OOPS LAB

ASSIGNMENT – 5

NAME : Swarnendu Banerjee

ROLL : 002311001016

DEPARTMENT : IT – A1

Complex.h

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

class Complex{
  int x,y;
  public:
  Complex(int xx=0, int yy=0){
    x=xx;
    y=yy;
  }
  bool operator>(Complex &ob){
    if(x>ob.x) return true;
    if(x==ob.x && y>ob.y) return true;
    return false;
  }
  friend ostream& operator<<(ostream &os, Complex &ob){
    os << "(" << ob.x << "," << ob.y << ")\n";
    return os;
  }
};
```

Stack.h

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

class Overflow {
  public:
```

```
  Overflow() {
    cout << "Stack is full\n";
  }
};

class Underflow {
 public:
  Underflow() {
    cout << "No elements to delete\n";
  }
};

class Stack {
 public:
  int top, size;
  int *arr;
  Stack(int s = 0) {
    size = s;
    top = -1;
    arr = new int[size];
  }

  void push(int val) {
    if (top >= size - 1) throw Overflow();
    arr[++top] = val;
  }

  int pop() {
    if (top == -1) throw Underflow();
```

```
    return arr[top--];
  }

  friend ostream &operator<<(ostream &os, Stack s) {
    for (int i = 0; i <= s.top; i++) {
      cout << s.arr[i] << " ";
    }
    cout << "\n";
        return os;
  }
};
```

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.

CODE:

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

class ZeroDivide{
    public:
        void disp(){
            cout<<"zero divide"<<endl;
        }
        ZeroDivide(int y=0){
            cout<<y<<"constructed"<<endl;}
```

```
};

int divi (int x,int y){
        if(y==0){
                throw ZeroDivide();}
                 return x/y;
        }

int main(){
        int a,b;
        cout<<"enter two no.-->";
        cin>>a>>b;
        try{
                int c=divi(a,b);
                cout<<c;
        }
        catch(ZeroDivide ob){
                ob.disp();}
}
```

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.

CODE:

```
#include<iostream>
#include<cstring>
using namespace std;
	int main(){
		int arr[]={-1,0,1};
		for(int i=0;i<3;i++){
			int ex=arr[i];
			try{
				if(ex>0){
					throw(ex);}
				else{
					throw "string";
				}
			}
			catch(int x){
				cout<<"integer-"<<x<<endl;}
			catch(const char *s){
				cout<<s<<endl;}
		}
	}
```

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.

CODE:

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


class outer{
        int a;
        int b;

        public:
        outer(int x=0,int y=0):a(x),b(y){}
                class divi{
                        const char *c;
                        public:
                        divi(const char *cc="Default message"){
                                c=cc;}
                        void disp(){
                                cout<<"Error: "<<c<<"\n";
                        }
                };
                int  division(int x,int y){
                        if(y==0) throw divi("division by zero");
                         return x/y;}
```

```
};

int main(){
	int a,b;
	cout<<"enter two no. :";
	cin>>a>>b;
	outer ob(a,b);
	try{
		cout<<ob.division(a,b)<<endl;
	}
	catch(outer::divi d){
		d.disp();}
}
```

Q47. Design a class Stack with necessary exception handling.

CODE:

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

class overflow{
	public:
		overflow(int e=0){
			cout<<"stack is full"<<endl;}
```

```
};

class underflow{
        public:
                underflow(int u=0){
                        cout<<"stack empty"<<endl;}
};

class stack{
        int *arr;
        int size;
        int top;
        public:
        stack(int s){
                size=s;
                top=-1;
                arr=new int[size];}
        void push(int x){
                if(top==size-1){
                        throw overflow(-1);}
                top++;
                arr[top]=x;
        }
        int pop(){
```

```
        if(top==-1){
            throw underflow(-1);
        }
        int val=arr[top];
        top-=1;
        return val;}
};

int main(){
    stack ob(5);
    try{
        ob.push(1);
        ob.push(2);
        ob.push(3);
        ob.push(4);
        ob.push(5);
        ob.push(6);
        }
    catch(overflow o){
        cout<<"no more space into stack"<<endl;
    }
    try{
        cout<<ob.pop()<<endl;
        cout<<ob.pop()<<endl;
```

```
            cout<<ob.pop()<<endl;
            cout<<ob.pop()<<endl;
            cout<<ob.pop()<<endl;
            cout<<ob.pop();
        }
        catch (underflow u){
            }
}
```

Q48. 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( ).

CODE:

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


class motor{
   int tr;
public:
motor(int z=0){
 tr=z;
 if(!tr) throw "Working";
 else throw "Not working";
```

```
    }
};

class car{
motor*v;   public:
car(int z=0){
try{       v=new
motor(z);
    }
    catch(const char*ch){
throw ;
    }

  }

};


class garage{
public:
garage(int f=0){
try{       car
ct(f);     }
    catch(const char *s){
```

```
      cout<<"the motor of the car is-->"<<s<<endl;
if(s=="Not working")throw "NEED to  repair";
else throw "No need to repair";
    }
  }
};

int main(){
int arr[]={1,0};
 for(int i=0;i<2;i++){
 try{
 cout<<"for "<<i+1<<" car ";
 garage g(arr[i]);
    }
    catch(const char *s){
cout<<s<<endl;
    }
  }
}
```

Q49. 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.

CODE:

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

class Collision {
  public:
   Collision() {
       cout << "Cars must be stopped. There may be collision\n";
   }
};

class Vehicle {
   int direction, speed, x;

  public:
   Vehicle(int d = 0, int s = 0, int xx = 0) {
       direction = d;
       speed = s;
       x = xx;
```

```
    }

    friend void detectCollision(Vehicle &v1, Vehicle &v2) {
        bool flag = false;
        if (v1.x == v2.x) flag = true;

        else if (v1.x < v2.x) {
            if ((v1.direction > 0 && v2.direction < 0) || (v1.direction > 0
&& v2.direction > 0 && v1.speed > v2.speed) || (v1.direction < 0
&& v2.direction < 0 && v2.speed > v1.speed))
                flag = true;
        } else {
            if ((v2.direction > 0 && v1.direction < 0) || (v2.direction >
0 && v1.direction > 0 && v2.speed > v1.speed) || (v2.direction < 0
&& v1.direction < 0 && v1.speed > v2.speed))
                flag = true;
        }

        if (flag)
            throw Collision();
        else
            cout << "No collision\n";
    }
};
```

```
int main() {

    int ss, dd, xx;
    cout << "Enter position, direction and speed: ";
    cin >> xx >> dd >> ss;
    Vehicle v1(dd,ss,xx);
    cout << "Enter position, direction and speed: ";
    Vehicle v2(dd,ss,xx);
    cin >> xx >> dd >> ss;
    try{
    detectCollision(v1, v2);
    }
    catch(Collision ob){
    }

    return 0;
}
```

Q50. 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.

CODE:

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

class Complex {
private:
    double real;
    double imag;

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

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

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

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

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

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

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

int main() {
    int x = 5, y = 8;
    cout << "Max of " << x << " and " << y << " is: " << maximum(x, y)
<< endl;

    Complex c1(3, 4), c2(5, 12);
    cout << "Max of complex numbers of ";
    c1.display();
     cout<<" and ";
    c2.display();
    cout<<" is";
    Complex maxComplex = maximum(c1, c2);
    maxComplex.display();
    cout << endl;

    const char* str1 = "apple";
    const char* str2 = "banana";
    cout << "Max of \"" << str1 << "\" and \"" << str2 << "\" is: " <<
maximum(str1, str2) << endl;

    return 0;
}
```

Q51. 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.

CODE:

```
#include "complex.h"
#include <iostream>
#include "stack.h"

using namespace std;

template<class T>
void swapTwo(T &x, T &y){
    T temp = x;
    x = y;
    y = temp;
}

template<>
void swapTwo(Stack &x, Stack &y){
    Stack temp;
    temp.size = x.size;
    x.size = y.size;
    y.size = temp.size;

    temp.top = x.top;
```

```
    x.top = y.top;
    y.top = temp.top;

    temp.arr = x.arr;
    x.arr = y.arr;
    y.arr = temp.arr;
}

int main(){
    int a,b;
    cout << "Enter two variables: ";
    cin >> a >> b;
    swapTwo<int>(a,b);
    cout << "After swapping, a= " << a << " b= " << b << "\n";

    cout << "Enter real and imaginary part of 1st complex
number: ";
    cin >> a >> b;
    Complex c1(a,b);

    cout << "Enter real and imaginary part of 2nd complex
number: ";
    cin >> a >> b;
    Complex c2(a,b);
```

```
    swapTwo(c1,c2);
    cout << "After swapping, first= " << c1 << "Second= " << c2;


    Stack s1(5),s2(4);
    s1.push(1);
    s1.push(2);
    s1.push(3);
    s1.push(4);
    s1.push(5);
    s2.push(6);
    s2.push(7);
    s2.push(8);
    s2.push(9);
    cout << "Before swapping:\nStack 1:\n" << s1 << "Stack 2:\n" <<
s2;
    swapTwo<Stack>(s1,s2);
    cout << "After swapping:\nStack 1:\n" << s1 << "Stack 2:\n" <<
s2;
    return 0;
}
```

Q52. 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.

CODE:

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

class Complex {
private:
    double real;
    double imag;

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

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

    void display(){
        cout << "(" << real << ", " << imag << ")";
    }
```

```
    bool operator==(const Complex& other)  {
        return (real == other.real && imag == other.imag);
    }
};

template <typename T>
class Stack {
private:
    T* arr;
    int topIndex;
    int size;

public:
    Stack(int s = 100) {
        arr = new T[s];
        topIndex = -1;
        size = s;
    }


    class Overflow {
    public:
        Overflow(const char* str) {
            cout << "Stack Overflow Exception: " << str << endl;
```

```
    }
};


class Underflow {
public:
    Underflow(const char* str) {
        cout << "Stack Underflow Exception: " << str << endl;
    }
};

void push(T value) {
    if (topIndex == size - 1) {
        throw Overflow("Stack is full");
    } else {
        arr[++topIndex] = value;
    }
}

T pop() {
    if (topIndex == -1) {
        throw Underflow("No elements to pop");
    } else {
        return arr[topIndex--];
```

```
        }
    }

    bool empty() {
        return topIndex == -1;
    }

    T top() {
        if (topIndex == -1) {
            throw Underflow("No elements");
        }
        return arr[topIndex];
    }

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

int main() {
    try {

        Stack<int> intStack(5);
        intStack.push(6);
```

```
    intStack.push(78);
    intStack.push(62);
    intStack.push(9);

    cout << "Top of int stack: " << intStack.top() << endl;

    intStack.pop();
    intStack.pop();
    cout << "Top after popping two elements: " <<
intStack.top() << endl;



    Stack<Complex> complexStack(3);
    complexStack.push(Complex(3, 4));
    complexStack.push(Complex(5, 12));
    complexStack.push(Complex(1, 1));

    cout << "Top of complex stack: ";
    complexStack.top().display();
    cout << endl;

    complexStack.pop();
    complexStack.pop();
```

```
        cout << "Top of complex stack after popping two
elements:";
        complexStack.top().display();
        cout << endl;
        complexStack.push(Complex(2, 2));
        complexStack.pop();
        complexStack.pop();
        complexStack.pop();


    }
    catch (Stack<int>::Overflow ob1) {
        cout << "Overflow exception handled for int stack!" << endl;
    }
    catch (Stack<int>::Underflow ob2) {
        cout << "Underflow exception handled for int stack!" <<
endl;
    }
    catch (Stack<Complex>::Overflow ob1) {
        cout << "Overflow exception handled for complex stack!"
<< endl;
    }
    catch (Stack<Complex>::Underflow ob2) {
```

```
        cout << "Underflow exception handled for complex stack!"
<< endl;
    }
    catch (...) {
        cout << "Default exception caught!" << endl;
    }

    return 0;
}
```