

DATA Lab

Lab-4



Name: Talib Husain

Roll# 21F-9070

**Task-1**

#include<iostream>

using namespace std;

struct node

{

int data;

struct node\* next;

struct node\* prev;

}

\*start;

//making a class list

class double\_llist

{

private:

node\* head;

node\* current\_ptr;

int count = 0;

public:

//methods of linked list

void insert\_at\_begin(int value) {

node\* temp = new node;

temp->data = value;

temp->next = head;

temp->prev = NULL;

head = temp;

count++;

}

void insert\_after(int value, int position) {

current\_ptr = head;

int c = 1;

//moving to desired position

while (c < position) {

current\_ptr = current\_ptr->next;

c++;

}

node\* newnode = new node;

newnode->data = value;

newnode->prev = current\_ptr;

newnode->next = current\_ptr->next;

current\_ptr->next = newnode;

count++;

}

void insert\_at\_end(int value) {

current\_ptr = head;

//moving to end of list

while (current\_ptr->next != NULL) {

current\_ptr = current\_ptr->next;

}

node\* newnode = new node;

newnode->data = value;

newnode->prev = current\_ptr;

newnode->next = current\_ptr->next;

current\_ptr->next = newnode;

count++;

}

void delete\_at\_begin() {

node\* temp = head;

head = head->next;

delete temp;

temp = NULL;

}

//deletion code

void delete\_before(int position) {

current\_ptr = head;

int c = 1;

//moving to desired position to deleete

while (c < position) {

current\_ptr = current\_ptr->next;

c++;

}

node\* temp = current\_ptr;

current\_ptr->prev->next = current\_ptr->next;

current\_ptr->prev = current\_ptr->prev;

delete temp;

temp = NULL;

}

void delete\_at\_end() {

current\_ptr = head;

while (current\_ptr->next != NULL) {

current\_ptr = current\_ptr->next;

}

delete current\_ptr;

current\_ptr = NULL;

}

//display function

void display\_dlist() {

current\_ptr = head;

//moving to end of list

while (current\_ptr != NULL) {

cout << current\_ptr->data << "->";

current\_ptr = current\_ptr->next;

}

cout << endl;

}

//checing ismepty?

bool is\_empty() {

if (head == NULL && count == 0);

return true;

return false;

}

double\_llist() {

head = nullptr;

current\_ptr = nullptr; //constructor for my class

count = 0;

}

//destructor to delete the list

~double\_llist() {

current\_ptr = head;

while (current\_ptr != nullptr) {

node\* next = current\_ptr->next;

delete current\_ptr;

current\_ptr = next;

}

}

};

int main() {

double\_llist l;

int choice = 0;

bool exit = true;

int elem, pos;

while (exit)

{

cout << "Menu " << endl;

cout << "1-Insert at Begin: " << endl;

cout << "2-Insert After: " << endl;

cout << "3-Insert at End: " << endl;

cout << "4-Delete at Begin: " << endl;

cout << "5-Delete After: " << endl;

cout << "6-Delete at End: " << endl;

cout << "7-Display List: " << endl;

cout << "8-Quit " << endl;

cin >> choice;

if (choice == 1) {

cout << "Enter Elem: ";

cin >> elem;

l.insert\_at\_begin(elem);

}

else if (choice == 2) {

cout << "Enter Elem: ";

cin >> elem;

cout << "Enter Position: ";

cin >> pos;

l.insert\_after(elem, pos);

}

else if (choice == 3) {

cout << "Enter Elem: ";

cin >> elem;

l.insert\_at\_end(elem);

}

else if (choice == 4)

l.delete\_at\_begin();

else if (choice == 5) {

cout << "Enter Position: ";

cin >> pos;

l.delete\_before(pos);

}

else if (choice == 6)

l.delete\_at\_end();

else if (choice == 7) {

l.display\_dlist();

system("pause");

}

else

exit = false;

system("cls");

}

}

**Text

Description automatically generated**

**Task-2**

#include<iostream>

using namespace std;

struct node

{

int data;

struct node\* next;

struct node\* prev;

}

\*start;

//making a class list

class double\_llist

{

private:

node\* head;

node\* current\_ptr;

int count = 0;

public:

//methods of linked list

void insert\_at\_begin(int value) {

node\* temp = new node;

temp->data = value;

temp->next = head;

temp->prev = NULL;

head = temp;

count++;

}

void insert\_after(int value, int position) {

current\_ptr = head;

int c = 1;

//moving to desired position

while (c < position) {

current\_ptr = current\_ptr->next;

c++;

}

node\* newnode = new node;

newnode->data = value;

newnode->prev = current\_ptr;

newnode->next = current\_ptr->next;

current\_ptr->next = newnode;

count++;

}

void insert\_at\_end(int value) {

current\_ptr = head;

//moving to end of list

while (current\_ptr->next != NULL) {

current\_ptr = current\_ptr->next;

}

node\* newnode = new node;

newnode->data = value;

newnode->prev = current\_ptr;

newnode->next = current\_ptr->next;

current\_ptr->next = newnode;

count++;

}

void delete\_at\_begin() {

node\* temp = head;

head = head->next;

delete temp;

temp = NULL;

}

//deletion code

void delete\_before(int position) {

current\_ptr = head;

int c = 1;

//moving to desired position to deleete

while (c < position) {

current\_ptr = current\_ptr->next;

c++;

}

node\* temp = current\_ptr;

current\_ptr->prev->next = current\_ptr->next;

current\_ptr->prev = current\_ptr->prev;

delete temp;

temp = NULL;

}

void delete\_at\_end() {

current\_ptr = head;

while (current\_ptr->next != NULL) {

current\_ptr = current\_ptr->next;

}

delete current\_ptr;

current\_ptr = NULL;

}

//display function

void display\_dlist() {

current\_ptr = head;

//moving to end of list

while (current\_ptr != NULL) {

cout << current\_ptr->data << "->";

current\_ptr = current\_ptr->next;

}

cout << endl;

}

//checing ismepty?

bool is\_empty() {

if (head == NULL && count == 0);

return true;

return false;

}

double\_llist() {

head = nullptr;

current\_ptr = nullptr; //constructor for my class

count = 0;

}

//code to remove duplicates

void removeDuplicates()

{

node\* ptr1, \* ptr2, \* dup;

ptr1 = head;

//getting elements one by one

while (ptr1 != NULL && ptr1->next != NULL) {

ptr2 = ptr1;

//comparing elements

while (ptr2->next != NULL) {

if (ptr1->data == ptr2->next->data) {

dup = ptr2->next;

ptr2->next = ptr2->next->next;

delete (dup);

}

else

ptr2 = ptr2->next;

}

ptr1 = ptr1->next;

}

}

//destructor to delete the list

~double\_llist() {

current\_ptr = head;

while (current\_ptr != nullptr) {

node\* next = current\_ptr->next;

delete current\_ptr;

current\_ptr = next;

}

}

};

int main() {

double\_llist l;

int choice = 0;

bool exit = true;

int elem,pos;

while (exit)

{

cout << "Menu " << endl;

cout << "1-Insert at Begin: " << endl;

cout << "2-Insert After: " << endl;

cout << "3-Insert at End: " << endl;

cout << "4-Delete at Begin: " << endl;

cout << "5-Delete After: " << endl;

cout << "6-Delete at End: " << endl;

cout << "7-Display List: " << endl;

cout << "8-Remove Duplicates: " << endl;

cout << "9-Quit " << endl;

cin >> choice;

if (choice == 1) {

cout << "Enter Elem: ";

cin >> elem;

l.insert\_at\_begin(elem);

}

else if (choice == 2) {

cout << "Enter Elem: ";

cin >> elem;

cout << "Enter Position: ";

cin >> pos;

l.insert\_after(elem, pos);

}

else if (choice == 3) {

cout << "Enter Elem: ";

cin >> elem;

l.insert\_at\_end(elem);

}

else if (choice == 4)

l.delete\_at\_begin();

else if (choice == 5){

cout << "Enter Position: ";

cin >> pos;

l.delete\_before(pos);

}

else if (choice == 6)

l.delete\_at\_end();

else if (choice == 7) {

l.display\_dlist();

system("pause");

}

else if (choice == 8)

l.removeDuplicates();

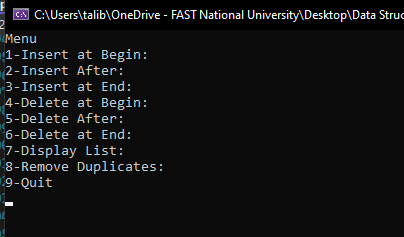
else

exit = false;

system("cls");

}

}

****

**Text

Description automatically generated Text

Description automatically generated**

**Task-3**

#include<iostream>

using namespace std;

struct node

{

int data;

struct node\* next;

struct node\* prev;

}

\*start;

//making a class list

class double\_llist

{

private:

node\* head;

node\* current\_ptr;

int count = 0;

public:

//methods of linked list

void insert\_at\_begin(int value) {

node\* temp = new node;

temp->data = value;

temp->next = head;

temp->prev = NULL;

head = temp;

count++;

}

void insert\_after(int value, int position) {

current\_ptr = head;

int c = 1;

//moving to desired position

while (c < position) {

current\_ptr = current\_ptr->next;

c++;

}

node\* newnode = new node;

newnode->data = value;

newnode->prev = current\_ptr;

newnode->next = current\_ptr->next;

current\_ptr->next = newnode;

count++;

}

void insert\_at\_end(int value) {

current\_ptr = head;

//moving to end of list

while (current\_ptr->next != NULL) {

current\_ptr = current\_ptr->next;

}

node\* newnode = new node;

newnode->data = value;

newnode->prev = current\_ptr;

newnode->next = current\_ptr->next;

current\_ptr->next = newnode;

count++;

}

void delete\_at\_begin() {

node\* temp = head;

head = head->next;

delete temp;

temp = NULL;

}

//deletion code

void delete\_before(int position) {

current\_ptr = head;

int c = 1;

//moving to desired position to deleete

while (c < position) {

current\_ptr = current\_ptr->next;

c++;

}

node\* temp = current\_ptr;

current\_ptr->prev->next = current\_ptr->next;

current\_ptr->prev = current\_ptr->prev;

delete temp;

temp = NULL;

}

void delete\_at\_end() {

current\_ptr = head;

while (current\_ptr->next != NULL) {

current\_ptr = current\_ptr->next;

}

delete current\_ptr;

current\_ptr = NULL;

}

//display function

void display\_dlist() {

current\_ptr = head;

//moving to end of list

while (current\_ptr != NULL) {

cout << current\_ptr->data << "->";

current\_ptr = current\_ptr->next;

}

cout << endl;

}

//checing ismepty?

bool is\_empty() {

if (head == NULL && count == 0);

return true;

return false;

}

double\_llist() {

head = nullptr;

current\_ptr = nullptr; //constructor for my class

count = 0;

}

//code to remove duplicates

void removeDuplicates()

{

node\* ptr1, \* ptr2, \* dup;

ptr1 = head;

//getting elements one by one

while (ptr1 != NULL && ptr1->next != NULL) {

ptr2 = ptr1;

//comparing elements

while (ptr2->next != NULL) {

if (ptr1->data == ptr2->next->data) {

dup = ptr2->next;

ptr2->next = ptr2->next->next;

delete (dup);

}

else

ptr2 = ptr2->next;

}

ptr1 = ptr1->next;

}

}

//code to separete even odd nodes

void EvenOdd(double\_llist even, double\_llist odd)

{

current\_ptr = head;

if (current\_ptr == NULL){

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

return;

}

else {

while (current\_ptr != NULL)

{

if (current\_ptr->data % 2 == 0)

even.insert\_at\_begin(current\_ptr->data);

else

odd.insert\_at\_begin(current\_ptr->data);

current\_ptr = current\_ptr->next;

}

}

cout << "Even List: ";

even.display\_dlist();

cout << "Odd List: ";

odd.display\_dlist();

system("pause");

}

//destructor to delete the list

~double\_llist() {

current\_ptr = head;

while (current\_ptr != nullptr) {

node\* next = current\_ptr->next;

delete current\_ptr;

current\_ptr = next;

}

}

};

int main() {

double\_llist l,l1,l2;

int choice = 0;

bool exit = true;

int elem, pos;

while (exit)

{

cout << "Menu " << endl;

cout << "1-Insert at Begin: " << endl;

cout << "2-Insert After: " << endl;

cout << "3-Insert at End: " << endl;

cout << "4-Delete at Begin: " << endl;

cout << "5-Delete After: " << endl;

cout << "6-Delete at End: " << endl;

cout << "7-Display List: " << endl;

cout << "8-Remove Duplicates: " << endl;

cout << "9-Separete Even Odd: "<<endl;

cout << "10-Quit " << endl;

cin >> choice;

if (choice == 1) {

cout << "Enter Elem: ";

cin >> elem;

l.insert\_at\_begin(elem);

}

else if (choice == 2) {

cout << "Enter Elem: ";

cin >> elem;

cout << "Enter Position: ";

cin >> pos;

l.insert\_after(elem, pos);

}

else if (choice == 3) {

cout << "Enter Elem: ";

cin >> elem;

l.insert\_at\_end(elem);

}

else if (choice == 4)

l.delete\_at\_begin();

else if (choice == 5) {

cout << "Enter Position: ";

cin >> pos;

l.delete\_before(pos);

}

else if (choice == 6)

l.delete\_at\_end();

else if (choice == 7) {

l.display\_dlist();

system("pause");

}

else if (choice == 8)

l.removeDuplicates();

else if (choice == 9) {

l.EvenOdd(l1,l2);

}

else

exit = false;

system("cls");

}

}

**Text

Description automatically generated Text

Description automatically generated**