Object Oriented Programming

Assignment # 04

Muhammad Shakaib Arsalan

Student ID: F2022266626

Course Code: CC1022L

Section: V2

Resource Person: Rehan Raza

School of Systems and Technology

UMT Lahore Pakistan

Question 01

Write a C++ program that demonstrates the use of templates. Follow the instructions below:

a) **Create a template function called maximum() that takes two parameters of the same type and returns the maximum value between them.**

```cpp
template <typename T>
T maximum(T a, T b)
{
    return (a > b) ? a : b;
}
```

b) **In the main() function, demonstrate the use of the maximum() function with different data types (e.g., integers, floating-point numbers, strings).**

```cpp
int main()
{
    cout << "     ======== Find Maximum by using Function ========\n\n";
    cout << " -> By comparing two integers \"" << maximum<int>(2, 3) << "\" is maximum." << endl;
    cout << " -> By comparing two strings \"" << maximum<string>("Eman", "Murtaza") << "\" is maximum." << endl;
    cout << " -> By comparing two Floating Point Numbers \"" << maximum<float>(2.2, 1.1) << "\" is maximum." << endl << endl;

    return 0;
}
```

c) **Provide a brief explanation of how templates allow for code reusability and flexibility in handling multiple data types.**

1. Code Reusability: Templates let you write generic code that works with different data types, reducing repetition and improving code organization.

2. Flexibility: Templates allow you to handle various data types effortlessly, making your code adaptable to different scenarios without the need for separate implementations.

3. Improved Performance: Templates generate optimized code for each data type at compile-time, eliminating the need for runtime type conversions and leading to faster execution.

d) Extend the program by creating a template class called Pair that represents a pair of values. The class should have two private data members of the same type and provide a member function called getMax() that returns the maximum value between the two data members.

```cpp
template <typename T>
class Pair
{
    T a;
    T b;

public:
    Pair() {}
    Pair(T a, T b)
    {
        this->a = a;
        this->b = b;
    }
    T getMax()
    {
        return maximum(a, b);
    }
};
```

e) In the main() function, create instances of the Pair class with different data types (e.g., integers, floating-point numbers, strings) and demonstrate the use of the getMax() function.

```cpp
int main()
{
    cout << "       ======== Find Maximum by using Class ========\n\n";
    Pair<int> p1(2, 4);
    cout << " -> int:- Maximum value: " << p1.getMax() << endl;

    Pair<float> p2(7.4, 6.6);
    cout << " -> float:- Maximum value: " << p2.getMax() << endl;

    Pair<string> p3("hello", "world");
    cout << " -> string:- Maximum value: " << p3.getMax() << endl
         << endl;

    return 0;
}
```

Sample Code:

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

template <typename T>
T maximum(T a, T b)
{
    return (a > b) ? a : b;
}

template <typename T>
class Pair
{
    T a;
    T b;

public:
    Pair() {}
    Pair(T a, T b)
    {
        this->a = a;
        this->b = b;
    }
    T getMax()
    {
        return maximum(a, b);
    }
};

int main()
{
    cout << "     ======== Find Maximum by using Function ========\n\n";
    cout << " -> By comparing two integers \"" << maximum<int>(2, 3) << "\" is
maximum." << endl;
    cout << " -> By comparing two strings \"" << maximum<string>("Eman",
"Murtaza") << "\" is maximum." << endl;
    cout << " -> By comparing two Floating Point Numbers \"" <<
maximum<float>(2.2, 1.1) << "\" is maximum." << endl
         << endl;

    cout << "       ======== Find Maximum by using Class ========\n\n";
    Pair<int> p1(2, 4);
    cout << " -> int:- Maximum value: " << p1.getMax() << endl;

    Pair<float> p2(7.4, 6.6);
    cout << " -> float:- Maximum value: " << p2.getMax() << endl;
```

```
    Pair<string> p3("hello", "world");
    cout << " -> string:- Maximum value: " << p3.getMax() << endl
         << endl;

    return 0;
}
```

OutPut:

```
     ======== Find Maximum by using Function ========

-> By comparing two integers "3" is maximum.
-> By comparing two strings "Murtaza" is maximum.
-> By comparing two Floating Point Numbers "2.2" is maximum.

      ======== Find Maximum by using Class ========

-> int:- Maximum value: 4
-> float:- Maximum value: 7.4
-> string:- Maximum value: world
```

Question 02

Wite a C++ program that demonstrates exception handling. Follow the instructions below:

a) **Create a function called divide(int numerator, int denominator) that divides the numerator by the denominator and returns the result.**

```
float divide(int numerator = 0, int denominator = 0)
{
    return numerator / denominator;
}
```

b) **Implement exception handling in the divide() function to handle the scenario when the denominator is zero. Throw an exception of type std::runtime_error in this case with the message "Division by zero is not allowed."**

```
float divide(int numerator = 0, int denominator = 0)
{
    if (denominator == 0)
    {
        throw runtime_error("Division by zero is not allowed.");
    }
    return numerator / denominator;
}
```

c) In your program, call the divide() function with different inputs, including a scenario where the denominator is zero. Use a try-catch block to handle the exception and display an appropriate error message.

```cpp
int main()
{
    int numerator = 0, denominator = 0;

    cout << "Enter Numerator: ";
    cin >> numerator;
    cout << "Enter Denominator: ";
    cin >> denominator;

    try
    {
        double result = divide(numerator, denominator);
        cout << "Result: " << result << endl;
    }
    catch (runtime_error e)
    {
        cout << "Error: " << e.what() << endl;
    }

    return 0;
}
```

Code Explanation: In my code, I created a function called divide() that performs division between two numbers and returns the result. I also implemented exception handling to address a specific scenario where the second number (denominator) is zero, which is not allowed in division. If the denominator is zero, it throws an exception of type std::runtime_error with the error message "Division by zero is not allowed."

Sample Code:

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

float divide(int numerator = 0, int denominator = 0)
{
    if (denominator == 0)
    {
        throw runtime_error("Division by zero is not allowed.");
    }
    return numerator / denominator;
}
```

```cpp
int main()
{
    int numerator = 0, denominator = 0;

    cout << "Enter Numerator: ";
    cin >> numerator;
    cout << "Enter Denominator: ";
    cin >> denominator;

    try
    {
        double result = divide(numerator, denominator);
        cout << "Result: " << result << endl;
    }
    catch (runtime_error e)
    {
        cout << "Error: " << e.what() << endl;
    }

    return 0;
}
```

OutPut:

```
Enter Numerator: 12
Enter Denominator: 0
Error: Division by zero is not allowed.
```

```
Enter Numerator: 24
Enter Denominator: 12
Result: 2
```

Question 03

Write a C++ program that demonstrates aggregation and composition. Follow the instructions below:

a) Create a class called Author with the following attributes: name (string), age (integer), country (string).

```cpp
class author
{
    string name;
    int age;
    string country;

public:
    author(string name, int age, string country)
    {
        this->name = name;
        this->age = age;
        this->country = country;
    }
};
```

b) Create a class called Book with the following attributes and methods:
Aggregation relationship: a pointer to an instance of the Author class
Method: getAuthorInfo() (prints the author's name, age, and country)

```cpp
class book
{
    string title;
    int year;
    author *a1;

public:
    book(string title, int year, author *author)
    {
        this->title = title;
        this->year = year;
        this->a1 = author;
    }

    void getAuthorInfo()
    {
        cout << "Author's Name: " << a1->getName() << endl;
        cout << "Author's Age: " << a1->getAge() << endl;
        cout << "Author's Country: " << a1->getCountry() << endl;
    }
};
```

c) **In your program, create an instance of the Author class and an instance of the Book class, demonstrating the aggregation relationship. Call the getAuthorInfo() method to display the author's information.**

Explanation: To achieve aggregation, a pointer to the Author class is created within the Book class. This pointer represents a relationship where a book "has an" author. An object of the Author class is then assigned to this pointer, establishing the aggregation relationship between the Book and Author classes.

Sample Code:

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

class author
{
    string name;
    int age;
    string country;

public:
    author(string name, int age, string country)
    {
        this->name = name;
        this->age = age;
        this->country = country;
    }

    string getName()
    {
        return name;
    }
    int getAge()
    {
        return age;
    }
    string getCountry()
    {
        return country;
    }
};

class book
{
    string title;
    int year;
    author *a1;
```

```
public:
    book(string title, int year, author *author)
    {
        this->title = title;
        this->year = year;
        this->a1 = author;
    }

    void getAuthorInfo()
    {
        cout << "Author's Name: " << a1->getName() << endl;
        cout << "Author's Age: " << a1->getAge() << endl;
        cout << "Author's Country: " << a1->getCountry() << endl;
    }
};
int main()
{
    author a1("Eman Murtaza", 40, "USA");
    book b1("Sample Book", 2023, &a1);

    b1.getAuthorInfo();

    return 0;
}
```

OutPut:

```
Author's Name: Eman Murtaza
Author's Age: 40
Author's Country: USA
```

d) Modify your program to demonstrate a composition relationship, where the Book class has a composition relationship with an instance of the Author class. Update the program accordingly and provide a brief explanation of the changes made.

Explanation: To achieve composition, an instance of the Book class is created within the Author class. This represents a stronger relationship where an author "consists of" a book. The Book object is directly created within the Author class, indicating a composition relationship between the two classes.

Sample Code:

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

class book;
class author
{
    string name;
```

```
    int age;
    string country;
    book *b1;

public:
    author(string name = "", int age = 0, string country = "");

    string getName()
    {
        return name;
    }
    int getAge()
    {
        return age;
    }
    string getCountry()
    {
        return country;
    }

    void getAuthorInfo()
    {
        cout << "Author's Name: " << name << endl;
        cout << "Author's Age: " << age << endl;
        cout << "Author's Country: " << country << endl;
    }
};

class book
{
    string title;
    int year;
    author *a1;

public:
    book(string title = "", int year = 0)
    {
        this->title = title;
        this->year = year;
    }

    void setTitle(string title = "")
    {
        this->title = title;
    }
    void setYear(int year = 0)
    {
        this->year = year;
```

```
    }
};

author ::author(string name, int age, string country)
{
    b1 = new book();
    this->name = name;
    this->age = age;
    this->country = country;
}

int main()
{
    author a1("Eman Murtaza", 40, "USA");

    a1.getAuthorInfo();

    return 0;
}
```

OutPut:

```
Author's Name: Eman Murtaza
Author's Age: 40
Author's Country: USA
```