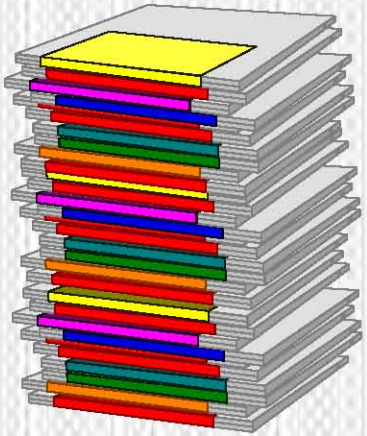
Chapter 7: Files

01000001



Mr. Dave Clausen

La Cañada High School

Interactive vs. Batch Processing Programs

- Interactive Program
  - what we have used to date
  - the program halts, and waits for a user to enter data from the keyboard, then proceeds...
- Batch Processing Programs
  - noninteractive input and output
  - the user and computer do not interact while the program is running
  - the data is stored as a separate file on a disk or HD

Streams

- Stream
  - a channel or conduit where data are passed to receivers from senders.
- Output Stream
  - a channel where data are sent out to a receiver
  - cout; the standard output stream (to monitor)
  - the monitor is a *destination* device
- Input Stream
  - a channel where data are received from a sender
  - cin; the standard input stream (from the keyboard)
  - the keyboard is a *source* device

Stream Processing

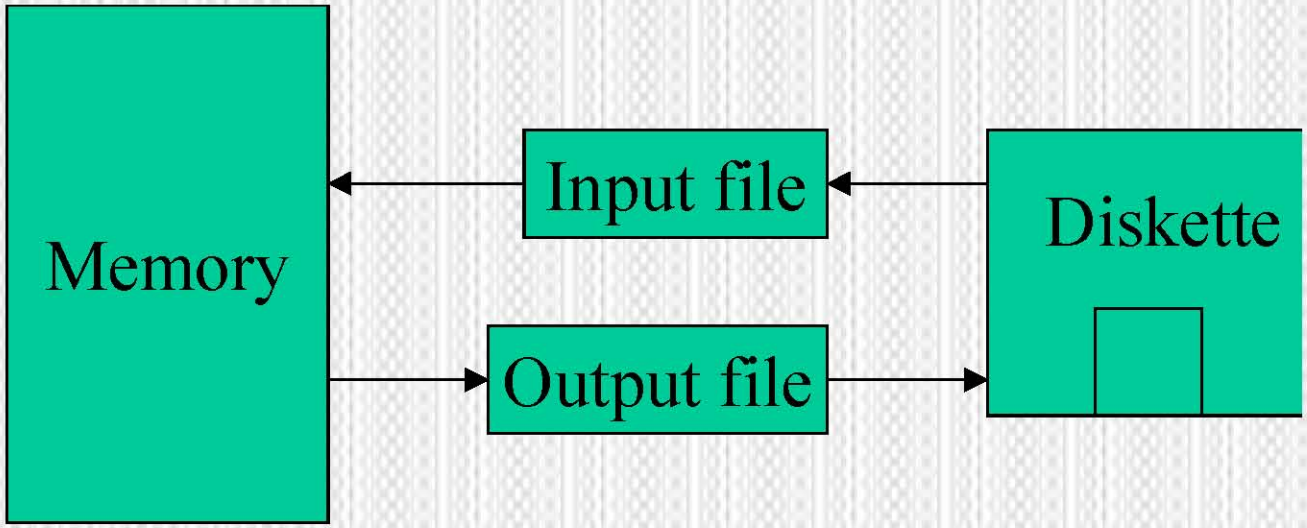
- Serial Processing
  - data elements must be sent to or received from a stream one element at a time
- Five operations necessary for stream processing
  - the stream must be opened for use
  - if it's an input stream, *get* the next element
  - detect the end of the input stream
  - if it's an output stream, *put* the next element
  - close the stream

Standard Input / Output Streams

- Standard Streams use #include <iostream.h>
  - Standard Input Stream
    - cin names the stream
    - the extractor operator >> extracts, gets, or receives the next element from the stream
  - Standard Output Stream
    - cout names the stream
    - the inserter operator << inserts, puts, or sends the next element to the stream
  - Opening & closing of the standard streams occur automatically when the program begins & ends.

Files

Used to transfer data to and from disk



File Streams

- Files
  - data structures that are stored separately from the program (using auxiliary memory)
  - streams must be connected to these files
  - Input File Stream
    - extracts, receives, or gets data from the file
  - Output File Stream
    - inserts, sends, or puts data to the file
  - #include <fstream.h> creates two new classes
    - ofstream (**o**utput **f**ile stream)
    - ifstream (**i**nput **f**ile stream)

Output File Streams

- #include <fstream.h>
  - allows use of the two classes: ofstream, ifstream
- ofstream out_file;
  - a variable or object (out_file) is declared to be of type or of the class ofstream
- out_file.open("myfile.dat");
  - connect the output file stream to a file on the disk **in the default directory** named "myfile"
  - dot notation calls the member function "open"
  - if "myfile" exists it is opened for output & connected to the data stream out_file. If data is there, it is erased!
  - If "myfile" doesn't exist, it is created & connected to the data stream out_file

Output File Streams

- General Form for output file stream

  ofstream <stream variable name>;

  <stream variable name>**.open**(<file name>);

  - the stream variable name can be any valid C++ identifier (**out_file** is a good name to use)
  - the file name must be a string that follows the conventions of your operating system (**8.3**)

- The file stream should be closed when you are finished. If out_file is the variable name then:
  - out_file.close( ) //no file name parameter used

String Variables for File Names

- To input the name of the file that the user would like to open.

```
#include <fstream.h>
#include "apstring.h"
ofstream out_file;
apstring file_name;
cout<<"Enter the output file name: ";
cin>>file_name;  //use cin to avoid white space
out_file.open(file_name);
//the open function expects a "C" style string, not a C++ string
out_file.open(file_name.c_str( ));
//converts C++ file name to C file name (using an apstring
  member function, hence the dot notation)
```

Errors Opening & Closing Files

- C++ has a "fail" function for use with file streams

```
#include <fstream.h>
#include <assert.h>
. . .
ofstream out_file;
. . .
out_file.open("myfile.dat");
assert( ! out_file.fail( ));
//send data to the file
out_file.close( );
assert( ! out_file.fail( ));
```

Output Streams: Point 1

- Operations on output streams are abstract.
  - Abstract: hiding the details
  - It doesn't matter if the output stream is a file on a disk or the monitor screen.
  - We only need to know the name of the stream to send the data to the stream
    - use cout for the monitor or out_file for a data file
    - out_file<<"This is going to the data file.";
    - cout<<"This is going to the monitor screen.";

Output Streams: Point 2

- Programs using output streams are portable.
  - Portable: can be transferred to another application or computer platform, and be recompiled without having to change the code.
  - This works even if the different platform uses a different method of saving files to a disk.
  - Of course, different systems may have different requirements for any filenames that are used.

Loops and Output File Streams

- One form of data processing:
  - inputs data from the user at the keyboard (standard input stream)
  - processes the data
  - writes the results of the data processing to a file (output file stream)
    - It is essential to separate each data item by a whitespace character when writing to the disk.
      - Using spaces, tabs, or carriage returns

kbdfile.cpp kbdfile.txt

Input File Streams

- #include <fstream.h>
  - allows use of the two classes: ofstream, ifstream
- ifstream in_file;
  - a variable or object (in_file) is declared to be of type or of the class ifstream
- in_file.open("myfile.dat");
  - connect the input file stream to a file on the disk in the default directory named "myfile"
  - dot notation calls the member function "open"
  - if "myfile" exists it is opened for input & connected to the input data stream in_file.
  - If "myfile" doesn't exist, it is created & connected to the data stream in_file (on some compilers)

Input File Streams

- General Form for input file stream

  ifstream <stream variable name>;

  <stream variable name>**.open**(<file name>);

  - the stream variable name can be any valid C++ identifier (**in_file** is a good name to use)
  - the file name must be a string that follows the conventions of your operating system (8.3)

- The file stream should be closed when you are finished. If in_file is the variable name then:
  - in_file.close( ) //no file name parameter used

Using Input File Streams

- Input Streams
  - we use the >> extractor operator to get data from the keyboard
  - With the Standard Input Stream
    - The user enters characters from the keyboard followed by a blank space, tab, or carriage return
    - The computer converts the characters into the data type represented by the identifier used with the cin statement
      - int, double, char, apstring
    - The computer stores the data in the variable following the >> (extractor) operator
    - Success depends on: 1) the format of the data typed by the user and 2) the matching data type of the variable

Using Input File Streams

- Input Streams
  - we use the >> extractor operator to get data from the data file
  - With the Input File Stream
    - Some user has already entered characters from the keyboard followed by a blank space, tab or carriage returns into a data file
    - We need to know
      - what type of data is stored in the file
        - int, double, char, apstring
      - what order the data is stored
      - sometimes even what type of whitespace separates the data
        - blank spaces, tabs, carriage returns

Filescr.ide
Filescr.cpp
Filescr.txt

Loops And Input File Streams

- We don't always know precisely how many data values are in a file, or if any values are in a data file.
  - We do need to know the general format of the file:
    - what type of data it contains,
    - the order that these data types are arranged
  - We use a pretest indefinite loop to read data **while** the end of the file has **not** been reached (**eof**)
    - **eof** returns **true** when it reaches the "end of file marker", and there is no more data to be read.
    - Otherwise **eof** returns **false**

Input File Stream Example

```
in_file>> data;
while( ! in_file.eof( ))
{
      process (data);
      in_file>> data;
}
```

- Use a "primed" while loop. You must read the first data item before checking to see if it is the eof.
- A while loop (pretest) will prevent trying to read data beyond the eof marker. (the first item could be eof, in an empty file.)
- Don't forget to read the data again inside the loop to update the Boolean expression and avoid infinite loops.

Compiler Differences

- Some compilers will indicate that the eof has been reached if the file stream member function fail ( ) returns true.
  - Use this compound Boolean Expression to guard against this:

```
while( (! in_file.fail( ) ) && ( ! in_file.eof( ) ) )
```

This sample program reads integers from an input file and displays them in a column on the screen.

intfile.cpp intfile.txt

Using Functions With Files

- Some useful functions:

```
void open_input_file(ifstream &in_file)
//always pass file streams as a reference parameter:  & in_file
{
    apstring in_file_name;                  //local variable

    cout << "Enter the input file name: ";
    cin >> in_file_name;
    in_file.open(in_file_name.c_str( ));
    assert(! in_file.fail( ));
}
```

Using Functions With Files 2

- Some useful functions:

```
void open_output_file(ofstream &out_file)
//always pass file streams as a reference parameter:  & out_file
{
   apstring out_file_name;                  //local variable

   cout << "Enter the output file name: ";
   cin >> out_file_name;
   out_file.open(out_file_name.c_str( ));
   assert(! out_file.fail( ));
}
```

Using Functions With Files 3

- Some useful functions:

```
void copy_integers(ifstream &in_file, ofstream & out_file)
{
    int data;
    in_file >> data;
    while (! in_file.eof( ) && ! in_file.fail( ))
    {
        out_file << data << " ";
        in_file >> data;
    }
}
```

Intcopy.ide
Intcopy.cpp
Intcopy.txt

Files and Strings

- String processing is an important part of word processing and data base applications.
- Files containing strings can be processed:
  - a string (word) at a time using an apstring variable
  - a line of strings at a time
    - using getline from the apstring library
    - getline is limited to 1024 characters per line
    - remember with getline, the end of line character is not stored with the actual line of text
  - or a character at a time
    - necessary to handle white space characters

Processing string by string

```
//Consider searching through a file of strings looking for a desired string (word)
void search_for_word(ifstream &in_file, const apstring &desired_word,
                                        int &position, bool &word_found)
{
   apstring input_word;  //error in text P389, not in corrections:  apstring
   in_file >> input_word;
   ++position;
   while( (! in_file.eof( )) && (input_word != desired_word))
   {       //order of above Boolean Expression is crucial, short circuit evaluation
           //We must make sure there is another string to compare.
     in_file>> input_word;
     ++position;
   }
   word_found = ! in_file.eof( );
}
```

Processing line by line

- getline reads characters and stores them into a string variable until the end of line character is reached or until the getline limit of 1024 characters have been read.
- getline supports "buffered file input"
  - the input of large blocks of data from a file into a "buffer"
  - a "buffer" is a block of memory of a definite size where data is placed while waiting to be sent to the program.
  - The main advantages are efficiency and speed
  - The main disadvantage is that some data may exceed the limits of the buffer (1024 characters per line)

Buffcopy.ide **Buffcopy.cpp** **Buffcopy.txt**

Processing Character by Character

- Some problems call for the input and output of individual characters.
  - Counting characters in a file
  - Counting lines if a file
- If our data includes white space (space, tab, carriage returns) we can't use the >> extractor operator.
  - The extractor operator >> treats white space as separators between data values in an input stream.
  - Therefore, we can't use >> to input or process white space characters as their own data values.
  - C++ includes two commands to process data a character at a time including processing the white space characters.

Character Output with "put"

- General format for "put" statement
  - <output file stream>**.put**(<character value>);
  - put is called as a function with a character value as its parameter.
  - The dot notation associates the member function call with the output stream
  - put is defined so that it can be used with any output stream

```
for(char ch = 'a'; ch <= 'd'; ++ch)
  cout .put(ch);
```

Character Input with "get"

- General format for "get" statement
  - <input file stream>.**get**(<character value>);
  - get is called as a function with a character value as its parameter.
  - The dot notation associates the member function call with the input stream
  - get is defined so that it will treat a blank space, tab, or carriage return as valid character data
  - Whether a white space character or other character, the following will place the first character of a file into the input stream:
    in_file.**get**(ch);

Detecting eof at the character level

- A special character marks the end of a file.
  - In an empty file, this is the only character present.
  - It is important to detect this character and not try to read any data beyond this marker.
  - When "get" reads the eof function it returns a Boolean value of true.
  - Standard form for processing a file, character by character is:

<input stream name>**.get** (<character variable>);
while(! <input stream name>.eof( ))
  process_data(<character variable>);
  <input stream name>**.get** (<character variable>);

Examples of Character Processing

- A program to count the number of characters in a file. (Ex 7.5)

  charcnt.cpp charcnt.txt

- A program to count the number of lines in a file, by counting the number of carriage return characters '\n' (Ex 7.6)

  linecnt.cpp linecnt.txt

- A program to copy one file into another, copying character by character. (Ex 7.7)

  copyfile.cpp copyfile.txt

Saving Files to Other Paths

- All of the examples from the text, have opened and closed files in the default directory.
- We are using a network server to save all our programs and data files. You will have to save your programs and data files to our "s:" , "q:", and maybe even "t:" network mappings. You may also have to read data files from our "r:" network mapping.
- So, how do we do this?

Change the default directory: Part 1

- When you save your program's source code using the File menu / Save as… command, to any of our network mappings, that becomes the default directory.
- Then follow the textbooks examples specifying the filename without a path.
- Don't forget to add the extension .dat to your filenames (the text doesn't do this).
- This will only work for opening and closing files in this directory.
- What do we do if we want to read a file in one directory and copy it to another directory?

Change the default directory: Part 2

- We can specify the path as well as the filename in the same command when opening files for reading and / or writing.
- In our system, if you are saving to another directory at the root level, use one backslash

```
in_file.open("s:\9999p7a.dat");
out_file.open("s:\9999p7b.dat");
```

- If you need to read or save files in a subdirectory, use double backslashes

```
in_file.open("r:\\data\\intfile.dat");
out_file.open("q:\\prog7\\9999p7b.dat");
```

pathcopy.cpp pathcopy.txt

Change the default directory: Part 3

- We can specify the path as well as the filename in the same command when opening files for reading and / or writing.
- When we allow the user to enter the name into an apstring variable, we need to use one backslash:

```
cout << "Enter the input file name: ";
cin >> in_file_name;
in_file.open(in_file_name.c_str( ));
```

- The user would type:

r:\data\intfile.dat

q:\prog7\9999p7b.dat

intcopy.ide intcopy.cpp intcopy.txt