CIS 330: Project #3F
Assigned: May 20th, 2018
Due May 27th, 2018
(which means submitted by 6am on May 28th, 2018)
Worth 3% of your grade

Please read this entire prompt!

Assignment:

Add logging and exception handling to your code.

Please note: a new main3F.C and a new Makefile are on the class page. I also added a header file called "logger.h" that has the base class for logging and exceptions.

== Logging ==

Add logging to your code. This means that your program will generate a log file, and generate log events as it executes. The log events will detail your program execution. The output of my log file is posted online. Your file does not need to match mine exactly, but the exceptions should match up. (It might be a good idea to match exactly, for what it is worth, since it helps you find subtle errors.)

There are two parts to logging:
(1) adding infrastructure for logging
(2) making your data flow code use that infrastructure

== adding infrastructure for logging ==

Check out the class Logger in the file logging.h. You need to add methods associated with that class.

== making your data flow code use that infrastructure ==
You will log an event by calling "Logger::LogEvent(msg);" with a "char *" named "msg".

For example, my routine:

```
void
Source::Update()
{
   Execute();
}
```

became:

```
void
```

```
Source::Update()
{
   char msg[128];
   sprintf(msg, "%s: about to execute", SourceName());
   Logger::LogEvent(msg);
   Execute();
   sprintf(msg, "%s: done executing", SourceName());
   Logger::LogEvent(msg);
}
```

You can see exactly where I put my logging code in the file “logger”.

== Exception handling ==

There are two parts to the exception handling:
(1) adding infrastructure for exception handling
(2) making your data flow code use that infrastructure

== adding infrastructure for exception handling ==

There is really only one thing to do here: add a constructor.

You will need to implement a constructor for DataFlowException that stores the exception in the “msg” data member.

If the constructor gets type = “Shrinker” and error = “bad sizes”, then it should make msg be equal to: “(Shrinker): bad sizes”. For what it is worth, my constructor also logs an event.

== making your data flow code use that infrastructure ==

There are many spots where you can add exception handling in your data flow code. My driver program only tests three of them.

Here is my code that checks for and throws exceptions

```
   if (input->GetWidth() != input2->GetWidth())
   {
      char msg[1024];
      sprintf(msg, "%s: widths must match: %d, %d", SinkName(), input->GetWidth(),
input2->GetWidth());
      DataFlowException e(SinkName(), msg);
      throw e;
   }
```

IMPORTANT: the final “stress test” project at the end of the term can include programs that should cause exceptions. So adding as many error checks as possible now is good.

Hint: it is convenient to add the following pure virtual functions:

```
const char *Source::SourceName() = 0;
const char *Sink::SinkName() = 0;
const char *Filter::FilterName() = 0;
```

they make coding faster in several spots. (Note my code above uses SinkName)

(also, define:

```
const char *Filter::SourceName() { return FilterName(); };
const char *Filter::SinkName() { return FilterName(); };
```

)

== “Turnin” ==

Brent and I will not be grading 3F and 3G in the normal way. Instead, they will be graded as we grade 3H. It is not possible to do 3H without 3F and 3G. So if your 3F and 3G contributions work sufficiently well for your 3H to work, then you will get full credit for 3F and 3G.

This means:
- there is no grader.sh for 3F or 3G
- you do not need to upload anything
- nothing will be "late" for these projects
- the grades will be assigned for these projects when your 3H grade is assigned