G51PRG Exercise Three: Arrays, Files and Strings

Steven R. Bagley

Introduction

In this coursework, we will write three different computer programs. The first program you will write extends what you have already done in Exercise 2, where you used the algorithm of successive division by 8 to generate the octal equivalent of an input decimal integer. I'm sure you'll all recall that it generated the octal digits in reverse order. Now we know about arrays we can store the generated digits in an array and then print them out in the *correct* order.

The second question is also uses arrays but, this time, asks you to analyse a 2D array of (hypothetical) marks for the four courseworks associated with a mythical Computer Science module. The final question asks you to read in a series of words from a text file and then output them to the screen with a set line length.

Again, there is no single correct solution for this exercise; any program which correctly implements the problem will be given a pass mark. To achieve higher grades, your program should make efficient and appropriate use of variables and functions in your solution.

1. Octal Numbers (in the right order)

Let's revisit your solution for question 1 of coursework 2. You are to modify your program so that it prints out the octal digits in the *correct* order. To do this, you should create a function `toOctal()` with the declaration:

```
void toOctal(int num, char octal[])
```

This function, which will be based on your octal converter from coursework 2, should take the number `num` and write the octal representation into the string `octal`, with preceding zeros if the value is less than three characters (you can assume for this exercise that we are only converting values less than `256` — your program should reject input that is out of range).

Every time your algorithm generates an octal digit (between `0` and `7`) as the remainder it should insert the ASCII value of this into the string (this can be found by adding the character literal `'0'` to the remainder, just as was done in the caesar cipher example in lectures). However, because our routine generates the digits in reverse order and we want to print this in the correct order — we need to start inserting the characters from the end of the string (i.e. array index of `2`) and count down rather than from the beginning and counting up.

In your main program, you can call `toOctal()` when you need to generate an octal version of a number, passing it both the number but also a `char` array in which it will write the string (and don't forget — your function will need to insert the null character '`\0`' into the string at the end, so you'll need to generate enough space to store four characters in the array). Once the string has been generated, you can pass it to `printf` to print it out.

When the user inputs a number your program should now print out the input number and its octal equivalent by calling your `toOctal()` function to generate the relevant string.

2. Processing marks

The file `http://g51prg.cs.nott.ac.uk/marks.txt` contains 40 values representing the scores for a particular Preceptor's set of ten tutees across all four of the courseworks set for a mythical module. The marks are set out on a separate line for each of the ten students in the tutorial group and the marks themselves are comma separated. Thus we might see:

```
77, 66, 80, 81
40,  5, 35, -1
51, 58, 62, 34
 0, -1, 21, 18
61, 69, 58, 49
81, 82, 90, 76
44, 51, 60, -1
64, 63, 60, 66
-1, 38, 41, 50
69, 80, 72, 75
```

Note that a mark of `0` implies "the coursework was handed in but all answers were so scrappy as to gain no marks", whereas a '`-1`' implies that the coursework wasn't handed in at all. However, a '`-1`' entry still means that the allocated mark for that coursework should be zero.

Write a C program that opens the file and firstly stores all the values in an appropriately declared two-dimensional array of integers. It then should analyse this array to find (from the 40 input values) the total number of coursework marks in the following ranges:

>= 70
between 60 and 69
between 50 and 59
between 40 and 49
between 0 and 39

with a particular note also being kept of the number of courseworks not handed in. The output of your program should be as follows:

```
The number of marks greater than 70 was 10
The number of marks between 60 and 69 was …
The number of marks between 50 and 59 was …
The number of marks between 40 and 49 was …
The number of marks less than 39 was …
The number of courseworks not handed in was …
```

(with, of course, appropriate integer values at the end of each line of the above.)

You will need to download the text file from the web so you can use it to test your program. You can either do this via Windows and save it to your `H:\` drive in the correct place, or you can do it from the Linux command prompt by using the `curl` program, like so:

```
curl -O http://g51prg.cs.nott.ac.uk/marks.txt
```

To read in the values, you should use `fscanf` appropriately. Your program should accept the name of the file to be opened as the first parameter on the command line. Therefore, if your program is called `processmarks` and the file containing the marks called `marks.txt`, you would call you problem as follows (this assumes that `marks.txt` is in the same directory as your program):

```
./processmarks marks.txt
```

When marking we will check that your program really is counting the number of marks in all of the above categories as the array is traversed (as opposed to just printing the correct numbers out).

3. Text Justification

In this exercise, you will develop a computer program that can format a block of ASCII text to fit a specific line length. Your program will be called with the length of a line (in characters) and the name of the file to format on the command line (you can extract the line length using `sscanf()`). If no arguments are given or if the file does not exist your program should print an error (to `stderr`) and exit.

Your program should then read each word in the file (a quick and simple way to do this is to use `fscanf()` with a `%s` format specifier) into a string (you can assume that no word will be longer than 32 characters). You should then print each word out with a space character between them. If printing a word would mean that there would now be more characters on the line than is permitted by the specified line length, then you should start a new line and print the word on this new line.

Therefore, if your program was run with the previous paragraph and a line length of 32, it should produce the following output:

```
$ ./justify 32 test.txt
Your program should then read
each word in the file (a quick
and simple way to do this is to
use fscanf() with a %s format
specifier) into a string (you
can assume that no word will be
longer than 32 characters). You
should then print each word out
with a space character between
them. If printing a word would
mean that there were more
characters on the line than
were permitted by the specified
line length, then you should
start a new line and print the
word on the new line.
```