Block 11: File Streams

RECAP:

There are:

C++ Streams ("easy", big, clumsy, labor-intensive, overly formalistic).

(You use objects, after learning all the methods they contain.)

\*C Standard Library Formatted (Text). Simple text-based.

\*C Standard Library Unformatted (Binary). Well-known, precise.

(You create and use FILE Pointers, to data structures.)

System Level. Assembly-language level, no real gain in use,

to the contrary: systems streams are unbuffered, so the program

must wait for i/o to complete.

Streams <iostream> are sold as being "type safe", which is fine

if the programmer can't remember the types s/he defined hir

variables to be, but generally, we WANT to "cheat" when doing

file I/O, and ignore type, because we want it to GET DONE,

not sit and grind the disk for minutes on end. Stream I/O

should be used ONLY when it's manifestly the EASIEST way.

Like for handling primitive data types without a lot of

file opening.

Otherwise, use C Standard Library techniques.

Since that comprises most of the code on the planet anyway,

it is a good thing to know well.

This study chapter is divided into two parts:

I. THE QUICK AND DIRTY HOW-TO.

and

II. THE THEORY, IN GREAT DETAIL AND COMPLETENESS.

My opinion: for this course, section I is what you

need, so work on that. After the semester is over,

you can read and re-read every aspect of file i/o,

and you will be a better programmer for doing so.

But for now, I'd focus on just what I needed.

-------------------

PART I. STREAMS QUICK AND DIRTY HOW-TO.

C++ Bare Minimum:

Read: Chapter 17 File I/O Streams.

There is a wealth of options in C++ for reading from

or writing to disk files.

I. std::fstream objects and methods

Generally, it's easy to "create" a stream to read a

given file. You do this by passing arguments to the

stream objects "constructor" like so:

#include iostream

#include fstream

std::ofstream ofile("struct.dat", std::ios::binary | std::ios::app);

(open it to be able to write binary data, and

to APPEND (not overwrite) the file if it's

already there.)

Read: <A HREF="http://209.129.16.61/~hhaller/data/cisc192/modules/iostream\_one">iostream\_one, a review

of cin and cout</A>

Read: <A

HREF="http://209.129.16.61/~hhaller/data/cisc192/modules/ifstream.ofstream.txt">ifstream.ofstream.txt,

how to use streams to read and write text files</A>

Read: <A HREF="http://209.129.16.61/~hhaller/data/cisc192/modules/ostream1.cpp.txt">ostream1.cpp, how to use

stream methods to open, write, read, close files</A>

All ofstream, ifstream, and fstream objects have the

.eof(), .get() and .put() methods we have been using

with cin and cout.

--

NOTE: And that is all the file processing you need

to pass this course.

OK? Calm down. Deep breathely.

-----------------------------

Now, for completeness in your notes, here is the

larger picture:

WHEN WRITING C/C++ PROGRAMS, YOU MAY SAFELY PRETEND

TO BE RUNNING UNDER UNIX, EVEN IF YOU'RE ON A WINDOWS

BOX, BECAUSE THE STANDARD LIBRARY IS WRITTEN THAT

WAY, AND THE ISO STANDARD REQUIRES THAT C/C++

METHODS BE VIABLE ON ALL OPERATING SYSTEMS.

So...even if you're running on Windows, your program

can act like it's on the Linux ext2 filesystem, or

ext3, or ReiserFS, or SYSVfs, or.... You get the idea.

Remember:

There are two types of files: text and binary.

Text files are a "translation" of what was stored

in memory, (text or numbers) into "human readable"

ASCII or UNICODE text which can be edited with Notepad

or a similar editor program.

Binary files are an untranslated "snapshot" of the contents

of memory.

Data Base Management Systems always save their

information in Binary files because to do so is much

faster, both to save and to read back in.

Read: <A HREF="http://209.129.16.61/~hhaller/data/cisc192/modules/structstream.cpp.txt">structstream.cpp,

how to use stream methods to open, write, read,

close files</A>

As an example, if you wanted to write a program which opened

and read a file whose name you placed in the first command-line

argument, you would do something like:

int main (int argc, char \* argv[])

{

string s ;

std::istream infile(argv[1] ) ;

if ( !infile.good()) exit (-1) ;

do

{

infile >> s ;

// some processing here

} while ( ! infile.eof()) ;

infile.close() ;

OK?

NOW: THE WINDOWS SIDE:

Read: <A HREF="http://209.129.16.61/~hhaller/data/cisc192/modules/windows\_filesystem">windows\_filesystem</A>

Read: <A HREF="http://209.129.16.61/~hhaller/data/cisc192/modules/doswindow.commands">doswindow.commands</A>

<A HREF="syllabus.html#Menu">Return to Menu</A>

/\*-----------------------------------------------------------------

\* staticmethod.cpp - run a method without an object to contain it

\*

-----------------------------------------------------------------\*/

#include <cstdio>

#include <cctype>

#include <iostream>

#include <iomanip>

using namespace std ;

class prs

{

public:

int hats , sox, shirts ;

static int getofus(void) { return ofus ; }

public:

static int ofus ; // static instance var counts all objects

prs() { ofus++ ; }

~prs() { ofus-- ; }

} ; // prs

int prs::ofus = 0 ; // must initialize at file scope, or nothing works

int main (int argc, char \*argv[], char \*\*env)

{

//int prs::ofus = 0 ;

cout << "No Objects, running static function getofus(): " << prs::getofus() << endl ;

prs X ;

cout << "One object:X: " << X.getofus() << endl ;

prs \*Y = new prs() ; // default constructor, ofus gets incrimented

cout << "Two objects:X & Y: " << prs::getofus() << endl ;

delete Y ;

cout << "One object?:X : " << prs::getofus() << endl ;

} // main ends

---

"friend" functions are not part of the class, but are

granted special permission to manipulate the class'

"private:" data objects.

class prs {

private: int a[500] ;

public:

friend void showMembers(void) ;

// now showMembers() is able to manipulate the private

//data objects in the class.

} ;

"this": Since all "instance variables" are "global" to

all of a class' member functions, you don't see "this"

as much as you might...but you can still use it, if it

makes code read more clearly to you. Static variables

aren't used with "this". Think about it: if a static

method or data item exists without any object being

around, they don't exist in any object, so "this->"

is irrelevant to them.

Here's running a static method:

/\*-----------------------------------------------------------------

\* staticmethod.cpp - run a method without an object to contain it

\*

-----------------------------------------------------------------\*/

#include <cstdio>

#include <cctype>

#include <iostream>

#include <iomanip>

using namespace std ;

class prs

{

public:

int hats , sox, shirts ;

static int getofus(void) { return ofus ; }

public:

static int ofus ; // static instance var counts all objects

prs(int n) { ofus = n ; }

prs() { ofus++ ; }

~prs() { ofus-- ; }

} ; // prs

int prs::ofus = 0 ; // must initialize at file scope, or nothing works

int main (int argc, char \*argv[], char \*\*env)

{

//int prs::ofus = 0 ;

cout << "No Objects, running static function getofus(): " << prs::getofus() << endl ;

prs X(1) ;

cout << "One object:X: " << X.getofus() << endl ;

prs \*Y = new prs() ; // default constructor, ofus gets incrimented

cout << "Two objects:X & Y: " << prs::getofus() << endl ;

delete Y ;

cout << "One object?:X : " << prs::getofus() << endl ;

} // main ends

====================================================

"this":

class prs {

private:

int a[50] ;

int size ;

int sort(void) ;

} ;

prs::sort(void)

{

int i, j ;

for (i = 0 ; i < size ; i++)

{

if (this.a[i] < this.a[i+1])......

I use "this" a lot to find the ADDRESS of a given OBJECT

when I'm doing dynamic list management.

...and that's enough class for the moment.

DYNAMIC ALLOCATION:

(Like we haven't seen this before.)

The old (C) way used void \* malloc(int) or void \* calloc(int)

and free(). Modern memory management uses new and delete.

Which ever you pick, USE THEM CONSISTENTLY. If you "new"

an object, don't free() it, and if you malloc() something,

don't delete it.

int a[34] ; // staticly allocated an array of ints.

int \* a = new int[34] ; // dynamically allocated an array of ints.

To delete an array, use brackets: delete[] a ;

----

Container classes and iterators.

You already know about containers, at least vaguely, since

you've been using arrays. Arrays are kind of simple containers

for basic types. The difference is, arrays are "indexed" or

"iterated through" with an INTEGER EXPRESSION.

int a[45] , i ;

a[i] = 0 ; // the int i is the iterator for simple arrays.

Some objects don't take simple int iterators, they take

an iterator defined in their class declaration. The iterator

knows just how many bytes to skip to get to the next object

of that class in memory.

prs::iterator p ; //define a new iterator

/\*-----------------------------------------------------------------

\* 01fileio.cpp - stream i/o.

\* Written Tue Jul 16 16:22:07 PDT 2002 TEK.

Generalized, A&D exception code, (pick an integer)

-----------------------------------------------------------------\*/

#include <iostream>

#include <fstream>

#include <stdlib.h>

#include <cstring>

using namespace std;

int main (int argc, char \*argv[], char \*\*env)

{

ifstream instream ;

ofstream outstream ;

static char c, s[81] ;

int a, b ;

try

{

// append to the file

outstream.open ("output.file", ios::app) ;

// did we open the file OK?

if (outstream.fail())

{

throw 20 ;

}

outstream << "appending to file." ;

outstream << endl ;

outstream.close() ;

}

catch (int e)

{

cout << "An exception occurred. Exception Nr. " << e << endl;

return -1 ;

}

strncpy(s,"Hello, World!\n", 13) ;

}

/\*-----------------------------------------------------------------

\* 02fileio.cpp - stream i/o.

\* Written Tue Jul 16 16:22:07 PDT 2002 TEK.

-----------------------------------------------------------------\*/

#include <iostream>

#include <fstream>

#include <stdlib.h>

using namespace std;

int main (int argc, char \*argv[], char \*\*env)

{

ifstream in\_s ;

ofstream ou\_s ;

char c , instring[81];

int a, b ;

in\_s.open (argv[1]) ; /\* open first cmd line arg for input \*/

if (in\_s.fail())

{

cout << "Error opening input file\n" ;

exit(-1) ;

}

ou\_s.open ("output.file") ;

if (ou\_s.fail())

{

cout << "Error onpening output file\n" ;

exit(-1) ;

}

while (in\_s >> instring ) /\* read from a file \*/

{

cout << instring ; /\* write to screen \*/

ou\_s << instring ; /\* write to a file \*/

}

ou\_s.close() ;

} // End main()

/\*-----------------------------------------------------------------

\* iostreams.cpp - demo C++ io methods

Thu Aug 14 17:48:30 PDT 2003

ifstream for input

ofstream for output

-----------------------------------------------------------------\*/

#include <stdio.h>

#include <iostream>

#include <fstream>

#include <stdlib.h>

using namespace std;

int main (int argc, char \*argv[], char \*\*env)

{

//////////////////////////////////////////

// simple i/o with stream objects

//////////////////////////////////////////

one: {

string s ; char c ;

ofstream ofs("test.cpp.dat", ios::out) ;

ofs << "Hello, World!\n" ;

ofs.close() ; // must close before reading

ifstream ifs("test.cpp.dat", ios::in) ;

if (!ifs) { cout << "cannot open " << argv[1] << " for input\n" ;}

while(ifs.get(c)) { cout.put(c) ; }

ifs.close() ;

} // one

//////////////////////////////////////////

// character and line counter

//////////////////////////////////////////

two: {

char buffer[128] ;

long nc = 0 , nl = 0 ;

ifstream ifs2(argv[1], ios::in) ;

if (!ifs2) { cerr << "Error, unable to open " << argv[1] << endl ; }

while (! ifs2.eof() ) {

ifs2.getline(buffer, sizeof(buffer), '\n') ;

if (!(ifs2.eof() && strlen(buffer) == 0)) {

nc += strlen(buffer) ;

nl ++ ;

cout << buffer << endl ;

}

}

cout << endl << endl << "Total characters: " << nc ;

cout << endl << "Number of lines: " << nl << endl ;

return 0 ;

} // two

} // main

/\*-----------------------------------------------------------------

\* format\_flags.cpp

\* you can control streams like printf() but with a lot less grace

-----------------------------------------------------------------\*/

#include <cstdio>

#include <cctype>

#include <iostream>

#include <iomanip>

#include <fstream>

#include <cstdlib>

using namespace std ;

int main (int argc, char \*argv[], char \*\*env)

{

double e = 6200000000 ;

cout << "e: " << e << endl ;

cout << "e: " << setprecision(3) << setw(6) << e << endl ;

cout.setf(ios\_base::scientific) ;

cout << "e: " << e << endl ;

return EXIT\_SUCCESS ;

} // main ends

/\*-----------------------------------------------------------------

\* iostreams3.cpp - sometimes, you want to fall back to

\* the standard output method when you

\* fail to open a file or stream.

run both with and without a command-line filename to write.

Wed Jul 7 20:03:14 PDT 2010

cin and cout are istream and ostream classes.

-----------------------------------------------------------------\*/

#include <iostream>

#include <fstream>

using namespace std;

int main(int argc, char \*argv[])

{

/\*-----------------

\* if you fail to open the filebuf, default

\* to using cout.

-----------------\*/

filebuf fb;

fb.open (argv[1],ios::out);

ostream \* c ;

if(fb.is\_open())

c = new ostream(&fb); // point istream at open buffered file

else

c = &cout ; // point to cin

\*c << "Hello world!" << endl;

return 0;

} // main ends

/\*-----------------------------------------------------------------

\* randomlines.cpp - pull a number of random lines out of a file

Begun Thu Sep 22 10:16:59 PDT 2011 TEK

-----------------------------------------------------------------\*/

#include <cstdio>

#include <cctype>

#include <iostream>

#include <fstream>

#include <iomanip>

using namespace std ;

int main (int argc, char \*argv[], char \*\*env)

{

if (argc != 2)

{

cerr << "randomlines 1/fraction\_of\_lines filename\n" ;

exit(-1) ;

}

srand(time(NULL)) ;

int i , percentage;

ifstream input(argv[2]) ;

if (input.fail()) exit(-1) ;

char buf[1024] ;

percentage = atoi(argv[1]) ;

while (!(input.eof()))

{

input.getline(buf, sizeof(buf)-1, '\n') ;

if (!(rand() % percentage))

cout << buf << endl ;

} // while

} // main

/\*----------------------------------------------------------

\* frdfwt.c - C Standard Library Unformatted

\* File I/O.

\* Create an array, write it in one operation to a binary file.

\* destroy it, read it back in with one binary operation.

\* TEK.

----------------------------------------------------------\*/

#include <string.h>

#include <stdio.h>

int main(void)

{

FILE \* fp ;

int array[250], x ;

/\* fill array with consecutive numbers \*/

for(x = 0 ; x < 250 ; x++)

array[x] = x ;

/\*

\* create file, but open it so that we can read

\* what we write to it as well. ("w+")

\*/

if ((fp = fopen("TEST.FIL", "w+")) == NULL) {

fprintf(stderr,"Cannot open output file.\n");

return 1;

}

/\*

\* write the whole array on the file in one burp

\*/

fwrite(array, sizeof(int), 250, fp); /\* write whole array to disk \*/

/\*

\* rewind the file to the start

\*/

fseek(fp, SEEK\_SET, 0); /\* rewind file \*/

for(x=0; x<250; x++) /\* zero out array \*/

array[x] = 0 ;

fread(array, sizeof(int), 250, fp); /\* read whole array \*/

for(x=0; x<250; x++) /\* prove it was read successfully \*/

printf("array[%d] : %d\n", x, array[x]) ;

fclose(fp);

return 0;

} /\* main ends \*/

/\*-----------------------------------------------------------------

rndtst.c - random access input and output from disk

This code illustrates that a file may be written as well as read

in ramdom order. The C techniques for laying down file records are

extremely flexible, and uniquely suited to database applications.

-----------------------------------------------------------------\*/

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <ctype.h>

#include <stdio.h>

main()

{

struct one

{

char line[20] ;

unsigned int num ;

} array[5] = { "this will never",0,

"come to pass",1,

"a back seat driver",2,

"out of gas",3,

"Burma-Shave!",4,} ;

FILE \*fp ;

unsigned int x, filepos[] = {3,1,4,2,0} ; /\* order to write items in \*/

if ((fp = fopen("test.dat","w")) == NULL)

{

puts("Error opening test.dat for write!") ;

exit(1) ;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*WRITE THE RECORDS IN THE ORDER OF filepos[]\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

for ( x = 0 ; x < 5 ; x++)

{

if (fseek(fp,(long) (sizeof(struct one) \* filepos[x]), 0) == -1L)

{

puts("Seek failed!") ;

exit(1) ;

}

fwrite(&array[x], sizeof(struct one), 1, fp) ;

} /\* for \*/

fclose(fp) ;

/\*\*\*\*\*\*\*\*\*DUMP THE DISKFILE IN THE ORDER IT APPEARS ON DISK\*\*\*\*\*\*\*\*\*\*\*\*\*/

if ((fp = fopen("test.dat","r")) == NULL)

{

puts("Error opening test.dat for read!") ;

exit(1) ;

}

puts("Order it occurs on disk file: ") ;

for ( x = 0 ; x < 5 ; x++)

{

if (fseek(fp,(long) (sizeof(struct one) \* x), 0) == -1L)

{

puts("Seek failed!") ; exit(1) ;

} /\* if \*/

fread(&array[x], sizeof(struct one), 1, fp) ;

printf("%u %s\n",array[x].num, array[x].line) ;

} /\* for \*/

/\*\*\*\*\*\*\*SEEK AND PRINT THE RECORDS IN THE ORDER THEY WERE WRITTEN\*\*\*\*\*\*\*\*/

puts("\n\nBut fetching it in random order: \n") ;

for ( x = 0 ; x < 5 ; x++)

{

if (fseek(fp,(long) (sizeof(struct one) \* filepos[x]), 0) == -1L)

{

puts("Seek failed!") ;

exit(1) ;

} /\* if \*/

fread(&array[x], sizeof(struct one), 1, fp) ;

printf("%u %s\n",array[x].num, array[x].line) ;

} /\* for \*/

getchar() ;

} /\* main \*/

/\* $ hexdump test.dat

HEXDUMP Written in Turbo C by T. E. Harrisburg.

00000000: 42 75 72 6D 61 2D 53 68 - 61 76 65 21 00 00 00 00 - Burma-Shave!....

00000010: 00 00 00 00 04 00 00 00 - 63 6F 6D 65 20 74 6F 20 - ........come.to.

00000020: 70 61 73 73 00 00 00 00 - 00 00 00 00 01 00 00 00 - pass............

00000030: 6F 75 74 20 6F 66 20 67 - 61 73 00 00 00 00 00 00 - out.of.gas......

00000040: 00 00 00 00 03 00 00 00 - 74 68 69 73 20 77 69 6C - ........this.wil

00000050: 6C 20 6E 65 76 65 72 00 - 00 00 00 00 00 00 00 00 - l.never.........

00000060: 61 20 62 61 63 6B 20 73 - 65 61 74 20 64 72 69 76 - a.back.seat.driv

00000070: 65 72 00 00 02 00 00 00 - - er......

\*/