**DAILY PRACTICE CODING**

1. **Write a program to check whether the number is even or odd.**

#include <iostream>

using namespace std;

int main() {

int num;

cout << "Enter a number: ";

cin >> num;

if (num % 2 == 0) {

cout << num << " is even." << endl;

} else {

cout << num << " is odd." << endl;

}

return 0;

}

1. **Write a program to check if a number is negative, positive or zero.**

#include <iostream>

using namespace std;

int main() {

int num;

cout << "Enter a number: ";

cin >> num;

if (num > 0) {

cout << num << " is positive." << endl;

} else if (num < 0) {

cout << num << " is negative." << endl;

} else {

cout << "The number is zero." << endl;

}

return 0;

}

1. **Write a program to find the maximum of three numbers.**

#include <iostream>

using namespace std;

int main() {

int num1, num2, num3, max;

cout << "Enter three numbers: ";

cin >> num1 >> num2 >> num3;

max = (num1 > num2) ? ((num1 > num3) ? num1 : num3) : ((num2 > num3) ? num2 : num3);

cout << "Maximum number: " << max << endl;

return 0;

}

1. **Write a program to swap two numbers using a temporary variable.**

#include <iostream>

int main() {

int num1, num2, temp;

cout << "Enter two numbers: ";

cin >> num1 >> num2;

temp = num1;

num1 = num2;

num2 = temp;

cout << "After swapping, num1 = " << num1 << ", num2 = " << num2 << endl;

return 0;

}

1. **Write a program to calculate the factorial of a number.**

#include <iostream>

int main() {

int num, factorial = 1;

cout << "Enter a number: ";

cin >> num;

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

factorial \*= i;

}

cout << "Factorial: " << factorial << endl;

return 0;

}

1. **Write a program to find maximum of two numbers.**

#include <iostream>

using namespace std;

int main() {

int num1, num2;

cout << "Enter two numbers: ";

cin >> num1 >> num2;

int max;

if (num1 > num2) {

max = num1;

} else {

max = num2;

}

cout << "The maximum is " << max << "\n";

return 0;

}

1. **Write a program to calculate the area of a circle.**

#include <iostream>

int main() {

double radius;

cout << "Enter the radius of the circle: ";

cin >> radius;

double area = 3.14159 \* radius \* radius;

cout << "The area of the circle is " << area << "\n";

return 0;

}

1. **Write a program to calculate the factorial of a number.**

#include <iostream>

using namespace std;

int main() {

int num;

cout << "Enter a number: ";

cin >> num;

int factorial = 1;

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

factorial \*= i;

}

cout << "The factorial of " << num << " is " << factorial << "\n";

return 0;

}

1. **Write a program to check if a number is prime.**

#include <iostream>

using namespace std;

int main() {

int num;

cout << "Enter a number: ";

cin >> num;

bool is\_prime = true;

for (int i = 2; i <= num / 2; i++) {

if (num % i == 0) {

is\_prime = false;

break;

}

}

if (is\_prime) {

cout << num << " is prime\n";

} else {

cout << num << " is not prime\n";

}

return 0;

}

1. **Write a program to find the sum of all even numbers between 1 and 100.**

#include <iostream>

using namespace std;

int main() {

int sum = 0;

for (int i = 2; i <= 100; i += 2) {

sum += i;

}

cout << "The sum of all even numbers between 1 and 100 is:” << sum << “\n”;

return 0;

}

1. **How to find the length of a string in C++?**

#include <iostream>

#include <string>

using namespace std;

int main() {

string str = "Hello World";

cout << "Length of string: " << str.length() << endl;

return 0;

}

1. **How to concatenate two strings in C++?**

#include <iostream>

#include <string>

using namespace std;

int main() {

string str1 = "Hello";

string str2 = "World";

string result = str1 + " " + str2;

cout << result << endl;

return 0;

}

1. **How to compare two strings in C++?**

#include <iostream>

#include <string>

using namespace std;

int main() {

string str1 = "Hello";

string str2 = "World";

if(str1 == str2) {

cout << "Strings are equal" << endl;

}

else {

cout << "Strings are not equal" << endl;

}

return 0;

}

1. **How to find a substring in a string in C++?**

#include <iostream>

#include <string>

using namespace std;

int main() {

string str = "Hello World";

string sub = "World";

if(str.find(sub) != string::npos) {

cout << "Substring found" << endl;

}

else {

cout << "Substring not found" << endl;

}

return 0;

}

1. **How to convert a string to uppercase in C++?**

#include <iostream>

#include <string>

#include <algorithm>

using namespace std;

int main() {

string str = "Hello World";

transform(str.begin(), str.end(), str.begin(), ::toupper);

cout << str << endl;

return 0;

}

1. **How to convert a string to lowercase in C++?**

#include <iostream>

#include <string>

#include <algorithm>

using namespace std;

int main() {

string str = "Hello World";

transform(str.begin(), str.end(), str.begin(), ::tolower);

cout << str << endl;

return 0;

}

1. **How to remove whitespace from a string in C++?**

#include <iostream>

#include <string>

#include <algorithm>

using namespace std;

int main() {

string str = " Hello World ";

str.erase(remove\_if(str.begin(), str.end(), ::isspace), str.end());

cout << str << endl;

return 0;

}

1. **How to reverse a string in C++?**

#include <iostream>

#include <string>

#include <algorithm>

using namespace std;

int main() {

string str = "Hello World";

reverse(str.begin(), str.end());

cout << str << endl;

return 0;

}

1. **How to convert a string to an integer in C++?**

#include <iostream>

#include <string>

#include <sstream>

using namespace std;

int main() {

string str = "12345";

int num;

stringstream(str) >> num;

cout << num << endl;

return 0;

}

1. **How to convert an integer to a string in C++?**

#include <iostream>

#include <string>

#include <sstream>

using namespace std;

int main() {

int num = 12345;

stringstream ss;

ss << num;

string str = ss.str();

cout << str << endl

1. **Write a program to find the sum of all elements of an array.**

#include<iostream>

using namespace std;

int main() {

int arr[5] = {1, 2, 3, 4, 5};

int sum = 0;

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

sum += arr[i];

}

cout << "Sum of array elements: " << sum << endl;

return 0;

}

1. **Write a program to find the largest element of an array.**

#include<iostream>

using namespace std;

int main() {

int arr[5] = {1, 2, 3, 4, 5};

int max = arr[0];

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

if(arr[i] > max) {

max = arr[i];

}

}

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

return 0;

}

1. **Write a program to find the smallest element of an array.**

#include<iostream>

using namespace std;

int main() {

int arr[5] = {1, 2, 3, 4, 5};

int min = arr[0];

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

if(arr[i] < min) {

min = arr[i];

}

}

cout << "Smallest element of array: " << min << endl;

return 0;

}

1. **Write a program to reverse an array.**

#include<iostream>

using namespace std;

int main() {

int arr[5] = {1, 2, 3, 4, 5};

int start = 0, end = 4;

while(start < end) {

int temp = arr[start];

arr[start] = arr[end];

arr[end] = temp;

start++;

end--;

}

cout << "Reversed array: ";

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

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

}

cout << endl;

return 0;

}

1. **Write a program to find the second largest element of an array.**

#include<iostream>

using namespace std;

int main() {

int arr[5] = {1, 2, 3, 4, 5};

int max = arr[0], secondMax = arr[0];

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

if(arr[i] > max) {

secondMax = max;

max = arr[i];

}

else if(arr[i] > secondMax && arr[i] != max) {

secondMax = arr[i];

}

}

cout << "Second largest element of array: " << secondMax << endl;

return 0;

}

1. **Write a program to find the frequency of each element of an array.**

#include<iostream>

using namespace std;

int main() {

int arr[5] = {1, 2, 3, 4, 1};

int freq[5];

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

freq[i] = -1;

}

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

int count = 1;

for(int j = i + 1

1. **How to declare and initialize an array in C++?**

#include <iostream>

using namespace std;

int main() {

int arr[5] = {1, 2, 3, 4, 5};

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

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

}

return 0;

}

1. **How to access elements of an array in C++?**

#include <iostream>

using namespace std;

int main() {

int arr[5] = {1, 2, 3, 4, 5};

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

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

}

return 0;

}

1. **How to find the length of an array in C++?**

#include <iostream>

using namespace std;

int main() {

int arr[5] = {1, 2, 3, 4, 5};

int length = sizeof(arr)/sizeof(arr[0]);

cout << "Length of array: " << length << endl;

return 0;

}

1. **How to sort an array in C++?**

#include <iostream>

#include <algorithm>

using namespace std;

int main() {

int arr[5] = {5, 4, 3, 2, 1};

sort(arr, arr+5);

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

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

}

return 0;

}

1. **How to search an element in an array in C++?**

#include <iostream>

#include <algorithm>

using namespace std;

int main() {

int arr[5] = {1, 2, 3, 4, 5};

int key = 3;

if(find(arr, arr+5, key) != arr+5) {

cout << "Element found" << endl;

}

else {

cout << "Element not found" << endl;

}

return 0;

}

1. **How to remove an element from an array in C++?**

#include <iostream>

using namespace std;

int main() {

int arr[5] = {1, 2, 3, 4, 5};

int n = 5, key = 3;

int j = 0;

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

if(arr[i] != key) {

arr[j] = arr[i];

j++;

}

}

n = j;

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

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

}

return 0;

}

1. **How to find the maximum element of an array in C++?**

#include <iostream>

using namespace std;

int main() {

int arr[5] = {1, 2, 3, 4, 5};

int max = arr[0];

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

if(arr[i] > max) {

max = arr[i];

}

}

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

return 0;

}

1. **How to find the minimum element of an array in C++?**

#include <iostream>

#include <algorithm>

int main() {

int arr[] = {5, 2, 9, 1, 7};

int size = sizeof(arr) / sizeof(arr[0]);

// Finding the minimum element

int\* minElement = std::min\_element(arr, arr + size);

std::cout << "Minimum element: " << \*minElement << std::endl;

return 0;

}

1. **Write a function to reverse a linked list.**

#include <iostream>

struct Node {

int data;

Node\* next;

};

Node\* reverseLinkedList(Node\* head) {

Node\* prev = nullptr;

Node\* curr = head;

Node\* next = nullptr;

while (curr != nullptr) {

next = curr->next;

curr->next = prev;

prev = curr;

curr = next;

}

return prev;

}

int main() {

Node\* head = new Node{1, nullptr};

head->next = new Node{2, nullptr};

head->next->next = new Node{3, nullptr};

head->next->next->next = new Node{4, nullptr};

head->next->next->next->next = new Node{5, nullptr};

Node\* current = head;

while (current != nullptr) {

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

current = current->next;

}

std::cout << std::endl;

head = reverseLinkedList(head);

current = head;

while (current != nullptr) {

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

current = current->next;

}

std::cout << std::endl;

// Free memory

current = head;

while (current != nullptr) {

Node\* temp = current;

current = current->next;

delete temp;

}

return 0;

}

1. **Write a function to check if a linked list has a cycle.**

#include <iostream>

struct Node {

int data;

Node\* next;

};

bool hasCycle(Node\* head) {

if (head == nullptr || head->next == nullptr)

return false;

Node\* slow = head;

Node\* fast = head;

while (fast != nullptr && fast->next != nullptr) {

slow = slow->next;

fast = fast->next->next;

if (slow == fast)

return true;

}

return false;

}

int main() {

Node\* head = new Node{1, nullptr};

head->next = new Node{2, nullptr};

head->next->next = new Node{3, nullptr};

head->next->next->next = new Node{4, nullptr};

head->next->next->next->next = new Node{5, nullptr};

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

// Check if the linked list has a cycle

bool cycleExists = hasCycle(head);

if (cycleExists)

std::cout << "The linked list has a cycle." << std::endl;

else

std::cout << "The linked list does not have a cycle." << std::endl;

// Free memory

Node\* current = head;

while (current != nullptr) {

Node\* temp = current;

current = current->next;

delete temp;

}

return 0;

}

1. **Write a function to search for a specific element in an array.**

#include <iostream>

int linearSearch(int arr[], int size, int target) {

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

if (arr[i] == target)

return i;

}

return -1; // Element not found

}

int main() {

int arr[] = {5, 2, 9, 1, 7};

int size = sizeof(arr) / sizeof(arr[0]);

int target = 9;

int index = linearSearch(arr, size, target);

if (index != -1)

std::cout << "Element found at index " << index << std::endl;

else

std::cout << "Element not found" << std::endl;

return 0;

}

1. **Write a function to search for a specific element in a sorted array using binary search.**

#include <iostream>

int binarySearch(int arr[], int size, int target) {

int left = 0;

int right = size - 1;

while (left <= right) {

int mid = left + (right - left) / 2;

if (arr[mid] == target)

return mid;

if (arr[mid] < target)

left = mid + 1;

else

right = mid - 1;

}

return -1; // Element not found

}

int main() {

int arr[] = {1, 2, 3, 4, 5, 6, 7, 8, 9};

int size = sizeof(arr) / sizeof(arr[0]);

int target = 6;

int index = binarySearch(arr, size, target);

if (index != -1)

std::cout << "Element found at index " << index << std::endl;

else

std::cout << "Element not found" << std::endl;

return 0;

}

1. **Write a function to search for a specific element in a rotated sorted array.**

#include <iostream>

int searchInRotatedArray(int arr[], int size, int target) {

int left = 0;

int right = size - 1;

while (left <= right) {

int mid = left + (right - left) / 2;

if (arr[mid] == target)

return mid;

if (arr[left] <= arr[mid]) {

if (arr[left] <= target && target < arr[mid])

right = mid - 1;

else

left = mid + 1;

} else {

if (arr[mid] < target && target <= arr[right])

left = mid + 1;

else

right = mid - 1;

}

}

return -1; // Element not found

}

int main() {

int arr[] = {7, 8, 9, 1, 2, 3, 4, 5, 6};

int size = sizeof(arr) / sizeof(arr[0]);

int target = 3;

int index = searchInRotatedArray(arr, size, target);

if (index != -1)

std::cout << "Element found at index " << index << std::endl;

else

std::cout << "Element

1. **Write a function to sort an array of integers in ascending order using the bubble sort algorithm.**

#include <iostream>

void bubbleSort(int arr[], int size) {

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

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

if (arr[j] > arr[j + 1]) {

// Swap elements

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

int main() {

int arr[] = {5, 2, 9, 1, 7};

int size = sizeof(arr) / sizeof(arr[0]);

bubbleSort(arr, size);

std::cout << "Sorted array: ";

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

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

}

std::cout << std::endl;

return 0;

}

1. **Write a function to sort an array of integers in ascending order using the insertion sort algorithm.**

#include <iostream>

void insertionSort(int arr[], int size) {

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

int key = arr[i];

int j = i - 1;

while (j >= 0 && arr[j] > key) {

arr[j + 1] = arr[j];

--j;

}

arr[j + 1] = key;

}

}

int main() {

int arr[] = {5, 2, 9, 1, 7};

int size = sizeof(arr) / sizeof(arr[0]);

insertionSort(arr, size);

std::cout << "Sorted array: ";

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

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

}

std::cout << std::endl;

return 0;

}

1. **Write a function to sort an array of integers in ascending order using the merge sort algorithm.**

#include <iostream>

void merge(int arr[], int left, int mid, int right) {

int n1 = mid - left + 1;

int n2 = right - mid;

int\* leftArr = new int[n1];

int\* rightArr = new int[n2];

for (int i = 0; i < n1; ++i)

leftArr[i] = arr[left + i];

for (int j = 0; j < n2; ++j)

rightArr[j] = arr[mid + 1 + j];

int i = 0;

int j = 0;

int k = left;

while (i < n1 && j < n2) {

if (leftArr[i] <= rightArr[j]) {

arr[k] = leftArr[i];

++i;

} else {

arr[k] = rightArr[j];

++j;

}

++k;

}

while (i < n1) {

arr[k] = leftArr[i];

++i;

++k;

}

while (j < n2) {

arr[k] = rightArr[j];

++j;

++k;

}

1. **Write a function to calculate the height of a binary tree.**

#include <iostream>

struct TreeNode {

int data;

TreeNode\* left;

TreeNode\* right;

};

int calculateHeight(TreeNode\* root) {

if (root == nullptr)

return 0;

int leftHeight = calculateHeight(root->left);

int rightHeight = calculateHeight(root->right);

return std::max(leftHeight, rightHeight) + 1;

}

int main() {

// Create a sample binary tree

TreeNode\* root = new TreeNode{1, nullptr, nullptr};

root->left = new TreeNode{2, nullptr, nullptr};

root->right = new TreeNode{3, nullptr, nullptr};

root->left->left = new TreeNode{4, nullptr, nullptr};

root->left->right = new TreeNode{5, nullptr, nullptr};

int height = calculateHeight(root);

std::cout << "Height of the binary tree: " << height << std::endl;

return 0;

}

1. **Write a function to check if a binary tree is balanced.**

#include <iostream>

struct TreeNode {

int data;

TreeNode\* left;

TreeNode\* right;

};

int calculateHeight(TreeNode\* root) {

if (root == nullptr)

return 0;

int leftHeight = calculateHeight(root->left);

int rightHeight = calculateHeight(root->right);

return std::max(leftHeight, rightHeight) + 1;

}

bool isBalanced(TreeNode\* root) {

if (root == nullptr)

return true;

int leftHeight = calculateHeight(root->left);

int rightHeight = calculateHeight(root->right);

int heightDiff = std::abs(leftHeight - rightHeight);

if (heightDiff > 1)

return false;

return isBalanced(root->left) && isBalanced(root->right);

}

int main() {

// Create a sample binary tree

TreeNode\* root = new TreeNode{1, nullptr, nullptr};

root->left = new TreeNode{2, nullptr, nullptr};

root->right = new TreeNode{3, nullptr, nullptr};

root->left->left = new TreeNode{4, nullptr, nullptr};

root->left->right = new TreeNode{5, nullptr, nullptr};

bool balanced = isBalanced(root);

if (balanced)

std::cout << "The binary tree is balanced." << std::endl;

else

std::cout << "The binary tree is not balanced." << std::endl;

return 0;

}

**66.Implement a Queue using an array with the enqueue and dequeue operations.**

#include <iostream>

const int MAX\_SIZE = 100;

class Queue {

private:

int arr[MAX\_SIZE];

int front;

int rear;

public:

Queue() {

front = -1;

rear = -1;

}

bool isEmpty() {

return (front == -1 && rear == -1);

}

bool isFull() {

return (rear + 1) % MAX\_SIZE == front;

}

void enqueue(int data) {

if (isFull()) {

std::cout << "Queue is full. Unable to enqueue." << std::endl;

return;

}

if (isEmpty()) {

front = 0;

rear = 0;

} else {

rear = (rear + 1) % MAX\_SIZE;

}

arr[rear] = data;

}

void dequeue() {

if (isEmpty()) {

std::cout << "Queue is empty. Unable to dequeue." << std::endl;

return;

}

if (front == rear) {

front = -1;

rear = -1;

} else {

front = (front + 1) % MAX\_SIZE;

}

}

int getFront() {

if (isEmpty()) {

std::cout << "Queue is empty." << std::endl;

return -1;

}

return arr[front];

}

};

int main() {

Queue q;

q.enqueue(5);

q.enqueue(2);

q.enqueue(9);

std::cout << "Front element: " << q.getFront() << std::endl;

q.dequeue();

std::cout << "Front element after dequeue: " << q.getFront() << std::endl;

return 0;

}

**67.Implement a Queue using linked list with the enqueue and dequeue operations.**

#include <iostream>

struct Node {

int data;

Node\* next;

};

class Queue {

private:

Node\* front;

Node\* rear;

public:

Queue() {

front = nullptr;

rear = nullptr;

}

bool isEmpty() {

return (front == nullptr && rear == nullptr);

}

void enqueue(int data) {

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

if (isEmpty()) {

front = newNode;

rear = newNode;

} else {

rear->next = newNode;

rear = newNode;

}

}

void dequeue() {

if (isEmpty()) {

std::cout << "Queue is empty. Unable to dequeue." << std::endl;

return;

}

Node\* temp = front;

front = front->next;

if (front == nullptr)

rear = nullptr;

delete temp;

}

int getFront() {

if (isEmpty()) {

std::cout << "Queue is empty." << std::endl;

return -1;

}

return front->data;

}

};

int main() {

Queue q;

q.enqueue(5);

q.enqueue(2);

q.enqueue(9);

std::cout << "Front element: " << q.getFront() << std::endl;

q.dequeue();

std::cout << "Front element after dequeue: " << q.getFront() << std::endl;

return 0;

}

**68.Implement a Stack using an array with the push and pop operations.**

#include <iostream>

const int MAX\_SIZE = 100;

class Stack {

private:

int arr[MAX\_SIZE];

int top;

public:

Stack() {

top = -1;

}

bool isEmpty() {

return top == -1;

}

bool isFull() {

return top == MAX\_SIZE - 1;

}

void push(int data) {

if (isFull()) {

std::cout << "Stack is full. Unable to push." << std::endl;

return;

}

arr[++top] = data;

}

void pop() {

if (isEmpty()) {

std::cout << "Stack is empty. Unable to pop." << std::endl;

return;

}

--top;

}

int getTop() {

if (isEmpty()) {

std::cout << "Stack is empty." << std::endl;

return -1;

}

return arr[top];

}

};

int main() {

Stack s;

s.push(5);

s.push(2);

s.push(9);

std::cout << "Top element: " << s.getTop() << std::endl;

s.pop();

std::cout << "Top element after pop: " << s.getTop() << std::endl;

return 0;

}

**69.Implement a Stack using linked list with the push and pop operations.**

#include <iostream>

struct Node {

int data;

Node\* next;

};

class Stack {

private:

Node\* top;

public:

Stack() {

top = nullptr;

}

bool isEmpty() {

return top == nullptr;

}

void push(int data) {

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

if (isEmpty()) {

top = newNode;

} else {

newNode->next = top;

top = newNode;

}

}

void pop() {

if (isEmpty()) {

std::cout << "Stack is empty. Unable to pop." << std::endl;

return;

}

Node\* temp = top;

top = top->next;

delete temp;

}

int getTop() {

if (isEmpty()) {

std::cout << "Stack is empty." << std::endl;

return -1;

}

return top->data;

}

};

int main() {

Stack s;

s.push(5);

s.push(2);

s.push(9);

std::cout << "Top element: " << s.getTop() << std::endl;

s.pop();

std::cout << "Top element after pop: " << s.getTop() << std::endl;

return 0;

}

1. **Implement a Max Heap data structure with the following operations: insert, extractMax, and getMax.**

#include <iostream>

#include <vector>

class MaxHeap {

private:

std::vector<int> heap;

int getParentIndex(int index) {

return (index - 1) / 2;

}

int getLeftChildIndex(int index) {

return (2 \* index) + 1;

}

int getRightChildIndex(int index) {

return (2 \* index) + 2;

}

void heapifyUp(int index) {

while (index > 0 && heap[index] > heap[getParentIndex(index)]) {

std::swap(heap[index], heap[getParentIndex(index)]);

index = getParentIndex(index);

}

}

void heapifyDown(int index) {

int largestIndex = index;

int leftChildIndex = getLeftChildIndex(index);

int rightChildIndex = getRightChildIndex(index);

if (leftChildIndex < heap.size() && heap[leftChildIndex] > heap[largestIndex]) {

largestIndex = leftChildIndex;

}

if (rightChildIndex < heap.size() && heap[rightChildIndex] > heap[largestIndex]) {

largestIndex = rightChildIndex;

}

if (largestIndex != index) {

std::swap(heap[index], heap[largestIndex]);

heapifyDown(largestIndex);

}

}

public:

void insert(int data) {

heap.push\_back(data);

heapifyUp(heap.size() - 1);

}

int extractMax() {

if (heap.empty()) {

std::cout << "Heap is empty." << std::endl;

return -1;

}

int maxElement = heap[0];

heap[0] = heap.back();

heap.pop\_back();

heapifyDown(0);

return maxElement;

}

int getMax() {

if (heap.empty()) {

std::cout << "Heap is empty." << std::endl;

return -1;

}

return heap[0];

}

};

int main() {

MaxHeap maxHeap;

maxHeap.insert(5);

maxHeap.insert(2);

maxHeap.insert(9);

maxHeap.insert(1);

maxHeap.insert(7);

std::cout << "Max element: " << maxHeap.getMax() << std::endl;

int extracted = maxHeap.extractMax();

std::cout << "Extracted max element: " << extracted << std::endl;

std::cout << "Max element after extraction: " << maxHeap.getMax() << std::endl;

return 0;

}

1. **Given an array of integers, find the kth largest element using a Min Heap.**

#include <iostream>

#include <queue>

#include <vector>

int findKthLargestElement(const std::vector<int>& nums, int k) {

std::priority\_queue<int, std::vector<int>, std::greater<int>> minHeap;

for (int num : nums) {

if (minHeap.size() < k) {

minHeap.push(num);

} else if (num > minHeap.top()) {

minHeap.pop();

minHeap.push(num);

}

}

return minHeap.top();

}

int main() {

std::vector<int> nums = {3, 2, 1, 5, 6, 4};

int k = 2;

int kthLargest = findKthLargestElement(nums, k);

std::cout << "The " << k << "th largest element is: " << kthLargest << std::endl;

return 0;

}

**71.**