OOPS ASSIGNMENT – 5

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.

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

class DivisionByZeroException
{
    public:
    DivisionByZeroException() {}
};

int main()
{
    float x, y, res;
    while (true)
    {
        cout << "enter x and y to get x/y \n";
        cin >> x >> y;
        try
        {
            if (y == 0){
                throw DivisionByZeroException();
            }
            else{
                res = x / y;
                cout << "Quotient=" << res << "\n";
            }
        }
        catch (DivisionByZeroException e)
        {
            cout << "Division by zero not possible \nException thrown...\nTerminating
program...\n";
            break;
        }
    }
}
```

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.

```
#include <iostream>
using namespace std;
int main()
{
    int x[] ={1,2,3,4,5};
```

```
        for (int i=0;i<=2)
        {
            try{
                if(x[i]<=2)
                throw -1;
                else
                throw "exception";
            }
            catch (int e) {
                cout << "Caught an integer exception: " << e << endl;
            }
            catch (const char* ch) {
                cout << "Caught a string exception: " << ch << 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 <iostream>
using namespace std;
class Test
{
        public:
        class Custom
        {
                const char* description;
                public:
                Custom(const char* desc)
                {
                        description=desc;
                }
                void disp()
                {
                        cout<<description<<endl;
                }
        };
        Void Test()
        {
                Throw Custom(“Testing the Exception class”);
        }
  };
```

```
Int main() {
        Test obj;
        try{
                Obj.test();
        }
        catch (Test:::Custom ob) {
                Ob.disp();
        }
        return 0;
}
```

47. Design a class Stack with necessary exception handling.

```
#include <iostream>
using namespace std;
class Stack
{
        int size;
        int *arr;
        int top;
        public:
        Stack(int s)
        {
                size=s;
                top=0;
                arr=new int[size];
        }
        void push(int n)
        {
                if(top<=size-1)
                arr[top++]=n;
                else
                throw "Stack Overflow";
        }
        int pop()
        {
                int p;
                if(top<1)
                throw “Stack Underflow”;
                return arr[--top];
        }
};
int main()
{
        Stack ob(3);
        try{
                ob.push(1);
```

```
                ob.push(2);
                ob.push(3);
                ob.push(4);
        }
        catch(const char * e)
        {
                cout<<e<<endl;
        }
        try{
                cout<<ob.pop()<<endl;
                cout<<ob.pop()<<endl;
                cout<<ob.pop()<<endl;
                cout<<ob.pop()<<endl;
        }
        catch(const char * e)
        {
                cout<<e<<endl;
        }
}
```

48. Write a Garage class that has a Car that is having troubles with its Motor. Use a functionlevel 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
{
        int parts;
        public:
        Motor()
        {
                parts=0;
        }
        void assign_parts(int p)
        {
                if(p<10000)
                throw “Not enough parts”;
                parts=p;
        }
};
class Garage : public Motor
{
        int tools;
        public:
        Garage(int p,int t)
        {
                try
```

```
                {
                        assign_parts(p);
                }
                catch(const char* e)
                {
                        cout<<e<<endl;
                }
                tools=0;
                if (t<10(
                throw “Not enough tools”;
                tools=t;
        }
};
int main()
{
        try{
                Garage g1(5000,15);
        }
        catch(cost char * e)
        {
                cout<<e<<endl;
        }
        try{
                Garage g2(10001,5);
        }
        catch(cost char * e)
        {
                cout<<e<<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 <iostream>
using namespace std;
class Lane
{
        int direction;
        public:
        Lane(int d)
        {
                direction=d;
        }
        int get_d()
        {
                return direction;
        }
```

```
};
int main()
{
	Lane vehicle1(1);
	Lane vehicle2(-1);
	try
	{
		if(vehicle1.get_d() != vehicle2.get_d())
		throw "Collision possible";
	}
	catch (cost char * e)
	{
		cout<<e>>endl;
	}
}
```

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 <cmath>
using namespace std;
double mod(int r,int i)
{
	return pow(r*r,i*I,0.5);
}
template <class T>
T maximum (T x, T y)
{
	if (x>y)
	return x;
	else
	return y;
}
template <>
const char* maximum (const char* x,const char* y)
{
	int x1=0, y1=0;
	while (x[x1]!='\0')
	{
		x1++;
	}
	while (y[y1]!='\0')
	{
```

```
                y1++;
        }
        if(xl>yl)
        return x;
        else
        return y;
}
int main()
{
        cout<<"Maximum of 2 integers: "<<maximum(5,6)<<endl;
        int x=1, ix=-1, y=2, iy=-2;
        cout<<"Maximum of mod of complex numbers: "<< maximum(mod(x,ix),
mod(y,iy))<<endl;
        cout<<"Maximum of 2 strings: "<< maximum("hi","hello")<<endl;
}
```

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;
class Complex
{
        float real, img;
        public:
        Complex()
        {
                real=imag=0;
        }
        Complex(float r, float i)
        {
                real = r;
                imag = i;
        }
        void disp()
        {
                cout << '(' << real << ',' << imag << ")";
        }
};
class Stack
{
        int *arr;
        int top;
        int size;
        public:
        Stack()
```

```
	{
		size = 0;
		top = -1;
	}
	Stack(int sz)
	{
		size = sz;
		arr = new int[size];
		top = -1;
	}
	void push(int x)
	{
		if (top == size - 1)
		{
			cout << "Overflow\n";
			return;
		}
		arr[++top] = x;
	}
	int pop()
	{
		if (top == -1)
		{
			return -99999;
		}
		else
		{
			return arr[top--];
		}
	}
	void disp()
	{
		int ptr = top;
		while (ptr >= 0)
		{
			cout << arr[ptr--] << " ";
		}
		cout << "\n";
	}
	void _swap(Stack, Stack);
};

template <class T>
void _swap (T &a, T &b)
{
	T temp = a;
	a = b;
	b = temp;
}
```

```
template <>
void _swap<Stack&>(Stack &s1, Stack &s2)
{
	Stack temp = s1;
	s1 = s2;
	s2 = temp;
}
int main()
{
	int x1 = 7, x2 = 10;
	cout << "Before swapping :\n";
	cout << "x1=" << x1 << "\nx2=" << x2 << "\n\n";
	 _swap(x1, x2);
	cout << "After swapping :\n";
	cout << "x1=" << x1 << "\nx2=" << x2 << "\n\n";

	Complex c1(3, -3), c2(-6, 6);
	cout << "Before swapping :\n";
	cout << "c1=";
	c1.disp();
	cout << "\nc2=";
	c2.disp();
	cout << "\n\n";

	_swap(c1, c2);

	cout << "After swapping :\n";
	cout << "c1=";
	c1.disp();
	cout << "\nc2=";
	c2.disp();
	cout << "\n\n";

	Stack s1(5), s2(6);
	s1.push(2), s1.push(3), s1.push(4), s1.push(5);
	s2.push(6), s2.push(7), s2.push(8), s2.push(9), s2.push(1);

	cout << "Before swapping :\n";
	cout << "s1: ";
	s1.disp();
	cout << "s2: ";
	s2.disp();
	_swap(s1, s2);
	cout << "After swapping :\n";
	cout << "s1: ";
	s1.disp();
	cout << "s2: ";
	s2.disp();
}
```

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>
using namespace std;
class Complex
{
        private:
        float real, imag;
        public:
         Complex(float r, float i)
        {
                real = r;
                imag = i;
        }
         void disp()
        {
                cout << '(' << real << ',' << imag << ")";
        }
};

template <class T>
class Stack
{
        private:
        T *arr;
        int top;
        int size;
        public:
        Stack(int);
        void push(T);
        T pop();
        void disp();
};

template <classT>
Stack<T>::Stack(int sz)
{
        size = sz;
        arr = (T *)malloc(size * sizeof(T));
        top = -1;
}

template <class T>
void Stack<T>::push(T x)
{
        if (top == size - 1)
```

```
	{
		cout << "Overflow\n";
		return;
	}
	arr[++top] = x;
}

template<class T>
T Stack<T>::pop()
{
	if (top == -1)
	{
		return -99999;
	}
	else
	{
		return arr[top--];
	}
}

template <>
Complex Stack<Complex>::pop()
{
	if (top == -1)
	{
		return Complex(-99999, -99999);
	}
	else
	{
		return arr[top--];
	}
}

template <class T>
void Stack<T>::disp()
{
	cout << "Contents of stack:\n";
	int ptr = top;
	while (ptr >= 0)
	{
		cout << arr[ptr--] << " ";
	}
	cout << "\n";
}

template <>
void Stack<Complex>::disp()
{
	cout << "Contents of stack:\n";
	int ptr = top;
	while (ptr >= 0)
```

```
        {
                arr[ptr--].disp();
                cout << " ";
        }
        cout << "\n";
}

int main()
{
        Stack st1(10);
        st1.push(1);
        st1.push(2);
        st1.push(3);
        st1.push(4);
        int d = st1.pop();
        st1.push(5);
        st1.disp();
        d = st1.pop();
        d = st1.pop();
        st1.push(6);
        st1.disp();

        Stack<Complex>  st2(10);
        st2.push(Complex(1, 2));
        st2.push(Complex(2, -1));
        st2.push(Complex(3, -4));
        st2.push(Complex(1, 4));
        st2.push(Complex(-6, 1));
        Complex c = st2.pop();
        st2.disp();
        c = st2.pop();
        st2.push(Complex(9, -2));
        st2.disp();
}
```