**Create a class hierarchy (e.g., animals with different sounds) and manage object lifetimes and relationships using smart pointers. Include error handling to gracefully handle situations where resources might not be available.**

#include <iostream>

#include <memory>

#include <stdexcept>

#include <vector>

using namespace std;

class Animal {

public:

virtual ~Animal() = default;

virtual void makeSound() const = 0;

};

class Dog : public Animal {

public:

void makeSound() const override {

cout << "Woof!" << endl; }

};

class Cat : public Animal {

public:

void makeSound() const override {

cout << "Meow!" << endl; }

};

class Cow : public Animal {

public:

void makeSound() const override {

cout << "Moo!" << endl; }

};

int main() {

try {

vector<shared\_ptr<Animal>> animals = {

make\_shared<Dog>(),

make\_shared<Cat>(),

make\_shared<Cow>()

};

for (const auto& animal : animals) {

if (animal) {

animal->makeSound(); }

else {

throw runtime\_error("Failed to create an animal"); }

}

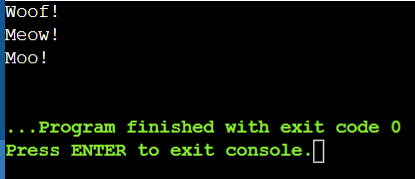
} catch (const exception& e) {

cerr << "Error: " << e.what() << endl; }

return 0;

}

**Output:**

****

**Simulate rolling dice, flipping coins, or generating random temperatures within a range. Users can choose the type of distribution and potentially customize parameters.**

#include <iostream>

#include <random>

#include <string>

#include <stdexcept>

#include <functional>

using namespace std;

template <typename T> // General function to generate random numbers

void generateRandomNumbers(int count, T distFunc, const string& description) {

random\_device rd;

mt19937 gen(rd()); //mt19937 is a MersenneTwister random number generator rd()generates speed

cout << description << " " << count << " times:" << endl;

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

cout << description.substr(0, description.size() - 1) << " " << i + 1 << ": " << distFunc(gen) << endl; }

}

void rollDice(int rolls) { // Function to roll dice

auto distFunc = uniform\_int\_distribution<>(1, 6);

generateRandomNumbers(rolls, distFunc, "Rolling dice");

}

void flipCoin(int flips) { // Function to flip coin

auto distFunc = uniform\_int\_distribution<>(0, 1);

generateRandomNumbers(flips, distFunc, "Flipping coin");

}

// Function to generate random temperatures

void generateRandomTemperatures(int count, double minTemp, double maxTemp) {

auto distFunc = uniform\_real\_distribution<>(minTemp, maxTemp);

generateRandomNumbers(count, distFunc, "Generating random temperatures between " + to\_string(minTemp) + " and " + to\_string(maxTemp) + " degrees");

}

int main() {

try {

int choice;

cout << "Choose an option:\n";

cout << "1. Roll dice\n";

cout << "2. Flip coin\n";

cout << "3. Generate random temperatures\n";

cout << "Enter your choice (1-3): ";

cin >> choice;

if (choice == 1) {

int rolls;

cout << "Enter the number of rolls: ";

cin >> rolls;

rollDice(rolls);

} else if (choice == 2) {

int flips;

cout << "Enter the number of flips: ";

cin >> flips;

flipCoin(flips);

} else if (choice == 3) {

int count;

double minTemp, maxTemp;

cout << "Enter the number of temperatures to generate: ";

cin >> count;

cout << "Enter the minimum temperature: ";

cin >> minTemp;

cout << "Enter the maximum temperature: ";

cin >> maxTemp;

generateRandomTemperatures(count, minTemp, maxTemp);

} else {

throw invalid\_argument("Invalid choice. Please enter a number between 1 and 3."); }

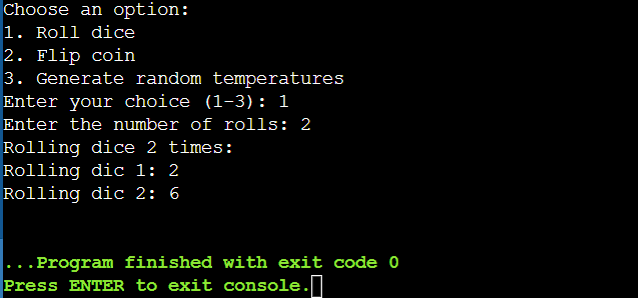
} catch (const exception& e) {

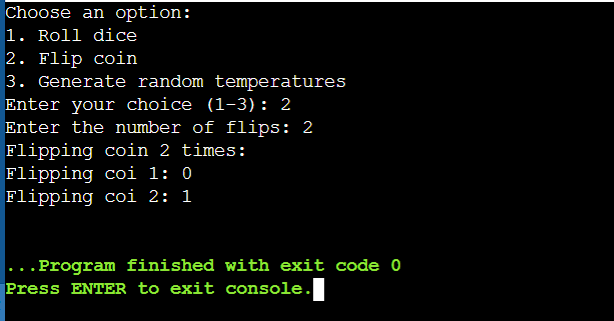
cerr << "Error: " << e.what() << endl; }

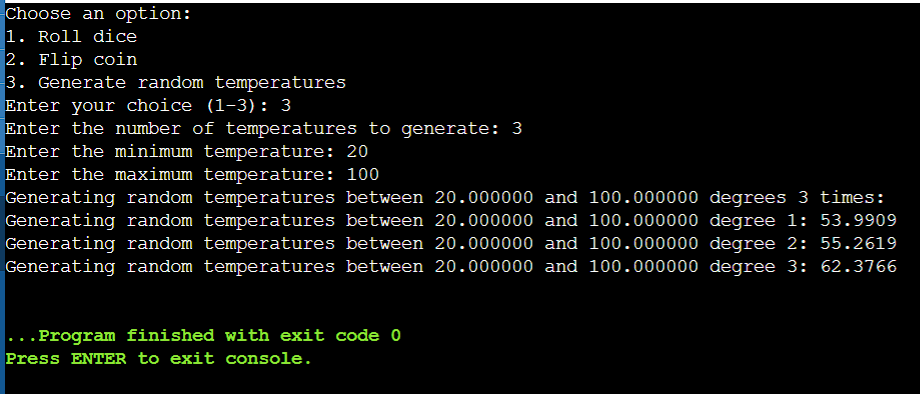
return 0;

}

**Output:**

****

****

****

**Project 4: File I/O with Regular Expressions (Enhanced with Error Handling and Performance)**

**Concept: Employ C++11 file I/O streams (ifstream, ofstream) to read from and write to files.**

**Enhancements:**

**Error Handling: Implement robust error handling to gracefully deal with file opening failures, I/O errors, or invalid data formats. Consider using exceptions or custom error codes for better diagnostics.**

**Regular Expressions: Utilize the <regex> library to search for patterns within text files, allowing for more complex data extraction or manipulation.**

**Example: Create a program that reads a log file, searches for specific error messages using regular expressions, and writes the matching lines to a new file, providing informative error messages if issues arise during file access or processing.**

#include <iostream>

#include <fstream>

#include <regex>

#include <stdexcept>

#include <string>

using namespace std;

bool processLogFile(const string& inputFile, const string& outputFile, const string& pattern) {

try {

// Simulating input file creation

ofstream tempInFile(inputFile);

if (!tempInFile) throw runtime\_error("Failed to create input file.");

tempInFile << "INFO: Application started.\n";

tempInFile << "ERROR: Failed to open database.\n";

tempInFile << "INFO: User logged in.\n";

tempInFile << "ERROR: Unable to fetch data.\n";

tempInFile << "INFO: Application terminated.\n";

tempInFile.close();

ifstream inFile(inputFile);

if (!inFile) throw runtime\_error("Failed to open input file.");

ofstream outFile(outputFile);

if (!outFile) throw runtime\_error("Failed to open output file.");

regex re(pattern);

string line;

while (getline(inFile, line)) {

if (regex\_search(line, re)) {

outFile << line << endl;

}

}

return true;

} catch (const exception& e) {

cerr << "Error: " << e.what() << endl;

return false;

}

}

int main() {

if (processLogFile("log.txt", "errors.txt", "ERROR")) {

cout << "File processed successfully." << endl;

} else {

cout << "File processing failed." << endl;

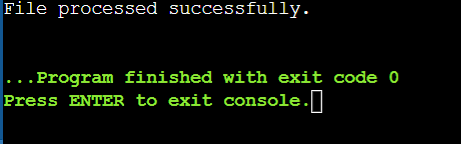
return 1;

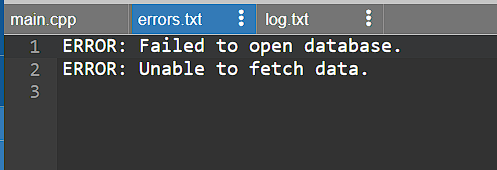
}

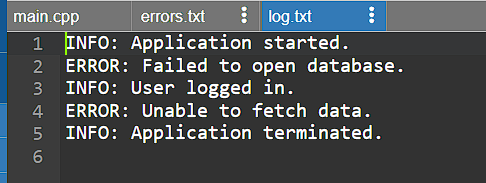
return 0;

}

**Output:**

****

****

****

**Project 5: Modern C++ Design Patterns (Using Move Semantics and Lambdas)**

**Concept: Explore modern C++ design patterns like move semantics (rvalue references) and lambdas to write efficient and expressive code.**

**Enhancements:**

**Move Semantics: Optimize code by understanding how to efficiently move resources (like large objects) to avoid unnecessary copies.**

**Lambdas: Utilize lambda expressions to create concise and readable anonymous functions, particularly for short-lived logic or event handling.**

**Example: Create a container class that efficiently stores and moves large objects like images or scientific data. Implement custom iterators or member functions using lambdas to process elements in the container.**

**These enhanced projects will significantly improve your proficiency in C++11 by:**

**Emphasizing robust error handling for real-world application reliability.**

**Leveraging regular expressions for powerful text manipulation.**

**Optimizing code with move semantics and lambdas.**

**Applying modern design patterns for well-structured and maintainable code.**

#include <iostream>

#include <vector>

#include <string>

#include <algorithm>

#include <functional>

#include <utility> // for std::move

using namespace std;

// Large object simulation

struct LargeImage {

string data;

LargeImage(const string& str) : data(str) {}

// Move constructor

LargeImage(LargeImage&& other) noexcept

: data(move(other.data)) {}

// Move assignment operator

LargeImage& operator=(LargeImage&& other) noexcept {

if (this != &other) {

data = move(other.data);

}

return \*this;

}

void display() const {

cout << "Image data: " << data << endl;

}

};

// Template class to store large objects

template<typename T>

class LargeObjectContainer {

public:

LargeObjectContainer() = default;

// Move constructor

LargeObjectContainer(LargeObjectContainer&& other) noexcept

: objects(move(other.objects)) {}

// Move assignment operator

LargeObjectContainer& operator=(LargeObjectContainer&& other) noexcept {

if (this != &other) {

objects = move(other.objects);

}

return \*this;

}

// Add object using move semantics

void addObject(T&& obj) {

objects.push\_back(move(obj));

}

// Process each object using a lambda

void processObjects(const function<void(T&)>& func) {

for (auto& obj : objects) {

func(obj);

}

}

// Custom iterators

auto begin() { return objects.begin(); }

auto end() { return objects.end(); }

private:

vector<T> objects;

};

int main() {

LargeObjectContainer<LargeImage> container;

// Adding large objects using move semantics

container.addObject(LargeImage("Dog"));

container.addObject(LargeImage("Cat"));

container.addObject(LargeImage("Lion"));

// Process each object with a lambda

container.processObjects([](LargeImage& img) {

img.display();

});

// Using custom iterators with lambdas

for\_each(container.begin(), container.end(), [](LargeImage& img) {

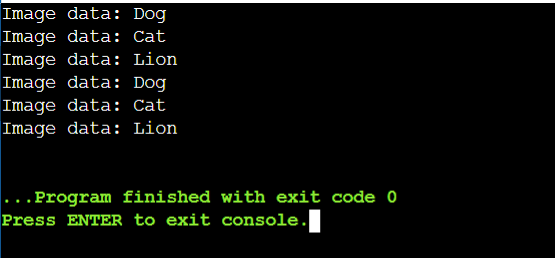
img.display();

});

return 0;

}

**Output:**

****

**Hash Map:**

#include <iostream>

#include <map>

#include <iterator>

using namespace std;

int main() {

// empty map container

map<int, int> gquiz1;

// insert elements in random order

gquiz1.insert(pair<int, int>(1, 40));

gquiz1.insert(pair<int, int>(2, 30));

gquiz1.insert(pair<int, int>(3, 60));

gquiz1.insert(pair<int, int>(4, 20));

gquiz1.insert(pair<int, int>(5, 50));

gquiz1.insert(pair<int, int>(6, 50));

gquiz1.insert(pair<int, int>(7, 10));

// printing map gquiz1

map<int, int>::iterator itr;

cout << "\nThe map gquiz1 is : \n";

cout << "\tKEY\tELEMENT\n";

for (itr = gquiz1.begin(); itr != gquiz1.end(); ++itr) {

cout << '\t' << itr->first << '\t' << itr->second << '\n';

}

cout << endl;

// assigning the elements from gquiz1 to gquiz2

map<int, int> gquiz2(gquiz1.begin(), gquiz1.end());

// print all elements of the map gquiz2

cout << "\nThe map gquiz2 after assignment from gquiz1 is : \n";

cout << "\tKEY\tELEMENT\n";

for (itr = gquiz2.begin(); itr != gquiz2.end(); ++itr) {

cout << '\t' << itr->first << '\t' << itr->second << '\n';

}

cout << endl;

// remove all elements up to element with key=3 in gquiz2

cout << "\ngquiz2 after removal of elements less than key=3 : \n";

cout << "\tKEY\tELEMENT\n";

gquiz2.erase(gquiz2.begin(), gquiz2.find(3));

for (itr = gquiz2.begin(); itr != gquiz2.end(); ++itr) {

cout << '\t' << itr->first << '\t' << itr->second << '\n';

}

// remove all elements with key = 4

int num;

num = gquiz2.erase(4);

cout << "\ngquiz2.erase(4) : ";

cout << num << " removed\n";

cout << "\tKEY\tELEMENT\n";

for (itr = gquiz2.begin(); itr != gquiz2.end(); ++itr) {

cout << '\t' << itr->first << '\t' << itr->second << '\n';

}

cout << endl;

// lower bound and upper bound for map gquiz1 key = 5

cout << "gquiz1.lower\_bound(5) : "

<< "\tKEY = ";

cout << gquiz1.lower\_bound(5)->first << '\t';

cout << "\tELEMENT = " << gquiz1.lower\_bound(5)->second << endl;

cout << "gquiz1.upper\_bound(5) : "

<< "\tKEY = ";

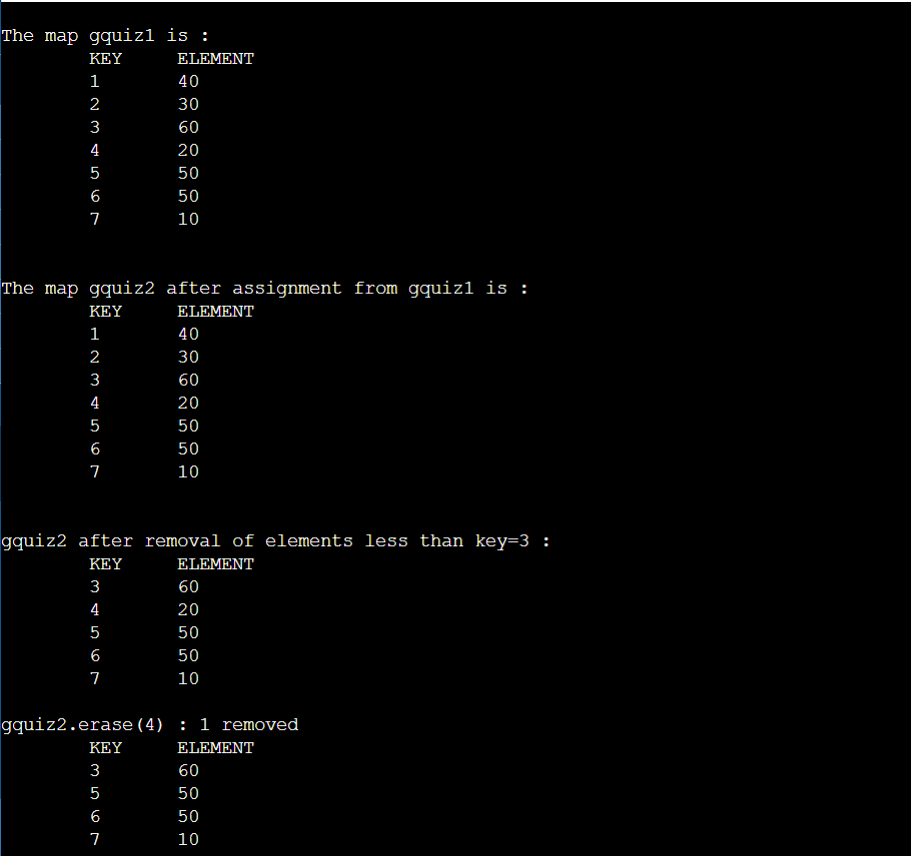
cout << gquiz1.upper\_bound(5)->first << '\t';

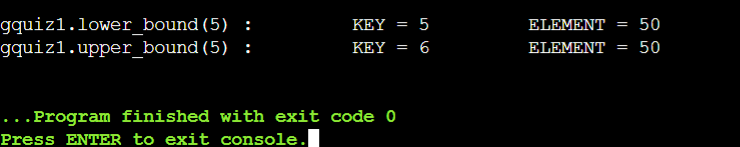
cout << "\tELEMENT = " << gquiz1.upper\_bound(5)->second << endl;

return 0;

}

**Output:**





**Develop a C++ program that allows users to enter and store contact details (name, phone number, email) in a map. The program should provide options for adding new contacts, searching for existing contacts, and displaying all stored contacts.**

#include <iostream>

#include <map>

#include <string>

using namespace std;

struct Contact {

string phone;

string email;

};

// Function to add a new contact

void addContact(map<string, Contact>& contacts) {

string name, phone, email;

cout << "Enter name: ";

cin.ignore();

getline(cin, name);

cout << "Enter phone number: ";

getline(cin, phone);

cout << "Enter email: ";

getline(cin, email);

contacts[name] = {phone, email};

cout << "Contact added successfully!\n";

}

// Function to search for a contact

void searchContact(const map<string, Contact>& contacts) {

string name;

cout << "Enter name to search: ";

cin.ignore();

getline(cin, name);

auto it = contacts.find(name);

if (it != contacts.end()) {

cout << "Name: " << it->first << "\n"

<< "Phone: " << it->second.phone << "\n"

<< "Email: " << it->second.email << "\n";

} else {

cout << "Contact not found.\n";

}

}

// Function to display all contacts

void displayContacts(const map<string, Contact>& contacts) {

if (contacts.empty()) {

cout << "No contacts found.\n";

return;

}

for (const auto& entry : contacts) {

cout << "Name: " << entry.first << "\n"

<< "Phone: " << entry.second.phone << "\n"

<< "Email: " << entry.second.email << "\n"

<< "--------------------------\n";

}

}

int main() {

map<string, Contact> contacts;

string choice;

while (true) {

cout << "1. Add Contact\n"

<< "2. Search Contact\n"

<< "3. Display All Contacts\n"

<< "4. Exit\n"

<< "Enter your choice: ";

cin >> choice;

if (choice == "1") {

addContact(contacts);

} else if (choice == "2") {

searchContact(contacts);

} else if (choice == "3") {

displayContacts(contacts);

} else if (choice == "4") {

cout << "Exiting...\n";

break;

} else {

cout << "Invalid choice. Please try again.\n";

}

}

return 0;

}

**Output:**

