**✅ C++ Program — Singly Linked List Operations with Input/Output**

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

};

class LinkedList {

private:

Node\* head;

public:

LinkedList() {

head = nullptr;

}

// 1. Insert at end

void insertEnd(int value) {

Node\* newNode = new Node{value, nullptr};

if (!head) {

head = newNode;

} else {

Node\* temp = head;

while (temp->next)

temp = temp->next;

temp->next = newNode;

}

}

// 2. Insert at beginning

void insertBeginning(int value) {

Node\* newNode = new Node{value, head};

head = newNode;

}

// 3. Insert at certain position

void insertAtPosition(int value, int pos) {

if (pos < 1) {

cout << "Invalid position.\n";

return;

}

if (pos == 1) {

insertBeginning(value);

return;

}

Node\* temp = head;

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

temp = temp->next;

if (temp == nullptr) {

cout << "Position out of range.\n";

return;

}

Node\* newNode = new Node{value, temp->next};

temp->next = newNode;

}

// 4. Delete from end

void deleteEnd() {

if (!head) {

cout << "List is empty.\n";

return;

}

if (!head->next) {

delete head;

head = nullptr;

return;

}

Node\* temp = head;

while (temp->next->next)

temp = temp->next;

delete temp->next;

temp->next = nullptr;

}

// 5. Delete from beginning

void deleteBeginning() {

if (!head) {

cout << "List is empty.\n";

return;

}

Node\* temp = head;

head = head->next;

delete temp;

}

// 6. Delete from certain position

void deleteAtPosition(int pos) {

if (!head || pos < 1) {

cout << "Invalid operation.\n";

return;

}

if (pos == 1) {

deleteBeginning();

return;

}

Node\* temp = head;

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

temp = temp->next;

if (!temp || !temp->next) {

cout << "Position out of range.\n";

return;

}

Node\* toDelete = temp->next;

temp->next = temp->next->next;

delete toDelete;

}

// 7. Search

void search(int value) {

Node\* temp = head;

int pos = 1;

while (temp) {

if (temp->data == value) {

cout << "Value found at position " << pos << endl;

return;

}

temp = temp->next;

pos++;

}

cout << "Value not found.\n";

}

// 8. Display

void display() {

Node\* temp = head;

if (!temp) {

cout << "List is empty.\n";

return;

}

cout << "Linked List: ";

while (temp) {

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

temp = temp->next;

}

cout << "NULL\n";

}

// 9. Update

void update(int oldVal, int newVal) {

Node\* temp = head;

while (temp) {

if (temp->data == oldVal) {

temp->data = newVal;

cout << "Value updated.\n";

return;

}

temp = temp->next;

}

cout << "Value not found to update.\n";

}

// 10. Reverse

void reverse() {

Node\* prev = nullptr;

Node\* current = head;

Node\* next = nullptr;

while (current) {

next = current->next;

current->next = prev;

prev = current;

current = next;

}

head = prev;

cout << "List reversed.\n";

}

};

int main() {

LinkedList list;

int choice, value, pos, oldVal, newVal;

do {

cout << "\n--- Menu ---\n";

cout << "1. Insert End\n2. Insert Beginning\n3. Insert at Position\n";

cout << "4. Delete End\n5. Delete Beginning\n6. Delete at Position\n";

cout << "7. Search\n8. Display\n9. Update\n10. Reverse\n11. Exit\n";

cout << "Enter choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter value: ";

cin >> value;

list.insertEnd(value);

break;

case 2:

cout << "Enter value: ";

cin >> value;

list.insertBeginning(value);

break;

case 3:

cout << "Enter position and value: ";

cin >> pos >> value;

list.insertAtPosition(value, pos);

break;

case 4:

list.deleteEnd();

break;

case 5:

list.deleteBeginning();

break;

case 6:

cout << "Enter position to delete: ";

cin >> pos;

list.deleteAtPosition(pos);

break;

case 7:

cout << "Enter value to search: ";

cin >> value;

list.search(value);

break;

case 8:

list.display();

break;

case 9:

cout << "Enter old value and new value: ";

cin >> oldVal >> newVal;

list.update(oldVal, newVal);

break;

case 10:

list.reverse();

break;

case 11:

cout << "Exiting...\n";

break;

default:

cout << "Invalid choice. Try again.\n";

}

} while (choice != 11);

return 0;

}

**✅ C++ Program — Operations on Array (Simulating Linked List Operations)**

#include <iostream>

using namespace std;

class ArrayList {

private:

int\* arr;

int capacity;

int size;

public:

ArrayList(int cap) {

capacity = cap;

arr = new int[capacity];

size = 0;

}

ArrayList() {

delete[] arr;

}

// 1. Insert at end

void insertEnd(int value) {

if (size == capacity) {

cout << "Array is full.\n";

return;

}

arr[size++] = value;

}

// 2. Insert at beginning

void insertBeginning(int value) {

if (size == capacity) {

cout << "Array is full.\n";

return;

}

for (int i = size; i > 0; i--) {

arr[i] = arr[i - 1];

}

arr[0] = value;

size++;

}

// 3. Insert at certain position

void insertAtPosition(int value, int pos) {

if (pos < 1 || pos > size + 1) {

cout << "Invalid position.\n";

return;

}

if (size == capacity) {

cout << "Array is full.\n";

return;

}

for (int i = size; i >= pos; i--) {

arr[i] = arr[i - 1];

}

arr[pos - 1] = value;

size++;

}

// 4. Delete from end

void deleteEnd() {

if (size == 0) {

cout << "Array is empty.\n";

return;

}

size--;

}

// 5. Delete from beginning

void deleteBeginning() {

if (size == 0) {

cout << "Array is empty.\n";

return;

}

for (int i = 0; i < size - 1; i++) {

arr[i] = arr[i + 1];

}

size--;

}

// 6. Delete from certain position

void deleteAtPosition(int pos) {

if (pos < 1 || pos > size) {

cout << "Invalid position.\n";

return;

}

for (int i = pos - 1; i < size - 1; i++) {

arr[i] = arr[i + 1];

}

size--;

}

// 7. Search

void search(int value) {

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

if (arr[i] == value) {

cout << "Value found at position " << i + 1 << endl;

return;

}

}

cout << "Value not found.\n";

}

// 8. Display

void display() {

if (size == 0) {

cout << "Array is empty.\n";

return;

}

cout << "Array: ";

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

cout << arr[i] << " ";

}

cout << endl;

}

// 9. Update

void update(int oldVal, int newVal) {

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

if (arr[i] == oldVal) {

arr[i] = newVal;

cout << "Value updated.\n";

return;

}

}

cout << "Value not found to update.\n";

}

// 10. Reverse

void reverse() {

int start = 0;

int end = size - 1;

while (start < end) {

int temp = arr[start];

arr[start] = arr[end];

arr[end] = temp;

start++;

end--;

}

cout << "Array reversed.\n";

}

};

int main() {

int capacity = 10; // Adjust the capacity as needed

ArrayList list(capacity);

int choice, value, pos, oldVal, newVal;

do {

cout << "\n--- Menu ---\n";

cout << "1. Insert End\n2. Insert Beginning\n3. Insert at Position\n";

cout << "4. Delete End\n5. Delete Beginning\n6. Delete at Position\n";

cout << "7. Search\n8. Display\n9. Update\n10. Reverse\n11. Exit\n";

cout << "Enter choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter value: ";

cin >> value;

list.insertEnd(value);

break;

case 2:

cout << "Enter value: ";

cin >> value;

list.insertBeginning(value);

break;

case 3:

cout << "Enter position and value: ";

cin >> pos >> value;

list.insertAtPosition(value, pos);

break;

case 4:

list.deleteEnd();

break;

case 5:

list.deleteBeginning();

break;

case 6:

cout << "Enter position to delete: ";

cin >> pos;

list.deleteAtPosition(pos);

break;

case 7:

cout << "Enter value to search: ";

cin >> value;

list.search(value);

break;

case 8:

list.display();

break;

case 9:

cout << "Enter old value and new value: ";

cin >> oldVal >> newVal;

list.update(oldVal, newVal);

break;

case 10:

list.reverse();

break;

case 11:

cout << "Exiting...\n";

break;

default:

cout << "Invalid choice. Try again.\n";

}

} while (choice != 11);

return 0;

}

**✅ C++ Program — Singly Linked List Using Pointers**

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

};

class LinkedList {

private:

Node\* head;

public:

LinkedList() {

head = nullptr;

}

// 1. Insert at end

void insertEnd(int value) {

Node\* newNode = new Node{value, nullptr};

if (head == nullptr) {

head = newNode;

} else {

Node\* temp = head;

while (temp->next != nullptr) {

temp = temp->next;

}

temp->next = newNode;

}

}

// 2. Insert at beginning

void insertBeginning(int value) {

Node\* newNode = new Node{value, head};

head = newNode;

}

// 3. Insert at a certain position

void insertAtPosition(int value, int pos) {

if (pos < 1) {

cout << "Invalid position.\n";

return;

}

if (pos == 1) {

insertBeginning(value);

return;

}

Node\* temp = head;

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

temp = temp->next;

}

if (temp == nullptr) {

cout << "Position out of range.\n";

return;

}

Node\* newNode = new Node{value, temp->next};

temp->next = newNode;

}

// 4. Delete from end

void deleteEnd() {

if (head == nullptr) {

cout << "List is empty.\n";

return;

}

if (head->next == nullptr) {

delete head;

head = nullptr;

return;

}

Node\* temp = head;

while (temp->next && temp->next->next) {

temp = temp->next;

}

delete temp->next;

temp->next = nullptr;

}

// 5. Delete from beginning

void deleteBeginning() {

if (head == nullptr) {

cout << "List is empty.\n";

return;

}

Node\* temp = head;

head = head->next;

delete temp;

}

// 6. Delete from certain position

void deleteAtPosition(int pos) {

if (head == nullptr || pos < 1) {

cout << "Invalid operation.\n";

return;

}

if (pos == 1) {

deleteBeginning();

return;

}

Node\* temp = head;

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

temp = temp->next;

}

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

cout << "Position out of range.\n";

return;

}

Node\* toDelete = temp->next;

temp->next = temp->next->next;

delete toDelete;

}

// 7. Search

void search(int value) {

Node\* temp = head;

int pos = 1;

while (temp != nullptr) {

if (temp->data == value) {

cout << "Value found at position " << pos << endl;

return;

}

temp = temp->next;

pos++;

}

cout << "Value not found.\n";

}

// 8. Display

void display() {

if (head == nullptr) {

cout << "List is empty.\n";

return;

}

Node\* temp = head;

cout << "Linked List: ";

while (temp != nullptr) {

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

temp = temp->next;

}

cout << "NULL\n";

}

// 9. Update

void update(int oldVal, int newVal) {

Node\* temp = head;

while (temp != nullptr) {

if (temp->data == oldVal) {

temp->data = newVal;

cout << "Value updated.\n";

return;

}

temp = temp->next;

}

cout << "Value not found to update.\n";

}

// 10. Reverse

void reverse() {

Node\* prev = nullptr;

Node\* curr = head;

Node\* next = nullptr;

while (curr != nullptr) {

next = curr->next;

curr->next = prev;

prev = curr;

curr = next;

}

head = prev;

cout << "List reversed.\n";

}

};

int main() {

LinkedList list;

int choice, value, pos, oldVal, newVal;

do {

cout << "\n--- Menu ---\n";

cout << "1. Insert End\n2. Insert Beginning\n3. Insert at Position\n";

cout << "4. Delete End\n5. Delete Beginning\n6. Delete at Position\n";

cout << "7. Search\n8. Display\n9. Update\n10. Reverse\n11. Exit\n";

cout << "Enter choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter value: ";

cin >> value;

list.insertEnd(value);

break;

case 2:

cout << "Enter value: ";

cin >> value;

list.insertBeginning(value);

break;

case 3:

cout << "Enter position and value: ";

cin >> pos >> value;

list.insertAtPosition(value, pos);

break;

case 4:

list.deleteEnd();

break;

case 5:

list.deleteBeginning();

break;

case 6:

cout << "Enter position to delete: ";

cin >> pos;

list.deleteAtPosition(pos);

break;

case 7:

cout << "Enter value to search: ";

cin >> value;

list.search(value);

break;

case 8:

list.display();

break;

case 9:

cout << "Enter old value and new value: ";

cin >> oldVal >> newVal;

list.update(oldVal, newVal);

break;

case 10:

list.reverse();

break;

case 11:

cout << "Exiting...\n";

break;

default:

cout << "Invalid choice. Try again.\n";

}

} while (choice != 11);

return 0;

}