What I learned about exceptions is that they have certain functions which are used to catch errors and report them. They are very useful when working on a project that has two or more people working on it and work by using a try block which checks if the data is correct if not then the catch statements catches the error. Before I go off and make my own exception classes the standard library already comes with some useful exception classes i.e. the assert. Last I learned that you can use an exception to tell when a constructor failed to construct however you should not do this in a destructor as this will create another exception and cause the program to freeze.

The section on namespace was really neat because it showed me how to build my own namespaces. It talked about how you can call two classes that have the same name by referencing them with their namespace i.e., using the example from chapter 18, std::map and acme::map. Though similar in name, these are two different classes because they come from two different libraries thus allowing a programmer to use them in the same program without errors occurring. To initialize a new namespace you set it up like a class however the brackets are opened {} allowing a programmer to add however many items he or she wants to. However when you build a new namespace you should make its name very long and unusual i.e. this\_is\_an\_example\_namespace. There is a slight problem giving it such a name and that is programmers won’t be happy constantly typing it out when they need to use it so to remedy this issue you can set the namespace to an alias case in point namespace example = this\_is\_an\_example\_namespace. If you decide to put the namespaces in a header file it is not good practice to have them already set to specific aliases because it leaves future programmers stuck with these aliases. Your best bet is to simply put them into the header file and let the next programmer build his or her own aliases. Last, namespaces are subject to the same rules as scopes. Basically meaning that local takes priority over global variables. The example in the book best describes what I mean:

using namespace std; // Allows stack to mean the standard library

void f(double data)

{

using acme::stack; // ACME version now shadows std::stack

...

}

This great for when you only need to use a specific library that has similar variables to those of the standard without having go through the entire program and state which is which.