**1.Implement a custom dynamic array class that supports basic operations like insertion, deletion, resizing, and clearing.**

#include <iostream>

#include <stdexcept>

#include <algorithm>

template<typename T>

class DynamicArray {

private:

T\* data;

size\_t capacity;

size\_t size;

void resize(size\_t new\_capacity) {

T\* new\_data = new T[new\_capacity];

std::copy(data, data + size, new\_data);

delete[] data;

data = new\_data;

capacity = new\_capacity;

}

public:

DynamicArray()

: data(nullptr), capacity(0), size(0) {}

~DynamicArray() {

delete[] data;

}

void insert(const T& value) {

if (size == capacity) {

resize(capacity == 0 ? 1 : capacity \* 2);

}

data[size++] = value;

}

void remove(size\_t index) {

if (index >= size) {

throw std::out\_of\_range("Index out of range");

}

std::copy(data + index + 1, data + size, data + index);

size--;

}

void clear() {

delete[] data;

data = nullptr;

size = 0;

capacity = 0;

}

void setSize(size\_t new\_size) {

if (new\_size > capacity) {

resize(new\_size);

}

size = new\_size;

}

T& operator[](size\_t index) {

if (index >= size) {

throw std::out\_of\_range("Index out of range");

}

return data[index];

}

const T& operator[](size\_t index) const {

if (index >= size) {

throw std::out\_of\_range("Index out of range");

}

return data[index];

}

size\_t getSize() const {

return size;

}

size\_t getCapacity() const {

return capacity;

}

};

int main() {

DynamicArray<int> arr;

arr.insert(1);

arr.insert(2);

arr.insert(3);

std::cout << "Array size: " << arr.getSize() << std::endl;

for (size\_t i = 0; i < arr.getSize(); ++i) {

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

}

std::cout << std::endl;

arr.remove(1);

std::cout << "After removing element at index 1:" << std::endl;

for (size\_t i = 0; i < arr.getSize(); ++i) {

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

}

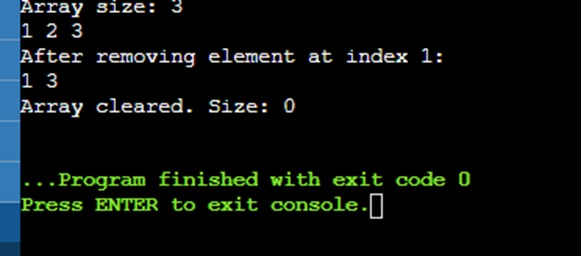
std::cout << std::endl;

arr.clear();

std::cout << "Array cleared. Size: " << arr.getSize() << std::endl;

return 0;

}



**2. Create a template-based stack class supporting push, pop, and peek operations. Implement it for different data types like int, float, and std::string.**

#include <iostream>

#include <stdexcept>

#include <string>

template<typename T>

class Stack {

private:

T\* data;

size\_t capacity;

size\_t top;

void resize(size\_t new\_capacity) {

T\* new\_data = new T[new\_capacity];

for (size\_t i = 0; i < top; ++i) {

new\_data[i] = data[i];

}

delete[] data;

data = new\_data;

capacity = new\_capacity;

}

public:

Stack()

: data(new T[1]), capacity(1), top(0) {}

~Stack() {

delete[] data;

}

void push(const T& value) {

if (top == capacity) {

resize(capacity \* 2);

}

data[top++] = value;

}

void pop() {

if (top == 0) {

throw std::out\_of\_range("Stack underflow");

}

top--;

}

T& peek() {

if (top == 0) {

throw std::out\_of\_range("Stack is empty");

}

return data[top - 1];

}

bool isEmpty() const {

return top == 0;

}

size\_t size() const {

return top;

}

};

int main() {

Stack<int> intStack;

intStack.push(1);

intStack.push(2);

intStack.push(3);

std::cout << "Integer Stack top element: " << intStack.peek() << std::endl;

intStack.pop();

std::cout << "Integer Stack top element after pop: " << intStack.peek() << std::endl;

Stack<float> floatStack;

floatStack.push(1.1f);

floatStack.push(2.2f);

floatStack.push(3.3f);

std::cout << "Float Stack top element: " << floatStack.peek() << std::endl;

floatStack.pop();

std::cout << "Float Stack top element after pop: " << floatStack.peek() << std::endl;

Stack<std::string> stringStack;

stringStack.push("Hello");

stringStack.push("World");

stringStack.push("!");

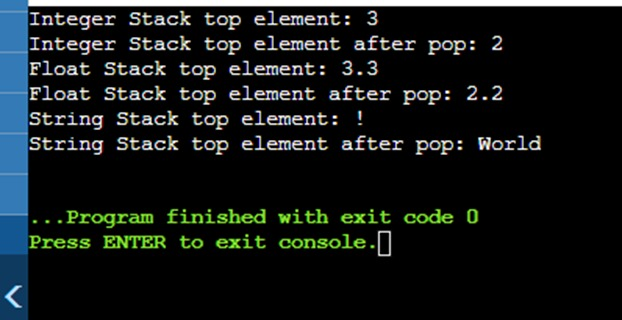
std::cout << "String Stack top element: " << stringStack.peek() << std::endl;

stringStack.pop();

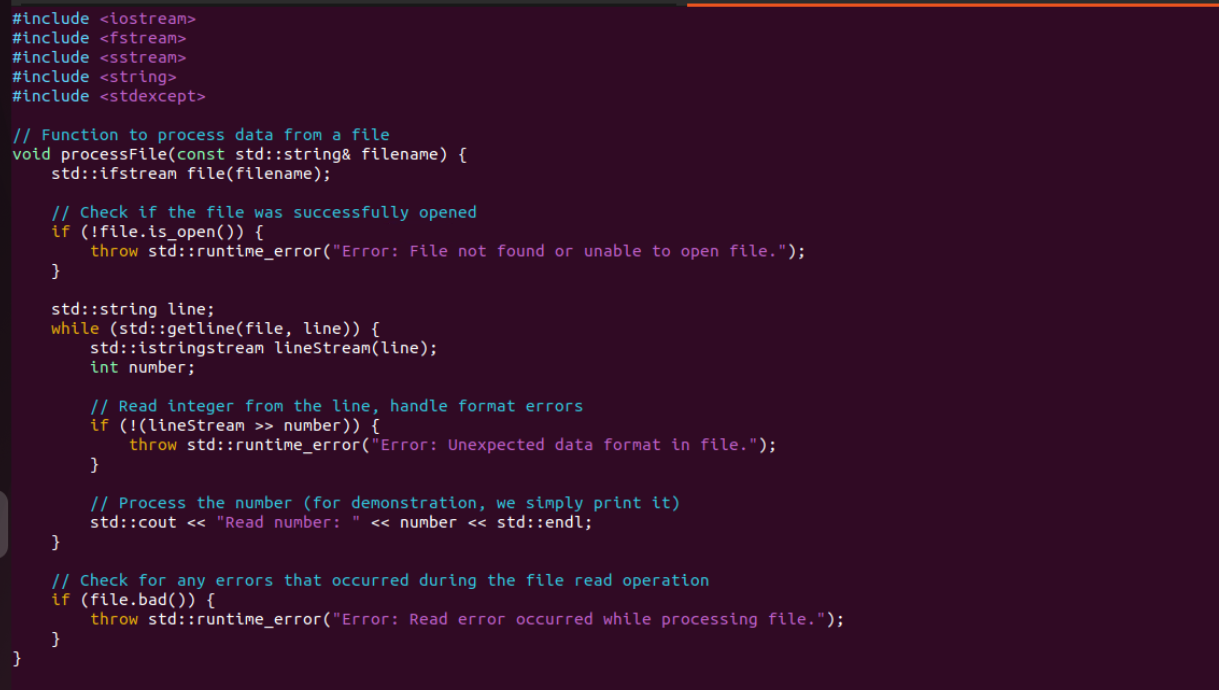
std::cout << "String Stack top element after pop: " << stringStack.peek() << std::endl;

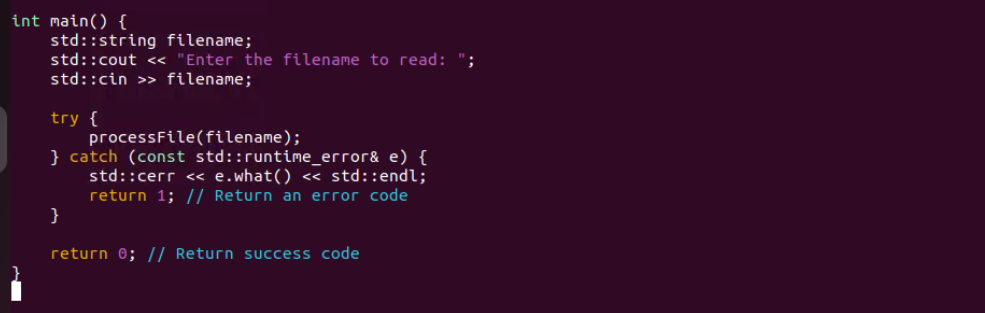
return 0;

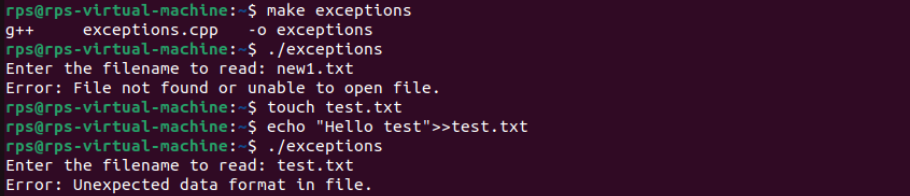
}



**3. Write a program that reads from a file and handles various exceptions such as file not found, read errors, and unexpected data formats.**

****

****

****