【C++】Day28

Class	C++
Date	@December 30, 2021
Material	
Series Number	
Summary	

【Ch8】The IO Library

8.1.3 Managing the Output Buffer

Each output stream manages a buffer, which it uses to hold the data that the program reads and writes. For example, when the following code is executed:

```
os << "Please enter a value: ";
```

the literal string might be printed immediately, or the operating system might store the data in a buffer to be printed later. Using a buffer allows the operating system to combine several output operations from our program into a single system-level write.

Because writing to a device can be time-consuming, letting the operating system combine several output operations into a single write can provide an important performance boost.

There are several conditions that cause the buffer to be flushed-that is, to be written-to the actual output device or file:

- The program completes normally. All output buffers are flushed as part of the return from `main`.

- At some indeterminate time, the buffer become full, in which case it will be flushed before writing the next value
- We can flush the buffer explicitly using a manipulator such as `endl`.
- We can use the `unitbuf` manipulator to set the stream's internal state to empty the buffer after each output operation.

  By default, `unitbuf` is set for `cerr`, so that writes to `cerr` are flushed immediately.
- An output stream might be tied to another stream. In this case, the buffer of the tied stream is flushed whenever the the tied stream is read or written.

  By default, `cin` and `cerr` are both tied to `cout`. Hence, reading `cin` or writing to `cerr` flushes the buffer in cout.

Flushing the Output Buffer

There are two other similar manipulators like `endl`: `flush` and `ends`.

- `flush` flushes the stream but adds no characters to the output
- `ends` inserts a null character into the buffer and then flushes it

```
cout << "hi!" << endl; //writes hi and a newline, then flushes the buffer
cout << "hi!" << flush; //writes hi, then flushes the buffer; adds no data
cout << "hi!" << ends; //writes hi and a null, then flushes the buffer.
```

The unitbuf Manipulator

If we want to flush after every output, we can use the `unitbuf` manipulator.

This manipulator tells the stream to do a flush after every subsequent write.

The `nounitbuf` manipulator restores the stream to use normal, system-managed buffer flushing:

```
cout << unitbuf; //all writes will be flushed immediately
//any output is flushed immediately, no buffering
cout << nounitbuf; //returns to normal buffering
```

Caution Buffers Are Not Flushed if the Program Crashes

Output buffers are not flushed if the program terminates abnormally. When a program crashes, it is likely that data the program wrote may be sitting in an output buffer waiting to be printed.

Trying Input and Output Streams Together

When an input stream is tied to an output stream, any attempt to read the input stream will first flush the buffer associated with the output stream. The library ties `cout` to `cin`, so the statement

```
cin >> ival;
```

causes the buffer associated with `cout` to be flushed.

Note: Interactive systems usually should tie their input stream to their output stream. Doing so means that all output, which might include promotes to the user, will be written before attempting to read the input.

There are two overloaded versions of `tie`: One version takes no argument and returns a pointer to the output stream, if any, to which this object is currently being tied. The function returns the null pointer if the stream is not tied.

The second version of `tie` takes a pointer to an `ostream` and ties itself to that `ostream`. That is

```
x.tie(&o);
```

ties the stream x to the output stream o.

We can tie either an `istream` or an `ostream` object to another `ostream`:

```
cin.tie(&cout); //illustration only: the library ties cin and cout for us
//old_tie points to the stream(if any) currently tied to cin
//cin.tie() will return the old-tied ostream, but will not untie the stream
ostream *old_tie = cin.tie(nullptr); //cin is no longer tied
//ties cin and cerr; not a good idea because cin should be tied to cout
cin.tie(&cerr);
cin.tie(old_tie); //reestablish normal tie between cin and cout
```

To tie a given stream to a new output stream, we pass `tie` a pointer to the new stream.

To untie the stream completely, we pass a null pointer.