Validating Data

When you read a value from `cin`, it is possible that the input may fail because the user entered invalid data. For instance:

```
cout << "Enter a number: ";
int n;
cin >> n;
cout << n << endl;
```

Suppose that the user types in `one` when asked to enter a number. Here's what happens:

1. The `cin` object enters a **failed** state and will stop accepting any more input.
2. The variable `n` will be set to `0`.

You can check for success by calling the member function `fail()` or by simply using a regular *`if`* statement. Here's a fragment that shows how to use *`if`*:

```
int n;
if (cin >> n) { cout << n << endl; }
else { cout << "Invalid input" << endl; }
```

And, here's a fragment which explicitly calles the `fail()` member function:

```
int n;
cin >> n;
if (cin.fail()) { cout << "Invalid input" << endl; }
else  { cout << n << endl; }
```

Recovering

Inside a sentinel loop, you'd like to **recover** if the user inadvertently entered bad data.

1. Call `cin.clear()` to allow `cin` to start accepting data once again.
2. **Consume** the bad data by creating a `string` variable and reading it.

```
while (true)                        // Endless loop
{
    cout << "> ";                   // Prompt and read item
    if (cin >> value) {
        if (value == 0) { break; }  // Sentinel? Leave loop
        total += value;             // No sentinel? Process
    }
    else {
        cin.clear();                // Clear the fail flag
        string bad_data;            // store the bad data
        in >> bad_data;             // read it and ignore it
    }
}
```