DAY -16 Assignment

Project 1 :

Concepts on Class Hierarchy, Custom Parameters and Exception Handling .

1. **Quest. -** 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.

Logic -

a. We will create a class Animal as abstract class by initializing a pure-virtual function.
b. Then we will create '2' base classes name it as Lion and Cow using the member function of base class having its own definition.
c. Use override keyword to ensure so that the derived class function that matches the base class function signature.
d. In the main function implement the smart pointers inside the **try – catch** block.

Code –

```
#include <iostream>

using namespace std;

#include <memory>

#include <stdexcept>    // We use this header file while we're dealing with exception.


// Base class Animal

class Animal {

public:

   virtual void makeSound() const = 0;

   virtual ~Animal() = default;     //  Destructor

};

// Derived class Lion
```

```
class Lion : public Animal {
public:
      void makeSound() const override {
       cout << "Lion always Roars!" << endl;
   }
};
// Derived class Cow
class Cow : public Animal {
public:
       void makeSound() const override {
       cout << "Cow always Moo’s!" << endl;
   }
};

int main() {
   try {
       unique_ptr<Animal> lion(new Lion());
       unique_ptr<Animal> cow(new Cow());
       lion -> makeSound();
       cow -> makeSound();
   } catch (const exception& e) {
       cerr << "Error: " << e.what() << endl;  // We can also use ‘ cout ‘ in place of  ‘cerr’
   }
   return 0;
}
```

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

   Logic -

   a. We include the standard libraries for input-output operations, random number generation, and string manipulation.
   b. Define an enumeration for the distribution types: Uniform and Normal.
   c. Define a function to convert **Distribution** enum values to strings.
   d. Define an enumeration for the simulation types: Dice, Coin, and Temperature.
   e. Define a function to convert **SimulationType** enum values to strings.
   f. Start the main function.
   g. Initialize a random device and a **Mersenne Twister random number generator.**
   h. Prompt the user to choose a simulation type and read their input.
   i. Set the simulationType variable based on the user's input and handle invalid input.
   j. Prompt the user to choose a distribution type and read their input.
   k. Set the distribution variable based on the user's input and handle invalid input.
   l. Prompt the user to enter the number of trials and read their input.
   m. Display Simulation Info.
   n. Simulate Dice Rolls.
   o. Run the simulation for the specified number of trials, using the chosen distribution type.
   p. Simulate Coin Flip.
   q. Run the simulation for the specified number of trials, using the chosen distribution type.
   r. Simulate Temperature Measurements
   s. Define the distributions for temperature based on user input.
   t. Run the simulation for the specified number of trials, using the chosen distribution type.

Code -

```
#include <iostream>

#include <random>      //  Includes the random number generation library.

#include <string>


enum class Distribution {

 Uniform,

Normal

 };

string distributionToString (Distribution dist) {
```

```
    switch (dist) {
        case Distribution::Uniform:
            return "Uniform";
        case Distribution::Normal:
            return "Normal";
        default:
            return "Unknown";
    }
}
enum class SimulationType {
Dice, Coin, Temperature
 };
string simulationTypeToString (SimulationType type) {
    switch (type) {
        case SimulationType::Dice:
            return "Dice";
        case SimulationType::Coin:
            return "Coin";
        case SimulationType::Temperature:
            return "Temperature";
        default:
            return "Unknown";
    }
}
int main() {
```

```
random_device rd;      //   creates a random device used to seed the random number generator.
mt19937 mt(rd());      //  ( reference from net ) Initializes the Mersenne Twister random number
                                                  generator with   the seed.

SimulationType simulationType;
cout << " Choose a simulation type (0 for Dice, 1 for Coin, 2 for Temperature) : ";
int choice;
cin >> choice;
if (choice == 0) {
    simulationType = SimulationType::Dice;
} else if (choice == 1) {
    simulationType = SimulationType::Coin;
} else if (choice == 2) {
    simulationType = SimulationType::Temperature;
} else {
    cerr << "Invalid choice. Exiting." << std::endl;    //  We can also use ‘cout’ .
    return 1;
}

Distribution distribution;
cout << "Choose a distribution (0 for Uniform, 1 for Normal): ";
cin >> choice;
if (choice == 0) {
    distribution = Distribution::Uniform;
} else if (choice == 1) {
```

```
        distribution = Distribution::Normal;
    } else {
        cerr << "Invalid choice. Exiting." << endl;
        return 1;
    }

    int numTrials;
    cout << "Enter the number of trials: ";
    cin >> numTrials;

 cout << "Simulating " << simulationTypeToString(simulationType) << " with " <<
distributionToString(distribution) << " distribution:" << endl;

    if (simulationType == SimulationType::Dice) {
          uniform_int_distribution<int> uniformDist(1, 6);
         normal_distribution<double> normalDist(3.5, 1.5);    // Mean and standard deviation for a
                                                              //  fair  six-sided die

        for (int i = 0; i < numTrials; ++i) {
            if (distribution == Distribution::Uniform) {
                cout << uniformDist(mt) << endl;
            } else if (distribution == Distribution::Normal) {
                cout << normalDist(mt) << endl;
            }
        }
    } else if (simulationType == SimulationType::Coin) {
```

```
uniform_int_distribution<int> uniformDist(0, 1);
normal_distribution<double> normalDist(0.5, 0.5); // Mean and standard deviation for a fair coin


        for (int i = 0; i < numTrials; ++i) {
            if (distribution == Distribution::Uniform) {
                cout << (uniformDist(mt) == 0 ? "Heads" : "Tails") << endl;
            } else if (distribution == Distribution::Normal) {
                cout << (normalDist(mt) < 0.5 ? "Heads" : "Tails") << endl;
            }
        }
    } else if (simulationType == SimulationType::Temperature) {
        double minTemp, maxTemp;
        cout << "Enter the minimum temperature: ";
        cin >> minTemp;
            cout << "Enter the maximum temperature: ";
            cin >> maxTemp;
uniform_real_distribution<double> uniformDist(minTemp, maxTemp);
normal_distribution<double> normalDist((minTemp + maxTemp) / 2, (maxTemp - minTemp) /
3); // Mean and standard deviation for a normal temperature distribution


     for (int i = 0; i < numTrials; ++i) {
            if (distribution == Distribution::Uniform) {
                cout << uniformDist(mt) << endl;
            } else if (distribution == Distribution::Normal) {
                cout << normalDist(mt) << endl;  }
```

```
        }
    }
    return 0;
}
```

Project 2 :

Quest. - File I/O with Regular Expressions (Enhanced with Error Handling and Performance)

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

Enhancements:

a. 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.
b. 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

Logic :

a. **Initialization**: We will include necessary libraries. Define the searchLogFile function.
b. **File Processing**: We will Open input and output files. Check if files are open; if not, throw an error.
c. **Line-by-Line Processing**: we will read each line from the input file. Check if the line matches the regex pattern.Write matching lines to the output file.
d. **Completion**: we will then close input and output files.
e. **Main Function Execution**: Define the regex pattern to search for. Call the searchLogFile function with appropriate arguments. Handle any exceptions. End the program successfully.

Code :

```
#include <iostream>

#include <fstream>

using namespace std;

#include <regex> // It is used for working with regular expressions, which are patterns that describe
// sets of strings

#include <string>


void searchLogFile (const string& inputFile, const string& outputFile, const regex& pattern) {
    ifstream inFile(inputFile);
    ofstream outFile(outputFile);


    if (!inFile.is_open() || !outFile.is_open()) {
        throw runtime_error("Failed to open input or output file.");
    }
    string line;
    while (getline(inFile, line)) {
        if (regex_search(line, pattern)) {
            outFile << line << endl;
        }
    }
    inFile.close();
    outFile.close();
}
int main() {
```

```
    try {
        regex errorPattern("ERROR");
        searchLogFile ("log.txt", "errors.txt", errorPattern);
    } catch (const exception& e) {
      cerr << "Error: " << e.what() << endl;
    }
    return 0;
}
```

Project 3 :

Quest. - Modern C++ Design Patterns (Using Move Semantics and Lambdas)

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

Enhancements:

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

Code –

```
#include <iostream>
#include <vector>        // We use it for dynamic array.
#include <algorithm>     // We use it for algorithm.
#include <memory>        // We use it for smart pointers.
using namespace std;
```

```
template <typename T>
class LargeObjectContainer {
public:
    // Default constructor
    LargeObjectContainer() = default;
    // Move constructor
    LargeObjectContainer(LargeObjectContainer&& other) noexcept {
        // Move elements from other to this
        data_ = move(other.data_);
    }

    // Move assignment operator
    LargeObjectContainer& operator=(LargeObjectContainer&& other) noexcept {
        if (this != &other) {
            data_ = move(other.data_);
        }
        return *this;
    }

    // Add an element to the container
    void add(T&& obj) {
        data_.push_back(move(obj));
    }
```

```
    // Member function using a lambda for element processing
    template <typename F>
    void for_each(F&& process) {
        for (auto& obj : data_) {
            process(obj);
        }
    }
    // Custom iterator for read-only access
    class LargeObjectIterator {
    public:
        using iterator_category = forward_iterator_tag;
        using value_type = T;
        using reference = const T&;  // Use const reference for read-only iteration
        using pointer = const T*;    // Use const pointer for read-only iteration

        explicit LargeObjectIterator(const LargeObjectContainer<T>& container, bool isEnd =
false) :
       container_(container), current_(isEnd ? container_.data_.end() : container_.data_.begin()) {}
        LargeObjectIterator& operator++() {
            ++current_;
            return *this;
        }

        LargeObjectIterator operator++(int) {
            LargeObjectIterator temp = *this;
```

```
            ++(*this);
            return temp;
        }
        // Read-only access through const reference and pointer
        reference operator*() const {
return *current_;
 }
        pointer operator->() const {
return &*current_;
 }
        bool operator!=(const LargeObjectIterator& other) const {
            return current_ != other.current_;
        }

    private:
        const LargeObjectContainer<T>& container_; // Use const reference for read-only access
        typename vector<T>::const_iterator current_;
    };

    // Begin and end iterators for custom read-only iteration
    LargeObjectIterator begin() const {
            return LargeObjectIterator(*this);
 }
    LargeObjectIterator end() const {
            return LargeObjectIterator(*this, true);
```

```
}
private:
    vector<T> data_;
};
// Example usage with a simple large object
class LargeObject {
public:
    LargeObject(int id) : id_(id) {
        cout << "LargeObject " << id_ << " created." << endl;
    }

    LargeObject(LargeObject&& other) noexcept : id_(other.id_) {
        other.id_ = -1; // Indicate the object has been moved
        cout << "LargeObject " << id_ << " moved." << endl;
    }

    LargeObject& operator=(LargeObject&& other) noexcept {
        if (this != &other) {
            id_ = other.id_;
            other.id_ = -1; // Indicate the object has been moved
            cout << "LargeObject " << id_ << " move-assigned." << endl;
        }
        return *this;
    }
    int getId() const { return id_; }
```

```
    private:
    int id_;
};
int main() {
    LargeObjectContainer<LargeObject> container;
    // Add large objects to the container
    container.add(LargeObject(1));
    container.add(LargeObject(2));
    container.add(LargeObject(3));
    // Process elements using a lambda
    container.for_each([](LargeObject& obj) {
        cout << "Processing LargeObject " << obj.getId() << endl;
    });
    // Iterate through elements using custom iterator
    for (const auto& obj : container) {
        cout << "Iterating LargeObject " << obj.getId() << endl;
    }
    // Move the container to a new container
    LargeObjectContainer<LargeObject> newContainer = move(container);
    // Iterate through elements in the new container
    for (const auto& obj : newContainer) {
       cout << "Iterating LargeObject in new container " << obj.getId() << endl;
    }
   return 0;
}
```

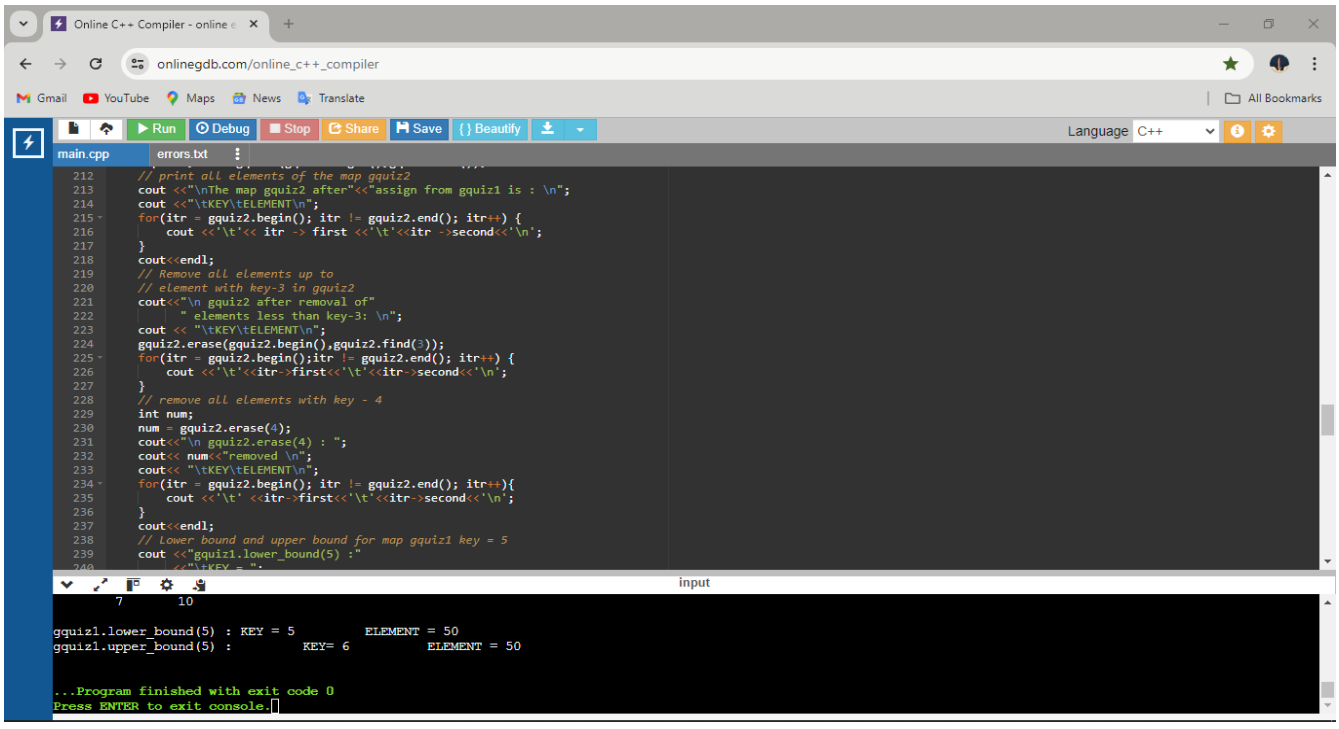
// Code 1 : Task Implement program in C++ on Map.

// Map concept

```
#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 <<"\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"<<"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 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<<"\n gquiz2.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;
}
```



Assignment : 2

Task : 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.

/* The **std::map** container is used to efficiently store and retrieve contacts based on their names, and the use of **std::string** ensures that the program can handle contact details with spaces. */

```
#include <iostream>

#include <map>

#include <string>

// 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, phone, email;

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

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

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

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

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

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


    contacts[name] = {phone, email};
```

```
    std::cout << "Contact added successfully!\n";
}
// Function to search for a contact by name
void searchContact(const std::map<std::string, Contact>& contacts) {
    std::string name;
    std::cout << "Enter name to search: ";
    std::getline(std::cin, name);

    auto it = contacts.find(name);
    if (it != contacts.end()) {
        std::cout << "Name: " << it->first << "\n"
                  << "Phone Number: " << it->second.phoneNumber << "\n"
                  << "Email: " << it->second.email << "\n";
    } else {
        std::cout << "Contact not found.\n";
    }
}
// Function to display all contacts
void displayContacts(const std::map<std::string, Contact>& contacts) {
    if (contacts.empty()) {
        std::cout << "No contacts available.\n";
        return;
    }
    for (const auto& pair : contacts) {
        std::cout << "Name: " << pair.first << "\n"
```

```
            << "Phone Number: " << pair.second.phoneNumber << "\n"
            << "Email: " << pair.second.email << "\n\n";
    }
}
// Main menu
void menu() {
    std::map<std::string, Contact> contacts;
    int choice;
    do {
        std::cout << "\nContact Management System\n";
        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;
        std::cin.ignore(); // Ignore newline character left in the input buffer
        switch (choice) {
            case 1:
                addContact(contacts);
                break;
            case 2:
                searchContact(contacts);
                break;
            case 3:
```

```
            displayContacts(contacts);
            break;
        case 4:
            std::cout << "Exiting the program.\n";
            break;
        default:
            std::cout << "Invalid choice. Please try again.\n";
        }
    } while (choice != 4);
}
int main() {
    menu();
    return 0;
}
```