list is empty

bool List::isempty() {

if (head == NULL)

return true;

return false;

}

//Function to display the data

void List::display() {

Node\* current = head;

int count = 0;

std::cout << "Displaying Data in Linked List\n";

//displayin data

while (current != NULL) {

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

current = current->next;

count++;

}

std::cout << "\nTotal Items: " << count << std::endl;

}

//function to insert element at position

bool List::insert\_pos(int val, int index) {

if (index <= 0) { return false; }

int currentIndex = 2;

Node\* current = head;

//itertrating current in linked list

while (current != NULL && index > currentIndex) {

current = current->next;

currentIndex++;

}

if (current == NULL && index > 1) { return false; }

//Node\* n = new Node(val);

Node\* n = createNode(val);

if (index == 1) {

n->next = head;

head = n;

}

else {

n->next = current->next;

current->next = n;

}

return true;

}

//main driver code

int main() {

List l;

int choice = 0, elem, position;

while (choice != 5) {

std::cout << "1-Insert\n2-Isempty\n3-Display\n4-Exit\nChoose an Option: ";

std::cin >> choice;

if (choice == 1) {

std::cout << "Inserting data\n";

std::cout << "Enter Element: ";

std::cin >> elem;

std::cout << "Enter Position: ";

std::cin >> position;

std::cout << "Is inserted: " << l.insert\_pos(elem, position) << std::endl;

}

else if (choice == 2) {

std::cout << "IsEmpty" << l.isempty() << std::endl;

}

else if (choice == 3) {

l.display();

}

else if (choice == 4) {

exit(1);

}

system("pause");

system("cls");

}

}

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

**Problem#02**

**//Searching and deleting the element**

#include<iostream>

//creating struct node for linked list

struct Node {

public:

int data;

Node\* next;

Node(int dat = 0) {

data = dat;

}

};

//creating a linked list

class List {

Node\* head;

public:

List(Node\* h = NULL) {

head = h;

}

//methode declaration

Node\* createNode(int val);

bool insert\_pos(int val, int index);

void display();

bool isempty();

bool deleteElement(int val);

};

//funtion to create node

Node\* List::createNode(int val) {

Node\* n = new Node(val);

return n;

}

//to check is list is empty

bool List::isempty() {

if (head == NULL)

return true;

return false;

}

//Function to display the data

void List::display() {

Node\* current = head;

int count = 0;

std::cout << "Displaying Data in Linked List\n";

//displayin data

while (current != NULL) {

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

current = current->next;

count++;

}

std::cout << "\nTotal Items: " << count << std::endl;

}

//function to insert element at position

bool List::insert\_pos(int val, int index) {

if (index <= 0) { return false; }

int currentIndex = 2;

Node\* current = head;

//itertrating current in linked list

while (current != NULL && index > currentIndex) {

current = current->next;

currentIndex++;

}

if (current == NULL && index > 1) { return false; }

//Node\* n = new Node(val);

Node\* n = createNode(val);

if (index == 1) {

n->next = head;

head = n;

}

else {

n->next = current->next;

current->next = n;

}

return true;

}

bool List::deleteElement(int val) {

Node\* current = head;

int c = 1;

Node\* temp;

//iterating to find the element

while (current != NULL) {

if (current->data == val && c == 1) {

temp = current;

head = current->next;

delete current;

return true;

}

if (current->data == val) {

temp = current->next;

current->next = temp->next;

delete temp;

return true;

}

c++;

current = current->next;

}

return false;

}

//main driver code

int main() {

List l;

int choice = 0,elem,position;

while (choice != 5) {

std::cout << "1-Insert\n2-Isempty\n3-Display\n4-Delete\n5-Exit\nChoose an Option: ";

std::cin >> choice;

if (choice == 1) {

std::cout << "Inserting data\n";

std::cout << "Enter Element: ";

std::cin >> elem;

std::cout << "Enter Position: ";

std::cin >> position;

std::cout << "Is inserted: " << l.insert\_pos(elem, position) << std::endl;

}

else if (choice == 2) {

std::cout << "IsEmpty" << l.isempty() << std::endl;

}

else if (choice == 3) {

l.display();

}

else if (choice == 4) {

std::cout << "Enter Element to Delete: ";

std::cin >> elem;

std::cout << "Is deleted: " << l.deleteElement(elem) << std::endl;

}

else if (choice == 5) {

exit(1);

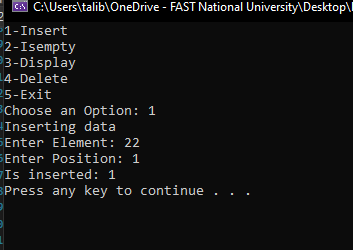
}

system("pause");

system("cls");

}

}

****

**Text

Description automatically generated**

**Text

Description automatically generated**

**Text

Description automatically generated**

**Problem#03**

**//error correction and then deleting after printing**

#include<iostream>

struct linkedList {

int number;

linkedList\* next;

};

void AddNode()

{

linkedList\* head = NULL;

linkedList\* current;

for (int i = 0; i < 10; i++) {

current = new linkedList;

current->number = i;

//error corrected

current->next = head;

head = current;

}

//printing the data in linked list

while (head->next != NULL) {

std::cout << head->number << std::endl;

//current = head->next;

//error corrected

current = head;

head = head->next;

delete current;

}

}

int main() {

AddNode();

}

**Text

Description automatically generated**

**Problem#04**

#include<iostream>

struct myList {

int data;

myList\* next;

};

int main() {

myList\* head; // Pointer of myList

myList\* cur;

myList\* previous = NULL;

for (int i = 0; i < 4; i++) {

head = new myList; // making new list and assiging data.

head->data = 0;

head->next = previous;

for (cur = previous; cur != NULL; cur = cur->next) // loop will iterate until the end of list

head->data += 1 + 2 \* cur->data; //

previous = head; // it will store the address of last node of list

}

while (previous != NULL) {

std::cout << previous->data << std::endl; //printing the data stored in a list

cur = previous; //

previous = previous->next;

delete cur; //deleting the list node after printing data

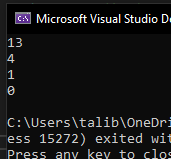
}

return 0;

}

**Code Explanation**

The list will store the elements and head will ne pointed to the end of list and after storing the list will be printed from the end because head is pointing at last.

****

**Problem#05**

**//Concatenation of two lists**

#include<iostream>

//creating struct node for linked list

struct Node {

public:

int data;

Node\* next;

Node(int dat = 0) {

data = dat;

}

};

//creating a linked list

class List {

Node\* head;

public:

List(Node\* h = NULL) {

head = h;

}

//methode declaration

Node\* createNode(int val);

bool insert\_pos(int val, int index);

void display();

bool isempty();

bool deleteElement(int val);

void concatenate(List\* l1, List\* l2);

};

//funtion to create node

Node\* List::createNode(int val) {

Node\* n = new Node(val);

return n;

}

//to check is list is empty

bool List::isempty() {

if (head == NULL)

return true;

return false;

}

//Function to display the data

void List::display() {

Node\* current = head;

int count = 0;

std::cout << "Displaying Data in Linked List\n";

//displayin data

while (current != NULL) {

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

current = current->next;

count++;

}

std::cout << "\nTotal Items: " << count << std::endl;

}

//function to insert element at position

bool List::insert\_pos(int val, int index) {

if (index <= 0) { return false; }

int currentIndex = 2;

Node\* current = head;

//itertrating current in linked list

while (current != NULL && index > currentIndex) {

current = current->next;

currentIndex++;

}

if (current == NULL && index > 1) { return false; }

//Node\* n = new Node(val);

Node\* n = createNode(val);

if (index == 1) {

n->next = head;

head = n;

}

else {

n->next = current->next;

current->next = n;

}

return true;

}

bool List::deleteElement(int val) {

Node\* current = head;

int c = 1;

Node\* temp;

//iterating to find the element

while (current != NULL) {

if (current->data == val && c == 1) {

temp = current;

head = current->next;

delete current;

return true;

}

if (current->data == val) {

temp = current->next;

current->next = temp->next;

delete temp;

return true;

}

c++;

current = current->next;

}

return false;

}

//Function to concatente two strings

void List::concatenate(List\* l1, List\* l2) {

Node\* List1\_tail;

Node\* current = head;

while (current->next != NULL) {

current = current->next;

}

List1\_tail = current;

List1\_tail->next = l2->head;

}

//main driver code

int main() {

List l1, l2;

l1.insert\_pos(1, 1);

l1.insert\_pos(2, 2);

l1.insert\_pos(3, 3);

l1.insert\_pos(4, 4);

l1.insert\_pos(5, 5);

l2.insert\_pos(10, 1);

l2.insert\_pos(11, 2);

l2.insert\_pos(12, 3);

std::cout << "List1 before concatanation\n";

l1.display();

std::cout << "List2 before concatanation\n";

l1.display();

l1.concatenate(&l1,&l2);

std::cout << "List1 after concatanation\n";

l1.display();

}

**Text

Description automatically generated**