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

#include <iostream>

#include <stdexcept>

using namespace std;

class DynamicArray {

private:

int\* arr;

int capacity;

int size;

void resizeIfNeeded(int new\_size) {

if (new\_size > capacity) {

resize(capacity \* 2);

} else if (new\_size <= capacity / 4 && capacity > 4) {

resize(capacity / 2); }

}

void resize(int new\_capacity) {

int\* new\_arr = new int[new\_capacity];

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

new\_arr[i] = arr[i]; }

delete[] arr;

arr = new\_arr;

capacity = new\_capacity; }

public:

DynamicArray(int initial\_capacity = 4)

: capacity(initial\_capacity), size(0) {

if (initial\_capacity <= 0) {

throw invalid\_argument("Capacity must be greater than zero.");

}

arr = new int[capacity];

}

~DynamicArray() {

delete[] arr;

}

void insert(int value) {

resizeIfNeeded(size + 1);

arr[size++] = value;

}

void remove(int index) {

if (index < 0 || index >= size) {

throw out\_of\_range("Index out of bounds.");

}

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

arr[i] = arr[i + 1];

}

--size;

resizeIfNeeded(size);

}

int get(int index) const {

if (index < 0 || index >= size) {

throw out\_of\_range("Index out of bounds.");

}

return arr[index];

}

void clear() {

delete[] arr;

size = 0;

capacity = 4;

arr = new int[capacity];

}

int getSize() const {

return size;

}

int getCapacity() const {

return capacity;

}

bool isEmpty() const {

return size == 0;

}

void print() const {

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

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

}

cout << endl;

}

};

int main() {

DynamicArray da;

da.insert(5);

da.insert(15);

da.insert(25);

da.insert(35);

da.insert(45);

cout << "Array contents: ";

da.print();

da.remove(1);

cout << "After removing index 1: ";

da.print();

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

cout << "Array capacity: " << da.getCapacity() << endl;

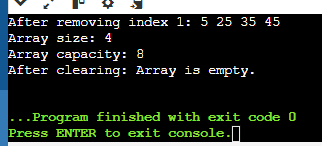
da.clear();

cout << "After clearing: " << (da.isEmpty() ? "Array is empty." : "Array is not empty.") << endl;

return 0;

}

OUTPUT:



**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>

using namespace std;

template <typename T>

class Stack {

private:

T\* arr;

int top;

int capacity;

void resize(int new\_capacity) {

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

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

new\_arr[i] = arr[i]; }

delete[] arr;

arr = new\_arr;

capacity = new\_capacity;

}

public:

Stack(int initial\_capacity = 4)

: top(0), capacity(initial\_capacity) {

if (initial\_capacity <= 0) {

throw invalid\_argument("Capacity must be greater than zero."); }

arr = new T[capacity];

}

~Stack() {

delete[] arr; }

void push(const T& value) {

if (top == capacity) {

resize(capacity \* 2); }

arr[top++] = value;

}

void pop() {

if (isEmpty()) {

throw out\_of\_range("Stack is empty."); }

--top;

}

T peek() const {

if (isEmpty()) {

throw out\_of\_range("Stack is empty."); }

return arr[top - 1];

}

bool isEmpty() const {

return top == 0; }

int size() const {

return top; }

void clear() {

top = 0; }

};

int main() {

Stack<int> intStack;

intStack.push(10);

intStack.push(20);

intStack.push(30);

cout << "Top element of int stack: " << intStack.peek() << endl;

intStack.pop();

cout << "Top element after pop: " << intStack.peek() << endl;

Stack<float> floatStack;

floatStack.push(1.5f);

floatStack.push(2.5f);

floatStack.push(3.5f);

cout << "Top element of float stack: " << floatStack.peek() << endl;

floatStack.pop();

cout << "Top element after pop: " << floatStack.peek() << endl;

Stack<string> stringStack;

stringStack.push("Hello");

stringStack.push("World");

cout << "Top element of string stack: " << stringStack.peek() << endl;

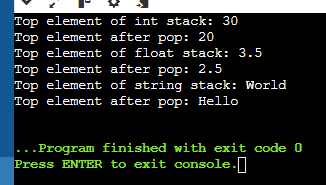
stringStack.pop();

cout << "Top element after pop: " << stringStack.peek() << endl;

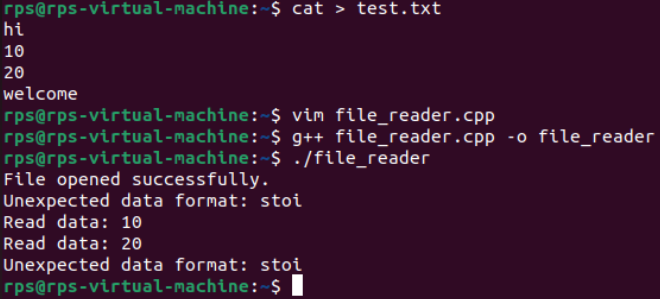
return 0;

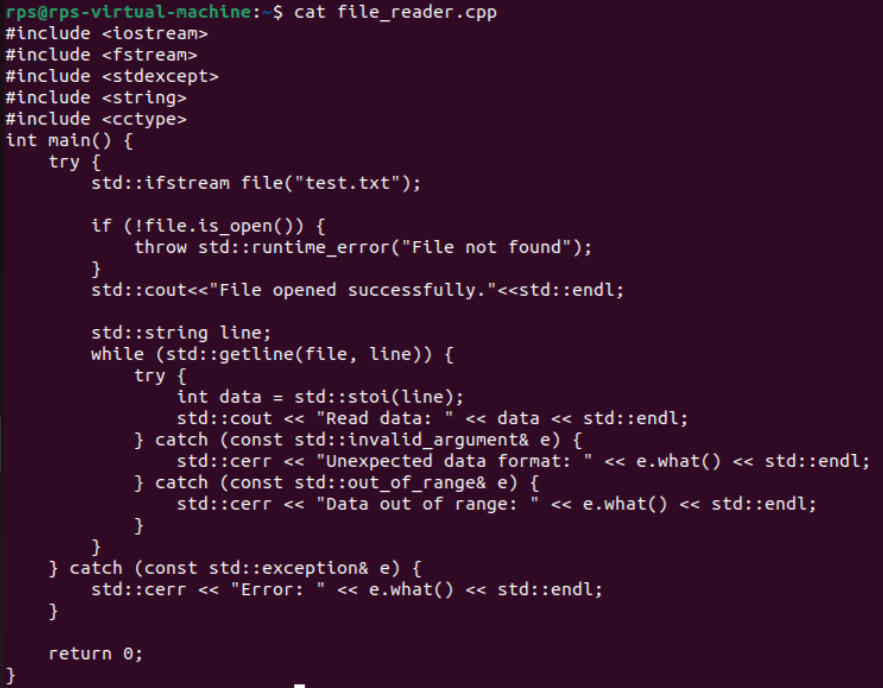
}

OUTPUT:



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

****

****

Write a unit test suite for the custom dynamic array class using a testing framework like Google Test or CppUnit.