In this lab we will learn how to throw exceptions, how to catch exceptions, and how to declare one class inside another.

If a function throws an exception, it will execute a statement like the following:

```
throw <constant>;
```

Where <constant> can be a constant of any type, including integers, floats, character strings (char *), and classes. When a class is used to throw an exception, it is usually called a *exception class*, and is used for no other purpose. It is common practice to declare exception classes inside of other classes.

An exception can be caught or ignored. If it is ignored, the program will normally be terminated when it occurs. An exception is caught by using the following construct.

```
try
{
        functions or other statements that might throw exceptions.
}
catch (char * message)
{
        exception handler for char * (value will be used)
}
catch (int x)
{
        exception handler for int (value will be used)
}
catch (MyClass)
{
        exception handler for the exception class MyClass.
        (value will be ignored)
}
```

Create a program called “Exceptions.cpp”.

In your program create three classes, exList, MonsterTruck, and Breakable. The class exList will be a container class for four exception classes named Ex1, Ex2, Ex3, and Ex4. Declare this class as follows:

```
class exList
{
        public:
                class Ex1 {};
                class Ex2 {};
                class Ex3 {};
                class Ex4 {};
};
```

Note that exception classes are very often empty classes. A constant of type `exList::Ex1()` is used to throw an exception of type Ex1 from class exList.

MonsterTruck will be an exception class with two public members, int a, and int b. MonsterTruck will have a default constructor that initializes a and b to zero and an int,int constructor that initializes both a and b to user-specified values. No other functions are required for this class.

The class Breakable will be a regular class with three public data items int a,b,c, and two public functions int func1(int x) and int func2(int x). No other functions are required for this class. The function func1 will have the body:

```
cout<<"func1 Body\n";
a=x;
b=x+1;
return a+b;
```

The function func2 will have the body:

```
cout<<"func2 Body\n";
c=x;
b=x+1;
return b+c;
```

The class Breakable will have two internal exception classes, Ex1 and Ex2, defined like Ex1 and Ex2 of exList. (It is OK for them to have the same names.)

The function func1 of class Breakable will throw Ex1( ) if its argument is equal to 7, exList::Ex1( ) if its argument is equal to 8, and exList::Ex2( ) if its argument is equal to 9.

The function func2 of class Breakable will throw Ex2( ) if its argument is equal to 9, “Hey! Stop That!” if its argument is equal to 10, and MonsterTruck(2,3) if its argument is equal to 11.

The main routine of the program will declare a variable B of type Breakable, and will call B.func1( ) and B.func2( ) 12 times with arguments from 0 through 11, using the following loop.

```
for (k=0 ; k<12 ; k++)
{
        cout<<k<<endl;
        try
        {
                B.func1(k);
                B.func2(k);
        }
        catch …
}
```

Start with five catch blocks, one for char *, which prints the character string from the exception on a line by itself, one for Breakable::Ex1 which prints “Breakable::Ex1” on a line by itself, one for Breakable::Ex2 which prints “Breakable::Ex2” on a line by itself, one for exList::Ex2, which prints “exList::Ex2” on a line by itself, and one for MonsterTruck x that prints “MonsterTruck(x.a,x,b)” on a line by itself. Note that the actual values returned in the MonsterTruck class must replace the values of x.a and x.b in the printout.

Run your program and print the output to turn in. There is an uncaught exception in Breakable::func1(int). If you can ignore this exception and continue, do so. If you can’t get the printout without catching the exception, add a catch block for exList::Ex1 that does nothing.

Print out your program and the output and turn in both.

Note from the printout that when func1 produces an exception that the body of func1 is skipped, and func2 is not called. Note that when func2 produces an exception (and func1 does not) the body of func2 is skipped.