Programming in Modern C++

Module M42: Input-Output: Streams in C++

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ac.in

All url's in this module have been accessed in September, 2021 and found to be functional

Module Recap

- Discussed formatted and unformatted I/O using C Standard Library
- Discussed I/O with file and string

Module Objectives

- To understand object-oriented stream input/output of C++

Source: Chapter 21 - C++ Stream Input/Output. © Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Module Outline

Features of C++ I/O

Sources:

- Input/output via `<iostream>` and `<cstdio>`, isocpp
- Input/Output Library: cplusplus.com
- Input/Output Library: cppreference.com
- Chapter 21 - C++ Stream Input/Output. © Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved

Features of C++ I/O

- Many C++ I/O features are object-oriented
  - Use references, function overloading and operator overloading
- C++ uses type safe I/O
  - Each I/O operation is automatically performed in a manner sensitive to the data type
- Extensibility
  - Users may specify I/O of user-defined types as well as standard types

Streams

Streams

Sources:

- Input/output via `<iostream>` and `<cstdio>`, isocpp
- Input/Output Library: cplusplus.com
- Input/Output Library: cppreference.com
- Chapter 21 - C++ Stream Input/Output. © Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved

Streams

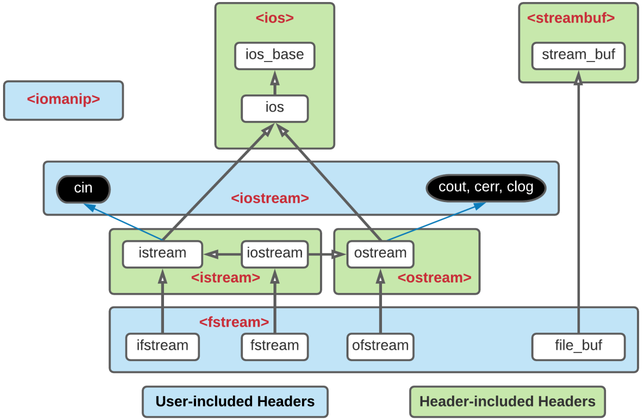
- Stream
  - A transfer of information in the form of a sequence of bytes
  - The term stream is an abstraction of a construct that allows you to send or receive an unknown number of bytes. The metaphor is a stream of water. You take the data as it comes, or send it as needed. Contrast this to an array, for example, which has a fixed, known length
- I/O Operations
  - Input: A stream that flows from an input device (that is, keyboard, disk drive, network connection) to main memory
    - ▷ `istream`
    - ▷ `ifstream`
  - Output: A stream that flows from main memory to an output device (that is, screen, printer, disk drive, network connection)
    - ▷ `ostream`
    - ▷ `ofstream`

Streams

- I/O operations are a bottleneck
  - The time for a stream to flow is many times larger than the time it takes the CPU to process the data in the stream
- Low-level I/O
  - Unformatted
  - Individual byte unit of interest
  - High speed, high volume, but inconvenient for people
- High-level I/O
  - Formatted
  - Bytes grouped into meaningful units: integers, characters, etc.
  - Good for all I/O except high-volume file processing

<iostream> Header Files

- iostream library
  - <iostream>: Contains cin, cout, cerr and clog objects
  - <iomanip>: Contains parameterized stream manipulators



Stream I/O Classes and Objects

- `ios`:
  - `istream` and `ostream` inherit from `ios`
    - ▷ `iostream` inherits from `istream` and `ostream`
- `<<` (left-shift operator)
  - Overloaded as stream insertion operator
- `>>` (right-shift operator)
  - Overloaded as stream extraction operator
  - Both operators used with `cin`, `cout`, `cerr` and `clog`, and with user-defined stream objects
- `istream`: input streams
  - `cin >>  grade`;
    - ▷ `cin` knows what type of data is to be assigned to grade (based on the type of `grade`)
- `ostream`: output streams
  - `cout <<  grade`;
    - ▷ `cout` knows the type of data to output
  - `cerr <<  errorMessage`;
    - ▷ *Unbuffered* - prints `errorMessage` immediately
  - `clog <<  errorMessage`;
    - ▷ *Buffered* - prints `errorMessage` as soon as output buffer is full or flushed

Stream Output

Stream Output

Sources:

- Input/output via `<iostream>` and `<cstdio>`, isocpp
- Input/Output Library: cplusplus.com
- Input/Output Library: cppreference.com
- Chapter 21 - C++ Stream Input/Output. © Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved

Stream Output

- `ostream`: performs formatted and unformatted output
  - Uses `put` for characters and `write` for unformatted output
  - Output of integers in decimal, octal and hexadecimal
  - Varying precision for floating points
  - Formatted text outputs
- `<<` is overloaded to output built-in types
  - Can also be used to output user-defined types
  - `cout << '\n';`
    - ▷ Prints newline character
  - `cout << endl;`
    - ▷ `endl` is a stream manipulator that issues a newline character and flushes the output buffer
  - `cout << flush;`
    - ▷ `flush` flushes the output buffer
- `put` member function
  - Outputs one character to specified stream: `cout.put('A');`
  - Returns a reference to the object that called it, so may be *cascaded*: `cout.put('A').put('\n');`
  - May be called with an ASCII-valued expression: `cout.put(65);`
    - ▷ Outputs A

Print Built-in Type Data

```
#include <iostream>
using namespace std;
int main() {
    int i = 17; long l = 0x012a78cb; // 19560651
    long long unsigned int i64 = 0x012a78cb2597ac3d; // 84012356964166717
    float f = 15.0 / 7; double d = 15.0 / 7;
    char c = 'x'; const char *s = "ppd";
    int *p = &i;

    cout << i << " ";             // int       // 17 Optional dec may be used
    cout << hex << i << endl;     // hex       // 11
    cout << oct << i << endl;     // oct       // 21
    cout << l << " ";             // long      // 19560651
    cout << i64 << " ";           // int 64    // 84012356964166717
    cout << f << " ";             // float     // 2.14286
    cout << d << " ";             // double    // 2.14286
    cout << c << " ";             // char      // x
    cout << s << " ";             // string    // ppd
    cout << (void*)(s) << endl; // pointer   // 0x55c825222009 // Address of 1st character of the string
    cout << p << " ";             // pointer   // 0x7fff9a17cf68
}
```

- An integer (`int`) may be printed in decimal (`dec`, by default), octal (`oct`) or hexadecimal (`hex`) format
- A `char*` pointer prints the string. To print the pointer value, cast to `void*` by `static_cast<const void*>` or `(void*)`

Print User-defined Type Data

```
#include <iostream>
using namespace std;

class Complex {
        double re, im; // Encapsulated
public:
        Complex(double r, double i) : re(r), im(i) { }

        // UDT Specific print function
        friend ostream& operator<<(ostream& os, const Complex& c) {
                cout << "(" << c.re << ", " << c.im << ")";
                return os;
        }
};
int main() {
        Complex c1 = { 2.5, 7.3 }, c2(4.3, 8.9);

        cout << c1 << "; " << c2 << endl; // Cascading the printing
}
```

```
(2.5, 7.3); (4.3, 8.9)
```

Print Built-in Type Data: C vis-a-vis C++

```
#include <cstdio>
#include <iostream>
using namespace std;
int main() {
    int i = 17;
    long l = 0x012a78cb; // 19560651
    long long unsigned int i64 = 0x012a78cb2597ac3d; // 84012356964166717
    float f = 15.0 / 7;
    double d = 15.0 / 7;
    char c = 'x';
    const char *s = "ppd";
    int *p = &i;
    cout << i << " ";           printf("%d\n", i);      // dec      // 17 17 Opt. dec may be used in C++
    cout << hex << i << endl;   printf("%x\n", i);      // hex      // 11 11
    cout << oct << i << endl;   printf("%o\n", i);      // oct      // 21 21
    cout << l << " ";           printf("%ld\n", l);     // long     // 19560651 19560651
    cout << i64 << " ";         printf("%llu\n", i64); // int 64   // 84012356964166717 84012356964166717
    cout << f << " ";           printf("%f\n", f);      // float    // 2.14286 2.142857
    cout << d << " ";           printf("%lf\n", d);     // double   // 2.14286 2.142857
    cout << c << " ";           printf("%c\n", c);      // char     // x x
    cout << s << " ";           printf("%s\n", s);      // string   // ppd ppd
    cout << p << " ";           printf("%p\n", p);      // pointer  // 0x7ffc28102988 0x7ffc28102988
}
```

- Note the use of hex and oct in C++ and the difference in default precision for float and double between C++ and C

Stream Input

Stream Input

Sources:

- Input/output via `<iostream>` and `<cstdio>`, isocpp
- Input/Output Library: cplusplus.com
- Input/Output Library: cppreference.com
- Chapter 21 - C++ Stream Input/Output. © Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved

Stream Input

- `>>` (stream-extraction)
  - Used to perform stream input
  - Normally ignores whitespaces (spaces, tabs, newlines)
  - Returns zero (`false`) when `EOF` is encountered, otherwise returns reference to the object from which it was invoked (that is, `cin`)
- `>>` controls the state bits of the stream
  - `endl` is a stream manipulator that issues a newline character and flushes the output buffer
- `>>` and `<<` have relatively high precedence
  - Conditional and arithmetic expressions must be contained in parentheses
- Common way to perform loops

```
while (cin >> grade)
```

  - Extraction returns 0 (`false`) when `EOF` encountered, and loop ends

Member Functions

- `cin.eof()`
  - returns true if end-of-file has occurred on `cin`
- `cin.get()`
  - inputs a character from stream (even white spaces) and returns it
- `cin.get(c)`
  - inputs a character from stream and stores it in `c`
- `cin.get(array, size)`
  - Accepts 3 arguments: array of characters, the size limit, and a delimiter (default of '`\n`')
  - Uses the array as a buffer
  - When the delimiter is encountered, it remains in the input stream
  - Null character is inserted in the array
  - Unless delimiter flushed from stream, it will stay there

Member Functions

- `cin.getline(array, size)`
  - Operates like `cin.get(buffer, size)` but it discards the delimiter from the stream and does not store it in array
  - Null character inserted into array
- `ignore`
  - Operates like `cin.get(buffer, size)` but it discards the delimiter from the stream and does not store it in array
  - Null character inserted into array
- `putback`
  - Places the previous character obtained by get back in to the stream
- `peek`
  - Returns the next character from the stream without removing it

File I/O

File I/O

Sources:

- Input/output via `<iostream>` and `<cstdio>`, isocpp
- Input/Output Library: cplusplus.com
- Input/Output Library: cppreference.com
- Chapter 21 - C++ Stream Input/Output. © Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved

Input / Output with Files

- Open
  - Like in C, files need to be first opened and associated with a stream

```
ofstream myfile;            // Output stream
myfile.open("example.txt"); // Open: Associate file example.txt to output stream myfile
-----
ofstream myfile("example.txt"); // Output stream opened and associated
myfile.is_open(); // Check if open has worked correctly
```

  - Unlike C (where stream is a pointer), stream is an object in C++
  - Unlike C (where mode is specified by a string flag), stream object itself is of i/p or o/p types
- Read / Write
  - Like in C, we perform formatted or unformatted I/O on an open stream (file)
  - Unlike C (where functions for formatted I/O are variadic and needs explicit format specification), objects are read / written using streaming operators for the data types
- Close
  - Like in C, streams need to be closed when done and disassociated from the file

```
myfile.close(); // Close: Flush stream to file and disassociate from stream
```

- Binary Files
  - Use `ios::binary` flag in the opening mode

Input / Output with Files

```
// Writing to Output File
#include <iostream>
#include <fstream>
using namespace std;
int main () { ofstream myfile;                     // Output stream
    myfile.open("example.txt");                    // Open: Associate file example.txt to output stream myfile
    myfile << "Writing this to a file.\n";         // Stream to output
    myfile.close();                                // Close: Flush stream to file and disassociate from stream
}

// Reading from Input File
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main () { ifstream myfile("example.txt");     // Input stream
    string line;
    if (myfile.is_open()) {                        // Open: Associate file example.txt to input stream myfile
        while (getline(myfile, line))              // Unformatted Read: Get by line from stream
            cout << line << '\n';
        myfile.close();                            // Close: Disassociate file from stream
    }
    else cout << "Unable to open file";
}
```

Type-safe I/O

Type-safe I/O

Sources:

- Input/output via `<iostream>` and `<cstdio>`, isocpp
- Input/Output Library: cplusplus.com
- Input/Output Library: cppreference.com
- Chapter 21 - C++ Stream Input/Output. © Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved

Type-safe I/O

- << and >> operators
  - Gets format from type of data being read / written
  - Overloaded to accept data of different types
  - Cascading for ease of expression
  - Avoids the use of error-prone variadic functions
  - When unexpected data encountered, error flags set
  - Program stays in control

Type-safe I/O: C vis-a-vis C++

```
#include <cstdio>
#include <iostream>
using namespace std;

int main() {
        int i = 5, j = 3;
        double d = 2.37483;

        // C I/O is type-unsafe
        printf("%d %d\n", i, j); // Okay: 5 3
        printf("%d\n", i, j);    // Error. Missing format spec. Prints garbage for j: 5 2757403
        printf("%d %d\n", i);    // Error. Missing second value. Ignored: 5

        printf("%lf\n", d);      // Okay: 2.374830
        printf("%d\n", d);       // Error. Wrong integer format for double value: -553878982
        printf("%lf\n", i);      // Error. Wrong double format for integer value: 0.000000

        // C++ I/O is type-safe
        cout << i << ' ';
        cout << j << ' ';
        cout << d << endl;       // Okay: 5 3 2.37483
}
```

Type-safe I/O: User-defined Operators

```
// Discussed in  Module 19: Program 19.06
#include <iostream>
using namespace std;

class Complex { double re, im; // Encapsulated
public: Complex(double r = 0.0, double i = 0.0) : re(r), im(i) { }

    friend ostream& operator<<(ostream& os, const Complex& c) { // UDT Specific print function
            cout << "(" << c.re << ", " << c.im << ")";
            return os;
    }
    friend istream& operator>>(istream& os, Complex& c) { // UDT Specific scan function
            cin >> c.re >> c.im;
            return os;
    }
};
int main() {
    Complex c1 = { 2.5, 7.3 }, c2(4.3, 8.9), c3, c4;

    cout << c1 << "; " << c2 << endl; // Cascading the printing: (2.5, 7.3); (4.3, 8.9)
    cout << c3 << "; " << c4 << endl; // Cascading the printing: (0, 0); (0, 0)
    cin >> c3 >> c4;                  // Cascading the scanning: 1.2 3.7 3.4 9.6
    cout << c3 << "; " << c4 << endl; // Cascading the printing: (1.2, 3.7); (3.4, 9.6)
}
```

Unformatted I/O

Unformatted I/O

Sources:

- Input/output via `<iostream>` and `<cstdio>`, isocpp
- Input/Output Library: cplusplus.com
- Input/Output Library: cppreference.com
- Chapter 21 - C++ Stream Input/Output. © Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved

Unformatted I/O

- `read` and `write` member functions
  - Unformatted I/O
  - Input/output raw bytes to or from a character array in memory
  - Since the data is unformatted, the functions will not terminate at a newline character for example
  - Instead, like `getline`, they continue to process a designated number of characters
  - If fewer than the designated number of characters are read, then the `failbit` is set
- `gcount`
  - Returns the total number of characters read in the last input operation

Stream Manipulators

Stream Manipulators

Sources:

- Input/output via <iostream> and <cstdio>, isocpp
- Input/Output Library: cplusplus.com
- Input/Output Library: cppreference.com
- Chapter 21 - C++ Stream Input/Output. © Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved

Functionality of Stream Manipulators

- Setting field widths
- Setting precisions
- Setting and unsetting format flags
- Setting the fill character in fields
- Flushing streams
- Inserting a newline in the output stream and flushing the stream
- Inserting a null character in the output stream and skipping whitespace in the input stream

Integral Stream Base

- `dec` (default), `oct` or `hex`
  - Change base of which integers are interpreted from the stream

    ```
    int n = 15;
    cout << hex << n;
    ```
  - Prints "F"
- `setbase`:
  - Changes base of integer output
  - Load `<iomanip>`
  - Accepts an integer argument (10, 8, or 16)

    ```
    cout << setbase(16) << n;
    ```
  - Parameterized stream manipulator - takes an argument

Floating-Point Precision

- `precision`
  - Member function
  - Sets number of digits to the right of decimal point

    ```
    cout.precision(2);
    ```
  - `cout.precision()` returns current precision setting
- `setprecision`:
  - Parameterized stream manipulator
  - Like all parameterized stream manipulators, `<iomanip>` required
  - Specify precision

    ```
    cout << setprecision(2) << x;
    ```
  - For both methods, changes last until a different value is set

Field Width

- `ios width` member function
  - Sets field width (number of character positions a value should be output or number of characters that should be input)
  - Returns previous width
  - If values processed are smaller than width, fill characters inserted as padding
  - Values are not truncated - full number printed

```
cin.width(5);
```

- `setw` stream manipulator

```
cin >> setw(5) >> string;
```

- Remember to reserve one space for the null character

User-Defined Manipulators

- We can create our own stream manipulators
  - `bell`
  - `ret` (carriage return)
  - `tab`
  - `endLine`
- Parameterized stream manipulators
  - Consult installation manuals

Stream States

Stream States

Sources:

- Input/output via <iostream> and <cstdio>, isocpp
- Input/Output Library: cplusplus.com
- Input/Output Library: cppreference.com
- Chapter 21 - C++ Stream Input/Output. © Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved

Stream Format States

- Format flags
  - Specify formatting to be performed during stream I/O operations
- `setf`, `unsetf` and `flags`
  - Member functions that control the flag settings

Format State Flags

- Format State Flags
  - Defined as an enumeration in class `ios`
  - Can be controlled by member functions
  - `flags` - specifies a value representing the settings of all the flags
  - Returns long value containing prior options
  - `setf` - one argument, "`or`s" flags with existing flags
  - `unsetf` - `unsets` flags
  - `setiosflags` - parameterized stream manipulator used to set flags
  - `resetiosflags` - parameterized stream manipulator, has same functions as `unsetf`
- Flags can be combined using bitwise `OR ( | )`

Trailing Zeros and Decimal Points

- `ios::showpoint`
  - Forces a float with an integer value to be printed with its decimal point and trailing zeros

    ```
    cout.setf(ios::showpoint)
    cout << 79;

    79 will print as 79.00000
    ```

    ▷ Number of zeros determined by precision settings

Justification

- `ios::left`
  - Fields to left-justified with padding characters to the right
- `ios::right`
  - Default setting
  - Fields right-justified with padding characters to the left
- Character used for padding set by
  - `fill` member function
  - `setfill` parameterized stream manipulator
  - Default character is space
- `internal` flag
  - Number's sign left-justified
  - Number's magnitude right-justified
  - Intervening spaces padded with the fill character
- `static` data member `ios::adjustfield`
  - Contains `left`, `right` and `internal` flags
  - Number's magnitude right-justified
  - `ios::adjustfield` must be the second argument to `setf` when setting the `left`, `right` or `internal` justification flags `cout.setf(ios::left, ios::adjustfield);`

Padding

- `fill`
  - Specifies the fill character
  - Space is default
  - Returns the prior padding character

```
cout.fill('*');
```

- `setfill` manipulator
  - Also sets fill character

```
cout << setfill ('*');
```

Integral Stream Base

- `ios::basefield` static member
  - Used similarly to `ios::adjustfield` with `setf`
  - Includes the `ios::oct`, `ios::hex` and `ios::dec` flag bits
  - Specify that integers are to be treated as octal, hexadecimal and decimal values
  - Default is decimal
  - Default for stream extractions depends on form inputted
    - Integers starting with `0` are treated as *octal*
    - Integers starting with `0x` or `0X` are treated as *hexadecimal*
  - Once a base specified, settings stay until changed

Floating-Point Numbers / Scientific Notation

- `ios::scientific`
  - Forces output of a floating point number in scientific notation:

    `1.946000e+009`

- `ios::fixed`
  - Forces floating point numbers to display a specific number of digits to the right of the decimal (specified with `precision`)
- `static` data member `ios::floatfield`
  - Contains `ios::scientific` and `ios::fixed`
  - Used similarly to `ios::adjustfield` and `ios::basefield` in `setf` like `cout.setf(ios::scientific, ios::floatfield);`
  - `cout.setf(0, ios::floatfield)` restores default format for outputting floating-point numbers

Uppercase/Lowercase Control

- `ios::uppercase`
  - Forces uppercase `E` to be output with scientific notation

    ```
    4.32E+010
    ```

  - Forces uppercase `X` to be output with hexadecimal numbers, and causes all letters to be uppercase

    ```
    75BDE
    ```

Setting and Resetting the Format Flags

- `flags`
  - Without argument, returns the current settings of the format flags (as a `long` value)
  - With a `long` argument, sets the format flags as specified
    - ▷ Returns prior settings
- `setf`
  - Sets the format flags provided in its argument
  - Returns the previous flag settings as a `long` value
  - Unset the format using `unsetf` member function as

    ```
    long previousFlagSettings =
    cout.setf(ios::showpoint | ios::showpos);
    ```
- `setf` with two `long` arguments `cout.setf(ios::left, ios::adjustfield);` clears the bits of `ios::adjustfield` then sets `ios::left`
  - This version of `setf` can be used with
  - `ios::basefield(ios::dec, ios::oct, ios::hex)`
  - `ios::floatfield(ios::scientific, ios::fixed)`
  - `ios::adjustfield (ios::left, ios::right, ios::internal)`
- `unsetf`
  - Resets specified flags
  - Returns previous settings

Stream Error States

- `eofbit`
  - Set for an input stream after end-of-file encountered
  - `cin.eof()` returns true if end-of-file has been encountered on `cin`
- `failbit`
  - Set for a stream when a format error occurs
  - `cin.fail()` - returns true if a stream operation has failed
  - Normally possible to recover from these errors

Stream Error States

- `badbit`
  - Set when an error occurs that results in data loss
  - `cin.bad()` returns true if stream operation failed
  - normally nonrecoverable
- `goodbit`
  - Set for a stream if neither `eofbit`, `failbit` or `badbit` are set
  - `cin.good()` returns `true` if the `bad`, `fail` and `eof` functions would all return `false`
  - I/O operations should only be performed on "good" streams
- `rdstate`
  - Returns the state of the stream
  - Stream can be tested with a switch statement that examines all of the state bits
  - Easier to use `eof`, `bad`, `fail`, and `good` to determine state

Stream Error States

- `clear`
  - Used to restore a stream's state to "good"
  - `cin.clear()` clears `cin` and sets `goodbit` for the stream
  - `cin.clear(ios::failbit)` actually sets the `failbit`
    - ▷ Might do this when encountering a problem with a user-defined type
- Other operators
  - `operator!`
    - ▷ Returns true if `badbit` or `failbit` set
  - `operator void*`
    - ▷ Returns false if `badbit` or `failbit` set
  - Useful for file processing

Standard I/O Library

Standard I/O Library

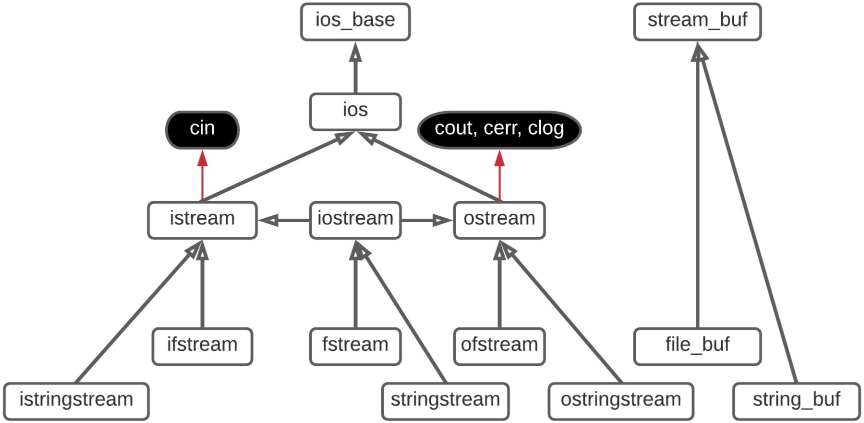
Sources:

- Input/output via `<iostream>` and `<cstdio>`, isocpp
- Input/Output Library: cplusplus.com
- Input/Output Library: cppreference.com
- Chapter 21 - C++ Stream Input/Output. © Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved

Library Organization

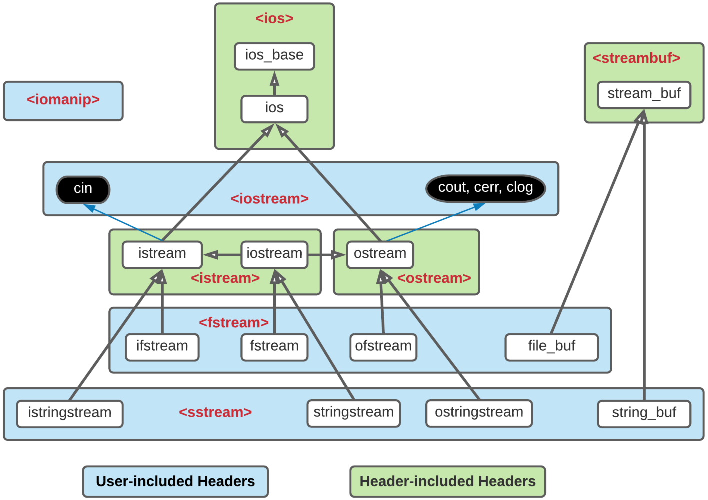
- `<ios>`, `<istream>`, `<ostream>`, `<streambuf>` and `<iosfwd>` are not usually included directly in most C++ programs. They describe the base classes of the hierarchy and are automatically included by other header files of the library that contain the derived classes.
- `<iostream>` declares the objects used to communicate through the standard input and output (including `cin` and `out`)
- `<fstream>` defines the file stream classes (like template `basic_ifstream` or class `ofstream`) as well as the internal buffer objects used (`basic_filebuf`). These classes are used to manipulate files with streams.
- `<sstream>`. The classes defined in this file are used to manipulate STL string objects as if they were streams.
- `<iomanip>` declares some standard manipulators with parameters to be used with extraction and insertion operators to modify internal flags and formatting options.

Input-Output Class Hierarchy



Sources: Input/Output Library: cplusplus.com, Input/Output Library: cppreference.com

Input-Output Classes in Header Files



Sources: Input/Output Library: cplusplus.com, Input/Output Library: cppreference.com

Header Organization

- `<iostream>`
  - `<istream>`
    - `<ostream>`
      - `<ios>`
- `<fstream>`
  - `<istream>`
- `<sstream>`
  - `<string>`
- `<iomanip>`
  - `<istream>`
- `<streambuf>`
  - `<xiosbase>`
- `<xiosbase>`
- `<iosfwd>`

Module Summary

- Understood object-oriented I/O of C++
- Learnt the major standard library components