**DAY 16 C++**

**1.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.**

To create a class hierarchy in C++ for animals with different sounds and manage object lifetimes and relationships using smart pointers, I did some clarification below from the problem :

1. **Define the Base Class**: Create an abstract base class Animal with a virtual function makeSound() that will be overridden by derived classes.
2. **Define Derived Classes**: Create derived classes like Dog and Cat that override the makeSound() function.
3. **Use Smart Pointers**: Use std::shared\_ptr or std::unique\_ptr to manage the lifetimes of the Animal objects.
4. **Error Handling**: Implement error handling to manage situations where resources might not be available.

#include <iostream>

#include <memory>

#include <stdexcept>

class Animal {

public:

virtual void makeSound() const = 0; // Pure virtual function

virtual ~Animal() = default; // Virtual destructor

};

class Dog : public Animal

{

public:

void makeSound() const override {

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

}

};

// Derived class Cat

class Cat : public Animal {

public:

void makeSound() const override {

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

}

};

// Function to create animals and make sounds

void createAndMakeSound() {

try {

std::shared\_ptr<Animal> dog = std::make\_shared<Dog>();

std::shared\_ptr<Animal> cat = std::make\_shared<Cat>();

dog->makeSound();

cat->makeSound();

} catch (const std::bad\_alloc& e) {

std::cerr << "Memory allocation failed: " << e.what() << std::endl;

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

std::cerr << "An error occurred: " << e.what() << std::endl;

}

}

int main() {

createAndMakeSound();

return 0;

}

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

**Step 1: Initialize Random Seed**

We'll initialize the random seed using srand() and time(NULL).

**Step 2: Simulate Rolling Dice**

We'll simulate rolling a six-sided die using rand() to generate random numbers.

**Step 3: Simulate Flipping Coins**

We'll simulate flipping a coin using rand() to generate random numbers.

**Step 4: Simulate Generating Random Temperatures**

We'll simulate generating random temperatures within a specified range using rand().

#include <iostream>

#include <cstdlib>

#include <ctime>

// Function to initialize the random seed

void initializeRandomSeed() {

srand(static\_cast<unsigned int>(time(0)));

}

// Function to simulate rolling dice

void rollDice(int numRolls) {

std::cout << "Rolling " << numRolls << " dice:" << std::endl;

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

int roll = (rand() % 6) + 1; // Generate a number between 1 and 6

std::cout << roll << " ";

}

std::cout << std::endl;

}

// Function to simulate flipping coins

void flipCoins(int numFlips) {

std::cout << "Flipping " << numFlips << " coins:" << std::endl;

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

int flip = rand() % 2; // Generate 0 or 1

std::cout << (flip == 0 ? "Heads" : "Tails") << " ";

}

std::cout << std::endl;

}

// Function to simulate generating random temperatures

void generateTemperatures(int numTemps, double minTemp, double maxTemp) {

std::cout << "Generating " << numTemps << " random temperatures (Uniform Distribution):" << std::endl;

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

double temp = minTemp + static\_cast<double>(rand()) / RAND\_MAX \* (maxTemp - minTemp);

std::cout << temp << " ";

}

std::cout << std::endl;

}

int main() {

initializeRandomSeed();

int choice;

std::cout << "Choose an option:" << std::endl;

std::cout << "1. Roll Dice" << std::endl;

std::cout << "2. Flip Coins" << std::endl;

std::cout << "3. Generate Random Temperatures" << std::endl;

std::cin >> choice;

if (choice == 1) {

int numRolls;

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

std::cin >> numRolls;

rollDice(numRolls);

} else if (choice == 2) {

int numFlips;

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

std::cin >> numFlips;

flipCoins(numFlips);

} else if (choice == 3) {

int numTemps;

double minTemp, maxTemp;

std::cout << "Enter the number of temperatures: ";

std::cin >> numTemps;

std::cout << "Enter the minimum temperature: ";

std::cin >> minTemp;

std::cout << "Enter the maximum temperature: ";

std::cin >> maxTemp;

generateTemperatures(numTemps, minTemp, maxTemp);

} else {

std::cout << "Invalid choice." << std::endl;

}

return 0;

}

**3. 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.

1. **File I/O with Regular Expressions**: We need to use C++11 file I/O streams (ifstream, ofstream) to read from and write to files.
2. **Error Handling**: We need to implement robust error handling to gracefully deal with file opening failures, I/O errors, or invalid data formats. We'll use exceptions for better diagnostics.
3. **Regular Expressions**: We need to use the <regex> library to search for patterns within text files. This will allow us to extract or manipulate data based on patterns.
4. **Example Program**: The example program should:
   * Read a log file.
   * Search for specific error messages using regular expressions.
   * Write the matching lines to a new file.
   * Provide informative error messages if issues arise during file access or processing.

**Example Program Explanation**

We'll create a program that:

* Reads from a log file named input.log.
* Searches for lines containing the word "ERROR".
* Writes the matching lines to a new file named errors.log.
* Includes error handling for file operations and invalid data formats.

#include <iostream>

#include <fstream>

#include <regex>

#include <string>

// Function to read from a log file, search for specific patterns, and write to a new file

void processLogFile(const std::string& inputFile, const std::string& outputFile, const std::string& pattern) {

std::ifstream inFile;

std::ofstream outFile;

try {

// Open input file

inFile.open(inputFile);

if (!inFile) {

throw std::ios\_base::failure("Error opening input file: " + inputFile);

}

// Open output file

outFile.open(outputFile);

if (!outFile) {

throw std::ios\_base::failure("Error opening output file: " + outputFile);

}

// Regular expression for the pattern

std::regex re(pattern);

std::string line;

// Read from input file and write to output file if pattern matches

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

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

outFile << line << std::endl;

}

}

// Close the files

inFile.close();

outFile.close();

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

// Handle exceptions and provide informative error messages

std::cerr << "An error occurred: " << e.what() << std::endl;

if (inFile.is\_open()) {

inFile.close();

}

if (outFile.is\_open()) {

outFile.close();

}

}

}

int main() {

std::string inputFile = "input.log";

std::string outputFile = "errors.log";

std::string pattern = "ERROR"; // Regular expression pattern to search for

processLogFile(inputFile, outputFile, pattern);

return 0;

}

**4. 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.

CODE: HERE I will give u my explanation also :

 **Class LargeObject**:

* Represents a large object using std::vector<int> to hold data.
* Includes a default constructor, a constructor that takes data, a move constructor, and a move assignment operator.
* The move constructor and move assignment operator use std::move to efficiently transfer resources.

 **Class LargeObjectContainer**:

* Contains a std::vector<LargeObject> to store large objects.
* The addObject method adds a LargeObject to the container using move semantics (std::move).
* The processObjects method accepts a lambda (or any callable) to process each object in the container.
* The printObjects method prints the data of each LargeObject in the container.

 **Main Function**:

* Creates instances of LargeObject.
* Creates an instance of LargeObjectContainer.
* Adds LargeObject instances to the container using move semantics.
* Uses a lambda to process objects in the container (doubles each value in the data).
* Prints the processed objects.

#include <iostream>

#include <vector>

#include <algorithm>

#include <functional>

// A class representing a large object

class LargeObject {

public:

std::vector<int> data;

// Default constructor

LargeObject() = default;

// Constructor with data

LargeObject(const std::vector<int>& d) : data(d) {}

// Move constructor

LargeObject(LargeObject&& other) noexcept : data(std::move(other.data)) {

std::cout << "Move constructor called" << std::endl;

}

// Move assignment operator

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

if (this != &other) {

data = std::move(other.data);

std::cout << "Move assignment called" << std::endl;

}

return \*this;

}

};

// A container class for storing large objects

class LargeObjectContainer {

private:

std::vector<LargeObject> objects;

public:

// Add a LargeObject to the container

void addObject(LargeObject obj) {

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

}

// Process each object with a lambda

void processObjects(const std::function<void(LargeObject&)>& processor) {

for (auto& obj : objects) {

processor(obj);

}

}

// Print the data of each LargeObject

void printObjects() const {

for (const auto& obj : objects) {

for (const auto& value : obj.data) {

std::cout << value << " ";

}

std::cout << std::endl;

}

}

};

int main() {

// Create some large objects

LargeObject obj1({1, 2, 3, 4, 5});

LargeObject obj2({6, 7, 8, 9, 10});

// Create a container

LargeObjectContainer container;

// Add objects to the container using move semantics

container.addObject(std::move(obj1));

container.addObject(std::move(obj2));

// Process objects in the container using a lambda

container.processObjects([](LargeObject& obj) {

for (auto& value : obj.data) {

value \*= 2; // Double each value

}

});

// Print the processed objects

container.printObjects();

return 0;

}

HASH:

#include <iostream>

#include<iterator>

#include<map>

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<<"\n The 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"<<"assign 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<<"\n gquiz2 after removal of""elements less thsn 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";

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

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<<"\t";

cout<<"gquiz1.upperr\_bound(5):"<<"\tKEY=";

cout<<gquiz1.upper\_bound(5)->first<<"\t";

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

return 0;

}

**EXAMPLE: 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>

// Define a struct to store contact details

struct Contact {

std::string phoneNumber;

std::string email;

};

// Function to add a new contact

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

std::string name, phoneNumber, email;

std::cout << "Enter contact name: ";

std::getline(std::cin, name);

std::cout << "Enter phone number: ";

std::getline(std::cin, phoneNumber);

std::cout << "Enter email: ";

std::getline(std::cin, email);

contacts[name] = {phoneNumber, email};

std::cout << "Contact added successfully!" << std::endl;

}

// Function to search for a contact

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

std::string name;

std::cout << "Enter contact name to search: ";

std::getline(std::cin, name);

auto it = contacts.find(name);

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

std::cout << "Name: " << it->first << std::endl;

std::cout << "Phone Number: " << it->second.phoneNumber << std::endl;

std::cout << "Email: " << it->second.email << std::endl;

} else {

std::cout << "Contact not found!" << std::endl;

}

}

// Function to display all contacts

void displayAllContacts(const std::map<std::string, Contact>& contacts) {

if (contacts.empty()) {

std::cout << "No contacts available." << std::endl;

} else {

for (const auto& [name, contact] : contacts) {

std::cout << "Name: " << name << std::endl;

std::cout << "Phone Number: " << contact.phoneNumber << std::endl;

std::cout << "Email: " << contact.email << std::endl;

std::cout << "-----------------------------" << std::endl;

}

}

}

int main() {

std::map<std::string, Contact> contacts;

int choice;

while (true) {

std::cout << "1. Add Contact" << std::endl;

std::cout << "2. Search Contact" << std::endl;

std::cout << "3. Display All Contacts" << std::endl;

std::cout << "4. Exit" << std::endl;

std::cout << "Enter your choice: ";

std::cin >> choice;

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n'); // Ignore newline character

switch (choice) {

case 1:

addContact(contacts);

break;

case 2:

searchContact(contacts);

break;

case 3:

displayAllContacts(contacts);

break;

case 4:

return 0;

default:

std::cout << "Invalid choice. Please try again." << std::endl;

}

}

}