Stream Input/Output

Contents

- Basic Concepts of Streams
- Stream I/O Classes and Objects
- I/O Manipulators
- Stream Error States

C++ Stream Input/Output

- The C++ standard libraries provide an extensive set of input/output (I/O) capabilities.
- Many of the I/O features are object oriented: The uses of other C++ features, such as references, function overloading and operator overloading.
- C++ uses **type-safe I/O**: Each I/O operation is executed in a manner sensitive to the data type.
- Users can specify how to perform I/O for objects of user-defined types by overloading the stream insertion operator (<<) and the stream extraction operator (>>)

Streams: Sequence of bytes

C++ Stream Input/Output

- C++ provides both “low-level” and “high-level” I/O capabilities.
- Low-level I/O capabilities (i.e., **unformatted I/O**) specify that some number of bytes should be transferred device-to-memory or memory-to-device
- High-level I/O (i.e., **formatted I/O**): Bytes are grouped into meaningful units, such as integers, floating-point numbers, characters, strings and user-defined type.
- C++ includes the standard stream libraries, which enable developers to build systems capable of performing I/O operations with Unicode characters.
- For this purpose, C++ includes an additional character type called wchar_t, which can store 2-byte Unicode characters

Library Header Files

- The C++ iostream library provides hundreds of I/O capabilities. Several header files contain portions of the library interface.
- Most C++ programs include the <iostream> header file, which declares basic services required for all stream-I/O operations.
- The <iostream> header file defines the cin, cout, cerr and clog objects, which
  - cin: standard input stream
  - cout: standard output stream
  - cerr: unbuffered standard error stream
  - clog: buffered standard error stream
- The <iomanip> header declares services useful for performing formatted I/O with so called parameterized stream manipulators, such as setw and setprecision.
- The <fstream> header declares services for user-controlled file processing.

Stream I/O Classes and Objects

- The iostream library provides many templates for handling common I/O operations. For example
- class template **basic_istream** supports stream-input operations,
- class template **basic_ostream** supports stream-output operations,
- class template **basic_iostream** supports both stream-input and stream-output operations.
- Also, the iostream library provides a set of typedefs that provide aliases for these template specializations

Stream I/O Classes and Objects

```
typedef int Length;
```

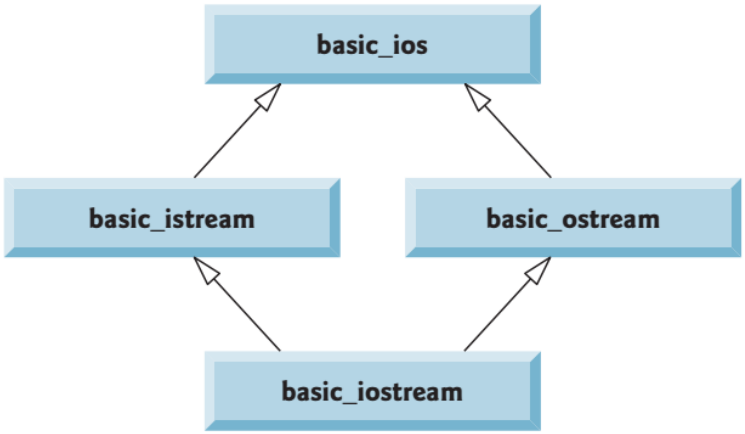
creates Length as a synonym for int.

```
typedef int (*PFI)(char *, char *);
```

creates PFI as a synonym for a pointer to a function of two char * arguments that returns an int.

- The **typedef istream** represents a specialization of basic_istream that enables char input.
- The **typedef ostream** represents a specialization of basic_ostream that enables char output.
- The **typedef iostream** represents a specialization of basic_iostream that enables both char input and output.

Stream I/O Template Hierarchy



- Templates **basic_istream** and **basic_ostream** both derive through single inheritance from base template basic_ios
- Template **basic_iostream** derives through multiple inheritance from templates basic_istream and basic_ostream.

- **The left-shift operator (<<) is overloaded** to designate stream output and is referred to as the stream insertion operator.
- **The right-shift operator (>>) is overloaded** to designate stream input and is referred to as the stream extraction operator.
- These operators are used with the standard stream objects cin, cout, cerr and clog and, commonly, with userdefined stream objects.

Standard Stream Objects

- Predefined object cin is an istream instance

connected to" (or attached to) the standard input device.

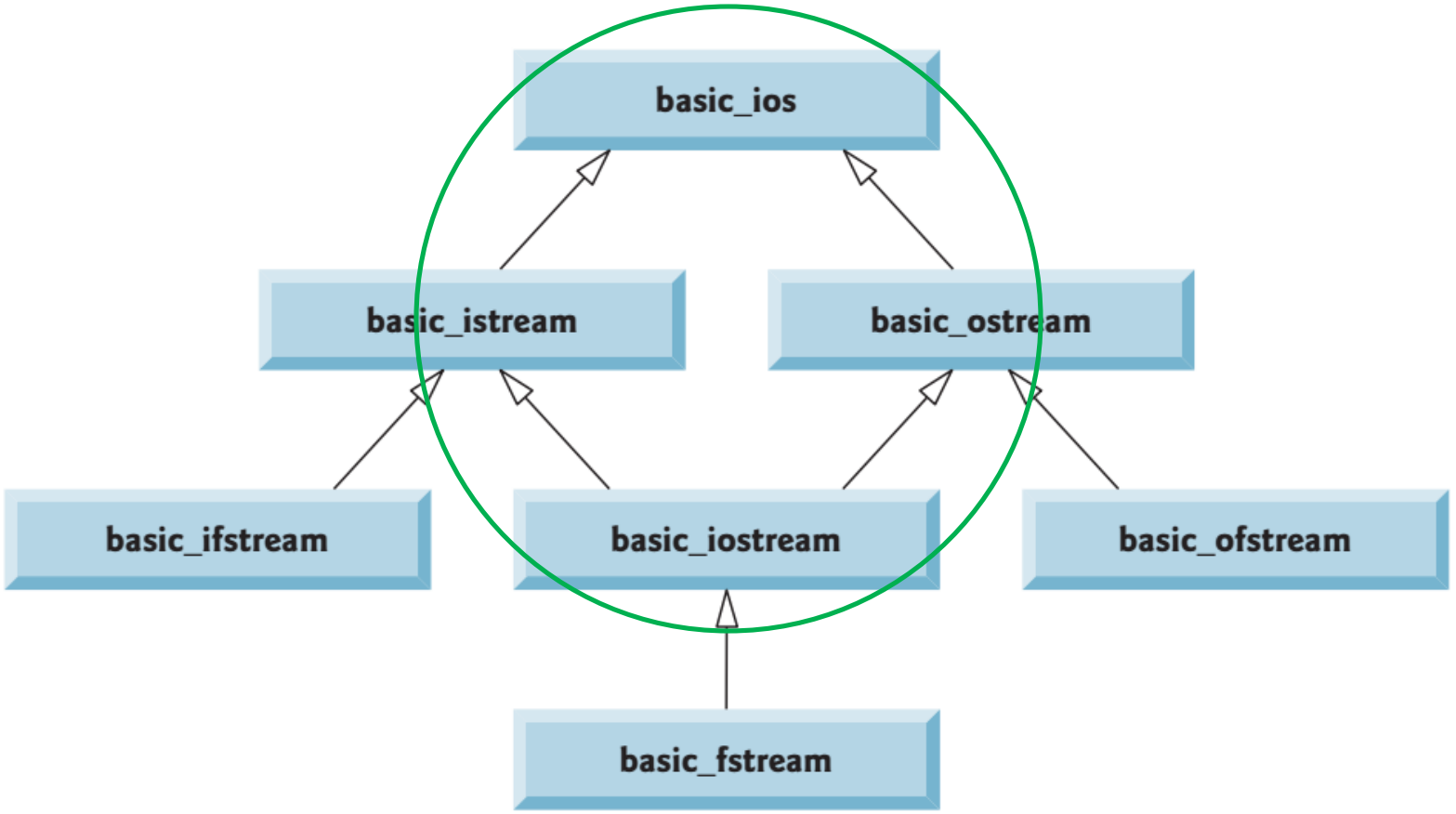
```
cin >> grade; // data "flows" in the direction of the arrows
```

- Predefined object cout is an ostream instance

"connected to" the standard output device.

```
cout << grade; // data "flows" in the direction of the arrows
```

File Processing Templates



Stream Output

- Output capabilities are provided by **ostream**.
  - Output of standard data types with the stream insertion operator (<<)
  - Output of characters via the put member function;
  - unformatted output via the write member function
  - Output of integers in decimal, octal and hexadecimal formats
  - Output of floating-point values with various precision
  - Output of data justified in fields of designated widths

```
cout.put( 'A' );
```

```
cout.put( 'A' ).put( '\n' )
```

```
cout.put( 65 );
```

Stream Input

- Input capabilities are provided by **istream**

```
// Using member functions get, put and eof.

#include <iostream>

using namespace std;

int main()
 {
    int character;

    cout << "Before input, cin.eof() is " << cin.eof() << endl
        << "Enter a sentence followed by end-of-file:" << endl;

    while ((character = cin.get()) != EOF)
        cout.put(character);


    cout << "\nEOF in this system is: " << character << endl;

    cout << "After input of EOF, cin.eof() is " << cin.eof() << endl;

    system("PAUSE");
}
```

Stream Input

```
Before input, cin.eof() is 0
Enter a sentence followed by end-of-file:
Testing the get and put member functions
Testing the get and put member functions
^Z

EOF in this system is: -1
After input of EOF, cin.eof() is 1
```

get() function: Another form

```
void main()
{
   char line[100];
   cout << " Type a line terminated by 't'" << endl;
   cin.getline( line, 100, 't' );
   cout << line;
}
```

cin vs. cin.get

```
// Contrasting input of a string via cin and cin.get.
#include <iostream>
using namespace std;

int main()
{
   // create two char arrays, each with 80 elements
   const int SIZE = 80;
   char buffer1[ SIZE ];
   char buffer2[ SIZE ];

   // use cin to input characters into buffer1
   cout << "Enter a sentence:" << endl;
   cin >> buffer1;

   // display buffer1 contents
   cout << "\nThe string read with cin was:" << endl
      << buffer1 << endl << endl;

   // use cin.get to input characters into buffer2
   cin.get( buffer2, SIZE );

   // display buffer2 contents
   cout << "The string read with cin.get was:" << endl
      << buffer2 << endl;
} // end main
```

getline() function

```
int main()
{
   const int SIZE = 80;
   char buffer[ SIZE ]; // create array of 80 characters

   // input characters in buffer via cin function getline
   cout << "Enter a sentence:" << endl;
   cin.getline( buffer, SIZE );

   // display buffer contents
   cout << "\nThe sentence entered is:" << endl << buffer << endl;
} // end main
```

ignore() function

- The ignore member function of istream reads and discards a designated number of characters (the default is one)

- Or, terminates upon encountering a designated delimiter (the default is EOF, which causes ignore to skip to the end of the file when reading from a file).

Unformatted I/O: read, write and gcount

- read(): istream: read inputs bytes to a character array in memory
- write(): ostream: write outputs bytes from a character array

- These bytes are not formatted in any way (raw bytes).

```
char buffer[] = "HAPPY BIRTHDAY";
cout.write( buffer, 10 );
```

```
cout.write( "ABCDEFGHIJKLMNOPQRSTUVWXYZ", 10 );
```

Practice

```
// Unformatted I/O using read, gcount and write.
#include <iostream>
using namespace std;

int main()
{
   const int SIZE = 80;
   char buffer[ SIZE ]; // create array of 80 characters

   // use function read to input characters into buffer
   cout << "Enter a sentence:" << endl;
   cin.read( buffer, 20 );

   // use functions write and gcount to display buffer characters
   cout << endl << "The sentence entered was:" << endl;
   cout.write( buffer, cin.gcount() );
   cout << endl;
} // end main
```

Stream Manipulators <iomanip>

- C++ provides various stream manipulators that perform formatting tasks.
- Setting field widths, setting precision, setting the fill character in fields
- Inserting a newline into the output stream, inserting a null character into the output stream and
- Skipping white space in the input stream.

Stream Manipulators: ios_base

`skipws`	Skip white-space characters on an input stream. This setting is reset with stream manipulator `noskipws`.
`left`	Left justify output in a field. Padding characters appear to the right if necessary.
`right`	Right justify output in a field. Padding characters appear to the left if necessary.
`internal`	Indicate that a number's sign should be left justified in a field and a number's magnitude should be right justified in that same field (i.e., padding characters appear between the sign and the number).
`dec`	Specify that integers should be treated as decimal (base 10) values.
`oct`	Specify that integers should be treated as octal (base 8) values.
`hex`	Specify that integers should be treated as hexadecimal (base 16) values.
`showbase`	Specify that the base of a number is to be output ahead of the number (a leading `0` for octals; a leading `0x` or `0X` for hexadecimals). This setting is reset with stream manipulator `noshowbase`.
`showpoint`	Specify that floating-point numbers should be output with a decimal point. This is used normally with `fixed` to guarantee a certain number of digits to the right of the decimal point, even if they're zeros. This setting is reset with stream manipulator `noshowpoint`.

Stream Manipulators

`uppercase`	Specify that uppercase letters (i.e., `X` and `A` through `F`) should be used in a hexadecimal integer and that uppercase `E` should be used when representing a floating-point value in scientific notation. This setting is reset with stream manipulator `nouppercase`.
`showpos`	Specify that positive numbers should be preceded by a plus sign (`+`). This setting is reset with stream manipulator `noshowpos`.
`scientific`	Specify output of a floating-point value in scientific notation.
`fixed`	Specify output of a floating-point value in fixed-point notation with a specific number of digits to the right of the decimal point.

Stream Error States

- The state of a stream may be tested through bits in class **ios_base**.

Cin object

- The **fail member function** reports whether a stream operation has failed (The failbit is set for a stream when a format error occurs on the stream and no characters are input. When such an error occurs, the characters are not lost)

- The **bad member function** reports whether a stream operation failed (The badbit is set for a stream when an error occurs that results in the loss of data)

- The **good member function** returns true if the bad, fail and eof functions would all return false

- The **rdstate member function** returns the stream's error state

- The **clear member function** is used to restore a stream's state to "good

Stream Error States

```
// Testing error states.
#include <iostream>
using namespace std;

int main()
{
   int integerValue;

   // display results of cin functions
   cout << "Before a bad input operation:"
      << "\ncin.rdstate(): " << cin.rdstate()
      << "\n    cin.eof(): " << cin.eof()
      << "\n   cin.fail(): " << cin.fail()
      << "\n    cin.bad(): " << cin.bad()
      << "\n   cin.good(): " << cin.good()
      << "\n\nExpects an integer, but enter a character: ";

   cin >> integerValue; // enter character value
   cout << endl;

   // display results of cin functions after bad input
   cout << "After a bad input operation:"
      << "\ncin.rdstate(): " << cin.rdstate()
      << "\n    cin.eof(): " << cin.eof()
      << "\n   cin.fail(): " << cin.fail()
      << "\n    cin.bad(): " << cin.bad()
      << "\n   cin.good(): " << cin.good() << endl << endl;

   cin.clear(); // clear stream

   // display results of cin functions after clearing cin
   cout << "After cin.clear()" << "\ncin.fail(): " << cin.fail()
      << "\ncin.good(): " << cin.good() << endl;
} // end main
```

Stream Error States

```
Before a bad input operation:
cin.rdstate(): 0
    cin.eof(): 0
   cin.fail(): 0
    cin.bad(): 0
   cin.good(): 1

Expects an integer, but enter a character: A

After a bad input operation:
cin.rdstate(): 2
    cin.eof(): 0
   cin.fail(): 1
    cin.bad(): 0
   cin.good(): 0

After cin.clear()
cin.fail(): 0
cin.good(): 1
```

Class Quiz

- Topics
  - Operator Overloading
  - Inheritance

Duration: 30 Min

Date: June 1, 2017 (Thursday)