University of Toronto
CSC228S, Winter 2002

Assignment 1

Deadline A :	*9 a.m., Friday January 25, 2002.*
Deadline B :	*9 a.m., Friday February 08, 2002.*
Total Weight:	*10% of your course grade. Part A is worth a small fraction of this total weight.*

Overview

This assignment will teach you about sequential file input in C++ and about working with binary files.

Character Recognition

One of the problems in computational vision is identifying text from an image file. One approach to this problem is to chop the text to be identified into separate images each containing a single character. This *unknown* character image is compared to a set of known character images or *pattern* images. The pattern that has the best match is used as the guess for the unknown character in any further processing.

In typical character recognition applications, the unknown character is not exactly the same as its corresponding pattern image. It is usually noisy - perhaps from a scanning process. It might even be a different font than the pattern. The chopping process may have left the unknown characters shifted both horizontally and vertically and possibly rotated. The unknown characters might be scaled so they are larger or smaller than the pattern images.

To make the problem simpler, the images that you will work with for this assignment will have no noise. Each unknown character will be an exact match to one of the character patterns. The image might be shifted but it won't be scaled or rotated.

You can find some test images on the course web-site. If you have access to a Windows machine you may also create some test images of your own using the Paint program. [1]

Your Job

You must write a c++ program which will take an unknown character image and a set of pattern images and return the most similar image, identified by its file name.

Program interface

Your program should not be interactive. The filenames of the image should be handled with a command-line arguments to your program i.e. arguments that the user gives when calling your program.

Your program should print to standard out a single value which is the file name of the most similar image.

[1] When you save your images from Paint, pick `Save As` and then when you select the type you can choose between a 256 colour bitmap (8 bits per pixel) and a 24-bit bitmap. The other standard Paint choices (Monocrome bitmap) and 16 colour bitmap would make the requirements for your program much harder.

Your program should accept an optional first argument *-v*. If *-v* is present your program should run in *verbose* mode. In verbose mode it should print to standard error information about each pair of of images which are compared. Each time it should print the dimensions of the images (in pixels), the calculated shift and the measure of similarity. If present, the **-v** must be first.

The next argument is the name of the unknown image. The remaining arguments are names of pattern images with which your program should compare the unknown character.

The Processing Algorithm

Here is an outline of the algorithm you should use to determine the measure of similarity between any two character images.

1. Read and process the headers of both files.
2. Read the blank lines from the top of each file.
3. Based on the location of the first foreground pixel in each file, calculate an estimated shift (both horizontal and vertical).
4. Assume the unknown image is the pattern character based on this estimated shift and process the rest of the pixels counting how many match the target value
5. Calculate and print the required information depending on the mode

Image File Format

An image file can have different formats. One of the basic ones is the Bitmap (bmp) file. Below is the file format description.

bytes	Purpose
2	`BM` signature to indicate bmp file
4	size of file in bytes
4	set to 0 (ignore)
4	offset from beginning of file to pixel date
4	ignore (default 40)
4	width of image in pixels
4	height of image in pixels
2	ignore
2	bits per pixel (either 8 or 24 for this assignment)
24	ignore
??	colour table (size varies)
remaining	pixel data

Using `od`, look at some of the .bmp images to help understand this format. In particular pay close attention to the byte order of the various 2-byte and 4-byte values you will need to read from the file.

Note that according to the bmp file specification, the number of bytes per row must be a multiple of 4. In order to do that, “empty” bytes are added at the end of each row, so we have a multiple of 4 number of bytes.

The images your program will process will either use 8 or 24 bits per pixel. [2] This will mean that each pixel will be an integer number of bytes. You will have to obtain this information by reading the proper field in the header. Your program must not assume that any pair of images being compared will have the same number of bits per pixel.

All the images are two colour. The background is represented by a zero and the foreground by some non-zero pixel value. Since we don't care about the colour of the character, your program should consider pixels of the foreground colour in one image to match the foreground pixels in the other image. Therefore, two zero values represent the same pixel. Two non zero values represent the same pixel as well regardless of the actual values. This means your don't have to understand or parse the colour table. Note that there is no color table in the case of 24 bits per pixel.

Some Restrictions

- You must read the image files sequentially. You may not skip back in the files.
- All the code you use must be your own. You must not use any API classes written specifically to work with `bmp` images or other graphics APIs. You may of course use standard c++ classes such as `string`.
- You must not store more than one line of each image in memory at one time. We recognize that this is a very unrealistic constraint for these small images but we require it none the less.

Erroneous input

You need not to do a lot of error checking in your program, but you should specify in your report what error checks would ideally be done, and which function would be responsible for each.

Using od when debugging your program

Working with binary files can be frustrating because you can't just print the files to the screen or create them with your favorite editor. You should use the unix tool **od** to view human-readable versions of the image files. And you must use `od` to produce printouts of your files for the testing section of your assignment report. In particular you should use `od` to demonstrate that the `-v` option of your program is correctly printing the image sizes. We will provide some tips on using `od` on the website.

Relevant Readings

We will cover some of the course material relevant to this assignment in class, but you are responsible for reading many of the details yourself in the text. Here are the sections of the text which you should begin reading now.

4.1.6	Binary files and using od
2.8.3	I/O redirection in unix
2.2	Opening files
2.4	Reading and writing files

[2] As mentioned earlier, the Paint program will produce bmp images with 1,4,8 or 24 bits per pixel. We will only consider the latter two cases for this assignment.

What to hand in for the first deadline (“A”)

There is no specific programming requirement for Part A; It is merely a step on the way to part B. You should have made a good start on the design, and have some code working.

You may need to create simple driver programs to test your code at each phase of development. Hand in a printout of your code (including any driver programs), and some test runs that demonstrate what works. Explain your test results with brief, hand-written comments on your printout.

Make your deadline A submission **as brief as possible,** and do not hand in a report.

What to hand in for the second deadline (“B”)

For deadline B, you are to hand in your assignment in the form of a report, as described in the 228 Course Guide. The style of a 228 report is different from what you are used to from past csc courses, so be sure to re-read the relevant sections of the Guide before getting far into the assignment. Your report must include all of your code and all relevant input and output test files. You can use a script file to demonstrate the execution of your programs (see `man script`).

Take care to meet the specifications as given in this handout and in any clarifications on the web-site. Notice also that many decisions are not specified. In these cases you must make decisions yourself. Remember to justify these decisions including explaining the alternatives you did not choose. This is a very important part of your report.

Your report should be thorough, well-written, concise, and honest. It should not be more than about five pages long, not including code and tests. Make sure that you run your report through a spelling checker.

How to package up your assignment submissions

See the course web site for important information about how we expect you to package up your assignment for submission and how to do the electronic submission.

A note about C++

We do not expect you to use any advanced features of C++ for this assignment. In particular, no inheritance or templates should be necessary for a good design.