Assignment 5

Name: Tanish Majumdar Roll No.: 002311001077

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 <bits/stdc++.h>
#define ll long long
#define vll vector<long long>
#define db double
#define vi vector<int>
using namespace std;

int main()
{
    int a, b;
    cout << "Enter two numbers: ";
    cin >> a >> b;
    try
    {
        if (b == 0)
        {
            throw string("Division by zero is not allowed");
        }
        cout << "Division: " << a / b << endl;
    }
    catch (const string &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 <bits/stdc++.h>
#define ll long long
#define vll vector<long long>
#define db double
#define vi vector<int>
using namespace std;

int main()
{
    int a;
    try
    {
        cout << "Enter a number: ";
        cin >> a;

        if (a < 0)
        {
            throw a;
        }
        else if (a == 0)
        {
            throw string("Zero is not allowed");
        }
        else
        {
            cout << "You entered: " << a << endl;
        }
    }
    catch (int e)
    {
        cout << "Caught an integer exception: " << e << endl;
    }
    catch (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.

```
#include <bits/stdc++.h>
#define ll long long
#define vll vector<long long>
#define db double
#define vi vector<int>
using namespace std;

class MyClass
{
public:
    class Exception
    {
        const char *description;
    public:
        Exception(const char *desc) : description(desc) {}
        const char *what() const { return description; }
    };

    void throwException()
    {
        throw Exception("An error occurred in MyClass");
    }
};

int main()
{
    MyClass obj;
    try
    {
        obj.throwException();
    }
    catch (const MyClass::Exception &e)
    {
        cout << "Caught exception: " << e.what() << endl;
    }

    return 0;
}
```

47. Design a class Stack with necessary exception handling.

```cpp
#include <bits/stdc++.h>
#define ll long long
#define vll vector<long long>
#define db double
#define vi vector<int>
using namespace std;
using namespace std;

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

public:
    Stack(int size = 10)
    {
        if (size <= 0)
        {
            throw invalid_argument("Stack size must be positive");
        }
        arr = new int[size];
        capacity = size;
        top = -1;
    }

    ~Stack()
    {
        delete[] arr;
    }

    void push(int x)
    {
        if (isFull())
        {
            throw overflow_error("Stack overflow");
        }
        arr[++top] = x;
    }

    int pop()
    {
        if (isEmpty())
        {
            throw underflow_error("Stack underflow");
        }
        return arr[top--];
    }
```

```
    int peek() const
    {
        if (isEmpty())
        {
            throw underflow_error("Stack is empty");
        }
        return arr[top];
    }

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

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

int main()
{
    try
    {
        Stack stack(5);

        stack.push(1);
        stack.push(2);
        stack.push(3);
        stack.push(4);
        stack.push(5);

        cout << "Top element is: " << stack.peek() << endl;

        stack.pop();
        stack.pop();
        stack.pop();
        stack.pop();
        stack.pop();

        // cout << "Stack is empty: " << stack.isEmpty() << endl;
        stack.pop();
    }
    catch (const exception &e)
    {
        cout << "Exception: " << e.what() << 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 handler and catch it in `main()`.

```
#include <bits/stdc++.h>
#define ll long long
#define vll vector<long long>
#define db double
#define vi vector<int>
using namespace std;

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

class Car
{
public:
    Motor motor;
    Car()
    try : motor()
    {
    }
    catch (const runtime_error &e)
    {
        cout << "Caught exception in Car constructor: " << e.what() <<
endl;
        throw;
    }
};

class Garage
{
public:
    Car car;
```

```
    Garage()
    try : car()
    {
    }
    catch (const runtime_error &e)
    {
        cout << "Caught exception in Garage constructor: " << e.what() <<
endl;
        throw logic_error("Garage initialization failed due to car motor
issue");
    }
};

int main()
{
    try
    {
        Garage garage;
    }
    catch (const logic_error &e)
    {
        cout << "Caught exception in main: " << e.what() << endl;
    }
}
```

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 <bits/stdc++.h>
#define ll long long
#define vll vector<long long>
#define db double
#define vi vector<int>
using namespace std;

class Vehicle
{
public:
    enum Direction
    {
        STOPPED,
        FORWARD,
        BACKWARD
```

```
    };
    Direction direction;

    Vehicle(Direction dir) : direction(dir) {}
};

class Lane
{
public:
    void checkForCollision(const Vehicle &v1, const Vehicle &v2)
    {
        if (v1.direction == Vehicle::FORWARD && v2.direction ==
Vehicle::BACKWARD)
        {
            throw runtime_error("Collision detected: Vehicles are running
in opposite directions!");
        }
        if (v1.direction == Vehicle::BACKWARD && v2.direction ==
Vehicle::FORWARD)
        {
            throw runtime_error("Collision detected: Vehicles are running
in opposite directions!");
        }
    }
};

int main()
{
    try
    {
        Vehicle car1(Vehicle::FORWARD);
        Vehicle car2(Vehicle::BACKWARD);

        Lane lane;
        lane.checkForCollision(car1, car2);

        cout << "No collision detected. Vehicles are safe." << endl;
    }
    catch (const runtime_error &e)
    {
        cout << "Exception: " << e.what() << 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 <bits/stdc++.h>
#define ll long long
#define vll vector<long long>
#define db double
#define vi vector<int>
using namespace std;

class Complex
{
private:
    double real;
    double imag;

public:
    Complex(double r = 0.0, double i = 0.0) : real(r), imag(i) {}

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

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

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

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

template <>
```

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

int main()
{
    int int1 = 10, int2 = 20;
    cout << "Maximum of " << int1 << " and " << int2 << " is " <<
customMax(int1, int2) << endl;

    Complex complex1(3.0, 4.0), complex2(1.0, 7.0);
    cout << "Maximum of " << complex1 << " and " << complex2 << " is " <<
customMax(complex1, complex2) << endl;

    const char *str1 = "apple";
    const char *str2 = "banana";
    cout << "Maximum of \"" << str1 << "\" and \"" << str2 << "\" is \""
<< customMax(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 <bits/stdc++.h>
using namespace std;

template <typename T>
void swapValues(T &a, T &b)
{
    T temp = a;
    a = b;
    b = temp;
}

class Complex
```

```
{
private:
    double real;
    double imag;

public:
    Complex(double r = 0.0, double i = 0.0) : real(r), imag(i) {}

    double getReal() const { return real; }
    double getImag() const { return imag; }

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

template <typename T>
class Stack
{
private:
    vector<T> elements;

public:
    void push(T value) { elements.push_back(value); }
    void pop()
    {
        if (!elements.empty())
            elements.pop_back();
    }
    T top() const { return elements.back(); }
    bool empty() const { return elements.empty(); }

    friend ostream &operator<<(ostream &os, const Stack<T> &stack)
    {
        for (const T &el : stack.elements)
            os << el << " ";
        return os;
    }
};

template <typename T>
void swapValues(Stack<T> &stack1, Stack<T> &stack2)
{
    Stack<T> temp = stack1;
    stack1 = stack2;
    stack2 = temp;
}
```

```
int main()
{

    int a = 10, b = 20;
    cout << "Before swapping: a = " << a << ", b = " << b << endl;
    swapValues(a, b);
    cout << "After swapping: a = " << a << ", b = " << b << endl;

    Complex c1(3, 4), c2(1, 7);
    cout << "\nBefore swapping: c1 = " << c1 << ", c2 = " << c2 << endl;
    swapValues(c1, c2);
    cout << "After swapping: c1 = " << c1 << ", c2 = " << c2 << endl;

    Stack<int> stack1, stack2;
    stack1.push(1);
    stack1.push(2);
    stack1.push(3);
    stack2.push(10);
    stack2.push(20);
    stack2.push(30);

    cout << "\nBefore swapping:\nStack1: " << stack1 << "\nStack2: " <<
stack2 << endl;
    swapValues(stack1, stack2);
    cout << "After swapping:\nStack1: " << stack1 << "\nStack2: " <<
stack2 << endl;

    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 <bits/stdc++.h>
#define ll long long
#define vll vector<long long>
#define db double
#define vi vector<int>
using namespace std;

template <typename T>
class Stack
```

```
{
private:
    vector<T> elements;

public:
    void push(T value)
    {
        elements.push_back(value);
    }

    void pop()
    {
        if (!elements.empty())
            elements.pop_back();
        else
            cout << "Stack is empty, cannot pop." << endl;
    }

    T top() const
    {
        if (!elements.empty())
            return elements.back();
        throw runtime_error("Stack is empty, no top element.");
    }

    bool empty() const
    {
        return elements.empty();
    }

    void print() const
    {
        if (elements.empty())
        {
            cout << "Stack is empty." << endl;
        }
        else
        {
            cout << "Stack elements: ";
            for (const T &el : elements)
            {
                cout << el << " ";
            }
            cout << endl;
        }
    }
};

class Complex
{
```

```
private:
    double real;
    double imag;

public:
    Complex(double r = 0.0, double i = 0.0) : real(r), 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);

    cout << "Integer Stack after pushes:" << endl;
    intStack.print();

    intStack.pop();
    cout << "Integer Stack after one pop:" << endl;
    intStack.print();

    Stack<Complex> complexStack;
    complexStack.push(Complex(3, 4));
    complexStack.push(Complex(1, 7));
    complexStack.push(Complex(2, 5));

    cout << "Complex Number Stack after pushes:" << endl;
    complexStack.print();

    complexStack.pop();
    cout << "Complex Number Stack after one pop:" << endl;
    complexStack.print();

    return 0;
}
```