CSC 412 – Operating Systems

Programming Assignment 03, Fall 2021

Tuesday, October 5th, 2021

Due date: Thursday, October 14th, 11:55pm.

1 What This Assignment Is About

1.1 Objectives

The objectives of this assignment are for you to

- Use 1D (and possibly 2D arrays, if you want) in ~~C++~~ C;
- Perform file I/O in ~~C++~~ C;
- Search a directory hierarchy;
- Perform a simplified, idealized form of "pattern matching" in "images;"
- Use scripting to solve a classical problem of cross-platform development.

This is an individual assignment.

1.2 Handout

The handouts for this assignment consist in sets of example "image" and "pattern" files that you can use to test your program. The "pattern" files are given in two versions: One with the incorrect, Windows-style line-ending, and another with the expected Unix-style line ending (see Section 3 for clarification of what this means).

2 Part I: Search for Patterns

2.1 Format of the data

As mentioned earlier, the "images" are represented as arrays of text data. The file format we are going to use to represent the "images" is extremely simple:

- the first line gives the width w and height h (number of columns and number of rows) of the "image;"
- the next h lines stores the different rows of the "image."

The following lines of text could encode a small "image"

```
12 10
ABEFGH345TYH
123456789012
ABC3DG123asd
345ASABCFTG0
224CG345D12t
SDFG3224Zd1z
130678901345
098765432101
098765432101
098765432101
```

The "patterns" to search for in the "images" are squares of three rows and three columns. Each pattern is stored in a separate file. We are going to reuse the same image format to store the pattern, so that all pattern files will start with a line `3 3`. This also means that you can use the same code to read the images and the patterns (and that a pattern is simply a small image).

2.2 An important note

You must understand that the "images" and "patterns" we are talking about here are abstractions and simplifications of a real-world problem. Please don't waste time searching online for solutions to "pattern matching in images." The solutions that you are going to find will try to address the **real** problem:

- real color images (encoded on multiple bytes);
- best-fit match rather than exact match;
- handling of noise and illumination variations in images;
- handling of scaling and rotation;
- etc.

In other words, the code you are going to find online will be so complicated and so much more general than what you need to do here that it will be completely worthless. Your solution will involve little more than string comparisons, and your main tools are going to be the `strcmp` function of `<string.h>` or simple character-by-character comparisons.

This being said, if you *are* interested in pattern matching, send me a note or come to my office (when I am back) to discuss your ideas. I approach such problems from the point of view of "classical computer vision" (that is, not machine learning-based). If you want to hear a different, machine learning-based point of view, Profs. Hamel, Alvarez, and Daniels are very good interlocutors.

2.3 The search for pattern matches

For a given image and pattern your program should look for every occurrence of the pattern in the image and report in an output file the number of matches found, and their location.
For example, if the pattern is

```
890
23a
CFT
```

then the result for this pattern should be

```
1 1 7
```

because the pattern has been found once in the image, at the location with its upper-left corner at row 1 and column 7 (Figure 1(a)).
If the pattern is

```
ABC
345
224
```

then the result for this pattern should be

```
2 2 0 3 5
```

because the pattern has been found twice in the image, once at the location with its upper-left corner at row 2 and column 0, once at the location with its upper-left corner at row 3 and column 5 (Figure 1(b)).

Figure 1: Localization of “patterns” in an “image.”

If the pattern is

```
xxx
yyy
zzz
```

then there is no match and the program shouldn’t produce any output. because the pattern is not found anywhere in the image.

2.4 Input to the program

Your program will be launched with the following arguments:

1. The path to a file storing a "pattern" (the files are identified by a `.pat` extension) that you want to identify occurrences of;
2. The path to a directory that you want to search through, to find "images" (the files are identified by a `.img` extension) inside which you want to look for occurrences of the pattern;
3. The path to an "output" folder where to print out the results of the search for the input "pattern."

2.5 Recursing through the folder hierarchy

Given the path to the "search" folder, your program should determine all "image" file contained within that folder. It should then perform "pattern matching" for that image and collate the results.

A piece of warning: Whenever you have an assignment of this type, where you need to wander throughout the file system, possibly reading and writing files, I highly recommend that you work with a test, "toy" folder hierarchy and not your real home folder or, worse, the root folder of your file system[1]. Pretty much every semester I have a case of a student in tears because some silly mistake in the code resulted in large portions of the disk being wiped out (a couple of hours before a deadline, needless to say). Please don't be that student for Fall 2020!

2.6 Format of the output

Each process will deal with *one* pattern file, specified by a file path. It will produce in the output folder a single file named after the pattern, in which are listed all the occurrences of the pattern that have been encountered.

If the name of the pattern file is `somePattern.pat`, then the name of the output file produced should be `somePattern_Matches.txt`

That file should consist of a list of all the image files in which matches have been found, along with the location of these matches, as indicated earlier. An example of output would be the following (obviously, the image names and match locations could be different for you):

```
image1.img
    2 2 0 3 5
SomeImage.img
    1 4 10
image2.img
    3 7 9 12 5 15 1
```

[1] Since, of course, you still do all your coding on an Admin account, right?

2.7 Extra credit (4 points)

Generalize your pattern matching code so that the patterns to search for can now be of any size (still specified in the first line of the pattern file).

2.8 Other extra credit

I may post more options for extra credit directly on BrightSpace. Right now I am just trying to get this assignment out.

3 Part II: `bash` Script

3.1 The end-of-line headache

This week, the `bash` script we are asking you to write is not only going to launch a C/C++ program. It is also going to perform some simple string manipulation to address a very old, vexing problem that all of us who have to use different platforms must run into, sooner or later.

The issue as stake here is that of the characters used to indicate a new line in a text file, that is, the end of the current line. This can be referred to as the "newline," "end of line," or line ending problem. It carries over from the days of mechanical typewriters. When typists arrived to the end of a line, they had to execute a "linefeed" (LF) to make the paper sheet move up by one line, and then a "carriage return" (CR) to reposition the head at the beginning of the new line.

When the different operating systems were developed in the 70s and 80s, they came with different scheme to represent the end of line, most based on the linefeed (LF, `'\n'`, `0x0A`, 10 in decimal) or the carriage return (CR, `'\r'`, `0x0D`, 13 in decimal).

Most notably:

- Early on, Unix systems (and later on, Amiga, BeOS, and Linux) opted for LF;
- The TRS 80, the Apple II, and the original Macintosh System (then Mac OS until Mac OS 9 up to the early 2000's), chose CR;
- CP/M, and therefore DOS, Windows, and OS/2 (along with Atari TOS and others) picked CR+LF.

As you can imagine, this posed several problems. For a start, typically text editors on Windows (for cause of being the dominant OS, with 95% of the installed base) and Unix (for "true OS" religious reasons) only recognized the native endline format. You can see this at work if you try to view in Notepad[2] one of your Linux programs freshly unzipped from the archive. Other common problems are that the "same" text file has a different length on Windows and Unix or that the same C code is not guaranteed to give the same results on the different platforms. Finally, developers of file transfer clients had to offer different transfer mode: one for text file, in which end of line

[2] Of course, you all have installed on your PC a decent text editor, Notepad++ or better, so you would never open a text file by default in Notepad, I am sure.

characters had to be translated, and one for "binary" files in which inserting a character `0x0A` before every occurrence of a `0x0D` could corrupt images, videos, etc.

Since then, Mac OS X (and now macOS and iOS) being a Unix system, Apple has switched to LF as its supported end of line format. So we are pretty much down to two formats: LF vs. CR+LF. This is the problem that you are going to address in this assignment. We are going to provide some text files (representing "images" and "patterns" to search for in the images) to be used as input for the C section of this assignment. All files will be in the Windows format, and your script will "translate" the files into the Unix LF end of line format.

3.2 What the script should do

Your script should reside in the same directory as the source (and possibly header) file(s) of the C++ program. It should take as arguments:

- the name of an executable for the program to build,
- the path to the directory within which to look for "images" (the files are identified by the extension `.img`);
- the path to a directory containing a number of pattern files (the files are identified by the extension `.pat`);
- the path to an output folder where the results of the searches should be written.

The pattern files are encoded using Windows line endings. The script must replace these line endings with Unix line endings. Before doing this, the script should copy the files into a new directory and make the modifications to the copy. After the modifications have been made to the copied file, the copied file should be moved back into the original directory and replace the original file. After the file encodings have been modified the script should remove the directory where the original files were copied to.

Once this task has been completed, your script should build the C program and launch a copy of it for each pattern found in the pattern data folder. Make sure that these instances of the program run concurrently and not sequentially.

3.3 What the script should *not* do

There exist utilities that allow you to perform the end-of-line fix on an entire file with a single-line command. You are not allowed to use these commands. Again, I want to force you to write a loop in `bash` to iterate through every line of a text file, because this is an fundamental building block of `bash` programming. If you also happen to know a few extra convenient commands, this is great, but you won't always find the "magic one-line command that solves exactly the problem." And in that case, you will have to be able to roll your own loop.

4 What to submit

4.1 The pieces

All your work should be organized inside a folder named `Prog02`. Inside this folder you should place:

- Your report;
- A folder named `Program` containing all the source and header files required to build the program, as well as your bash script;
- A folder named `Data Folder` containing a hierarchy of subfolders, some of which contain "image" files;
- A folder named `Patterns` containing pattern files that you used for your testing.

Please note that we may test your program and script with data other than the ones that you provide, but at least we want to be able to test your code in the same conditions that you did.

4.2 Grading

4.3 What's on the report

The main sections I want to see in this report are:

- A presentation and discussion of the choices you made for the implementation of the pattern search. This is an algorithm and data structure question, with considerations on style, modularity, and performance.
- A discussion of possible difficulties encountered in this assignment[3].
- A discussion of possible current limitations of your C program and script. Is it possible to make them either crash or fail?

4.4 Grading

- The C++ program follows the specifications: 20 points
- The C++ program performs the task specified (execution): 30 points
- the `bash` script performs the task specified (execution): 15 points
- C and `bash` code quality (readability, indentation, identifiers, etc.): 15 points
- Report: 15 points
- Folder organization: 5 points

[3] Here, I don't mean "my laptop's battery is dying" or "I couldn't get a pizza delivered to my dorm room."