

**DATA STRUCTURES**

**LAB MANUAL**

**NAME: AYESHA IMRAN**

**REGISTRATION: 2023-BS-AI-061**

**SUBMITTED TO: MS. IRSHA QURESHI**

|  |  |  |
| --- | --- | --- |
| INTRODUCTION | LAB 1 | 2 |
| ARRAY | LAB 2 | 2 |
| MULTIDIMENTIONAL ARRAY | LAB 3 | 5 |
| VECTOR | LAB 4 | 9 |
| LIST | LAB 5 | 12 |
| STACK | LAB 6 | 15 |
| QUEUE | LAB 7 | 22 |
| SINGLE LINK LIST | LAB 8 | 30 |
| DOUBLE LINK LIST | LAB 9 | 32 |
| CIRCULAR LINK LIST | LAB 10 | 40 |
| BINARY SEARCH TREE | LAB 11 | 47 |

**TABLE OF CONTENT**

**LAB 1:**

Introduction to c++ and reviewing websites such as w3school also memorizing syntax of basic c++

**LAB 2**

**What is Array?**

An array is a data structure that stores a fixed-size sequential collection of elements of the same type. In other words, it’s a collection of variables (called elements), all of the same type, stored under a single variable name.

**PROGRAM 1:**

#include <iostream>

OUTPUT:

Ayesha

--------------------------------

Process exited after 0.1261 seconds with return value 0

#include <string>

using namespace std;

int main() {

string names[4] = {"Ayesha", "laiba", "shiza", "haiqa"};

cout << names[0];

return 0;

}

**PROGRAM 2:**

**OUTPUT:**

10

20

30

40

50

--------------------------------

Process exited after 0.1933 seconds with return value 0

#include <iostream>

using namespace std;

int main() {

int myNumbers[5] = {10, 20, 30, 40, 50};

for (int i = 0; i < 5; i++) {

cout << myNumbers[i] << "\n";

}

return 0;}

**PROGRAM 3:**

**OUTPUT:**

20

--------------------------------

Process exited after 0.1153 seconds with return value 0

#include <iostream>

using namespace std;

int main() {

int myNumbers[5] = {10, 20, 30, 40, 50};

cout << sizeof(myNumbers);

return 0;

}

**PROGRAM 4:**

**OUTPUT:**

Largest element: 33

--------------------------------

Process exited after 0.08207 seconds with return value 0

#include <iostream>

using namespace std;

int main() {

int arr[5] = {10, 25, 7, 33, 15};

int max = arr[0];

for (int i = 1; i < 5; i++) {

if (arr[i] > max) {

max = arr[i];}

}

cout << "Largest element: " << max << endl;

return 0;

}

**PROGRAM 5:**

**OUTPUT:**

Reversed Array: 50 40 30 20 10

--------------------------------

Process exited after 0.09812 seconds with return value 0

#include <iostream>

using namespace std;

int main() {

int arr[5] = {10, 20, 30, 40, 50};

cout << "Reversed Array: ";

for (int i = 4; i >= 0; i--) {

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

}

cout << endl;

return 0;

}

**PROGRAM 6:**

#include <iostream>

**OUTPUT:**

Average: 30

--------------------------------

Process exited after 0.1107 seconds with return value 0

using namespace std;

int main() {

int arr[5] = {10, 20, 30, 40, 50};

int sum = 0;

float average;

for (int i = 0; i < 5; i++) {

sum += arr[i];

}

average = sum / 5.0;

cout << "Average: " << average << end;

**LAB 3:**

**MULTIDIMENSIONAL ARRAY**

A **multidimensional array** is an array of arrays, where each element is itself an array. In a **2D array**, elements are arranged in rows and columns, forming a matrix-like structure. This allows you to store data in a tabular form, making it ideal for scenarios like storing matrices, tables, or grids.

**PROGRAM 1:**

**OUTPUT:**

1 2 3

4 5 6

--------------------------------

Process exited after 0.1092 seconds with return value 0

#include <iostream>

using namespace std;

int main() {

// Declare and initialize a 2x3 array (2 rows, 3 columns)

int matrix[2][3] = {

{1, 2, 3}, // First row

{4, 5, 6} // Second row

};

// Print the 2D array

for (int i = 0; i < 2; i++) {

for (int j = 0; j < 3; j++) {

cout << matrix[i][j] << " "; // Access elements using row and column indices

}

cout << endl; // Newline after each row

}

return 0;

}

**PROGRAM 2:**

#include <iostream>

using namespace std;

int main() {

// Declare and initialize a 3x2 array to store marks for 3 students in 2 subjects

int marks[3][2] = {

{85, 90}, // Marks for Student 1 in Subject 1 and Subject 2

**OUTPUT:**

Student 1 Marks: 85 90

Student 2 Marks: 78 82

Student 3 Marks: 92 88

--------------------------------

Process exited after 0.1233 seconds with return value 0

{78, 82}, // Marks for Student 2 in Subject 1 and Subject 2

{92, 88} // Marks for Student 3 in Subject 1 and Subject 2

};

// Display the marks

for (int i = 0; i < 3; i++) {

cout << "Student " << i+1 << " Marks: ";

for (int j = 0; j < 2; j++) {

cout << marks[i][j] << " "; // Print each student's marks

}

cout << endl;

}

return 0;

}

**PROGRAM 3:**

#include <iostream>

**OUTPUT:**

Amina

--------------------------------

Process exited after 0.2121 seconds with return value 0

#include <vector>

using namespace std;

int main() {

vector<string> names = {"Ayesha", "Laiba", "shiza", "haiqa"};

// Change the value of the first element

names[0] = "Amina";

cout << names[0];

return 0;

}

**PROGRAM 4:**

**OUTPUT:**

Ayesha

Ali

ahmad

haiqa

sara

-------------------------------

Process exited after 0.1991 seconds with return value 0

#include <iostream>

#include <vector>

using namespace std;

int main() {

vector<string> names = {"Ayesha", "Ali", "ahmad", "haiqa"};

names.push\_back("sara");

for (string name : names) {

cout << name << "\n";

}

return 0;

}

**PROGRAM 5:**

#include <iostream>

**OUTPUT:**

Volvo

BMW

Ford

--------------------------------

Process exited after 0.1348 seconds with return value 0

#include <vector>

using namespace std;

int main() {

vector<string> cars = {"Volvo", "BMW", "Ford", "Mazda"};

cars.pop\_back();

for (string car : cars) {

cout << car << "\n";

return 0;

}

**PROGRAM 6:**

#include <iostream>

**OUTPUT:**

Sum of the arrays:

6 8

10 12

--------------------------------

Process exited after 0.2499 seconds with return value 0

using namespace std;

int main() {

int arr1[2][2] = {{1, 2}, {3, 4}};

int arr2[2][2] = {{5, 6}, {7, 8}};

int sum[2][2];

for (int i = 0; i < 2; i++) {

for (int j = 0; j < 2; j++) {

sum[i][j] = arr1[i][j] + arr2[i][j];}

cout << "Sum of the arrays:" << endl;

for (int i = 0; i < 2; i++) {

for (int j = 0; j < 2; j++) {

cout << sum[i][j] << " ";

cout << endl;

**LAB 4:**

**What is a Vector?**

A vector is a dynamic array class in C++ that provides a flexible and efficient way to store and manipulate collections of elements.

**PROGRAM 1:**

**1. Declare and Initialize a Vector**

**OUTPUT:**

--------------------------------

Process exited after 0.06491 seconds with return value 0

#include <vector>

using namespace std;

int main() {

vector<int> myVector = {1, 2, 3, 4, 5};

return 0;

}

**PROGRAM 2:**

**1. Add Elements to a Vector**

**OUTPUT:**

--------------------------------

Process exited after 0.08569 seconds with return value 0

#include <vector>

using namespace std;

int main() {

vector<int> myVector;

myVector.push\_back(10);

myVector.push\_back(20);

return 0;

}

**PRORAM 3:**

**1. Access Vector Elements**

**OUTPUT:**

1

--------------------------------

Process exited after 0.09945 seconds with return value 0

#include <vector>

#include <iostream>

using namespace std;

int main() {

vector<int> myVector = {1, 2, 3, 4, 5};

cout << myVector[0] << endl; // Output: 1

return 0;

}

**PROGRAM 4:**

**1. Modify Vector Elements**

#include <vector>

**OUTPUT:**

10

--------------------------------

Process exited after 0.1515 seconds with return value 0

#include <iostream>

using namespace std;

int main() {

vector<int> myVector = {1, 2, 3, 4, 5};

myVector[0] = 10;

cout << myVector[0] << endl; // Output: 10

return 0;

}

**PROGRAM 5:  
1. Get Vector Size**

**OUTPUT:**

5

--------------------------------

Process exited after 0.07575 seconds with return value 0

#include <vector>

#include <iostream>

using namespace std;

int main() {

vector<int> myVector = {1, 2, 3, 4, 5};

cout << myVector.size() << endl; // Output: 5

return 0;

}

**PROGRAM 6:**

**Vector Access and Modify**

**OUTPUT:**

Before modification: 1

After modification: 10

--------------------------------

Process exited after 0.08732 seconds with return value 0

#include <vector>

#include <iostream>

using namespace std;

int main() {

vector<int> myVector = {1, 2, 3, 4, 5};

cout << "Before modification: " << myVector[0] << endl;

myVector[0] = 10;

cout << "After modification: " << myVector[0] << endl;

return 0

# LAB NO 5

**LIST:**

A list is similar to a [vector](https://www.w3schools.com/cpp/cpp_vectors.asp) in that it can store multiple elements of the same type and dynamically grow in size.

#include <iostream>

#include <list>

using namespace std;

**PROGRAM 1:**

**OUTPUT:**

Ayesha

Laiba

Shiza

Haiqa

int main() {

// Create a list called cars that will store strings

list<string> names = {"Ayesha", "laiba", "shiza", "haiqa"};

// Print list elements

for (string name: names) {

cout << name << "\n";

}

return 0;

}

**PROGRAM 2:**

**OUTPUT:**

Ayesha

haiqa

--------------------------------

Process exited after 0.09327 seconds with return value 0

#include <iostream>

#include <list>

using namespace std;

int main() {

list<string> names = {"Ayesha", "laiba", "shiza", "haiqa"};

cout << names.front() << "\n";

cout << names.back() << "\n";

**PROGRAM 3:**

#include <iostream>

**OUTPUT:**

Amina

sara

--------------------------------

Process exited after 0.09945 seconds with return value 0

#include <list>

using namespace std;

int main() {

list<string> names = {"Ayesha", "laiba", "shiza", "haiqa"};

// Change the value of the first element

names.front() = “Amina ";

// Change the value of the last element

names.back() = "sara";

cout << names.front() << "\n";

cout << names.back() << "\n";

return 0;

}

**PROGRAM 4:**

int main() {

**OUTPUT:**

Laiba

Shiza

list<string> names = {"Ayesha", "laiba", "shiza", "haiqa"};

// Remove the first element

names.pop\_front();

// Remove the last element

names.pop\_back();

// Print list elements

for (string name : names) {

cout << name << "\n";

}

return 0}

**PROGRAM 5:**

**OUTPUT:**

4

--------------------------------

Process exited after 0.1427 seconds with return value 0

#include <iostream>

#include <list>

using namespace std;

int main() {

list<string> names = {"Ayesha", "laiba", "shiza", "haiqa"};

cout << names.size();

return 0;

}

**PROGRAM 6:**

#include <iostream>

**OUTPUT:**

Ayesha

laiba

shiza

haiqa

--------------------------------

Process exited after 0.107 seconds with return value 0

#include <list>

using namespace std;

int main() {

// Create a list called cars that will store strings

list<string> names = {"Ayesha", "laiba", "shiza", "haiqa"};

// Print list elements

for (string name : names) {

cout << name << "\n";

}

return 0;

}

# LAB NO 6

# STACK:

# A stack stores multiple elements in a specific order, called LIFO.

LIFO stands for Last in, First Out. To vizualise LIFO, think of a pile of pancakes, where pancakes are both added and removed from the top. So when removing a pancake, it will always be the last one you added.

**PROGRAM 1:**

**OUTPUT:**

Ayesha

--------------------------------

Process exited after 0.107 seconds with return value 0

#include <stack>

#include<iostream>

Using namespace std;

stack<string> names;

stack<string> names = {"Ayesha", "laiba", "shiza", "haiqa"};

cout << names.top();

return 0;

}

**PROGRAM 2:**

#include <iostream>

**OUTPUT:**

sara

--------------------------------

Process exited after 0.1085 seconds with return value 0

#include <stack>

using namespace std;

int main() {

stack<string> names;

names.push("ayesha");

names.push("laiba");

names.push("shiza");

names.top() = "sara";

cout << names.top();

**PROGRAM 3:**

#include <iostream>

**OUTPUT:**

sara

--------------------------------

Process exited after 0.1036 seconds with return value 0

#include <stack>

using namespace std;

int main() {

stack<string> names;

names.push("ayesha");

names.push("laiba");

names.push("shiza");

names.push("haiqa");

names.top() = "sara";

// Access the top element

cout << names.top();

return 0;

}

**PROGRAM 4:**

**OUTPUT:**

4

--------------------------------

Process exited after 0.09614 seconds with return value 0

#include <iostream>

#include <stack>

using namespace std;

int main() {

stack<string> names;

names.push("ayesha");

names.push("laiba");

names.push("shiza");

names.push("haiqa");

cout << names.size();

return 0;}

**PROGRAM 5:**

**OUTPUT:**  
0

--------------------------------

Process exited after 0.09951 seconds with return value 0

#include <iostream>

#include <stack>

using namespace std;

int main() {

// Create a stack of strings called cars

stack<string> names;

// Add elements to the stack

names.push("ayesha");

names.push("laiba");

names.push("shiza");

names.push("haiqa");

// Get the size of the stack

cout << names.empty();

return 0;

}

**INFIX TO POSTFIX:**

**PROGRAM:**

#include <iostream>

**OUTPUT:**

Infix: (A + B) \* (C - D)

Postfix: A B + C D - \*

#include <stack>

#include <string>

// Function to check the precedence of operators

int precedence(char op) {

if (op == '+' || op == '-') return 1;

if (op == '\*' || op == '/') return 2;

return 0;

}

// Function to convert infix to postfix

std::string infixToPostfix(std::string infix) {

std::stack<char> stack;

std::string postfix = "";

for (char c : infix) {

if (c == ' ') continue;

if (c == '(') stack.push(c);

else if (c == ')') {

while (stack.top() != '(') {

postfix += stack.top();

stack.pop();

}

stack.pop();

}

else if (c == '+' || c == '-' || c == '\*' || c == '/') {

while (!stack.empty() && precedence(c) <= precedence(stack.top())) {

postfix += stack.top();

stack.pop();

}

stack.push(c);

}

else postfix += c;

}

while (!stack.empty()) {

postfix += stack.top();

stack.pop();

}

return postfix;

}

int main() {

std::string infix = "(A + B) \* (C - D)";

std::string postfix = infixToPostfix(infix);

std::cout << "Infix: " << infix << std::endl;

std::cout << "Postfix: " << postfix << std::endl;

return 0;

**PROGRAM:**

#include <iostream>

using namespace std;

**OUTPUT:**

Stack elements: 10 20 30 40 50

Popped element: 50

Popped element: 40

Stack elements after popping: 10 20 30

class Stack {

int arr[10], top = -1;

public:

void push(int x) { arr[++top] = x; }

int pop() { return arr[top--]; }

bool isEmpty() { return top == -1; }

void print() {

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

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

}

cout << endl;

}

};

int main() {

Stack s;

s.push(10);

s.push(20);

s.push(30);

s.push(40);

s.push(50);

cout << "Stack elements: ";

s.print();

cout << "Popped element: " << s.pop() << endl;

cout << "Popped element: " << s.pop() << endl;

cout << "Stack elements after popping: ";

s.print();

return 0;

}

**LAB 7:**

**QUEUE:**

**PROGRAM 1:**

**OUTPUT:**

Front: 1

Back: 3

--------------------------------

Process exited after 0.09793 seconds with return value 0

**Queue Implementation**

#include <iostream>

#include <queue>

using namespace std;

int main() {

queue<int> q;

q.push(1);

q.push(2);

cout << "Front: " << q.front() << endl;

cout << "Back: " << q.back() << endl;

return 0;

**PROGRAM 2:**

**OUTPUT:**

Queue size: 3

--------------------------------

Process exited after 0.09523 seconds with return value 0

**Queue Size**

#include <iostream>

#include <queue>

using namespace std;

int main() {

queue<int> q;

q.push(1);

q.push(2);

q.push(3);

cout << "Queue size: " << q.size() << endl;

return 0;

**PROGRAM 3:**

**Enqueue and Dequeue**

**OUTPUT:**  
Dequeued: 1

Dequeued: 2

--------------------------------

Process exited after 0.09337 seconds with return value 0

#include <iostream>

#include <queue>

using namespace std;

int main() {

queue<int> q;

q.push(1);

q.push(2);

cout << "Dequeued: " << q.front() << endl;

q.pop();

cout << "Dequeued: " << q.front() << endl;

q.pop();

return 0;

}

**PROGRAM 4:**

**Empty Queue**

**OUTPUT:**

Is queue empty? Yes

Is queue empty? No

--------------------------------

Process exited after 0.08892 seconds with return value 0

#include <iostream>

#include <queue>

using namespace std;

int main() {

queue<int> q;

cout << "Is queue empty? " << (q.empty() ? "Yes" : "No") << endl;

q.push(1);

cout << "Is queue empty? " << (q.empty() ? "Yes" : "No") << endl;

return 0;

}

**PROGRAM 5:  
Front and Back**

**OUTPUT:**

Front: 1

Back: 3

--------------------------------

Process exited after 0.147 seconds with return value 0

#include <iostream>

#include <queue>

using namespace std;

int main() {

queue<int> q;

q.push(1);

q.push(2);

q.push(3);

cout << "Front: " << q.front() << endl;

cout << "Back: " << q.back() << endl;

return 0;}

**PROGRAM 6:**

**Queue Operations**

**OUTPUT:**

Front: 1

Front: 2

Back: 4

--------------------------------

Process exited after 0.1558 seconds with return value 0

#include <iostream>

#include <queue>

using namespace std;

int main() {

queue<int> q;

q.push(1);

q.push(2);

cout << "Front: " << q.front() << endl;

q.pop();

cout << "Front: " << q.front() << endl;

q.push(4);

cout << "Back: " << q.back() << endl;

**LAB 8:**

**SINGLE LINK LIST**

**PROGRAM 1:**

**INSERTION:**

#include <iostream>

**OUTPUT:**

Linked List: 30 20 10 50 40

--------------------------------

Process exited after 0.2315 seconds with return value 0

using namespace std;

// Node structure

struct Node {

int data;

Node\* next;

};

**// Function to insert a new node at the front**

void insertFront(Node\*\* head, int data) {

Node\* newNode = new Node();

newNode->data = data;

newNode->next = \*head;

\*head = newNode;

}

**// Function to insert a new node at the middle**

void insertMid(Node\*\* head, int data, int pos) {

Node\* newNode = new Node();

newNode->data = data;

Node\* temp = \*head;

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

temp = temp->next;

newNode->next = temp->next;

temp->next = newNode;

**// Function to insert a new node at the end**

void insertLast(Node\*\* head, int data) {

Node\* newNode = new Node();

newNode->data = data;

if (\*head == NULL) {

\*head = newNode;

return;

Node\* temp = \*head;

while (temp->next != NULL) {

temp = temp->next;

temp->next = newNode;

}

void printList(Node\* head) {

Node\* temp = head;

while (temp != NULL) {

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

temp = temp->next;

cout << endl;

}

int main() {

Node\* head = NULL;

insertFront(&head, 10);

insertFront(&head, 20);

insertFront(&head, 30);

cout << "Linked List: "

**PROGRAM 2:**

**DELETION:**

#include <iostream>

**OUTPUT:**

Linked List: 10 20 30 40 50

Linked List after deletion: 20 30

--------------------------------

Process exited after 0.2323 seconds with return value 0

using namespace std;

// Node structure

struct Node {

int data;

Node\* next;

};

**// Function to delete a node from the front**

void deleteFront(Node\*\* head) {

if (\*head == NULL) return;

Node\* temp = \*head;

\*head = (\*head)->next;

free(temp);

**// Function to delete a node from the middle**

void deleteMid(Node\*\* head, int pos) {

if (\*head == NULL) return;

Node\* temp = \*head;

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

temp = temp->next;

Node\* nodeToDelete = temp->next;

temp->next = nodeToDelete->next;

free(nodeToDelete);

}

void deleteLast(Node\*\* head) {

if (\*head == NULL) return;

if ((\*head)->next == NULL) {

free(\*head);

\*head = NULL;

Node\* temp = \*head;

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

temp = temp->next;

free(temp->next);

temp->next = NULL;

}

void printList(Node\* head) {

Node\* temp = head;

while (temp != NULL) {

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

temp = temp->next;

cout << endl;

int main() {

Node\* head = NULL;

insertLast(&head, 10);

insertLast(&head, 20);

insertLast(&head, 30);

insertLast(&head, 40);

insertLast(&head, 50);

cout << "Linked List: ";

cout << "Linked List after deletion: ";

printList(head);

return 0;

}

**PROGRAM 3:**

**SEARCHING:**

#include <iostream>

**OUTPUT:**

Found

Not Found

--------------------------------

Process exited after 0.1029 seconds with return value 0

using namespace std;

struct Node {

int data;

Node\* next;

};

void insertLast(Node\*\* head, int data) {

Node\* newNode = new Node();

newNode->data = data;

newNode->next = NULL;

}

bool search(Node\* head, int data) {

Node\* temp = head;

while (temp) {

if (temp->data == data) return true;

temp = temp->next;

}

return false;

}

int main() {

Node\* head = NULL;

insertLast(&head, 10);

insertLast(&head, 20);

cout << (search(head, 20) ? "Found" : "Not Found") << endl; // Output: Found

cout << (search(head, 40) ? "Found" : "Not Found") << endl; // Output: Not Found

**PROGRAM 4:**

**TRAVERSING:**

#include <iostream>

using namespace std;

**OUTPUT:**

10 20 30

--------------------------------

Process exited after 0.09761 seconds with return value 0

struct Node {

int data;

Node\* next;

};

void insertLast(Node\*\* head, int data) {

Node\* newNode = new Node();

newNode->data = data;

newNode->next = NULL;

if (\*head == NULL) \*head = newNode;

else {

Node\* temp = \*head;

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

temp->next = newNode;

void traverse(Node\* head) {

Node\* temp = head;

while (temp) {

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

temp = temp->next;

cout << endl;

int main() {

Node\* head = NULL;

insertLast(&head, 10);

insertLast(&head, 30);

**PROGRAM 5:**

**FIND INDEX:**

**OUTPUT:**

Index of 20: 1

Index of 40: -1

--------------------------------

Process exited after 0.1048 seconds with return value 0

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

};

void insertLast(Node\*\* head, int data) {

Node\* newNode = new Node();

newNode->data = data;

newNode->next = NULL;

if (\*head == NULL) \*head = newNode;

int findIndex(Node\* head, int data) {

while (temp) {

if (temp->data == data) return index;

temp = temp->next;

index++;

}

return -1; // Node not found

int main() {

Node\* head = NULL;

insertLast(&head, 10);

insertLast(&head, 20);

insertLast(&head, 30);

cout << "Index of 20: " << findIndex(head, 20) << endl; // Output: 1

cout << "Index of 40: " << findIndex(head, 40) << endl; // Output: -1

**LAB 9:**

**DOUBLE LINK LIST:**

**PROGRAM 1:**

#include <iostream>

using namespace std;

**// Node structure**

OUTPUT:

Linked List: 30 20 10 50 40

--------------------------------

Process exited after 0.2315 seconds with return value 0

struct Node {

int data;

Node\* prev;

Node\* next;

};

**// Constructor**

DoublyLinkedList() {

head = NULL;

}

**// Insert at front**

void insertFront(int data) {

Node\* newNode = new Node();

newNode->data = data;

newNode->prev = NULL;

if (head == NULL) {

newNode->next = NULL;

head = newNode;

} else {

newNode->next = head;

head->prev = newNode;

head = newNode;

**// Insert at middle**

void insertMid(int data, int pos) {

Node\* newNode = new Node();

newNode->data = data;

if (pos == 0) {

insertFront(data);

return;

}

Node\* temp = head;

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

temp = temp->next;

}

newNode->prev = temp;

newNode->next = temp->next;

if (temp->next != NULL) {

temp->next->prev = newNode;

}

temp->next = newNode;

**// Insert at last**

void insertLast(int data) {

Node\* newNode = new Node();

newNode->data = data;

newNode->next = NULL;

if (head == NULL) {

newNode->prev = NULL;

head = newNode;

**PROGRAM 2:**

**Searching**

**OUTPUT:**

Search for 20: Found

Search for 40: Not Found

--------------------------------

Process exited after 0.09327 seconds with return value 0

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* prev;

Node\* next;

};

bool search(Node\* head, int data) {

Node\* temp = head;

while (temp != NULL) {

if (temp->data == data) return true;

temp = temp->next;

}

int main() {

Node\* head = NULL;

// Insert nodes...

head = new Node();

head->data = 10;

head->prev = NULL;

head->next = new Node();

head->next->next->data = 30;

head->next->next->prev = head->next;

head->next->next->next = NULL;

cout << "Search for 20: " << (search(head, 20) ? "Found" : "Not Found") << endl;

cout << "Search for 40: " << (search(head, 40) ? "Found" : "Not Found") << endl;

**PROGRAM 3:**

**Editing/Updating**

#include <iostream>

using namespace std;

**OUTPUT:**

Original List: 10 20 30

Updated List: 10 40 30

--------------------------------

Process exited after 0.09181 seconds with return value 0

struct Node {

int data;

Node\* prev;

Node\* next;

};

void update(Node\* head, int oldData, int newData) {

Node\* temp = head;

while (temp != NULL) {

if (temp->data == oldData) {

temp->data = newData;

return;

temp = temp->next;

void printList(Node\* head) {

Node\* temp = head;

while (temp != NULL) {

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

temp = temp->next;

}

cout << endl;

}

int main() {

Node\* head = NULL;

// Insert nodes...

head = new Node();

head->data = 10;

head->prev = NULL;

head->next = new Node();

head->next->data = 20;

head->next->prev = head;

head->next->next = new Node();

head->next->next->data = 30;

head->next->next->prev = head->next;

head->next->next->next = NULL;

cout << "Original List: ";

printList(head);

update(head, 20, 40);

cout << "Updated List: ";

printList(head);

return 0;

}

**PROGRAM 4:**

**Finding Index**

#include <iostream>

**OUTPUT:**

Index of 20: 1

Index of 40: -1

--------------------------------

Process exited after 0.08062 seconds with return value 0

using namespace std;

struct Node {

int data;

Node\* prev;

Node\* next;

};

int findIndex(Node\* head, int data) {

Node\* temp = head;

while (temp != NULL) {

if (temp->data == data) return index;

temp = temp->next;

index++;

int main() {

Node\* head = NULL;

// Insert nodes...

head = new Node();

head->data = 10;

head->prev = NULL;

head->next = new Node();

head->next->data = 20;

head->next->next = new Node();

head->next->next->next = NULL;

cout << "Index of 20: " << findIndex(head, 20) << endl;

cout << "Index of 40: " << findIndex(head, 40) << endl;

**PROGRAM 5:**

#include <iostream>

**OUTPUT:**

Original List: 10 20 30

List after deletion: 10 30

--------------------------------

Process exited after 0.1364 seconds with return value 0

using namespace std;

struct Node {

int data;

Node\* prev;

Node\* next;

};

void deleteNode(Node\*\* head, int data) {

Node\* temp = \*head;

while (temp != NULL) {

if (temp->data == data) {

if (temp->prev != NULL) temp->prev->next = temp->next;

else \*head = temp->next;

if (temp->next != NULL) temp->next->prev = temp->prev;

delete temp;

return;

}

temp = temp->next;

}

}

void printList(Node\* head) {

Node\* temp = head;

while (temp != NULL) {

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

temp = temp->next;

}

cout << endl;

}

int main() {

Node\* head = NULL;

head = new Node();

head->data = 10;

head->prev = NULL;

head->next = new Node();

head->next->data = 20;

head->next->prev = head;

head->next->next = new Node();

head->next->next->data = 30;

head->next->next->prev = head->next;

head->next->next->next = NULL;

cout << "Origial List: ";

printList(head);

deleteNode(&head, 20);

cout << "List after deletion: ";

printList(head);

return 0;

}

**LAB 10:**

**CIRCULAR LINK LIST:**

**PROGRAM1:**

**OUTPUT:**

Circular Linked List: 1 2 3 4 5

--------------------------------

Process exited after 0.08969 seconds with return value 0

#include <iostream>

using namespace std;

// Node structure

struct Node {

int data;

Node\* next;

};

// Create function to add values to the list

void create(int data) {

Node\* newNode = new Node();

newNode->data = data;

if (head == NULL) {

head = newNode;

newNode->next = head;

} else {

Node\* temp = head;

while (temp->next != head) {

temp = temp->next;

}

temp->next = newNode;

newNode->next = head;

}

}

// Display function to print all elements

void display() {

Node\* temp = head;

if (head == NULL) {

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

return;

}

do {

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

temp = temp->next;

} while (temp != head);

cout << endl;

}

};

int main() {

CircularLinkedList cll;

// Create a circular linked list with numbers 1, 2, 3, 4, and 5

cll.create(1);

cll.create(2);

cll.create(3);

cll.create(4);

cll.create(5);

// Display the circular linked list

cout << "Circular Linked List: ";

cll.display();

return 0;

}

**PROGRAM 2:**

#include <iostream>

using namespace std;

**OUTPUT:**

Original List: 5 4 3 2 1

Updated List: 0 5 4 3 2 1

--------------------------------

Process exited after 0.08308 seconds with return value 0

struct Node {

int data;

Node\* next;

};

void insertAtBeginning(Node\*\* head, int data) {

Node\* newNode = new Node();

newNode->data = data;

if (\*head == NULL) {

\*head = newNode;

newNode->next = \*head;

}

temp->next = newNode;

newNode->next = \*head;

\*head = newNode;

void display(Node\* head) {

Node\* temp = head;

int main() {

Node\* head = NULL;

insertAtBeginning(&head, 1);

insertAtBeginning(&head, 2);

insertAtBeginning(&head, 3);

insertAtBeginning(&head, 4);

insertAtBeginning(&head, 5);

cout << "Original List: ";

**PROGRAM 3:**

#include <iostream>

**OUTPUT:**

Original List: 1 2 3 4 5

Updated List: 1 2 3 4 5 19

--------------------------------

Process exited after 0.09242 seconds with return value 0

using namespace std;

struct Node {

int data;

Node\* next;

};

void insertAfter(Node\*\* head, int value, int data) {

Node\* newNode = new Node();

newNode->data = data;

Node\* temp = \*head;

do {

if (temp->data == value) {

newNode->next = temp->next;

temp->next = newNode;

return;

}

temp = temp->next;

} while (temp != \*head);

}

void display(Node\* head) {

Node\* temp = head;

do {

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

temp = temp->next;

} while (temp != head);

cout << endl;

}

int main() {

Node\* head = new Node();

head->data = 1;

head->next = new Node();

head->next->data = 2;

head->next->next = new Node();

head->next->next->data = 3;

head->next->next->next = new Node();

head->next->next->next->data = 4;

head->next->next->next->next = new Node();

head->next->next->next->next->data = 5;

head->next->next->next->next->next = head;

cout << "Original List: ";

display(head);

insertAfter(&head, 5, 19);

cout << "Updated List: ";

display(head);

return 0;

}

**PROGRAM 4:**

#include <iostream>

using namespace std;

**OUTPUT:**

Original List: 1 2 3 4 5

Updated List: 1 2 3 5

--------------------------------

Process exited after 0.07979 seconds with return value 0

struct Node {

int data;

Node\* next;

};

void deleteNode(Node\*\* head, int value) {

if (\*head == NULL) return;

if ((\*head)->data == value) {

Node\* temp = \*head;

\*head = (\*head)->next;

if (\*head != NULL) (\*head)->next = temp->next;

delete temp;

return;

}

Node\* temp = \*head;

while (temp->next != NULL) {

if (temp->next->data == value) {

Node\* nodeToDelete = temp->next;

temp->next = nodeToDelete->next;

if (nodeToDelete->next != NULL) nodeToDelete->next->prev = temp;

delete nodeToDelete;

return;

}

temp = temp->next;

}

void display(Node\* head) {

Node\* temp = head;

do {

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

temp = temp->next;

} while (temp != head);

cout << endl;

}

int main() {

Node\* head = new Node();

head->data = 1;

head->next = new Node();

head->next->data = 2;

head->next->next = new Node();

head->next->next->data = 3;

head->next->next->next = new Node();

head->next->next->next->data = 4;

head->next->next->next->next = new Node();

head->next->next->next->next->data = 5;

head->next->next->next->next->next = head;

cout << "Original List: ";

display(head);

deleteNode(&head, 4);

cout << "Updated List: ";

display(head);

return 0;

**LAB 11:**

**Binary Search Tree**

A Binary Search Tree (BST) is a data structure in which each node has at most two children (i.e., left child and right child) and each node represents a value. All the values in the left subtree of a node are less than the value in the node, and all the values in the right subtree of a node are greater than the value in the node.

1. Insert: Inserts a new value into the BST.

2. Delete: Deletes a value from the BST.

3. Search: Searches for a value in the BST.

4. Traversal: Traverses the BST in a specific order (e.g., inorder, preorder, postorder).

**PROGRAM 1:**

**INSERTION:**

struct Node {

**OUTPUT:**

Inorder traversal: 20 30 40 50 60 70 80

--------------------------------

Process exited after 0.09774 seconds with return value 0

int data; **// data stored in the node**

node\* left; **// pointer to the left child node**

node\* right; **// pointer to the right child node**

};

**// Function to insert a new node into the BST**

Node\* insertNode(Node\* root, int data) {

if (root == NULL) {

return new Node{data, NULL, NULL}; **// Create a new node with the given data**

if (data < root->data) {

root->left = insertNode(root->left, data); **// Insert into the left subtree**

} else if (data > root->data) {

root->right = insertNode(root->right, data);} **// Insert into the right subtree**

return root; }**// return the updated root node**

**PROGRAM 2:**

**DELETION:**

**OUTPUT:**

Original BST: 20 30 40 50 60 70 80

BST after deletion: 20 40 50 60 70 80

--------------------------------

Process exited after 0.108 seconds with return value 0

struct Node {

int data;

Node\* left;

Node\* right;

};

**// Function to delete a node from the BST**

Node\* deleteNode(Node\* root, int data) {

if (root == NULL) return NULL;

if (data < root->data) root->left = deleteNode(root->left, data);

else if (data > root->data) root->right = deleteNode(root->right, data);

else {

if (root->left == NULL) return root->right;

else if (root->right == NULL) return root->left;

Node\* temp = root->right;

while (temp->left != NULL) temp = temp->left;

root->data = temp->data;

root->right = deleteNode(root->right, temp->data) }

return root;

}

**PROGRAM 3:**

**SEARCHING:**

struct Node {

**OUTPUT:**

Node found with data: 40

--------------------------------

Process exited after 0.08493 seconds with return value 0

int data;

Node\* left;

Node\* right;

};

// Function to search for a node in the BST

Node\* searchBST(Node\* root, int data) {

// Base case: If the tree is empty, return NULL

if (root == NULL) return NULL;

// If the data is found, return the node

if (root->data == data) return root;

// Recursive case: Search in the left or right subtree

if (data < root->data) return searchBST(root->left, data);

return searchBST(root->right, data);

}

**PROGRAM 4:**

**DUPLICATION:**

**OUTPUT:**

Duplicated BST:

--------------------------------

Process exited after 0.08939 seconds with return value 0

struct Node {

int data;

Node\* left;

Node\* right;

};

**// Function to create a new BST node**

Node\* createNode(int data) {

Node\* newNode = new Node();

newNode->data = data;

newNode->left = newNode->right = NULL;

return newNode;}

**// Function to duplicate a BST**

Node\* duplicateBST(Node\* root) {

if (root == NULL) return NULL;

Node\* newRoot = createNode(root->data);

newRoot->left = duplicateBST(root->left);

newRoot->right = duplicateBST(root->right);

return newRoot;}

**PROGRAM 5:**

**TRAVERSING:**

struct Node {

**OUTPUT:**  
Inorder Traversal: 20 30 40 50 60 70 80

Preorder Traversal: 50 30 20 40 70 60 80

Postorder Traversal: 20 40 30 60 80 70 50

--------------------------------

Process exited after 0.09832 seconds with return value 0

int data;

Node\* left;

Node\* right;

};

**// Function to create a new BST node**

Node\* createNode(int data) {

Node\* newNode = new Node();

newNode->data = data;

newNode->left = newNode->right = NULL;

return newNode;

}

**// Function for inorder traversal**

void inorderTraversal(Node\* root) {

if (root) {

inorderTraversal(root->left);

cout << root->data << " ";

inorderTraversal(root->right); }

}

**// Function for preorder traversal**

void preorderTraversal(Node\* root) {

if (root) {

cout << root->data << " ";

preorderTraversal(root->left);

preorderTraversal(root->right); }

}

**// Function for postorder traversal**

void postorderTraversal(Node\* root) {

if (root) {

postorderTraversal(root->left);

postorderTraversal(root->right);

cout << root->data << " "; }

}

**PROGRAM 6:**

**OUTPUT:**

Inorder Traversal: 3 5 7 10 15 20

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* left;

Node\* right;

};

Node\* createNode(int data) {

Node\* newNode = new Node();

if (!newNode) {

cout << "Memory error\n";

return NULL;

}

newNode->data = data;

newNode->left = newNode->right = NULL;

return newNode;

}

Node\* insert(Node\* root, int data) {

if (root == NULL) {

root = createNode(data);

return root;

}

if (data <= root->data) {

root->left = insert(root->left, data);

}

else {

root->right = insert(root->right, data);

}

return root;

}

void inorder(Node\* root) {

if (root) {

inorder(root->left);

cout << root->data << " ";

inorder(root->right);

}

}

int main() {

Node\* root = NULL;

root = insert(root, 10);

root = insert(root, 5);

root = insert(root, 20);

root = insert(root, 3);

root = insert(root, 7);

root = insert(root, 15);

cout << "Inorder Traversal: ";

inorder(root);

cout << endl;

return 0;

}