(EXCEPTION HANDLING)

(MADE BY ALI AKBER)

(BSCS 2ND SS1)

/// Input Mismatch.

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


int main()
{
	int num;
	try
	{
		cout<<"Enter no ";
		cin>>num;
		if(cin.fail())
			throw "Input must be integer";
		cout<<"No is "<<num<<endl;
	}
	catch(const char *msg)
	{
	     cout<<msg<<endl;
	}
	cout<<"Hello class"<<endl;
	cout<<"C++ exception"<<endl;
```

```
    return 0;
}
```

/// Input Mismatch.

///Runtime_error.

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

int main()
{
    int num;
    try
    {
        cout<<"Enter no ";
        cin>>num;
        if(cin.fail())
            throw runtime_error("input must be integer");
        cout<<"No is "<<num<<endl;
    }
    catch(runtime_error& e)
    {
       cout<<"Error..."<<e.what()<<endl;
    }

    cout<<"Hello class"<<endl;
    cout<<"C++ exception"<<endl;
```

```
return 0;
}
```

//Input Mismatch + denominator must be > Zero

/// C++ Exceptions

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

int main()
{
	int no1, no2;
	try
	{
		cout<<"Enter No-1 ";
		cin>>no1;
		if(cin.fail())
			throw runtime_error("No-1 must be integer");
		cout<<"Enter No-2 ";
		cin>>no2;
		if(cin.fail())
			throw runtime_error("No-2 must be integer");
		if(no2 == 0)
			throw runtime_error("/ by zero");
		cout<<"Division result is "<<(float) no1/no2<<endl;
	}
	catch(runtime_error& e)
```

```
    {
        cout<<"Error....."<<e.what()<<endl;
    }
    cout<<"Hello Class"<<endl;
    cout<<"C++ Exceptions"<<endl;

    return 0;
}
```

/// Stack Class with build in Exception

```
#include <iostream>
#include <stdlib.h>
#include <stdexcept>
using namespace std;

class Stack
{
    private:
        int arr[5];
        int top;
    public:
        Stack():top(-1){}
        void Push(int var)
        {
            if(top >= 4)
            {
                cout<<"Stack overflow"<<endl;
```

```
            exit(1);
        }
        arr[++top] = var;
    }
    int pop()
    {
        if(top == -1)
        {
            cout<<"Stack under flow"<<endl;
            exit(1);
        }
        return arr[top--];
    }
};
int main()
{
    Stack s;
        s.Push(11);
        s.Push(12);
        s.Push(13);
        s.Push(14);
        s.Push(15);
        //s.Push(16);
        cout<<"value is "<<s.pop()<<endl;
        cout<<"value is "<<s.pop()<<endl;
        cout<<"value is "<<s.pop()<<endl;
```

```
        cout<<"value is "<<s.pop()<<endl;
        cout<<"value is "<<s.pop()<<endl;
        ///cout<<"value is "<<s.pop()<<endl;
    cout<<"Bye Bye"<<endl;
    return 0;
}
```

/// Stack Class with build in Exception

///Runtime_error.

```
#include <iostream>
#include <stdlib.h>
#include <stdexcept>
using namespace std;

class Stack
{
    private:
        int arr[5];
        int top;
    public:
        Stack():top(-1){}
        void Push(int var)
        {
            if(top >= 4)
                throw runtime_error("Stack overflow");
            arr[++top] = var;
        }
```

```
        int pop()
        {
            if(top == -1)
                throw runtime_error("Stack under flow");
            return arr[top--];
        }
};
int main()
{
    Stack s;
    try
    {
        s.Push(11);
        s.Push(12);
        s.Push(13);
        s.Push(14);
        s.Push(15);
        ///s.Push(16);
        cout<<"value is "<<s.pop()<<endl;
        cout<<"value is "<<s.pop()<<endl;
        cout<<"value is "<<s.pop()<<endl;
        cout<<"value is "<<s.pop()<<endl;
        cout<<"value is "<<s.pop()<<endl;
        cout<<"value is "<<s.pop()<<endl;
    }
    catch(runtime_error& e)
```

```
	{
		cout<<"Error..."<<e.what()<<endl;
	}
	cout<<"Bye Bye"<<endl;
	return 0;
}
```

/// Stack Class with build in Exception (Inherited)

```
#include <iostream>
#include <stdlib.h>
#include <stdexcept>
using namespace std;

class Stack : runtime_error
{
	private:
		int arr[5];
		int top;
	public:
		Stack(): runtime_error(""), top(-1){}
		void Push(int var)
		{
			if(top >= 4)
			{
				throw runtime_error ("Stack overflow");
			}
			arr[++top] = var;
```

```
        }
        int pop()
        {
            if(top == -1)
            {
                throw runtime_error ("Stack under flow");
            }
            return arr[top--];
        }
};
int main()
{
    Stack s;
    try
    {
        s.Push(11);
        s.Push(12);
        s.Push(13);
        s.Push(14);
        s.Push(15);
        s.Push(16);
        cout<<"value is "<<s.pop()<<endl;
        cout<<"value is "<<s.pop()<<endl;
        cout<<"value is "<<s.pop()<<endl;
        cout<<"value is "<<s.pop()<<endl;
        cout<<"value is "<<s.pop()<<endl;
```

```
        ///cout<<"value is "<<s.pop()<<endl;
    }
    catch(runtime_error& e)
    {
        cout<<"Error..."<<e.what()<<endl;
    }
    cout<<"Bye Bye"<<endl;
    return 0;
}
```

/// Distance Class with build in Exception.

//Insertion and extraction operator overloading.

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

class Distance
{
    private:
        int feets;
        float inches;
    public:
        Distance(): feets(0),inches(0.0f){}
        friend istream& operator>>(istream&, Distance&);
        friend ostream& operator<<(ostream&,Distance&);

        Distance& operator / (const Distance& d)
```

```
        {
            if(d.feets == 0 || d.inches == 0.0f)
                throw "/ by zero (Denominator)";

            feets = feets / d.feets;
            inches = inches / d.inches;
            return *this;
        }
};
istream& operator >>(istream& in, Distance& d)
{
    cout<<"Enter Feets ";
    in>>d.feets;
    if(cin.fail())
        //throw runtime_error("Feets must be Integer ");
        throw "Feets must be Integer";

    cout<<"Enter Inches ";
    in>>d.inches;
    if(cin.fail())
        //throw runtime_error("Feets must be Integer ");
        throw "Inches must be Float";
}

ostream& operator << (ostream& out, Distance& d)
{
```

```
    out<<"Distance is "<<d.feets<<'\''<<d.inches<<'\"'<<endl;
}

int main()
{
    try
    {
        Distance d1, d2, d3;
        cin >> d1;
        cin >> d2;
        d3 = d1 / d2;

        cout<<"\nDivision result is "<<endl;
        cout << d3;
    }
    catch(const char* msg)
    {
        cout<<"Error....."<<msg<<endl;
    }

    return 0;
}
```

///EMPOLYEE CLASS WITH BUILD IN EXCEPTION.

//INSERTION AND EXTRACTION OPERATOR OVERLOADING.

///FRIEND FUNCTION.

/// EXCEPTION HANDLING IN MULTIPLE INHERITANCE.

```
#include <iostream>
#include <stdexcept>
#include <string.h>
using namespace std;
class Empolyee
{
private:
        char name[50];
        int id;

public:
        Empolyee() : id(0) { strcpy(name, "NULL"); }                    // no arg constructor.
        Empolyee(int i, char na[]) : id(i) { strcpy(name, na); } // two arg constructor.
        friend istream &operator>>(istream &in, Empolyee &e);
        friend ostream &operator<<(ostream &out, Empolyee &e);
};

    istream &operator>>(istream &in, Empolyee &e)
        {
                cout << "Enter Empolyee name:" << endl;
                cin >> e.name;
                in.ignore();
                cout << "Enter Empolyee id:" << endl;
                in >> e.id;
                    in.ignore();
                  if (cin.fail())
```

```
            throw runtime_error("Input must be character");
        return in;
    }

    ostream &operator<<(ostream &out, Empolyee &e)
    {
        out << "Empolyee name is:" << e.name << endl;
        out << "Empolyee id is:" << e.id << endl;
        return out;
    }
class Student
{
private:
    string university;
    string degree;

public:
    Student() : university(""), degree(){};
    Student(string uni, string deg) : university(uni), degree(deg){};
    friend istream &operator>>(istream &in, Student &s);
    friend ostream &operator<<(ostream &out, const Student &s);
};

istream &operator>>(istream &in, Student &s)
    {
        cout << "Enter the University in which student read:" << endl;
```

```
        getline(in, s.university);
        cin.ignore();
        cout << "Enter the Degree earned by the student:" << endl;
        getline(in, s.degree);
        in.ignore();
        return in;
    }

     ostream &operator<<(ostream &out, const Student &s)
    {
        out << "University In which student read is:" << s.university << endl;
        out << "Degree which the student earned is:" << s.degree << endl;
        return out;
    }
class Manager : private Empolyee, private Student
{
private:
    string title;
    double dues;

public:
    Manager() : Empolyee(), Student(), title(""), dues(0) {} // no arg constructor.
    friend istream &operator>>(istream &in, Manager &m);
    friend ostream &operator<<(ostream &out, const Manager &m);
};
```

```
istream &operator>>(istream &in, Manager &m)
	{
		cout << "Enter Manager title: " << endl;
		getline(in, m.title);
		in.ignore();
		cout << "Enter Manager dues:" << endl;
		in >> m.dues;
		in.ignore();
		  if (cin.fail())
		  throw runtime_error("Input must be character");
		return in;
	}

	ostream &operator<<(ostream &out, const Manager &m)
	{
		out << " Manager title is:" << m.title << endl;
		out << "Manager dues is:" << m.dues << endl;
		return out;
	}
class Scientist : private Empolyee, private Student
{
private:
	int publications;

public:
	Scientist() : Empolyee(), Student(), publications(0) {} // no arg constructor.
```

```
    friend istream &operator>>(istream &in, Scientist &s);
    friend ostream &operator<<(ostream &out, Scientist &s);
};

istream &operator>>(istream &in, Scientist &s)
    {
        cout << "Enter Scientist Publications :" << endl;
        in >> s.publications;
          in.ignore();
          int var=0;
        if (s.publications>10){
       cout<<"Publications overloaded";
        }
       else
       var=s.publications;
        return in;
    }

    ostream &operator<<(ostream &out, Scientist &s)
    {
        out << "Scientist Publications are:" << s.publications << endl;
        return out;
    }
class Laborer : public Empolyee
{
};
```

```
int main()
{
	try
	{
		Empolyee e1;
		cin >> e1;
		cout << e1;
		cout<<endl;
	}
	catch (runtime_error &e)
	{
		cout << "Error....." << e.what() << endl;
	}
	Manager m;
	Scientist s1, s2;
	Laborer l;
	try{
	  cout << "Enter data of the manager:" << endl;
	cin >> m;
	cout << endl;
	cout << "Manager data is as follows:" << endl;
	cout << m << endl;
	cout<<endl;
	}
      catch (runtime_error &e)
	{
```

```
        cout << "Error....." << e.what() << endl;
    }
    try{
cout << "Enter data of the ist scientist :" << endl;
    cin >> s1;
    cout << endl;
    cout << "Ist scientist data is as follows:" << endl;
    cout << s1 << endl;

    cout << "Enter data of the 2nd scientist :" << endl;
    cin >> s2;
    cout << endl;
    cout << "2nd scientist data is as follows:" << endl;
    cout << s2;
    cout << endl;
    }
      catch (runtime_error &e)
    {
        cout << "Error....." << e.what() << endl;
    }

    cout << "Enter data of the laborer:" << endl;
    cin >> l;
    cout << endl;
    cout << "Laborer data is as follows:" << endl;
    cout << l;
```

```
	cout << endl;


	return 0;
}
```

//USER DEFINED EXCEPTION HANDLING.

//DIVIDE BY ZERO CLASS.

```
#include<iostream>
#include<stdexcept>
#include<string.h>
using namespace std;
class DivideByZero : public runtime_error
{
	public:
		DivideByZero(char ch[]) : runtime_error(ch) {}
};
int main()
{
	try
	{
		int num,dnum;
		cout<<"Enter num ";
		cin>>num;
		cout<<"Enter D-num ";
		cin>>dnum;
		if(dnum == 0)
		{
```

```
                throw DivideByZero("/ by zero");
            }
            cout<<"Result :"<<(float) num / dnum<<endl;
        }
        catch(DivideByZero& d)
        {
            cout<<"Error... "<<d.what()<<endl;
        }
        return 0;
}
```

/// Stack Class with user define Exception.

```
#include <iostream>
#include <stdlib.h>
#include <stdexcept>
using namespace std;

class StackException : public runtime_error
{
    public:
        StackException(char ch[]) : runtime_error(ch) {}
};

class Stack
{
    private:
        int arr[5];
```

```
        int top;
    public:
        Stack():top(-1){}
        void Push(int var)
        {
            if(top >= 4)
                throw StackException("Stack overflow");

            arr[++top] = var;
        }
        int pop()
        {
            if(top == -1)
                throw StackException("Stack underflow");

            return arr[top--];
        }
};
int main()
{
    Stack s;
    try
    {
        s.Push(11);
        s.Push(12);
        s.Push(13);
```

```
            s.Push(14);
            s.Push(15);
            //s.Push(16);
            cout<<"value is "<<s.pop()<<endl;
            cout<<"value is "<<s.pop()<<endl;
            cout<<"value is "<<s.pop()<<endl;
            cout<<"value is "<<s.pop()<<endl;
            cout<<"value is "<<s.pop()<<endl;
            cout<<"value is "<<s.pop()<<endl;
        }
        catch(StackException& s)
        {
            cout<<"Error....."<<s.what()<<endl;
        }

        cout<<"Bye Bye"<<endl;
        return 0;
}
```

///User Define Exception Handling class.

///CLASS WITHIN THE CLASS.

```
#include <iostream>
#include <stdlib.h>
#include <stdexcept>
using namespace std;

class Stack
```

```
{
    private:
        int arr[5];
        int top;
    public:

        class StackException : public runtime_error
        {
            public:
                StackException(char ch[]) : runtime_error(ch) {}
        };

        Stack():top(-1){}
        void Push(int var)
        {
            if(top >= 4)
                throw StackException("Stack overflow");

            arr[++top] = var;
        }
        int pop()
        {
            if(top == -1)
                throw StackException("Stack underflow");

            return arr[top--];
```

```
        }
};

int main()
{
    Stack s;
    try
    {
        s.Push(11);
        s.Push(12);
        s.Push(13);
        s.Push(14);
        s.Push(15);
        //s.Push(16);
        cout<<"value is "<<s.pop()<<endl;
        cout<<"value is "<<s.pop()<<endl;
        cout<<"value is "<<s.pop()<<endl;
        cout<<"value is "<<s.pop()<<endl;
        cout<<"value is "<<s.pop()<<endl;
        cout<<"value is "<<s.pop()<<endl;
    }
    catch(Stack::StackException& s)
    {
        cout<<"Error....."<<s.what()<<endl;
    }
```

```
	cout<<"Bye Bye"<<endl;
	return 0;
}
```

/// Stack Class with user define Exception.

///MULTIPLE EXCEPTION CLASSES AND CATCH BLOCK.

```
#include <iostream>
#include <stdlib.h>
#include <stdexcept>
using namespace std;

class IsEmpty
{
};

class IsFull
{
};



class Stack
{
	private:
		int arr[5];
		int top;
	public:
		Stack():top(-1){}
```

```
        void Push(int var)
        {
            if(top >= 4)
                throw IsFull();

            arr[++top] = var;
        }
        int pop()
        {
            if(top == -1)
                throw IsEmpty();

            return arr[top--];
        }
};
int main()
{
    Stack s;
    try
    {
        s.Push(11);
        s.Push(12);
        s.Push(13);
        s.Push(14);
        s.Push(15);
        //s.Push(16);
```

```
            cout<<"value is "<<s.pop()<<endl;
            cout<<"value is "<<s.pop()<<endl;
            cout<<"value is "<<s.pop()<<endl;
            cout<<"value is "<<s.pop()<<endl;
            cout<<"value is "<<s.pop()<<endl;
            cout<<"value is "<<s.pop()<<endl;
        }
        catch(IsEmpty& e)
        {
            cout<<"Error.....Stack underflow"<<endl;
        }
        catch(IsFull& e)
        {
            cout<<"Error.....Stack overflow"<<endl;
        }

        cout<<"Bye Bye"<<endl;
        return 0;
}
```

ASSIGNMENT QUESTION.

Create a Professor class that has data members to holds the Id (int), name (string) and Pub (int).

Class also includes parameterized constructors and overloaded insertion (<<) and extraction (<<) operators' that displays and get all fields of class Professor.

Create an ProException class that holds EstimPub (type int). When the user enters Professor data,

if the pub is below then 10, then throw an ProException object with an appropriate message (Pass this String to the ProException's parent so it can be used in a what () call).

Write a main () function that instantiates a Professor object, allows the user to enter data,

and displays the data members.

///ASSIGNMENT QUESTION.

///USER DEFINED EXCEPTIONAL HANDLING.

///FRIEND FUNCTIONS.

///INSERTION AND EXTRACTION OPERATOR OVERLOADING.

```
#include<iostream>
#include<stdexcept>
#include<string.h>
using namespace std;
class ProException: public runtime_error{
    public:
    int EstimPub;
      ProException(string message,int EP):runtime_error(message),EstimPub(EP){}
};
class Professor{
      private:
      int id;
      string name;
      int pub;
      public:
      Professor():id(0),name(""),pub(0){}    //no arg constructor.
      Professor(int i,string n,int p):id(i),name(n),pub(p){}    //three arg constructor.
```

```
        friend istream& operator>>(istream& in,Professor &p);
        friend ostream& operator<<(ostream& out,Professor &p);
};
  istream& operator>>(istream& in,Professor &p){
        cout<<"Enter id of the professor:"<<endl;
        in>>p.id;
        cout<<"Enter name of the professor:"<<endl;
        in>>p.name;
        in.ignore();
        cout<<"Enter publications of the professor:"<<endl;
        in>>p.pub;
        if (p.pub<10)
        throw ProException("PUBLICATIONS ARE LESS THAN 10!!!!",p.pub);
        return in;
  }
  ostream& operator<<(ostream& out,Professor &p){
        out<<"Professor id is:"<<p.id<<endl;
        out<<"Professor name is:"<<p.name<<endl;
        out<<"Professor publications are:"<<p.pub<<endl;
        return out;
  }
  int main(){
     try{
        Professor p;
     cin>>p;
     cout<<p;
```

```
    }
  catch(ProException &s){
    cout<<"Note....."<<s.what()<<endl;
  }
  cout<<"PROGRAM ENDED....."<<endl;
    return 0;
}
```