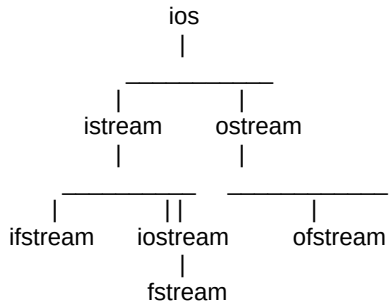
I/O STREAMS : objects and manipulators, examples

C++ uses a set of input/output (I/O) stream class libraries for input and output. Objects defined in these classes can be overloaded and extended just as any other C++ objects. Although you may still use the <stdio.h> function calls in the ANSI C library, you should probably get use to using streams. The following figure is the class hierarchy for C++ I/O classes and a brief description of the class names:



Generally, you'll be using the iostream class for standard I/O, and the fstream class for file I/O. In addition, the iomanip.h header file is frequently used and contains parameterized stream manipulators which are function-like calls which are used to change various I/O settings.

The following snippet shows how to use the basics of using C++ I/O streams. The insert operator '<<' and extraction operator '>>' are used to send and get data from the screen and keyboard.

```
// io.cp
#include <iostream.h>
#include <iomanip.h>

main()

   int     i;
   float f;
   char    c;

   cout << "Enter an integer: ";
   cin >> i;
   cout << "Enter a float: ";
   cin >> f;
   cout << "Enter a character: ";
   cin >> c;

   cout << endl;
   cout << "Default formats" << endl;
   cout << i << endl;
   cout << f << endl;
   cout << c << endl;

   cout << endl << "Other formats" << endl;
   cout << hex << i << endl;
   cout << setprecision(2) << f << endl;
   cout << setiosflags( ios::scientific ) << f << endl;

   return 0;

// end io.cp
```

INPUT STREAMS

Predefined input stream is cin (standard input). The >> operator is overloaded for input streams.

GCOUNT
int gcount()
Returns number of characters read by last unformatted read.

GET
int get()
istream &get(char &)
istream &get(unsigned char &)
istream &get(char *buf, int limit, char delim='\n')
istream &get(unsigned char *buf, int limit, char delim='\n')
Gets single character or series of characters (until end-of-file is reached). Delimiter, if read, is not included in characters read and is left in stream.

GETLINE
istream &getline(char *buf, int limit, char delim='\n')
istream &getline(unsigned char *buf, int limit, char delim='\n')
Reads characters until delimiter is read or limit-1 characters are read (or until end-of-file). The delimiter, if read, is included in character sequence.

IGNORE
istream &ignore(int limit=1, int delim=EOF)
Discards number of characters or until delimiter is encountered.

PEEK
int peek()
Look at next character to be read without reading it.

PUTBACK
istream &putback(char)
Puts character back onto stream. Can safely put one character back between successive calls to get().

READ
istream &read(char *buf, int count)
istream &read(unsigned char *buf, int count)
Reads a string of characters from stream. Sets "failbit" if end-of-file encountered.

SEEKG
istream &seekg(streampos, seek_dir=ios::beg)
Moves position of "get" pointer. The type streampos is an alias for type long.

TELLG
streampos tellg()
Returns current position of "get" pointer in file stream.

WS
istream &ws(istream &)
Discards whitespace characters from stream.

OUTPUT STREAMS

Predefined output streams are cout (standard output) and cerr (standard error). The << operator is overloaded for output streams.

ENDL
ostream &endl(ostream &)
Writes '\n' and flushes stream.
  Example:
    cout << endl;

FLUSH
ostream &flush()
ostream &flush(ostream &)
Flushes stream buffer.
  Examples:
    cout.flush();
    cout << flush;

PUT
ostream &put(char)
Writes single character to stream.

SEEKP
ostream &seekp(int &streampos, seek_dir=ios::beg)
Moves position of "put" pointer. seek_dir can be beginning (ios::beg), current (ios::cur), or end of the file (ios::end).

TELLP
streampos tellp()
Returns current "put" pointer.
  Example:
    streampos place = theFile.tellp();

WRITE
ostream &write(const char *buf, int count)
ostream &write(const unsigned char *buf, int count)
Writes specified number of characters to stream.

FILE I/O

Reading and writing to files is achieved by associating a file stream with a variable with statements like:

```
ifstream inputFile( "myInputFile.txt" );
if( !inputFile )
  doInputError();

ofstream outputFile( "myOutputFile.txt" );
if( !outputFile )
  doOutputError();
```

OPEN
void open(char *name, int mode=ios::out, int prot=filebuf::openprot)
Protection mode is operating-system dependent. Values for "mode" are:
  ios::app Data appended to file (implies ios::out).
  ios::ate Data appended to file (does not imply ios::out).
  ios::in File is opened for input.
  ios::out File opened for output.
  ios::trunc Discard previous contents of file.
  ios::nocreate If file does not exist, open() will fail.
  ios::noreplace If file exists, open() will fail.
  Example:
    ifstream inputFile;
    inputFile.open("myInputFile.txt");

ERROR STATES

Error states are maintained for every stream. These states are contained in the 'eofbit', 'failbit', and 'badbit'.

BAD
int bad()
Returns true if some operation on stream failed (recovery unlikely).

CLEAR
void clear(int state)
Sets error state of stream.

EOF
int eof()
Returns true if stream encounters end of file.

FAIL
int fail()
Returns true if some operation on stream failed. Stream is useable once the condition is cleared.

GOOD
int good()
Returns true if eof(), bad(), and fail() are false.

OPERATOR
int operator!()
operator void*()
Overloaded operators return true/false if failbit or badbit is set.

RDSTATE
int rdstate()
Returns current error state.

FORMATTING

Many format statements require the inclusion of the standard <iomanip.h> header file. These statements are identified as such. The following are a list of format flags and bitfields used by streams:

skipws	Skips whitespace on input.
left, right, internal	Sets justification within field.
dec, oct, hex	Set base for insertion/extraction of integral types. Comprise static member ios::basefield.
showbase	Display 0 before octal and 0x before hexadecimal values.
showpoint	Show decimal point and trailing zeros.
showpos	Insert + sign before positive values.
scientific, fixed	Sets floating point notation. Comprise static member ios::floatfield.

uppercase Use uppercase for hexadecimal X and exponential E.

The following is a list of format statements. Those statements which have a return type of ios& are stream manipulators (placed into stream with << and >> operators).

DEC
ios& dec(ios &)
Sets decimal base.

FILL
char fill()
char fill(char)
Sets fill character (if provided), returns previous fill character.

FLAGS
long flags()
long flags(long)
Returns current flags, or if flags provided returns previous flags.

HEX
ios &hex(ios &)
Sets hexidecimal base.

OCT
ios &oct(ios &)
Sets octal base.

PRECISION
int precision()
int precision(int)
Sets number of significant digits (if provided), returns current/previous value.

RESETIOSFLAGS
ios &resetiosflags(long) -- requires <iomanip.h>
Turns off specified flags.

SETBASE
ios &setbase(int) -- requires <iomanip.h>
Sets numerical base based on integer provided.

SETF
long setf(long bitFlags)
long setf(long bitFlags, long bitField)
Clears bitField and sets format flags, returns previous flags.

SETFILL
ostream &setfill(char) -- requires <iomanip.h>
Sets fill character.

SETIOSFLAGS
ios &setiosflags(long) -- requires <iomanip.h>
Sets format flags.

SETPRECISION
ios &setprecision(int) -- requires <iomanip.h>
Sets number of significant digits.

SETW
ios &setw(int size) -- requires <iomanip.h>
Sets size (width) of line buffer.

UNSETF
long unsetf(long)
Turns off specified flags and returns previous flags.

WIDTH
int width()
int width(int minimum)
Sets minimum field width if provided (zero = no minimum). Returns current width. Reset to zero after each insertion/extraction.