1) Write a program to sort given array using bubble sort technique

Code:-

#include <iostream>

void swap(int &a, int &b) {

int temp = a;

a = b;

b = temp;

}

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

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

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

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

// Swap if the element found is greater than the next element

swap(arr[j], arr[j + 1]);

}

}

}

}

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

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

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

}

std::cout << std::endl;

}

int main() {

int arr[] = {64, 34, 25, 12, 22, 11, 90};

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

std::cout << "Unsorted array: ";

printArray(arr, n);

bubbleSort(arr, n);

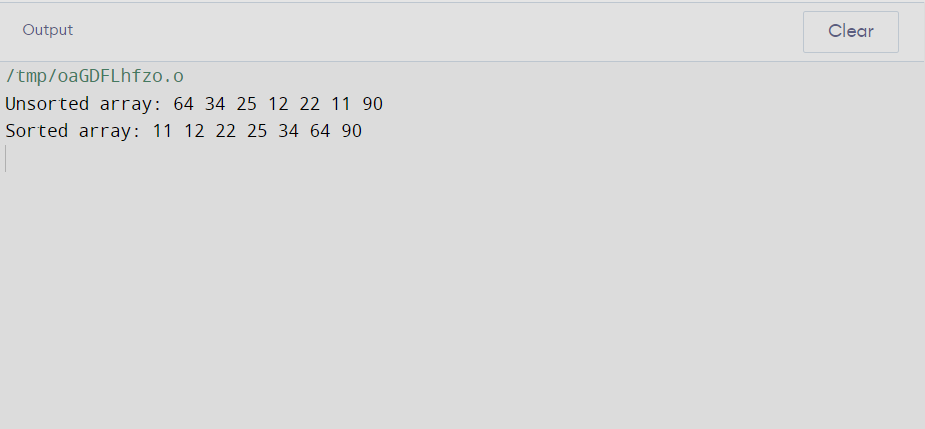
std::cout << "Sorted array: ";

printArray(arr, n);

return 0;

}

Output:-



2) Write a program to search element in given array using linear search algorithm

Code:-

#include <iostream>

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

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

if (arr[i] == target) {

// Element found, return its index

return i;

}

}

// If element is not found, return -1

return -1;

}

int main() {

int arr[] = {64, 34, 25, 12, 22, 11, 90};

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

int target;

std::cout << "Enter the element to search: ";

std::cin >> target;

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

if (index != -1) {

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

} else {

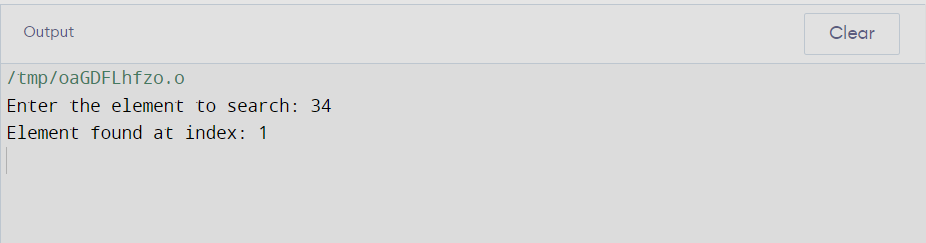
std::cout << "Element not found in the array." << std::endl;

}

return 0;

}

Output:-



3) Write a program to implement stack using array

Code:-

#include <iostream>

#include <climits> // Include the header for INT\_MIN

#include <cstdlib>

#define MAX\_SIZE 100

class Stack {

private:

int arr[MAX\_SIZE];

int top;

public:

Stack() {

top = -1;

}

void push(int data) {

if (top >= MAX\_SIZE - 1) {

std::cout << "Stack Overflow: Cannot push element, stack is full." << std::endl;

return;

}

arr[++top] = data;

std::cout << data << " pushed into the stack." << std::endl;

}

void pop() {

if (top < 0) {

std::cout << "Stack Underflow: Cannot pop element, stack is empty." << std::endl;

return;

}

std::cout << arr[top--] << " popped from the stack." << std::endl;

}

int peek() {

if (top < 0) {

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

return INT\_MIN; // Assuming INT\_MIN represents an empty stack

}

return arr[top];

}

bool isEmpty() {

return (top < 0);

}

};

int main() {

Stack stack;

stack.push(10);

stack.push(20);

stack.push(30);

std::cout << "Top element: " << stack.peek() << std::endl;

stack.pop();

stack.pop();

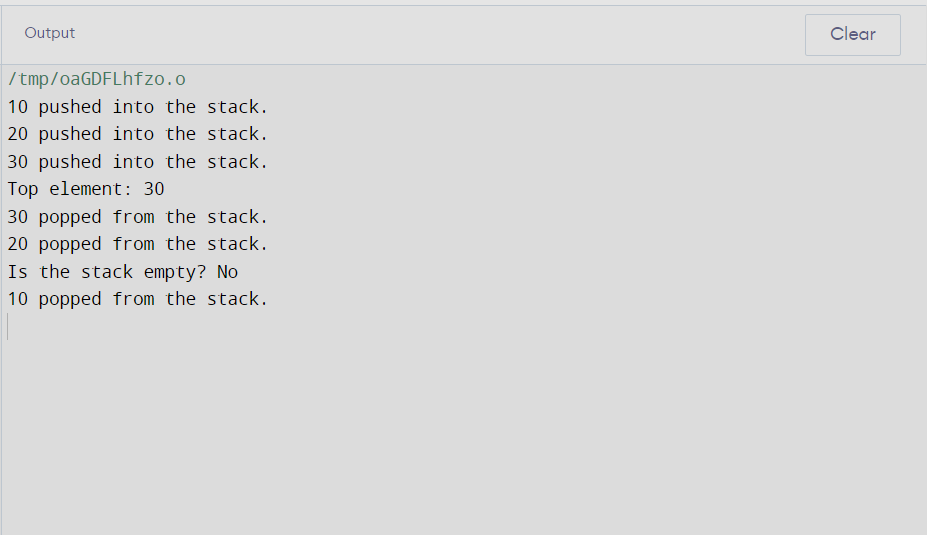
std::cout << "Is the stack empty? " << (stack.isEmpty() ? "Yes" : "No") << std::endl;

stack.pop(); // popping the last element

return 0;

}

Output:-



4)Write a program to implement balancing of parentheses using stack

Code:-

#include <iostream>

#include <stack>

bool isBalanced(const std::string& expression) {

std::stack<char> stack;

for (char bracket : expression) {

if (bracket == '(' || bracket == '[' || bracket == '{') {

// Push opening brackets onto the stack

stack.push(bracket);

} else if (bracket == ')' || bracket == ']' || bracket == '}') {

// Check for matching closing brackets

if (stack.empty()) {

// If stack is empty, there is no matching opening bracket

return false;

}

char top = stack.top();

stack.pop();

// Check if the closing bracket matches the top of the stack

if ((bracket == ')' && top != '(') ||

(bracket == ']' && top != '[') ||

(bracket == '}' && top != '{')) {

return false;

}

}

}

// If the stack is empty, all brackets are balanced

return stack.empty();

}

int main() {

std::string expression;

std::cout << "Enter an expression with parentheses: ";

std::cin >> expression;

if (isBalanced(expression)) {

std::cout << "Parentheses are balanced." << std::endl;

} else {

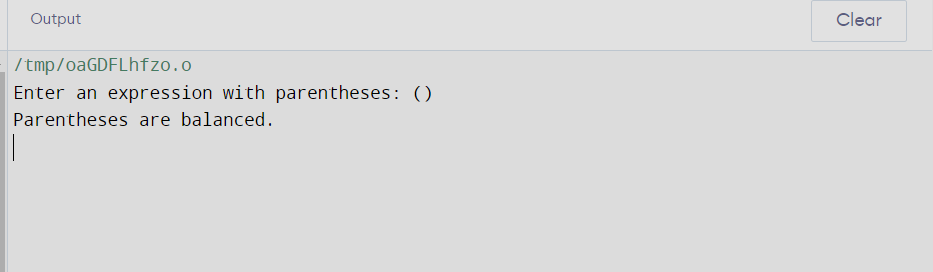
std::cout << "Parentheses are not balanced." << std::endl;

}

return 0;

}

Output:-



5)write a program to implementation of linear or ordered queue using array

Code:-

#include <iostream>

#define MAX\_SIZE 100

class Queue {

private:

int arr[MAX\_SIZE];

int front, rear;

public:

Queue() {

front = -1;

rear = -1;

}

bool isEmpty() {

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

}

bool isFull() {

return (rear == MAX\_SIZE - 1);

}

void enqueue(int data) {

if (isFull()) {

std::cout << "Queue Overflow: Cannot enqueue, queue is full." << std::endl;

return;

}

if (isEmpty()) {

// If the queue is empty, set both front and rear to 0

front = rear = 0;

} else {

// Increment rear for the next element

rear++;

}

arr[rear] = data;

std::cout << data << " enqueued into the queue." << std::endl;

}

void dequeue() {

if (isEmpty()) {

std::cout << "Queue Underflow: Cannot dequeue, queue is empty." << std::endl;

return;

}

std::cout << arr[front] << " dequeued from the queue." << std::endl;

if (front == rear) {

// If there was only one element, reset front and rear

front = rear = -1;

} else {

// Increment front for the next dequeue operation

front++;

}

}

void display() {

if (isEmpty()) {

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

return;

}

std::cout << "Queue elements: ";

for (int i = front; i <= rear; i++) {

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

}

std::cout << std::endl;

}

};

int main() {

Queue queue;

queue.enqueue(10);

queue.enqueue(20);

queue.enqueue(30);

queue.display();

queue.dequeue();

queue.display();

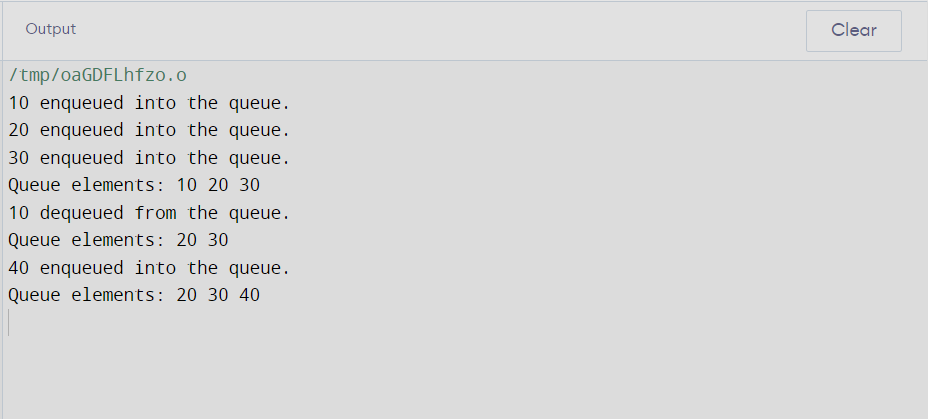
queue.enqueue(40);

queue.display();

return 0;

}

Output:-



6)write a program to implement a circular linked list

Code:-

#include <iostream>

class Node {

public:

int data;

Node\* next;

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

};

class CircularLinkedList {

private:

Node\* head;

public:

CircularLinkedList() : head(nullptr) {}

// Function to insert a node at the end of the circular linked list

void insert(int data) {

Node\* newNode = new Node(data);

if (!head) {

// If the list is empty, make the new node the head and point it to itself

head = newNode;

head->next = head;

} else {

// Find the last node and update its next pointer to the new node

Node\* last = head;

while (last->next != head) {

last = last->next;

}

last->next = newNode;

newNode->next = head;

}

std::cout << data << " inserted into the circular linked list." << std::endl;

}

// Function to display the circular linked list

void display() {

if (!head) {

std::cout << "Circular linked list is empty." << std::endl;

return;

}

Node\* current = head;

do {

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

current = current->next;

} while (current != head);

std::cout << std::endl;

}

};

int main() {

CircularLinkedList circularList;

circularList.insert(10);

circularList.insert(20);

circularList.insert(30);

std::cout << "Circular Linked List: ";

circularList.display();

return 0;

}

Output:-

