OOP Lab

Name: **Soham Das**

Section: **A1**

Roll No: **002311001004**

Assignment - 5

IT-UG2

44. Two integers are taken from keyboard. Then perform division operation. Write a try block to throw an exception when division by zero occurs and appropriate catch block to handle the exception thrown.

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

int main()
{
    int a, b;
    cout << "Enter two integers: ";
    cin >> a >> b;

    try
    {
        if (b == 0)
            throw "Division by zero!!!";
        cout << "Result: " << a / b << endl;
    }
    catch (const char *msg)
    {
        cout << "Error: " << msg << endl;
    }

    return 0;
}
```

45. Write a C++ program to demonstrate the use of try, catch block with the argument as an integer and string using multiple catch blocks.

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

int main() {
    try {
        int choice;
        cout << "Enter 1 for integer exception or 2 for string
exception: ";
        cin >> choice;

        if (choice == 1)
            throw -1;
        else if (choice == 2)
```

```cpp
            throw string("String exception");
        else
            cout << "No exception thrown" << endl;
    } catch (int e) {
        cout << "Caught an integer exception: " << e << endl;
    } catch (const string& e) {
        cout << "Caught a string exception: " << e << endl;
    }

    return 0;
}
```

46. Create a class with member functions that throw exceptions. Within this class, make a nested class to use as an exception object. It takes a single const char* as its argument; this represents a description string. Create a member function that throws this exception. (State this in the function s exception specification.) Write a try block that calls this function and a catch clause that handles the exception by displaying its description string.

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

class Sample
{
public:
    class Exception
    {
        const char *message;

    public:
        Exception(const char *msg)
        {
            message = msg;
        }
        const char *err() const
        {
            return message;
        }
    };

    void throwException()
    {
        throw Exception("An exception occurred");
    }
};

int main()
{
    Sample obj;
    try
    {
        obj.throwException();
    }
    catch (const Sample::Exception &e)
    {
        cout << e.err() << endl;
    }
    return 0;
}
```

47. Design a class Stack with necessary exception handling.

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

class Stack
{
    int *arr;
    int top;
    int capacity;

public:
    class StackOverflow
    {
    };
    class StackUnderflow
    {
    };

    Stack(int size) : capacity(size), top(-1)
    {
        arr = new int[size];
    }

    ~Stack()
    {
        delete[] arr;
    }
```

```cpp
    void push(int value)
    {
        if (top == capacity - 1)
            throw StackOverflow();
        arr[++top] = value;
    }

    int pop()
    {
        if (top == -1)
            throw StackUnderflow();
        return arr[top--];
    }

    int peek() const
    {
        if (top == -1)
            throw StackUnderflow();
        return arr[top];
    }

    bool isEmpty() const
    {
        return top == -1;
    }
};

int main()
{
    Stack s(3);

    try
    {
        s.push(10);
        s.push(20);
        s.push(30);
        cout << s.pop() << endl;
        cout << s.pop() << endl;
        cout << s.pop() << endl;
        cout << s.pop() << endl;
    }
    catch (const Stack::StackOverflow &)
    {
        cout << "Stack overflow occurred!" << endl;
    }
    catch (const Stack::StackUnderflow &)
    {
        cout << "Stack underflow occurred!" << endl;
    }

    return 0;
}
```

48. Write a Garage class that has a Car that is having troubles with its Motor. Use a function-level try block in the Garage class constructor to catch an exception (thrown from the Motor class) when its Car object is initialized. Throw a different exception from the body of the Garage constructor s handler and catch it in main( ).

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

class Motor
{
public:
    Motor()
    {
        throw "Motor failure!";
    }
};

class Car
{
    Motor motor;

public:
    Car()
    try : motor()
    {
    }
    catch (const char *e)
    {
        throw "Car's motor has problems!";
    }
};

class Garage
{
    Car car;

public:
    Garage()
    try : car()
    {
    }
    catch (const char *e)
    {
        throw "Garage encountered a car problem!";
    }
};

int main()
{
    try
    {
        Garage garage;
    }
    catch (const char *e)
    {
        cout << e << endl;
    }
    return 0;
}
```

49. Vehicles may be either stopped of running in a lane. If two vehicles are running in opposite direction in a single lane there is a chance of collision. Write a C++ program using exception handling to avoid collisions. You are free to make necessary assumptions.

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

class CollisionException
{
public:
    const char *err() const
    {
        return "Collision detected! Vehicles are running in opposite 
directions.";
    }
};

class Vehicle
{
    string direction;
public:
    Vehicle(const string &dir)
    {
        direction = dir;
    }

    const string &getDirection() const
    {
        return direction;
    }
};

void checkForCollision(const Vehicle &v1, const Vehicle &v2)
{
    if (v1.getDirection() != v2.getDirection())
        throw CollisionException();
}

int main()
{
    try
    {
        Vehicle car1("North");
        Vehicle car2("South");
        checkForCollision(car1, car2);
        cout << "Vehicles are safe." << endl;
    }
    catch (const CollisionException &e)
    {
        cout << e.err() << endl;
    }
    return 0;
}
```

50. Write a template function max() that is capable of finding maximum of two things (that can be compared). Used this function to find (i) maximum of two integers, (ii) maximum of two complex numbers (previous code may be reused). Now write a specialized template function for strings (i.e. char *). Also find the maximum of two strings using this template function.

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

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

template <>
const char *max(const char *a, const char *b)
{
    return (strcmp(a, b) > 0) ? a : b;
}

class Complex
{
    double real, imag;

public:
    Complex(double r, double i) : real(r), imag(i) {}

    double magnitude() const
    {
        return real * real + imag * imag;
    }

    bool operator>(const Complex &other) const
    {
        return magnitude() > other.magnitude();
    }

    friend ostream &operator<<(ostream &os, const Complex &c)
    {
        os << c.real << " + " << c.imag << "i";
        return os;
    }
};

int main()
{
    cout << "Max of 10 and 20: " << max(10, 20) << endl;

    Complex c1(3, 4), c2(1, 7);
    cout << "Max of complex numbers: " << max(c1, c2) << endl;

    const char *str1 = "Apple";
    const char *str2 = "Orange";
    cout << "Max of strings: " << max(str1, str2) << endl;

    return 0;
}
```

51. Write a template function swap() that is capable of interchanging the values of two variables. Used this function to swap (i) two integers, (ii) two complex numbers (previous code may be reused). Now write a specialized template function for the class Stack (previous code may be reused). Also swap two stacks using this template function.

```
#include <iostream>

using namespace std;

// Template function for swapping two variables
template <typename T>
void swap(T &a, T &b)
{
    T temp = a;
    a = b;
    b = temp;
}

// Class definition for Complex
class Complex
{
public:
    double real, imag;

    Complex(double r = 0, double i = 0) : real(r), imag(i) {}

    void display() const
    {
        cout << real << " + " << imag << "i";
    }
};

// Template class for Stack
template <typename T>
class Stack
{
    T arr[100];
    int top;
```

```cpp
public:
    Stack() : top(-1) {}

    void push(T value)
    {
        if (top < 99)
        {
            arr[++top] = value;
        }
    }

    T pop()
    {
        if (top >= 0)
        {
            return arr[top--];
        }
        return T(); // Return default value if stack is empty
    }

    bool isEmpty() const
    {
        return top == -1;
    }

    void display() const
    {
        for (int i = top; i >= 0; --i)
        {
            cout << arr[i] << " ";
        }
        cout << endl;
    }
};

// Specialized template function for swapping two stacks
template <typename T>
void swap(Stack<T> &s1, Stack<T> &s2)
{
    Stack<T> temp = s1;
    s1 = s2;
    s2 = temp;
}

int main()
```

```
{
    // Swap two integers
    int int1 = 10, int2 = 20;
    swap(int1, int2);
    cout << "Swapped integers: " << int1 << " " << int2 << endl;

    // Swap two complex numbers
    Complex c1(1.0, 2.0), c2(3.0, 4.0);
    swap(c1, c2);
    cout << "Swapped complex numbers: ";
    c1.display();
    cout << ", ";
    c2.display();
    cout << endl;

    // Swap two stacks
    Stack<int> stack1, stack2;
    stack1.push(1);
    stack1.push(2);
    stack2.push(3);
    stack2.push(4);

    cout << "Stack 1 before swap: ";
    stack1.display();
    cout << "Stack 2 before swap: ";
    stack2.display();

    swap(stack1, stack2);

    cout << "Stack 1 after swap: ";
    stack1.display();
    cout << "Stack 2 after swap: ";
    stack2.display();

    return 0;
}
```

52. Create a C++ template class for implementation of Stack data structure. Create a Stack of integers and a Stack of complex numbers created earlier (code may be reused). Perform some push and pop operations on these stacks. Finally print the elements remained in those stacks.

```
#include <iostream>
```

```cpp
using namespace std;

template <typename T>
class Stack
{
    T arr[100];
    int top;

public:
    Stack() : top(-1) {}

    void push(T value)
    {
        if (top < 99)
        {
            arr[++top] = value;
        }
    }

    T pop()
    {
        if (top >= 0)
        {
            return arr[top--];
        }
        return T();
    }

    void display() const
    {
        for (int i = top; i >= 0; --i)
        {
            cout << arr[i] << " ";
        }
        cout << endl;
    }
};

class Complex
{
public:
    double real, imag;

    Complex(double r = 0, double i = 0) : real(r), imag(i) {}

    void display() const
    {
        cout << real << " + " << imag << "i";
    }

    friend ostream &operator<<(ostream &os, const Complex &c)
    {
        os << c.real << " + " << c.imag << "i";
        return os;
    }
};

int main()
{
    Stack<int> intStack;
    intStack.push(10);
    intStack.push(20);
    intStack.push(30);
    intStack.pop();
    intStack.push(40);
    cout << "Integer Stack: ";
    intStack.display();

    Stack<Complex> complexStack;
    complexStack.push(Complex(1.0, 2.0));
    complexStack.push(Complex(3.0, 4.0));
    complexStack.pop();
    complexStack.push(Complex(5.0, 6.0));
    cout << "Complex Stack: ";
    complexStack.display();

    return 0;
}
```