**CreatCreate 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.e 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>

class Animal {                          // Base class Animal

public:

    virtual ~Animal() = default;

    virtual void makeSound() const = 0;

};

class Dog : public Animal {                         // Derived class Dog

public:

    void makeSound() const override {

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

    }

};

class Cat : public Animal {             // Derived class Cat

public:

    void makeSound() const override {

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

    }

};

class Cow : public Animal {                         // Derived class Cow

public:
```

```
    void makeSound() const override {
        std::cout << "Moo!" << std::endl;
    }
};

std::shared_ptr<Animal> createAnimal(const std::string& type) {              // Function to create an animal based on type
    if (type == "Dog") {
        return std::make_shared<Dog>();
    } else if (type == "Cat") {
        return std::make_shared<Cat>();
    } else if (type == "Cow") {
        return std::make_shared<Cow>();
    } else {
        throw std::invalid_argument("Unknown animal type");
    }
}

int main() {                                                                 // Main function demonstrating the usage
    try {
        std::shared_ptr<Animal> dog = createAnimal("Dog");
        std::shared_ptr<Animal> cat = createAnimal("Cat");
        std::shared_ptr<Animal> cow = createAnimal("Cow");
        dog->makeSound();
        cat->makeSound();
        cow->makeSound();
    } catch (const std::exception& e) {
```

```
        std::cerr << "Error: " << e.what() << std::endl;
    }
    std::weak_ptr<Animal> weakDog;
    {
        std::shared_ptr<Animal> dog = createAnimal("Dog");
        weakDog = dog;
    }
    if (auto sharedDog = weakDog.lock()) {
        sharedDog->makeSound();
    } else {
        std::cout << "Dog has been destroyed" << std::endl;
    }
    return 0;
}
```

**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>
int rollDie(int sides) {                    // Function to simulate rolling a fair n-sided die
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_int_distribution<> dis(1, sides);
    return dis(gen);
}
```

```
int flipCoin() {                                    // Function to simulate flipping a fair coin (1 for heads, 0 for tails)
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_int_distribution<> dis(0, 1);
    return dis(gen);
}
double generateRandomTemperature(double minTemp, double maxTemp) {          // Function to generate a random temperature within a specified range
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_real_distribution<> dis(minTemp, maxTemp);
    return dis(gen);
}
int main() {
    srand(time(0));                                 // Seed for rand() function (optional)
    std::cout << "Simulating rolling a 6-sided die: " << rollDie(6) << std::endl;
    std::cout << "Simulating flipping a coin: " << (flipCoin() ? "Heads" : "Tails") << std::endl;
    double minTemp = 10.0, maxTemp = 15.0;
    std::cout << "Generating random temperature between " << minTemp << " and " << maxTemp << " degrees Celsius: ";
    std::cout << generateRandomTemperature(minTemp, maxTemp) << " C" << std::endl;
    return 0;
}
```

**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>

#include <stdexcept>

class FileIOException : public std::runtime_error {                    // Custom exception for file I/O errors

public:

    explicit FileIOException(const std::string& message)

        : std::runtime_error(message) {}

};

// Function to read from a file, search for error messages, and write to another file

void processLogFile(const std::string& inputFilePath, const std::string& outputFilePath, const std::string& pattern) {

    std::ifstream inputFile(inputFilePath);

    if (!inputFile) {

        throw FileIOException("Error opening input file: " + inputFilePath);

    }

    std::ofstream outputFile(outputFilePath);

    if (!outputFile) {
```

```
            throw FileIOException("Error opening output file: " + outputFilePath);
        }
        std::regex errorPattern(pattern);
        std::string line;
        while (std::getline(inputFile, line)) {
            if (std::regex_search(line, errorPattern)) {
                outputFile << line << std::endl;
            }
        }
        if (inputFile.bad()) {
            throw FileIOException("Error reading input file: " + inputFilePath);
        }
        if (outputFile.bad()) {
            throw FileIOException("Error writing to output file: " + outputFilePath);
        }
}
int main() {                                        // Main function to demonstrate the usage
        std::string inputFilePath;
        std::string outputFilePath;
        std::string pattern;
        std::cout << "Enter the input log file path: ";
        std::cin >> inputFilePath;
        std::cout << "Enter the output file path: ";
        std::cin >> outputFilePath;
        std::cout << "Enter the regex pattern to search for: ";
```

```
    std::cin.ignore();    // Clear the newline character from the input buffer
    std::getline(std::cin, pattern);
    try {
        processLogFile(inputFilePath, outputFilePath, pattern);
        std::cout << "Log file processed successfully. Matching lines written to " << outputFilePath << std::endl;
    } catch (const FileIOException& e) {
        std::cerr << "File I/O Error: " << e.what() << std::endl;
    } catch (const std::regex_error& e) {
        std::cerr << "Regex Error: " << e.what() << std::endl;
    } catch (const std::exception& e) {
        std::cerr << "Unexpected Error: " << e.what() << std::endl;
    }
    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.**

**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 <memory>
#include <algorithm>
#include <stdexcept>
class Image {                                   // Example large object class (Image)
public:
    Image(size_t size) : size_(size), data_(new int[size]) {
        std::cout << "Image of size " << size_ << " created." << std::endl;
    }
    ~Image() {
        delete[] data_;
        std::cout << "Image of size " << size_ << " destroyed." << std::endl;
    }
    Image(Image&& other) noexcept : size_(other.size_), data_(other.data_) {
// Move constructor
        other.size_ = 0;
        other.data_ = nullptr;
        std::cout << "Image moved." << std::endl;
    }
    Image& operator=(Image&& other) noexcept {                  // Move assignment
operator
```

```
        if (this != &other) {
            delete[] data_;
            size_ = other.size_;
            data_ = other.data_;
            other.size_ = 0;
            other.data_ = nullptr;
            std::cout << "Image moved (assignment)." << std::endl;
        }
        return *this;
    }
    Image(const Image&) = delete;                    // Deleted copy constructor and copy assignment operator
    Image& operator=(const Image&) = delete;
    size_t size() const { return size_; }
private:
    size_t size_;
    int* data_;
};
template<typename T>                    // Container class for large objects
class Container {
public:
    void add(T&& item) {
        items_.emplace_back(std::move(item));
    }
    template<typename Func>
    void forEach(Func func) {
```

```
        std::for_each(items_.begin(), items_.end(), func);
    }
private:
    std::vector<T> items_;
};
int main() {
    try {
        Container<Image> imageContainer;
        imageContainer.add(Image(1000));
        imageContainer.add(Image(2000));
        imageContainer.add(Image(3000));
        imageContainer.forEach([](const Image& img) {                                    //
Process images with a lambda function
            std::cout << "Processing image of size " << img.size() << std::endl;
        });
    } catch (const std::exception& e) {
        std::cerr << "Error: " << e.what() << std::endl;
    }
    return 0;
}
```

**MAP QUIZ :-**

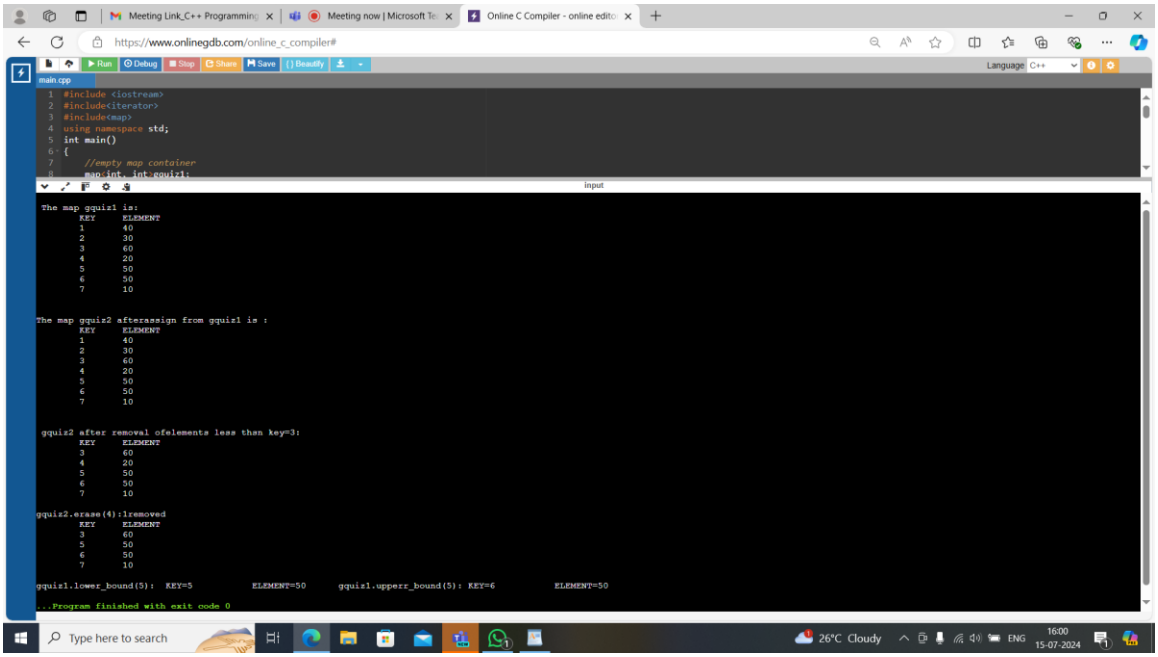
```
#include <iostream>
#include<iterator>
#include<map>
using namespace std;
```

```
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;
}
```

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>
struct Contact {                                    // Structure to hold contact details
    std::string phone_number;
```

```
    std::string email;
};
void addContact(std::map<std::string, Contact>& contacts) {                    // Function to add a new contact to the map
    std::string name, phone, email;
    std::cout << "Enter name: ";
    std::cin >> name;
    std::cout << "Enter phone number: ";
    std::cin >> phone;
    std::cout << "Enter email: ";
    std::cin >> email;
    Contact new_contact = {phone, email};
    contacts[name] = new_contact;
    std::cout << "Contact added successfully!\n";
}
void searchContact(const std::map<std::string, Contact>& contacts) {                    // Function to search for a contact by name
    std::string name;
    std::cout << "Enter name to search: ";
    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.phone_number << std::endl;
        std::cout << "Email: " << it->second.email << std::endl;
    } else {
        std::cout << "Contact not found.\n";
    }
```

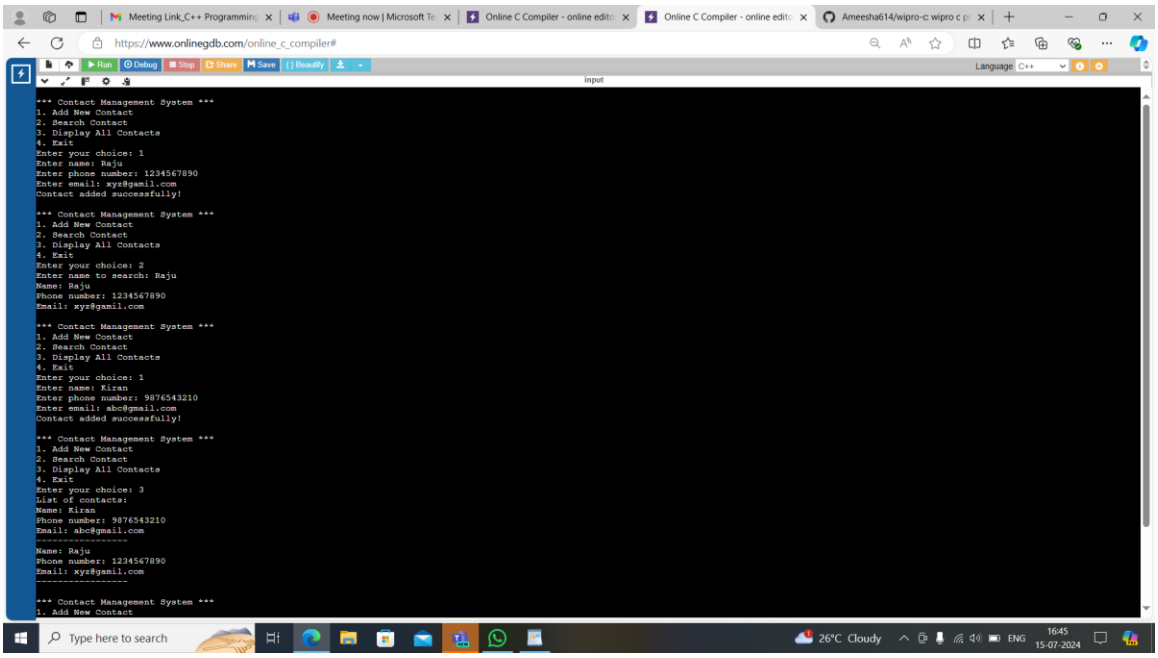
```
}

void displayContacts(const std::map<std::string, Contact>& contacts) {                    // Function to display all contacts
    if (contacts.empty()) {
        std::cout << "No contacts to display.\n";
    } else {
        std::cout << "List of contacts:\n";
        for (const auto& pair : contacts) {
            std::cout << "Name: " << pair.first << std::endl;
            std::cout << "Phone number: " << pair.second.phone_number << std::endl;
            std::cout << "Email: " << pair.second.email << std::endl;
            std::cout << "-----------------\n";
        }
    }
}

int main() {
    std::map<std::string, Contact> contacts;
    int choice;
    while (true) {
        std::cout << "\n*** Contact Management System ***\n";
// Display menu
        std::cout << "1. Add New Contact\n";
        std::cout << "2. Search Contact\n";
        std::cout << "3. Display All Contacts\n";
        std::cout << "4. Exit\n";
        std::cout << "Enter your choice: ";
        std::cin >> choice;
        switch (choice) {
```

```
                case 1:
                    addContact(contacts);
                    break;
                case 2:
                    searchContact(contacts);
                    break;
                case 3:
                    displayContacts(contacts);
                    break;
                case 4:
                    std::cout << "Exiting program...\n";
                    return 0;
                default:
                    std::cout << "Invalid choice. Please try again.\n";
            }
        }
        return 0;
}
```

OUTPUT :-


Meeting Link_C++ Programming
Meeting now | Microsoft Te
Online C Compiler - online editc
Online C Compiler - online editc
Ameesha614/wipro-c wipro c pr
https://www.onlinegdb.com/online_c_compiler#
Run
Debug
Stop
Share
Save
{ } Beautify
Language C++
Input
*** Contact Management System ***
1. Add New Contact
2. Search Contact
3. Display All Contacts
4. Exit
Enter your choice: 1
Enter name: Raju
Enter phone number: 1234567890
Enter email: xyz@gamil.com
Contact added successfully!
*** Contact Management System ***
1. Add New Contact
2. Search Contact
3. Display All Contacts
4. Exit
Enter your choice: 2
Enter name to search: Raju
Name: Raju
Phone number: 1234567890
Email: xyz@gamil.com
*** Contact Management System ***
1. Add New Contact
2. Search Contact
3. Display All Contacts
4. Exit
Enter your choice: 1
Enter name: Kiran
Enter phone number: 9876543210
Enter email: abc@gmail.com
Contact added successfully!
*** Contact Management System ***
1. Add New Contact
2. Search Contact
3. Display All Contacts
4. Exit
Enter your choice: 3
List of contacts:
Name: Kiran
Phone number: 9876543210
Email: abc@gmail.com
Name: Raju
Phone number: 1234567890
Email: xyz@gamil.com
*** Contact Management System ***
1. Add New Contact
Type here to search
26°C Cloudy
ENG
16:45
15-07-2024