

A1

Due Sep 27 by 4p.m. **Points** 5

Update Sept 21st:

Keep in mind that you will have to modify the main function we have provided you. Right now, it is dangerously checking `argv[2]` (before knowing if there *even is* a third argument). The if block with `strcmp` in it, is meant to show you an example as to how you could check if `argv[2] == "-p"`, (this is so if `0 == strcmp(argv[2], "-p")`). Modify this code as you see fit to implement the assignment as we have asked (it is likely that if you don't change it, you will have made a mistake).


Also, have a look at the example `fscanf.c` from lecture 02-2 if you are having trouble reading files.

Error Checking

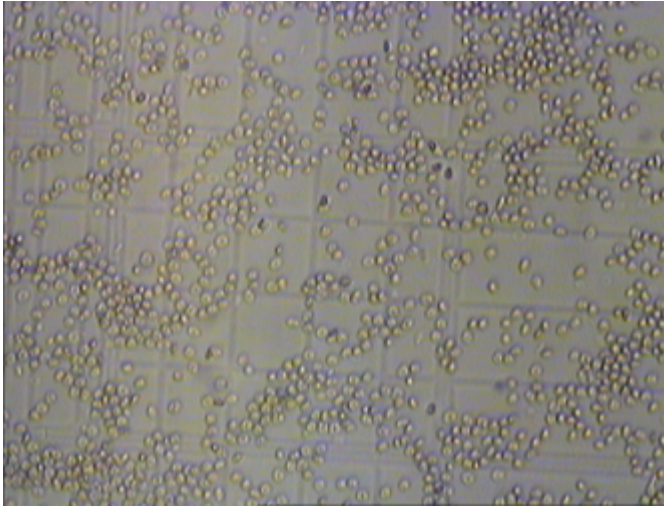
You can assume that the file will open successfully and be formatted correctly. You do not need to do error checking besides making sure the command line arguments are correct -- as in, there is at least one argument and if there is a second, it must be `-p` (and then print the array if so). If there is an error, the starter files mention printing something to `stderr` and then exiting with a value of 1 -- notice the if block in the main we provided demonstrates one way you could do this.

Introduction

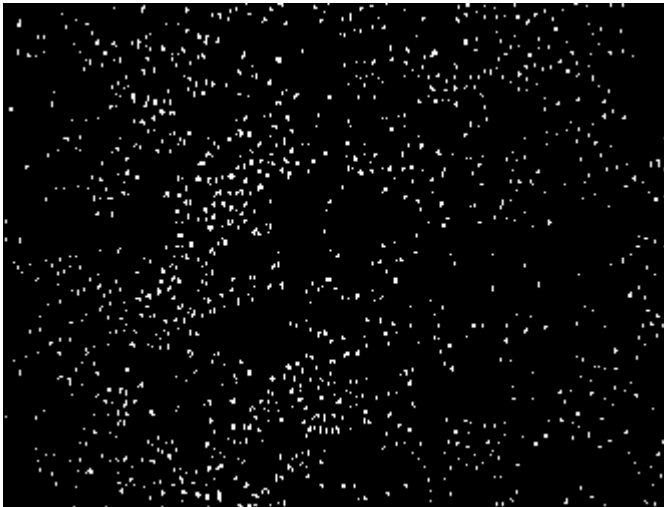
In this assignment we will investigate an image processing application.

Analyzing blood smear images, and extracting features of the cells is an important application in the field of medicine to help detect and monitor various types of diseases. In this assignment, you will implement a simple form of red blood cell counting. This assignment was inspired by an [article](https://www.sciencedirect.com/science/article/pii/S2352340915001626?via%3Dihub#bib1)  (<https://www.sciencedirect.com/science/article/pii/S2352340915001626?via%3Dihub#bib1>) about a data set and algorithm for counting red blood corpuscles.

The images we originally started with were stored in BMP format.



We have already taken care of the first steps, so you will be working with a text file that has already been segmented. We have taken the above image, and produced black and white images where all pixels have the values of 0 (black) or 255 (white).



We then converted the file to ASCII (plain text) format so that you can read it using `fscanf()`. Read the man page for `fscanf()` by typing `man fscanf` at the command line.

The file contains only integers separated by a space, and you can assume that the file format is correct when writing your code. The first two integers (first line) are the number of rows and columns of the image in pixels. There is a newline character after each row of pixels to make it slightly easier to visually inspect the file.

Step 0: Set up

Your program will consist of two files: `count_cells.c` and `image.c`

These files have already been added to you repo. You should be able to compile your program using the following gcc line:

```
gcc -Wall -g -std=gnu99 -o count_cells count_cells.c image.c
```

Step 1: Reading and writing an image file

The main function in `count_cells.c` takes one or two command-line arguments:

- The first argument the name of blood cell image in the text file format described above.
- The second argument, "`-p`" is optional. If present, this argument indicates that the pixel matrix should be printed *before other program output*. In `image.c` you will implement two functions. Their function prototypes are given below:

```
/* Reads the image from the open file fp into the two-dimensional array arr
 * num_rows and num_cols specify the dimensions of arr
 */
void read_image(int num_rows, int num_cols,
               int arr[num_rows][num_cols], FILE *fp);

void print_image(int num_rows, int num_cols, int arr[num_rows][num_cols]);
```

Because we are passing in a variable-length, multi-dimensional array as an argument to a function, the first two arguments must be the dimensions of the array, and the third the array itself. (See King pages 197-199) We could call `read_image` as

```
int p, q; // assume p and q have valid values
int a[p][q];
FILE *fp; // assume fp holds a valid open file pointer
read_image(p, q, a, fp);
```

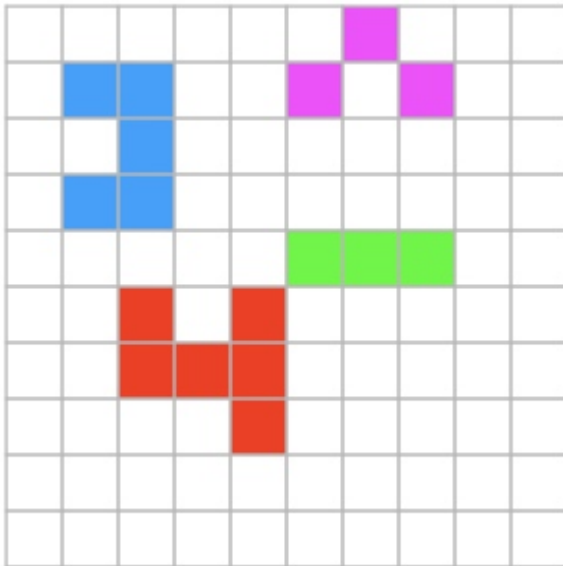
This means that we need to read the dimensions of the array before calling read image. Write these two functions and call them from the main function in `count_cells.c` code and test it before moving on. Remember to add the prototypes to `count_cells.c` . This might also be a good time to learn the basics of gdb.

It would also be a great idea to make sure to commit and push your work at this point.

Step 2: Count cells

Write a function `count_cells` to count the number of blood cells in the image. One cell is a region of adjacent white pixels. The function will be implemented in `image.c`

A pixel is adjacent to another pixel if there is a pixel of the same colour above, below, left or right of the pixel. In the example image below there are 6 cells. The "blue cell" is composed of 5 pixels, the red cell is composed of 6 pixels, the green cell is composed of 3 pixels (and has no pixels adjacent to the red cell), and there are three pink cells of one pixel each.



The main function will call `count_cells` and will print a message to stdout using the following format string:

```
"Number of Cells is %d\n"
```

You must include a statement using this exact format string to receive credit for your cell count. Be sure to print this only once.

Exactly *how* you count these cells is up to you, but I'd like to direct your attention to the flood fill algorithm, which is a recursive algorithm that recursively visits and somehow marks all adjacent pixels starting at some seed pixel within (in our case) a cell. Recursion terminates whenever a pixel visited does not have the same value as the seed pixel. This is how the paint-bucket works in any basic image creation software.

If you use any other algorithm (e.g. authored and developed yourself) be sure to add comments. If you decide to use some other algorithm described by someone else (note, you **cannot copy any code in any of these cases**, but you *can* look up algorithms) be sure to include a clear citation in your code with the author(s) names and how you accessed it.

Input

You probably want to create a few smaller image files so that you can verify that your algorithms are working correctly, and so that you can test some edge cases. For example, here is one that I wrote: (I will leave it to you to figure out how many cells are in this image).

```
4 7
0 0 255 255 0 0 0
0 255 0 0 255 255 0
0 255 0 0 0 0 0
255 255 0 0 0 0 0
```

There are a number of `.bmp` files and the resulting `.txt` files in `/u/csc209h/fall/pub/a1` on teach.cs that you can also use as input. Note that these aren't that useful for testing, unless you want to count by hand the number of cells in them.

What to submit

Add, commit, and push your `image.c` and `count_cells.c` files to the `a1` directory in your repository. Please refer to the [Syllabus](https://q.utoronto.ca/courses/309775/pages/syllabus?wrap=1) (<https://q.utoronto.ca/courses/309775/pages/syllabus?wrap=1>) to make sure you are following the expectations for assignment submissions, mainly, never pushing your program's binaries to the repo and ensuring that it compiles.