Stream Manipulation and Other I/O

CS 211

Adam Sweeney

November 8, 2017

Wichita State University, EECS

Introduction

- Discuss in a bit more detail stream output editing and manipulations

Agenda

- Remainder of Chapter 6
- Formatting stream output
- Streams as arguments to functions
- Character I/O
- Default parameters in functions

Formatting Stream Output

Formatting Stream Output

```
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
```

- We hopefull recognize the above code as something that was provided earlier in the semester as a means of ensuring that we print out a monetary value correctly
- We were invoking class member functions of the stream object cout in order to manipulate the stream
- This lecture will cover stream manipulation in more detail

`setf()`

- We should be able to at least deduce that `setf()` is a class member function
- It stands for "set flag"
- A flag is a binary indicator on whether we are doing a thing, or not doing a thing
- When we set a flag, we are saying that now we are going to do a thing

So What Flags Were We Setting?

```
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
```

- Because `setf()` is only called twice, we are only setting two flags
- The first flag, `fixed` denotes that we will never use scientific notation, but always use traditional fixed floating point numbers
- The second flag indicates that we will always show the decimal point
- The third item is not a flag (we did not call `setf()`), but it still changes the stream to only show two decimal digits

There Are Other Flags

Flag	Description
`ios::fixed`	Will never use scientific notation
`ios::scientific`	Always uses scientific notation
`ios::showpoint`	Always shows decimal point
`ios::showpos`	Show '+' in front of positive numbers
`ios::hex`	Displays numbers in hexadecimal
`ios::oct`	Displays numbers in octal
`ios::dec`	Displays numbers in decimal
`ios::showbase`	Shows what base system a number is
`ios::left`	Left aligns text within a field
`ios::right`	Right aligns text within a field

Notes on the Flags

- The flags ‘fixed’ and ‘scientific’ cannot both be on at the same time
- Setting one will turn the other off
- Similarly, we can only show numbers with one base system at a time
- We can only align left or right in a given field
- If we can set a flag, we can also “unset” a flag
    - The function is aptly named `unsetf()`
    - We use it the exact same way as `setf()`
    - But unlike `setf()`, it will turn off the flag

Another Way to Manipulate Streams

- So far, we've only used functions to manipulate our streams
- It is possible to make changes to the stream, from within the stream
- We've actually already done it before, when we used `setw()` and `endl`
- Remember that `setw()` requires the library <`iomanip`>
- When we change the stream from within the stream, we are still using a function, but because of the way they are used, they have a different name
    - They're called manipulators
- Every function and flag that we've seen so far can also be used as a manipulator, meaning they can be invoked from within a stream
    - `precision()` is called `setprecision()` if used as a manipulator

Another Handy Function

- There are times where we may use `setw()`, but would like something other than a giant empty space in a field
- One example is a directory type listing, where the name appears on the left edge, and a trail of dots connects the name to the phone number
- We are able to change the ‘fill’ character when we have empty space in field
- The function is called `fill()`, and it looks like this:

```
char prev = cout.fill('.');
```

- Instead of empty space, we would instead see it filled with dots

Streams as Arguments to Functions

Streams as Arguments to Functions

- Like any other type of variable, we are able to pass streams to functions
- There are a few things we need to keep in mind for it to work as expected
- Let's take a look at a function declaration

```
void writeOut(ostream& sout);
```

- One thing that we can observe is that the stream has been passed by reference
- We should **ALWAYS** pass a stream by reference to a function

What About the Type?

```
void writeOut(ostream& sout);
```

- Note the data type for the parameter; `ostream`
- All output streams, whether `cout` or an output file stream, are of type ostream
- We could have made the type `ofstream`, but then our function would be restricted to only accepting output file streams
- By using the type `ostream`, we give our function more flexibility
- If we only intend to pass `cout` to the function, we still have to use `ostream`
- The same applies to input streams

If We Want to Pass a File Stream

- If we want to pass a file stream, there are a couple things we have to be sure of
- The stream needs to be ready to be used
  - This means we have declared the stream, and attached it to a file already
  - This should not be done in the function
- Do *not* close the stream in the function
- When we pass a stream to a function, it is only to use it, never to maintain it

Putting it All Together

- Code files are also uploaded to Blackboard to demonstrate these functions
- While `cout` was used throughout the slides, any output stream has access to these same functions
- The source file `cleanup.cpp` puts everything that's been covered in Chapter 6 so far into practice
- Do take some time to play with some of the flags and experiment with the code

Character I/O

Character I/O

- There are times where we may want to manipulate individual characters in a stream
- We have already seen one function for doing that, `cin.get()`
- `cin.get()` will grab a single character out of the stream
- We can then manipulate and/or test that character as needed
- We can also place a single character into an output stream using `cout.put()`
- Note that while these slides use `cin` and `cout`, all input and output streams have access to these functions, including our filestreams

One Other Function

- There is one more function worth discussing, `cin.putback()`
- When we are using `cin.get()`, we are able to grab whitespace characters
- There may arise situations where we read a sentinel value to know that we are done fetching characters out of the stream, but that sentinel value may be required to be in the stream for logic elsewhere in our program
- `putback()` can place a character back into an input stream

Default Parameters

Default Parameters

- There are many times where a function takes a parameter, and a majority of the time that parameter is always the same value
- We could overload the function and have a version where the parameter uses a hard- coded value
- But that is not a good practice, and a wasteful repitition of code
- Instead, what we can do is specify a default value for that parameter

What That Looks Like

```
void printCoords(int x, int y = 0);
```

- As we can see, setting a default value is fairly straightforward
- We just assign it
- It is important to note that a default value should **only** appear when we declare a function
- We will get compile errors if we attempt to assign a default value when we define a function
- With a default value, we are able to call the function in two ways

```
printCoords(2, 4);
printCoords(2);
```

What’s Going On

```
void printCoords(int x, int y = 0);
printCoords(2, 4);
printCoords(2);
```

- The first call to the function is a normal call, it’s what we’ve been doing all semester
- The second call is a bit more interesting
- Our function has two parameters listed, but we only provide one
- When we declared our function, the second parameter was given a default value of 0
- If we choose not to provide the second parameter, the 0 is automatically used
- We should be aware that there are some limitations to default parameters

Limitations

- Whenever we call a function and provide arguments, the compiler will take the first argument and check it against the first parameter of the function
- This is always how it goes
- If the types don't match, we get an error
- If the argument can be type cast to suit the parameter it will be
- Because of this behavior, default parameters need to be at the end of parameter list
- If the parameter list has multiple default parameters, we are not able to skip one becaue of the compiler behavior to apply parameters starting at the left
- A file has been uploaded to Blackboard, experiment with different combinations to get a feel for what will and won't work