C++ I/O Stream Classes

Steam classes are more extensible, simpler to use and consistent i.e. they work the same way for console i/o as they do for file i/o

C++ I/O Streams Basics

Writing Output with <<

`cout` used to write to console. Modifiers can be used to change how data is presented

`cout << modifier << ...`

Stream manipulators

- `hex`: change format to hex

Reading Input >>

`cin` is used to read data in. Objects received my be *lvalues*, except in the case of C-strings which are addresses.

For strings, input is read up to and not including the first whitespace.

Limitations:

- `cin`: no input appears until the user presses enter
- non reliable input causes the program to freeze
- no more reads are possible until the error flag is cleared
- leading white-spaces are ignored

`>>` should mainly be used in simple cases

Reading Line Input with `getline`

Two major versions of `getline`

- `istream_obh.getline(cstr, n [, delim_ch = '\n'])`: reads line input into a C-string
- `getline(istream_obj, str_obj [, delim_ch = '\n'])`: reads a line of input into a `string` object

`getline` member to read input into a C-string. `getline` global function to read a line of input into a `string` object.

To interpret complex input, input can be tokenised with `strtok`

C++ Stream Class Hierarchy

`cout` object is an instance of the `ostream` class - derived from the base class `ios`.

Stream output objects are instances of `ostream`, or in the case of file output, `ofstream`. Input is derived from `istream` and `ifstream`.

Stream objects include

- `cout`: Console output, instance of `ostream`, can be written to with the `<<` operator.
- `cin`: Console input, instance of `istream`. Input is not available until `Enter` is pressed. Stream input operator `>>` skips over leading spaces, errors can be tested in a `while` loop
- `cerr`: Console error output, instance of `ostream`, can be redirected to any file.
- `clog`: Console log output. Instance of `ostream`

Stream Objects: Manipulators and Flags

Many ways to control and format output - `<iostream>` header gives access to most predefined objects, `<iomanip>` provides access to `set` and `reset` manipulators.

Stream Manipulators

Manipulators modify a stream in some way - they change the format of subsequent operations on the stream until the format is reset.

Manipulators fall into a number of categories:

- Manipulators that set the base
- Manipulators that set flags
- Manipulators that set the width, fill and precision
- Affect floating point behaviour
- Control print justification
- Termination `endl`, `ends`

Stream manipulators:

- `boolalpha`: causes true/false to be printed as `true` / `false`
- `dec`: Decimal format for integers
- `endl`: prints a newline character
- `ends`: prints a null character
- `fixed`: fixed point format fro floating point integers
- `flush`: flushes the buffer associated with the stream
- `hex`: specifies hex format for integers
- `internal`: sets print-field justification to internal rather than left / right
- `left`: print fields are left-justified
- `noboolalpha`: turns off `bools` printed as `true` / `false`
- `noshowbase`: turns off `showbase` setting, octal and hex prefixes are not printed

- `noshowpoint`: turns off `showpoint` flag, floating point numbers are not forced to be printed with a `.`
- `noshowpos`: turns off `showpos` flag positive signed numbers are not displayed with a `+`
- `noskipws`: turns off `skipws` flag, leading spaces are not skipped
- `nounitbuf`: turns off `unitbuf` flag, which causes stream to be flushed after every read / write
- `nouppercase`: turns off hex and exponents being printed in uppercase letters
- `oct`: octal as integer format
- `resetiosflags(bitmask)`: clears the format flags indicated by the *bitmask* arg
- `right`: print fields are right-justified
- `scientific`: scientific format for floating-point numbers, so exp portion is always displayed
- `setfill(ch)`: sets the fill character to *ch*
- `setiosflags(bitmask)`: sets the format flags indicated by *bitmask* arg
- `setprecision(n)`: Set precision of floating pint numbers to *n* - `showpoint` in effect, precision is total digits to left and right, `scientific` or `fixed`, precision is total digits to the right
- `setw(n)`: sets minimum print field with to *n*
- `showbase`: hex and octal numbers are printed with prefix
- `showpoint`: floating point numbers are printed with the decimal point even if they are whole numbers
- `showpos`: positive numbers are printed with a leading `+`
- `skipws`: leading whitespaces are skipped during stream input operations
- `unitbuf`: Output buffer flushed on each write operation
- `uppercase`: hex and exp printed in uppercase
- `ws`: Reads an ignore whitespaces until non-whitespace character found

Stream Format Flags

Stream flags are used with `setiosflags` and `resetiosflags`, the bitwise OR operator `|` can be used to combine flags

```
cout << setiosflags(ios_base::fixed | ios_base::showpoint);
```

Flags

- `ios_base::boolalpha`: displays `true/false` bool values as “true” and “false”
- `ios_base::dec`: display and read integers as decimal format
- `ios_base::fixed`: display fixed floating point format
- `ios_base::hex`: display and read integers as hex
- `ios_base::internal`: internal justifying for printing
- `ios_base::left`: left-justifying in print fields
- `ios_base::oct`: display and read integers as octal

- `ios_base::right`: right-justify fields
- `ios_base::scientific`: display floating points values in scientific format
- `ios_base::shwobase`: prefix hex and octal
- `ios_base::showpoint`: specifies that floating point number should be printed
- `ios_base::showpos`: display `+` for positive numbers
- `ios_base::skipws`: prevents an input stream from skipping leading whitespace
- `ios_base::unitbuf`: causes input buffer to be flushed with each read / write op
- `ios_base::uppercase`: displays hex prefix and digits as uppercase

Steam Member Functions

Input stream functions (`cin`)

- `istream_obj.gcount()`: number of characters rest by the last call of `getline` or `get`
- `istream_obj.get()`: gets the next character from the input buffer and returns it as an integer
- `istream_obj.get(ch)`: gets next char and puts it in *ch*
- `istream_obj.get(cstr, n [, delim])`: gets up to *n* chars stopping at *delim*
- `istream_obj.getline(cstr, n [, delim])`: reads input into C-string.
- `istream_obj.ignore([n = 1] [, delim = EOF])`: reads up to *n* characters or until the delim is read, and ignores them
- `istream_obj.peak()`: returns the next character in the input buffer without removing it from the buffer, enables look ahead functionality
- `istream_obj.putback(ch)`: Puts a character back into the input buffer, the position counter is decremented by 1 and *ch* is the next character to be read

Output Stream Functions

- `ostream_obj.put(ch)`: puts a character *ch* onto the output stream
- `ostream_obj.fill(ch)`: changes the fill character to *ch*
- `ostream_obj.flush()`: flushed the output buffer so that data in the buffer is written to the file
- `ostream_obj.precision(n)`: sets floating point precision to *n*
- `ostream_obj.width(n)`: specifies the minimum print-field width

Flag-Setting Stream Functions

Functions are a way to set format flags

- `stream_obj.bad()`: returns `true` if bad bit is set, indicating an error
- `stream_obj.clear()`: clears the error flag

- `stream_obj.copyfmt(stream)`: copies the format flag setting from another stream
- `stream_obj.fail()`: returns `true` if the fail bit or bad bit is set, indicating an error
- `stream_obj.flags()`: returns the current state of the format flags as an integer
- `stream_obj.flags(bitmask)`: sets the flag values according to the values in *bitmask*
- `stream_obj.good()`: returns `true` if the "good" bit is set for the file stream
- `stream_obj.setf(bitmask)`: turns on the flags indicated by *bitmask*
- `stream_obj.setf(bitmask1, bitmask2)`: only the flags specified by *bitmask2* are changed
- `stream_obj.unsetf(bitmask)`: switches off the flags indicated by *bitmask*
- `stream_obj.sync_with_stdio([on/off = true])`: turns on sync between stream classes and the C standard IO functions

File Stream Operations

C++ stream classes support both text and binary file I/O. Binary files are more efficient as data does not need to be encoded, and therefore copied directly.

Creating a File Object

`<fstream>` header is required.

- `ifstream`: object for file input
- `ofstream`: object for file output

Mode flags

- `ios_base::app`: Append mode, if the file already exists open it in append mode, otherwise create the file
- `ios_base::ate`: At-End mode, the files position indicator is moved to the end of the file, needs to be combined with `in`, `out`, `app`
- `ios_base::binary`: Binary mode, no translation of newlines is performed
- `ios_base::in`: Input mode (default for `ifstream`), the file must already exist
- `ios_base::out`: Output mode, (default for `ofstream`), if the file does not exist it is created
- `ios_base::trunc`: if the file exists, file contents are erased

File-specific member functions

- `fstream_obj.close()`: closes the file associated with the stream object
- `fstream_obj.eof()`: returns true if the end-of-file condition is detected

- `fstream_obj.is_open()`: returns true if the stream object has a file associated with it open for reading and writing
- `fstream_obj.open(filename [, mode])`: opens a file, function returns a reference to the stream itself

Reading and Writing in Binary mode

- `istream.read(p_data, size)`: Reads *size* bites directly from the stream and stores it at the address specified by *p_data*, a pointer of type `char*`
- `ostream.write(p_data, size)`: writes *size* bites from the address specified by *p_data* directly to the stream.

Random-Access Operations

- `istream.seekg(n [, ref_point])`: moves the stream position indicator to *n* where *n* is the offset, *ref_point* is the initial reference point
- `ostream.seekp(n [, ref_point])`: moves the stream position to *n*, where *n* is the offset, *ref_point* is the initial reference point
- `istream.tellg()`: returns the integer of type `streamoff` containing the current position indicator
- `ostream.tellp()`: returns the integer of type `streamoff` containing the current position indicator

Reading and Writing String Streams

A string stream allows you to leverage all the tools of formatted I/O to strings i.e. it is much easier to write formatted output to a `stringstream` object

```
#include <string>
#include <sstream>
// ....
stringstream s_out;
int n = 255;
s_out << "The value of n in hex is: << hex << showbase << n << endl;
```

- `stringstream name( [mode])`: `stringstream` constructor, initialised with empty string
- `stringstream name(str, [mode])`: `stringstream` constructor, initialised with the contents of *str*
- `stringstream_obj.str()`: returns a `string` objects containing the current text data stored
- `stringstream_obj.str(str)`: copies text data to *str* reference

Overloading Shift Operators

```
ostream& operator<< (ostream &os, const my_class &cls) P
    os << // ...
    return os;
```