G51PRG Exercise Three: Arrays, Files and Hidden Characters

Steven R. Bagley
David F. Brailsford

Introduction

In the second part of this coursework, we will finish our implementation of the hidden character viewing program that we started in coursework two by adding support for reading the input from a file, and integrating your octal conversion routines.

In the first part of this coursework, you will write a simple program that analyzes a 2D array of (hypothetical) marks for the four courseworks associated with the G51CSA module. These are stored in a file which you will need to read in and process.

There is no single correct solution for these exercises; any program which meets the specification will be given a pass mark. To achieve higher grades, your program should make appropriate use of variables and functions in your solution. In particular, to obtain the top grades it is necessary to design your own functions and make correct use of them.

1. Processing G51CSA 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 G51CSA. 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
<= 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).

2. Updating the Print File program

In the last coursework, you began work on a program that could print a file ensuring that characters that aren't normally visible (such as those with ASCII values below 32) could be seen. As we left it, the program wasn't complete — in particular, the following features were missing:

- The octal conversion was unfinished, instead of printing an octal representation of some characters it just printed a '`?`'
- It didn't read in from a file, rather it read its input from the standard input

In this coursework, you will finish off this program adding the missing features described above. To ensure that everyone is starting from the same position, you should add the missing features to the implementation of the program found at:

```
http://g51prg.cs.nott.ac.uk/code/printfile.c
```

Another advantage of making you use this version is that you'll gain some experience at reading and understanding other people's code — which is just as an important skill as being able to write your own programs.

There are two modifications that need to be made to this program. Firstly, you need to alter it so that it reads its input from a file, specified as the first argument on the command line. If the file doesn't exist then it should exit gracefully (either by `return`ing from `main`, or by calling `exit()`), printing the following message to the *standard error output*.

```
File <filename> does not exist.
```

Where `<filename>` should be replaced with the name of the file given on the command line.

Secondly, you need to modify the program so that it prints an octal representation of the characters instead of a question mark (e.g. for ASCII code 7 it should print `\007`). 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`).

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.