# Stream Operators

A “stream” is a stream of bytes flowing in (as in CIN) or out (as in COUT). The stream operators “<<” and “>>” push the builtins to and from bytes as ascii.

These are TEXT processors … to and from TEXT.

int a = 12; // An "int" is 4 bytes

cout << a; // TWO bytes are sent here: '1' and '2'.

There are methods you can do to all byte streams.

There are lots of sources of byte streams. You have interacted with these before:

* Keyboard Stream of bytes to read from
* Monitor Stream of bytes to write to
* Socket TWO streams … one to read and one to write
* File TWO streams … one to read and one to write

Once you know how to get hold of a particular stream you can talk to anything using these common operations.

Streams you write to “<<” are “ostream”. Streams you write to “>>” are “istream”. The specific types of stream inherit from these.

A file stream, for instance, has all of “ostream” plus extra methods to open and close the stream.

# Getting a File Stream

#include <iostream>

#include <fstream>

using namespace std;

int main() {

fstream someFile; // Local object manages the stream

someFile.open("c:\\cpp\_7\_2012\\test.txt", fstream::out);

int a = 12;

//cout << a;

someFile << a << " is a value " << endl << a << endl;

someFile.flush(); // Make sure nothing is in the buffer

someFile.close(); // Close the file

system("pause");

}

Create a stream … use it … then close it.

someFile.open("c:\\cpp\_7\_2012\\test.txt",

fstream::out | fstream::in);

Or the mode constants together:

* in Allow input operations on the stream
* out Allow output operations
* app Set position to end before EACH output operation
* ate Set the position to the end when opening
* binary Consider the stream as binary rather than text
* trunc Discard any existing content when opening

You can set the stream to use EXCEPTIONS when something bad happens. But by default you have to check the fail status:

if(!someFile.is\_open()) { // someFile.fail()

cout << "Could not open the file" << endl;

}

## ostream

// There are several of these modifier functions:

// endl ... print endl

// hex ... switch output format to hexadecimal

// oct ... octal

// dec ... back to decimal

int a = 100;

cout << a << hex << a << oct << a << dec << a << endl;

endl is a function!

endl(cout);

What is “cout << endl;” ??

cout.operator<<(endl);

operator<<(&cout,endl); --> this function calls the function you pass passing cout

// http://www.cplusplus.com/reference/iostream/manipulators/

// There are methods you can call on cout to control text formatting:

cout.width(12); // Next output is 12 characters at least. Then reset width for next output.

cout.fill('\*'); // Set the padding character

cout << 1 << endl; // \*\*\*\*\*\*\*\*\*\*1

// Control of the floating point format

cout.setf(ios::scientific, ios::floatfield);

cout.precision(3); // Digits after the decimal point here

cout << 1.2 << endl;

// Binary in the form of pointer to bytes and a number of bytes.

char\* p = new char[1024];

// Fill up the data

cout.write(p,1024);

Point pnt;

cout.write((char \*)&pnt,sizeof(Point));

## istream

int a;

cin >> hex >> a;

If you type in “-123Z45” then “-123” is read to a. Stops at invalid character, which is next to be read.

char c;

cin.get(ch); // gets a single character

char p[1024];

cin.getline(p,20); // Read a line up to CR but not more than 20

We will discuss “string” shortly … there is a helper function to read a string from a stream with no length requirement.

cin.read(char\*,int size)

Seeking around:

<http://www.cplusplus.com/reference/iostream/istream/seekg/>

## Reading Data

These methods require you know exactly what is coming next in the stream. Text is often dynamic.

* Line-at-a-time into a string and use string methods to parse the string
* XML … use a library to read the file into a model (Document Object Model … DOM)
* Binary … warning! Sizes, endianess, and padding changes. Look at bytes and convert.