**Pointers:**

#include <iostream>

using namespace std;

int main() {

int num = 10;

int\* ptr = &num;

\*ptr = 20; // Modifying value using pointer

cout << "Variable Value: " << num << endl;

cout << "Pointer Value: " << \*ptr << endl;

return 0;

}

**Big O Notation (Finding Maximum in Array):**

#include <iostream>

using namespace std;

int findMax(int arr[], int n) {

int maxVal = arr[0];

for (int i = 1; i < n; i++) {

if (arr[i] > maxVal)

maxVal = arr[i];

}

return maxVal;

}

int main() {

int arr[] = {3, 7, 2, 9, 5};

cout << "Max Value: " << findMax(arr, 5) << endl;

return 0;

}

**Singly Linked List (Insert at Start & End)**

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

Node(int val) : data(val), next(nullptr) {}

};

class LinkedList {

public:

Node\* head = nullptr;

void insertAtEnd(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 insertAtStart(int val) {

Node\* newNode = new Node(val);

newNode->next = head;

head = newNode;

}

void display() {

Node\* temp = head;

while (temp) {

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

temp = temp->next;

}

cout << "NULL" << endl;

}

};

int main() {

LinkedList list;

list.insertAtStart(5);

list.insertAtEnd(10);

list.insertAtStart(3);

list.display();

return 0;

}

**Singly Linked List (Insert at Specific Location)**

void insertAtPosition(Node\*& head, int val, int pos) {

Node\* newNode = new Node(val);

if (pos == 0) {

newNode->next = head;

head = newNode;

return;

}

Node\* temp = head;

for (int i = 0; i < pos - 1 && temp; i++) temp = temp->next;

if (!temp) return;

newNode->next = temp->next;

temp->next = newNode;

}

**Display Nodes (First, Last, Nth, Centre)**

void displayNodes(Node\* head) {

if (!head) return;

cout << "First: " << head->data << endl;

Node\* temp = head;

Node\* slow = head;

Node\* fast = head;

while (temp->next) {

if (fast && fast->next) {

fast = fast->next->next;

slow = slow->next;

}

temp = temp->next;

}

cout << "Last: " << temp->data << endl;

cout << "Centre: " << slow->data << endl;

}

**Delete Nodes (First, Last, Nth, Centre)**

void deleteFirst(Node\*& head) {

if (!head) return;

Node\* temp = head;

head = head->next;

delete temp;

}

void deleteLast(Node\*& head) {

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

delete head;

head = nullptr;

return;

}

Node\* temp = head;

while (temp->next->next) temp = temp->next;

delete temp->next;

temp->next = nullptr;

}

**Doubly Linked List (Insert & Display)**

struct DNode {

int data;

DNode\* prev, \*next;

DNode(int val) : data(val), prev(nullptr), next(nullptr) {}

};

class DoublyLinkedList {

public:

DNode\* head = nullptr;

void insertAtEnd(int val) {

DNode\* newNode = new DNode(val);

if (!head) head = newNode;

else {

DNode\* temp = head;

while (temp->next) temp = temp->next;

temp->next = newNode;

newNode->prev = temp;

}

}

void displayForward() {

DNode\* temp = head;

while (temp) {

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

temp = temp->next;

}

cout << "NULL" << endl;

}

};

**Merge Linked Lists (Singly & Doubly)**

Node\* mergeSinglyLists(Node\* head1, Node\* head2) {

if (!head1) return head2;

if (!head2) return head1;

Node\* temp = head1;

while (temp->next) temp = temp->next;

temp->next = head2;

return head1;

}

**Circular Linked List (Insert & Display)**

struct CNode {

int data;

CNode\* next;

CNode(int val) : data(val), next(nullptr) {}

};

void insertAtEndCircular(CNode\*& head, int val) {

CNode\* newNode = new CNode(val);

if (!head) {

head = newNode;

head->next = head;

} else {

CNode\* temp = head;

while (temp->next != head) temp = temp->next;

temp->next = newNode;

newNode->next = head;

}

}

void displayCircular(CNode\* head) {

if (!head) return;

CNode\* temp = head;

do {

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

temp = temp->next;

} while (temp != head);

cout << "(back to head)" << endl;

}