OOP asg#4

Q16. Research the STL and write a program that uses `vector`, `map`, and `set` containers. Explain the functionality of each container and its applications in programming.

Explanation:

1. **Vector**: A dynamic array that resizes automatically when elements are added or removed. Commonly used when the size of the array is unknown.
2. **Map**: A collection of key-value pairs, where keys are unique, and values can be accessed using keys. Useful for fast lookups.
3. **Set**: A container that stores unique elements in sorted order. Useful for ensuring no duplicate values.

Program:

```
#include <iostream>
#include <vector>
#include <map>
#include <set>
using namespace std;

int main() {
    // Vector Example
    vector<int> vec = {1, 2, 3, 4, 5};
    cout << "Vector Elements: ";
    for (int v : vec) cout << v << " ";
    cout << endl;

    // Map Example
    map<string, int> studentMarks = {{"Alice", 85}, {"Bob", 90}, {"Charlie",
78}};
    cout << "Map (Student Marks):" << endl;
    for (auto &pair : studentMarks)
        cout << pair.first << " -> " << pair.second << endl;

    // Set Example
    set<int> uniqueNums = {5, 3, 8, 3, 5}; // Duplicates will be removed
    cout << "Set Elements (Unique and Sorted): ";
    for (int num : uniqueNums) cout << num << " ";
    cout << endl;

    return 0;
}
```

Q17. Design a `Student` class and implement serialization using file streams to save and load student data. Discuss the importance of object serialization in data persistence and transfer.

Explanation:

- **Serialization**: The process of converting an object's state into a format that can be saved (e.g., file) or transferred (e.g., over a network).
- **Importance**: Ensures data persistence and easy transfer of complex data structures between systems.

Program:

```
#include <iostream>
#include <fstream>
using namespace std;

class Student {
    string name;
    int age;
public:
    Student() : name(""), age(0) {}
    Student(string n, int a) : name(n), age(a) {}

    void saveToFile(const string &filename) {
        ofstream file(filename);
        if (file) file << name << "\n" << age;
        file.close();
    }

    void loadFromFile(const string &filename) {
        ifstream file(filename);
        if (file) file >> name >> age;
        file.close();
    }

    void display() { cout << "Name: " << name << ", Age: " << age << endl; }
};

int main() {
    Student s1("Alice", 20);
    s1.saveToFile("student.txt");

    Student s2;
    s2.loadFromFile("student.txt");
    s2.display();
    return 0;
}
```

Q18. Write a program that saves object data both in binary and text format. Compare the two approaches in terms of readability, file size, and data integrity. When would each approach be more appropriate?

Explanation:

- **Text Format**: Human-readable but less compact.

- **Binary Format**: Compact, faster for large data, but not human-readable.
- **Use Cases**:
    - Text format for debugging or sharing files.
    - Binary format for performance-critical applications.

Program:

```
#include <iostream>
#include <fstream>
using namespace std;

class Student {
    string name;
    int age;
public:
    Student() : name(""), age(0) {}
    Student(string n, int a) : name(n), age(a) {}

    void saveAsText(const string &filename) {
        ofstream file(filename);
        if (file) file << name << "\n" << age;
        file.close();
    }

    void saveAsBinary(const string &filename) {
        ofstream file(filename, ios::binary);
        if (file) file.write((char *)this, sizeof(*this));
        file.close();
    }

    void loadFromBinary(const string &filename) {
        ifstream file(filename, ios::binary);
        if (file) file.read((char *)this, sizeof(*this));
        file.close();
    }

    void display() { cout << "Name: " << name << ", Age: " << age << endl; }
};

int main() {
    Student s1("Bob", 22);

    // Save as text
    s1.saveAsText("student.txt");

    // Save as binary
    s1.saveAsBinary("student.bin");

    // Load from binary
    Student s2;
    s2.loadFromBinary("student.bin");
    s2.display();
    return 0;
}
```

Q19. Design a program that opens a file, reads data, and then processes it. Implement exception handling to manage potential file I/O errors and resource cleanup. Explain how exception handling ensures safe program execution.

Explanation:

- Exception handling ensures safe program execution by catching errors and preventing crashes.
- **Common File I/O Errors**: Missing files, read/write failures.
- **Resource Cleanup**: Ensures resources like file handles are released properly, even in case of errors.

Program:

```
#include <iostream>
#include <fstream>
#include <stdexcept>
using namespace std;

void processFile(const string &filename) {
    ifstream file(filename);
    if (!file) throw runtime_error("File not found!");

    string data;
    while (getline(file, data)) {
        cout << data << endl;
    }
    file.close();
}

int main() {
    try {
        processFile("example.txt");
    } catch (const runtime_error &e) {
        cout << "Error: " << e.what() << endl;
    }
    return 0;
}
```

Q20. Create a custom exception class `InvalidAgeException` to handle errors when a user enters an invalid age for registration in a system. Demonstrate its use in a program and discuss how custom exceptions improve code clarity and error handling.

Explanation:

- **Custom Exceptions** improve error handling by defining specific error conditions.
- Example: `InvalidAgeException` handles cases where an invalid age is entered.

Program:

```cpp
#include <iostream>
#include <stdexcept>
using namespace std;

class InvalidAgeException : public runtime_error {
public:
    InvalidAgeException() : runtime_error("Invalid age entered!") {}
};

void registerUser(int age) {
    if (age <= 0 || age > 120) throw InvalidAgeException();
    cout << "User registered successfully. Age: " << age << endl;
}

int main() {
    try {
        int age;
        cout << "Enter age for registration: ";
        cin >> age;
        registerUser(age);
    } catch (const InvalidAgeException &e) {
        cout << "Error: " << e.what() << endl;
    }
    return 0;
}
```