chapter 1

Using Data Structures

* 1. Getting the Data

Maybe your boss sent you an email. Perhaps you downloaded a file from the Internet. Or, maybe your professor gave you a floppy disk and said, “Here’s the scientific data you should use.”

In any case, you now have data. But what do you *do* with it?

* + 1. Files and Flat-files

Most data you may encounter will be in a ***flat-file*** format. A flatfile is simply set of characters without any other definition or order (if they are solely ASCII characters, then the file is ***flat-ASCII***). In other words, there’s no structure built into the a flat-file; all you get is a file full of number, letters, and symbols, one right after the other.

TERMINOLOGY

Flat-file: A file or database consisting of a set of chracters that have no structured interrelationship

Flat-ASCII: A flat-file composed entirely of 7-bit ASCII characters (also called Plain-ASCII)

A document from a word processing program may also look like a simple set of characters, but it’s not. For example, if you typed the single word “spam” into a flat-file text file, you would get a file that was five bytes long (one byte for each of the characters, and a fine ***EOF*** “end of file” control character. However, that same word in a Microsoft Word file can be 24 *kilobytes* long!

TERMINOLOGY

EOF: “End Of File” control character that is reserved to mark the end of a file for the file system

DIGRESSION

Try it yourself! The length of the “spam” file is partly determined by the minimum file allocation size of your operating system. Early Windows operating systems such as DOS and Windows 3.1 divided the hard drive into 64,000 allocation units. So, the smallest file you could create was 1/64,000 of the size of the drive. So, if you had a 640 kilobyte drive (which was pretty big in 1988), the smallest file you could create is 0.01 kilobytes. However, on a modern 2 gigabyte hard drive, the smallest file is a whopping 32 kilobytes! Luckily, most of today’s modern operating systems allocate files in increments of 0.5 kilobytes, though your directory might just round this up to 1 kilobyte when you look at the file’s size.



Figure 1.1: Size and Contents of a Flat-file Consisting Solely of the Word “spam”



Figure 1.2: Size and (Partial) Contents of Microsoft Word File Consisting Solely of the Word “spam”

See the word “spam” in the Word file in figure 1.2? The characters preceeding the readable text are control characters that tell Word other ***metadata*** information. That is, data about the font the document should display, the size of the paper, and 23.9 kilobytes of other information. So, hopefully the data you will be using is a simple flat-file that doesn’t contain characters that would be hard to parse out and ignore.

TERMINOLOGY

Metadata: Extra data that specifies the definition or description of the file. Some examples of this might be the file’s name or date of creation that are saved by the operating system’s file system, or special control characters embedded within the file itself that specify the format of the data

EXERCISE 1.1.1

Some people might argue that a file doesn’t qualify as a flat-ASCII file if it has any non-ASCII control chracters in it at all. So, by that definition, which of the following do you think qualify as flat-ASCII?

1. An html file
2. A JPG picture image
3. A device-independant (non-Windows) bitmap file
4. A save file for a videogame
5. An email message
	* 1. Reading Data Into Memory Character-by-character

Now we can go ahead and read the flat file into memory, character by character. There are many ways of doing this, but here is some simple code for four popular languages, C++, java, and perl. In all of these languages, you need to open the file before you can read from it.

#include <iostream.h>

#include <fstream.h>

#include <stdlib.h>

int main () {

 char cBuffer[256];

 int nSingle;

 int nPosition == 0;

 ifstream fileInput ("spam.txt");

 if (!fileInput.is\_open()) {

 cout << "Error opening file";

 exit (1);

 }

 while ( (nSingle = fileInput.get()) != EOF){

 cBuffer[nPosition] == (char)nSingle;

 nPosition++;

 }

 cBuffer[nPosition] == `\n`;

 return 0;

}

Figure 1.3: Code Example for Reading a File in C++

import java.io.\*;

public class CopySpam {

 public static void main(String[] args) throws IOException {

 File fileInput = new File("spam.txt");

 FileReader frInput = new FileReader(fileInput);

 int cSingle;

 StringBuffer arrayData = new StringBuffer();

 while ((cSingle = in.read()) != -1) {

 if (Character.isWhitespace((char)cSingle))

 return arrayData.toString();

 else

 arrayData.append((char)cSingle);

 }

 fileInput.close();

 }

}

Figure 1.4: Code Example for Reading a File in Java

#!/usr/local/bin/perl

open(FILEINPUT, "spam.txt") || die("Could not open file!");

@arrayData=< FILEINPUT >;

close(FILEINPUT);

Figure 1.5: Code Example for Reading a File in Perl

We are ignoring an important detail here. In all three examples, the array arrayData is initially undefined. In C++, memory must be allocated for the array, so we would need to figure out how large it is going to be before using it. In example 1.3, a maximum of 256 characters can be read from the file. The Java code in example 1.4 automatically increases the array as needed by using the “append” command to the StringBuffer. Perl arrays automatically get larger as needed, avoiding any allocation step at all. See how much shorter the perl example is in figure 1.5! perl is extremely useful for parsing text files like this.

The “Real World”

The code in examples 1.3 - 1.5 assumes that you want to read in the entire file at one time. However, what if your data file is 2 gigabytes and you only have 128 megabytes RAM? As most operating systems read a file that large, they need to swap out most of it to the hard drive as virtual memory. So, there’s a lot of disk access and your operating system might “thrash” or stop working altogether. A better idea in many cases is to use file streaming, where only a segment of the file is read into a block of memory, and the then another page is read in as needed.

* + 1. Parsing data

But wait! Right now, we just have a large array of ASCII characters. For example, if the text file was made up of three-digit numbers, it might look like figure 1.6:

100101102103104105106107108109110

Figure 1.6: arrayData for a Sample File of Three Digit Numbers

We could access the first three elements in the array, but they would be three individual ASCII characters. So far, there is nothing in our program to identify that the first item in the data file is the number 100. So we need to ***parse*** the data into a series integers instead of a long string of text.

TERMINOLOGY

Parse: To analyze something by dividing it up into its constituent parts. For example, English sentences can be parsed into words, which can then be classified as nouns, adjectives, dependent clauses, and so forth. Every compiler uses a parser to divide up lines of code into the commands, variables, and other syntactic elements so the code can be compiled.

Every element in figure 1.6 is a three-digit number, so we easily know where each one starts and stops. However, what if we wanted to create a file with a mixture of two, three, or sixteen digit numbers? There are many ways to represent this in a data file by using a ***delimiter*** to separated each number. Some flat-files are ***comma-delimited***, ***tab-delimited***, ***fixed-length***, or have another delimiter. Comma-delimited files have a comma separating each element in the array, such as in figure 1.7. Tab-delimited files are similar, except they use the “TAB” to separate the data elements. Or, the data file could have any aribtrary character divide up the data, as long as the delimiter was easily identifiable, and not used in the actual data inself (for example, the number “0” would probably make a bad delimiter.

TERMINOLOGY

Delimiter: A special character or set of characters that divide a flat-file into individual data elements

Comma-delimited: A file that has data elements separated by an ASCII number 44 “,” comma (also know as a Comma-Separated Values file or CSV)

Tab-delimited: A file where every data element is separated by an ASCII #9 “TAB” character. These files may *look* like they take up more space than comma-delimited files, but the tab delimiters are still only one byte long

Fixed-length: A file where every data element takes up the same amount of characters, and is padded with spaces if the element is too short. These files have the disadvantage that they often take up more space than other delimited files, and the specified length must be long enough to hold the longest possible data element, or data-corrupting truncation will occur

Joe,Smith,111 Madison Avenue,Phoenix,AZ,85016,Ann,Smithy,4570 Washington Street,New York,NY,10011,Davesh Smiti,1919 Pine Street,Dallas,TX,76135

Figure 1.7: Comma-delimited Example of Address Information

Joe Smith 111 Madison Avenue Phoenix AZ 85016

Ann Smithy 4570 Washington Street New York NY 10011

Davesh Smiti 1919 Pine Street Dallas TX 76135

Figure 1.8: Tab-delimited Example of Address Information

Joe Smith 111 Madison Avenue Phoenix AZ85016

Ann Smithy4570 Washington StreetNew YorkNY10011

DaveshSmiti 1919 Pine Street Dallas TX76135

Figure 1.9: Fixed-length Example of Address Information

PUZZLE 1.1.3

On no! All of the commas were removed from this comma-delimited file. However, all the elements in the data file were standard English words. Can you replace the commas and restore the data?

THEREALLOVENOTTERMEVENALLOWEDDYEPICTURN

* + 1. Parsing Student Test Scores

Let’s say the data file we are given is a set of test scores for various students, and tha the file is comma-delimited. The first item in the file is a student’s last name, followed by a score from zero to 100. Then, there is the second student’s name, followed by their test score, and so on, until the end of the file.

"Jackson",89,"Marywell",65,"Anderson",41,"Nakamura",75,"Claude",90,"Reding",44,"Speers",8,"Fontaine",17,"Zannick",94,"Welsh",70,"Babic",14,"Shannon",83,"Quant",66

Figure 1.10: Our Student Test Score Data File

Note that in figure 1.10, it is not merely enough to know that the file is comma-delimited, but there are also quotation marks around all the student names. The quotation marks might help if a student’s last name is more than one word, or has spaces in the name. Even if the student’s name has a comma in it (unlikely, but still legal), we can figure out that the comma is part of the last name since it will be “protected” by the quotation marks. In short, every data file is different, and many may require some tricky data ***munging*** to eliminate the idiosyncratic elements.

TERMINOLOGY

Munging: To transform information, often in a tricky or imperfect way

Now we are ready to turn our linear arrayData set of ASCII characters into a useful array of data with the correct data type. The student’s last names will all remain as strings of characters (without the quotation marks), while the test scores will all be converted into integer values.

XXXXXXXXXXXXXXX

Figure 1.11: Code Example for Parsing the Student File in C++

XXXXXXXXXXXXXX

Figure 1.12: Code Example for Parsing the Student File in Java

XXXXXXXXXXXXXXX

Figure 1.13: Code Example for Parsing the Student File in Perl

* 1. Using Data Structures

Are we done yet? Well, we could stop here, and use the data in our new string/integer/string format, but it’s not very safe. For example, other programmers could get confused when using our data and try to connect a student’s name with the score *before* it, instead of the score following the name. It’s better to create a data structure that combines a student’s name and test score in the same object. This technique is called ***data hiding*** and is one of the cornerstones of object-oriented programming.

TERMINOLOGY

Data hiding: The technique of surrounding several data elements inside a larger container class. This ensures that outside code won’t accidentally damage the important data inside the object.

Digression

Encapsulation is just one of the hallmarks of object-oriented programming. Other important features are instantiation (the ability to create one or more copies of a class), inheritance (the ability for a new class to expand the features of its parent class), polymorphism (allowing a function or operator to act differently depending on the data is is acting upon), and encapsulation (a class that includes not only data, but all of the methods and functions needed to manipulate the data), as well as other arguably essential techniques such as “operator overloading”, “extensibility”, and others.

* + 1. Creating a Data Structure

Here is some sample code to create a data structure using differnet languages…

* + 1. Why Use Data Structures?

Another reason to use data structures is due to the increased time needed to search through an unparsed strign array. As we discussed in the inrtoduction, searching through an array requires O(n) time in the worst-case scenario. The code may find the necessary element before reaching the end of the list, but on average, at least half of the n characters will be examined before finding a match.

* 1. The Linked List

ANSWERS To EXERCISES AND PUZZLES

EXERCISE 1.1.1

1. HTML files often look complicated, especially if they have embedded objects in them. However, looking at the underlying “source” of any html file will show that it is made up entirely of printable ASCII characters. So, it is a flat-ASCII file, though it might be tricky to parse.
2. JPG picture images are ***not*** flat-ASCII. Even though they look like they are made up of individual pixels, the file is compressed into a binary format that is impossible to parse if you treated each byte as an ASCII character
3. A device-independant bitmap file is made up of individual pixels, one right after the other, and if the user knows the number of bytes of each pixel (the pixel “depth”), they could parse the file by hand. This results in a flat-file, though not one made up of ASCII characters, so it’s ***not*** flat-ASCII
4. Many early videgames created saved the game in a plain text files. This allow the player to cheat, by editing the number of lives, or the amount of gold they had won! Modern saved games usually create large binary files that are ***not*** flat-ASCII, though the creative hacker might still be able to edit the raw hexadecimal data in order to cheat
5. Ten years ago, every email message would qualify as a flat-ASCII file. Even though there is a header and routing information at the top of every email message, it is still written entirely in ASCII characters. Some modern email programs send messages that are written in html, which confuses things. However, html files are still flat-ASCII

PUZZLE 1.1.3

Two answers are possible:

THERE ALL OVEN OTTER ME VENAL LOW EDDY EPIC TURN

THE REAL LOVE NOT TERM EVEN ALL OWED DYE PICT URN