Day-16:

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>

class Animal {

public:

virtual ~Animal() {}

virtual void makeSound() const = 0;

};

class Dog : public Animal {

public:

void makeSound() const override {

std::cout << "Woof!" << std::endl;

}

};

class Cat : public Animal {

public:

void makeSound() const override {

std::cout << "Meow!" << std::endl;

}

};

class Cow : public Animal {

public:

void makeSound() const override {

std::cout << "Moo!" << std::endl;

}

};

std::unique\_ptr<Animal> createAnimal(const std::string& type) {

if (type == "dog") {

return std::make\_unique<Dog>();

} else if (type == "cat") {

return std::make\_unique<Cat>();

} else if (type == "cow") {

return std::make\_unique<Cow>();

} else {

return nullptr;

}

}

int main() {

std::string animalType = "dog";

std::unique\_ptr<Animal> animal = createAnimal(animalType);

if (animal) {

animal->makeSound();

} else {

std::cerr << "Error: Unknown animal type \"" << animalType << "\"" << std::endl;

}

return 0;

}

Output:

Woof

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 <ctime>

// Function to simulate rolling a dice with given number of sides

int rollDice(int sides) {

std::random\_device rd;

std::mt19937 gen(rd());

std::uniform\_int\_distribution<> dis(1, sides);

return dis(gen);

}

// Function to simulate flipping a coin 'numCoins' times

void flipCoins(int numCoins) {

std::random\_device rd;

std::mt19937 gen(rd());

std::uniform\_int\_distribution<> dis(0, 1);

std::cout << "Coin flips: ";

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

int result = dis(gen);

if (result == 0) {

std::cout << "Tails ";

} else {

std::cout << "Heads ";

}

}

std::cout << std::endl;

}

// Function to generate a random temperature within a specified range

double generateRandomTemperature(double minTemp, double maxTemp) {

std::random\_device rd;

std::mt19937 gen(rd());

std::uniform\_real\_distribution<> dis(minTemp, maxTemp);

return dis(gen);

}

int main() {

int choice;

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

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

std::cout << "2. Flip coins\n";

std::cout << "3. Generate random temperature\n";

std::cin >> choice;

if (choice == 1) {

int sides;

std::cout << "Enter the number of sides on the dice: ";

std::cin >> sides;

if (sides <= 0) {

std::cerr << "Error: Number of sides must be positive." << std::endl;

return 1;

}

int result = rollDice(sides);

std::cout << "Result of rolling a " << sides << "-sided die: " << result << std::endl;

} else if (choice == 2) {

int numCoins;

std::cout << "Enter the number of coins to flip: ";

std::cin >> numCoins;

if (numCoins <= 0) {

std::cerr << "Error: Number of coins must be positive." << std::endl;

return 1;

}

flipCoins(numCoins);

} else if (choice == 3) {

double minTemp, maxTemp;

std::cout << "Enter the minimum and maximum temperatures (e.g., 0.0 100.0): ";

std::cin >> minTemp >> maxTemp;

if (minTemp >= maxTemp) {

std::cerr << "Error: Minimum temperature must be less than maximum temperature." << std::endl;

return 1;

}

double temp = generateRandomTemperature(minTemp, maxTemp);

std::cout << "Generated random temperature: " << temp << std::endl;

} else {

std::cerr << "Error: Invalid choice." << std::endl;

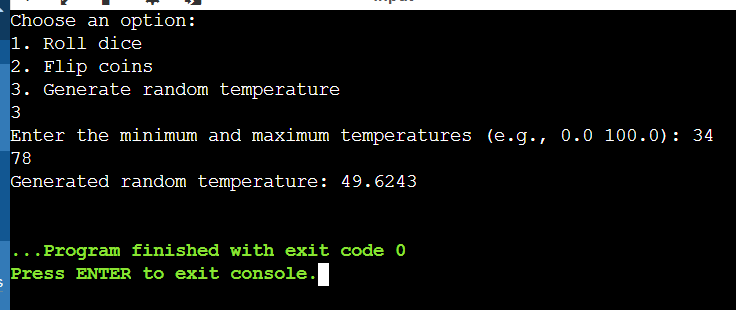
return 1;

}

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 <string>

// Function to read a log file, search for error messages using regex, and write matching lines to a new file

void processLogFile(const std::string& inputFilename, const std::string& outputFilename) {

try {

std::ifstream inputFile(inputFilename);

if (!inputFile.is\_open()) {

throw std::runtime\_error("Error: Failed to open input file.");

}

std::ofstream outputFile(outputFilename);

if (!outputFile.is\_open()) {

throw std::runtime\_error("Error: Failed to create or open output file.");

}

// Regular expression to search for error messages

std::regex errorRegex(R"(\b(error|exception)\b)", std::regex\_constants::icase);

std::string line;

while (std::getline(inputFile, line)) {

// Search for error messages in the current line

if (std::regex\_search(line, errorRegex)) {

outputFile << line << "\n";

}

}

std::cout << "Matching lines have been written to " << outputFilename << std::endl;

} catch (const std::exception& e) {

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

}

}

int main() {

std::string inputFilename = "logfile.txt"; // Replace with your actual log file path

std::string outputFilename = "errors.txt"; // Output file to store matching error lines

processLogFile(inputFilename, outputFilename);

return 0;

}

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.

#include <iostream>

#include <vector>

#include <memory>

#include <algorithm>

class LargeObject {

public:

explicit LargeObject(int id) : id(id) {

std::cout << "Creating LargeObject " << id << std::endl;

}

~LargeObject() {

std::cout << "Destroying LargeObject " << id << std::endl;

}

LargeObject(LargeObject&& other) noexcept : id(other.id) {

other.id = -1;

std::cout << "Moving LargeObject " << id << std::endl;

}

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

if (this != &other) {

id = other.id;

other.id = -1;

std::cout << "Moving LargeObject " << id << std::endl;

}

return \*this;

}

void process() const {

std::cout << "Processing LargeObject " << id << std::endl;

}

private:

int id;

};

class LargeObjectContainer {

public:

void addLargeObject(LargeObject&& obj) {

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

}

void processAllObjects() {

std::for\_each(objects.begin(), objects.end(), [](LargeObject& obj) {

obj.process();

});

}

private:

std::vector<LargeObject> objects;

};

int main() {

LargeObjectContainer container;

container.addLargeObject(LargeObject(1));

container.addLargeObject(LargeObject(2));

container.addLargeObject(LargeObject(3));

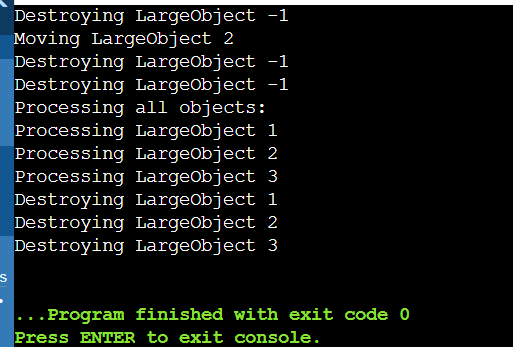
std::cout << "Processing all objects:\n";

container.processAllObjects();

return 0;

}

Output:



#include<iostream>

#include<iterator>

#include<map>

using namespace std;

int main()

{

map<int,int>gquiz1;

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));

map<int,int>::iterator itr;

cout<<"\n the map gquizl is:\n";

cout<<"\tkey\telement\n";

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

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

}

cout<<endl;

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

cout<<"\n the map gquiz2 after"<<"assign from 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;

cout<<"\n gquiz2 after removal of" "elements less than key-3:\n";

cout<<"\t key\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';

}

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;

cout<<"gquiz1.lower\_bound(s):"<<"\tkey -";

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

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

cout<<"gquiz1.upper\_bound(s):"<<"\tKEY -";

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

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

return 0;

}

Output:

the map gquizl is:

key element

1 40

2 30

3 60

4 20

5 50

6 50

7 10

the map gquiz2 afterassign from gquiz1 is:

KEY ELEMENT

1 40

2 30

3 60

4 20

5 50

6 50

7 10

gquiz2 after removal ofelements less than key-3:

key element

3 60

4 20

5 50

6 50

7 10

gquiz2.erase(4):1removed

key Element

3 60

5 50

6 50

7 10

gquiz1.lower\_bound(s): key -5 ELEMENT -50

gquiz1.upper\_bound(s): KEY -6 EMENT-50

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;

};

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

string name, phone, email;

cout << "Enter name: ";

getline(std::cin, name);

cout << "Enter phone number: ";

getline(std::cin, phone);

cout << "Enter email: ";

getline(cin, email);

contacts[name] = { phone, email };

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

}

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

string name;

cout << "Enter name to search: ";

getline(cin, name);

if (contacts.count(name) > 0) {

cout << "Name: " << name << "\n";

cout << "Phone: " << contacts.at(name).phone << "\n";

cout << "Email: " << contacts.at(name).email << "\n";

} else {

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

}

}

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

if (contacts.empty()) {

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

} else {

for (const pair<string, Contact>& entry : contacts) {

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

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

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

}

}

}

int main() {

map<string, Contact> contacts;

int choice;

while (true) {

cout << "1. Add New Contact\n";

cout << "2. Search Contact\n";

cout << "3. Display All Contacts\n";

cout << "4. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

cin.ignore();

switch (choice) {

case 1:

addContact(contacts);

break;

case 2:

searchContact(contacts);

break;

case 3:

displayContacts(contacts);

break;

case 4:

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

return 0;

default:

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

}

}

}