Error Flags

With the advent of object-oriented programming, a variation on completion codes was birthed—**error state** which is encapsulated in objects. Of course, you've already encountered this with the input stream classes.

Here's an example. What happens if the user enters *twelve*?

```
cout << "Enter an integer: ";
int n;
cin >> n;    // Error state is set here
```

Each stream object has an **internal data member** that contains an individual error code, or **error flag**. These flags are given names like `badbit`, `goodbit` and `failbit`. If the user enters *twelve*, then the `failbit` is "set". If the keyboard isn't working, the `badbit` is set.

In the C-style of programming, you use **bitwise logical operators** (something we won't cover in this class, but you'll probably encounter in Computer Architecture) to read or set each of these error codes. In C++, however, you have **member functions**:

```
cin >> n;    // Error state may be set here
if (cin.fail()) // Check if failbit is set
{
    cin.clear(); // clear all of the error flags
    // empty the input stream and try again
}
```

The big problem with completion codes and with error states, is that **you can ignore the return value without encountering any warnings**. Research has shown that programmers almost **never** check them. To better handle these kinds of problems, C++ introduced **exception handling**. If an error occurs inside a function, rather than returning a value, you report the problem and **jump to the proper error-handling code**.