Streams II

•••

Original from : Mike Precup (mprecup@cs.stanford.edu)
ENJMIN Edition 2016

Quick Recap

- Streams are a general way to handle I/O
- Most frequent ways to access data are getline, >>, <<, and get
- Can be an istream, ostream, or iostream

Today

- Stream Internals
- Stream Shortcuts
- Stream Manipulators

ostream Internals

- We glossed over using ostreams last time
- It turns out ostreams also have an internal sequence of data!

Stream Buffers

- The internal sequence of data stored in a stream is called a **buffer**
- istreams use them to store data we haven’t used yet
- ostreams use them to store data they haven’t passed along yet
    - Depends on the implementation used

Flushing

If you want to force the buffer to get used, you **flush** the stream:

- `stream.flush()`
  - Use by default
- `stream << flush`
  - Use if you're already printing on that line without a newline
- `flush(stream)`
  - Just don't use this
- `stream << endl`
  - Use if you're printing a newline

stream.fail()

What exactly does stream.fail() actually tell us?

Stream Bits

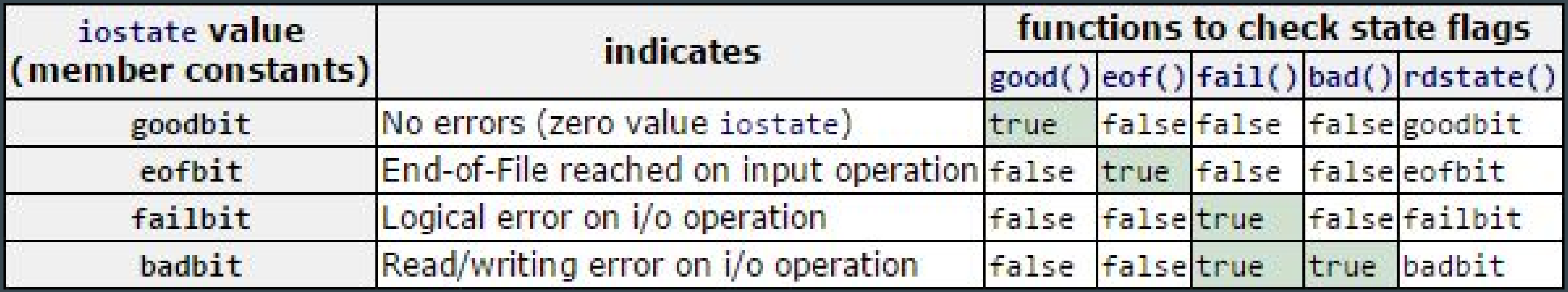
Has four bits:

- Good bit - No errors, the stream is good to go
- EOF bit - End-of-file was reached during a previous operation
- Fail bit - Logical error on a previous operation
- Bad bit - Likely unrecoverable error on previous operation

Stream Bits

Has four bits:

- Good bit - No errors, the stream is good to go
- EOF bit - End-of-file was reached during a previous operation
- Fail bit - Logical error on a previous operation
- Bad bit - Likely unrecoverable error on previous operation



iostate value (member constants)	indicates	functions to check state flags				
		good()	eof()	fail()	bad()	rdstate()
goodbit	No errors (zero value iostate)	true	false	false	false	goodbit
eofbit	End-of-File reached on input operation	false	true	false	false	eofbit
failbit	Logical error on i/o operation	false	false	true	false	failbit
badbit	Read/writing error on i/o operation	false	false	true	true	badbit

Which Bit is Best?

1. Read data
2. Check if data is valid, if not, break
3. Use data
4. Go back to step 1

Which Bit is Best?

1. Read data
2. Check if data is valid, if not, break
3. Use data
4. Go back to step 1

```
while(true) {

    stream >> temp;

    if (stream.fail()) break;

    foo(temp);

}
```

Clear

- `stream.clear()` sets the stream's bit to `goodbit`
- You must do this if you want to continue using a stream that isn't in a good state

Today

- Stream Internals
- Stream Shortcuts
- Stream Manipulators

Chaining << and >>

We’ve been writing code like:

```
    cout << “Hello World!” << endl;
```

Why can we chain together multiple << operators?

Operator Returns

C++ treats operators as functions, and they thus have return types

What are the return types for the following?

- int + int
- int / double
- ostream << int

Chaining << and >>

If we take our hello world line and parenthesize it, we get the equivalent:

```
(cout << “Hello World!”) << endl;
```

Chaining << and >>

If we take our hello world line and parenthesize it, we get the equivalent:

```
    (cout) << endl;
```

Converting the Stream

```
int x = 0;

double y = x;  // Converted to a double implicitly
```

Converting the Stream to a …bool?

```
int x = 0;

double y = x;  // Converted to a double implicitly

bool z = cout; // Converted to a bool implicitly
```

Converting the Stream to a …bool?

```
int x = 0;

double y = x;  // Converted to a double implicitly

bool z = cout; // Converted to a bool implicitly

bool isGood = !cout.fail(); // Equivalent to the above line
```

Using Our New Tools

1. Read data
2. Check if data is valid, if not, break
3. Use data
4. Go back to step 1

```
while(true) {

    stream >> temp;

    if (stream.fail()) break;

    foo(temp);

}
```

Using Our New Tools

1. Read data
2. Check if data is valid, if not, break
3. Use data
4. Go back to step 1

```
while(true) {

    stream >> temp;

    if (!stream) break;

    foo(temp);

}
```

Using Our New Tools

1. Read data
2. Check if data is valid, if not, break
3. Use data
4. Go back to step 1

```
while(true) {

    bool isGood = stream >> temp;

    if (!isGood) break;

    foo(temp);

}
```

Using Our New Tools

1. Read data
2. Check if data is valid, if not, break
3. Use data
4. Go back to step 1

```
while(stream >> temp) {

    foo(temp);

}
```

Today

- Stream Internals
- Stream Shortcuts
- Stream Manipulators

How Does endl Work?

- A brief overview:
  - endl is actually something called a **stream manipulator**
  - If you check the type of endl, it’s actually a function

`stream << endl` is equivalent to `endl(stream)`

Mixing >> and getline

```
int age;
string name;
cin >> age;
getline(cin, name);
```

Doesn’t do what we want!

Mixing >> and getline

```
int age;
string name;
cin >> age >> ws;
getline(cin, name);
```

Does do what we want!

Common Stream Manipulators

- endl: inserts a newline and flushes the stream
- ws: skips all currently available whitespace
- boolalpha: prints “true” and “false” for bools
- Numeric:
  - hex: prints numbers in hex
  - setprecision: adjusts the precision numbers print with
- Padding:
  - setw
  - setfill

Padding Example

```
cout << “[“ << setw(10) << “Hi!” << “]” << endl;
```

outputs

```
[       Hi!]
```

Padding Example

```
cout << “[“ << right << setw(10) << “Hi!” << “]” << endl;
```

outputs

```
[Hi!       ]
```

Padding Example

```
cout << “[“ << right << setfill(‘!’) << setw(10)
     << “Hi!” << “]”  << endl;
```

outputs

```
[Hi!!!!!!!!!]
```

Numeric Example

```
cout << hex << 10; //prints a
cout << oct << 10; //prints 12
cout << dec << 10; //prints 10
```

Stream Manipulators Recap

- Stream manipulators are things you can pass into streams to change how they behave
- They have a variety of uses, and if you'd like to format something differently, there's probably a manipulator for it
- The most important are probably `endl` and `ws`
- You can find a list of the most common at http://www.cplusplus.com/reference/library/manipulators/