CMSC 216 Project #1 Spring 2013

Date due: Wednesday, February 13, 10:00:00 p.m.

# 1 Introduction

This is a short (in terms of the amount of code to be written) first project intended primarily to give you more practice with the code development facilities that are different in this course than what you are used to in Java and Eclipse. If you took CMSC 131 here the project might seem familiar, because sometimes CMSC 131 uses a similar project. There isn't anything special about the problem to be solved, and it doesn't really have anything to do with the content of the course; it's just a problem that can be solved using only the simple facilities of C that have been covered so far this semester.

The project can be written in less than 150 lines of code. If you find it easy and can write it quickly, that's great (of course keep in mind that may not be the case for everyone). Since the project is simple, however, it is being assigned with just a short time to be done in.

All programming coursework in this class, including this project, assumes you set up your account correctly in discussion section last week. If you didn't set up your account because you weren't in discussion, you're on your own for getting things to work. Get notes from a classmate who was there, and set things up yourself.

If you have questions about the project while you are working on it ask them in person; we're not going to be able to answer them via email. **Note**: there will not be any extra office hours for this short first project. Please come to any of the TAs' regular office hours if you have questions.

# 2 Overview

Imagine that upon graduation you are fortunate enough to get a job with a large, exciting multinational corporation, with offices in many interesting nations around the world. Suppose that, due to your programming background, one of your first tasks is to do some work on the company's website. Of course it would like to have a nice–looking presence on the web, and to properly acknowledge the contributions of employees at all of the international branches the company would like to display the flag of each nation when a user visits the webpage of the office in each country. Therefore, as part of your webpage revisions you've been tasked with creating images of flags of the various nations in which the company does business, which will be displayed in visitors' web browsers. Of course, depending upon the browsers that people use to visit the site, and the sizes of their displays, the flags may have to be shown in different sizes to really look good when drawn on the screen.

Since your knowledge of how to create fancy color graphics is limited (i.e., nonexistent), but your C programming background is extremely strong (or it will be by then), and in any case your boss doesn't care whether things are fancy, just what the flags look like, imagine that imagine that you've been asked to just work up a prototype program that will show what the flags will look like when drawn. Some poor summer college intern will have to actually set up things for the flags to be drawn, based on your specifications, meaning the output of your program. Therefore your program will just use characters to represent the colors of the flags. In particular, it will use a two–dimensional array of characters to represent a flag, and will store different characters in areas of the array to represent different–colored parts of flags.

# 3 Flag descriptions

As mentioned, some of your functions will be filling in the elements of two–dimensional character arrays with different characters, to represent the colors of parts of flags. (We could use arrays of a type other than character, but using character arrays means it will be easier to print and inspect the flags produced by your functions.) In this section the four flags that your program should be able to create, and how they are to be constructed, are described. The specifics of the functions you must write, as well as how you should start working on the project, what files you are given and what files you must create, etc., are all addressed to later sections.

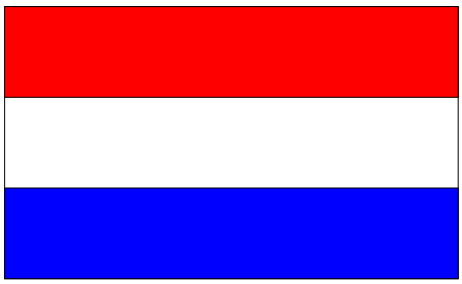
The four flags that your program should create are those of Luxembourg, Romania, Sweden, and the Czech Republic. As will be seen below, the colors red, white, blue, and yellow are needed to create these flags. When your program needs to fill a region of an array representing a flag it should indicate these colors using the capitalized first letter of their names: `B` for blue, `R` for red, `W` for white, and `Y` for yellow.

Flags of different sizes can be created, as described below. The size of a flag is determined by specifying its height, which is the number of rows of characters in the array that should be filled in. To be constructed properly when using characters for graphics, each flag has specific requirements for what its height can be; if the height given to your functions doesn't satisfy the requirements a flag cannot be created. Each flag's width is determined based upon its height as described below.

Note that in real life a character printed on either a computer screen or on a printer is normally not square, in other words, its width is not the same as its height. The sad fact is that, for this reason, if flags created by your program are printed they will not actually look all that great. Your imagination, applied liberally, can help in making the flags appear pleasing and beautiful to the eye.

### 3.1 Flag of Luxembourg

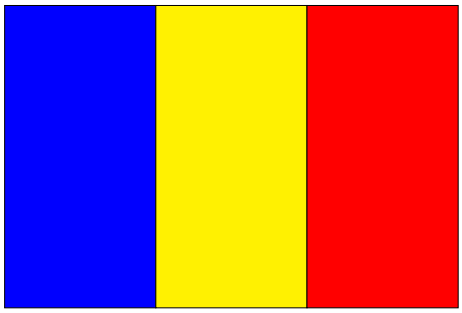
The flag of Luxembourg has a red stripe in its upper third, a white stripe in its middle third, and a blue stripe in its bottom third, as pictured below. The width of the three rows are the full flag's width. To be valid, the height of the Luxembourgish flag must be a multiple of 3. The ratio of its height to its width will be 3:5. For example, if the flag is to be created with a height of 9, its width would be 15, as in the example shown on the right.



```
RRRRRRRRRRRRRRR
RRRRRRRRRRRRRRR
RRRRRRRRRRRRRRR
WWWWWWWWWWWWWWW
WWWWWWWWWWWWWWW
WWWWWWWWWWWWWWW
BBBBBBBBBBBBBBB
BBBBBBBBBBBBBBB
BBBBBBBBBBBBBBB
```

### 3.2 Flag of Romania

The flag of Romania is composed of a blue region in its left third, a yellow region in its middle third, and a red region in its right third. The height of the three stripes are the full flag's height. To be valid, the height of the Romanian flag must be a multiple of 2. The ratio of its height to its width will be 2:3. For example, if the flag is to be created with a height of 12, its width would be 18, as in the example shown on the right.



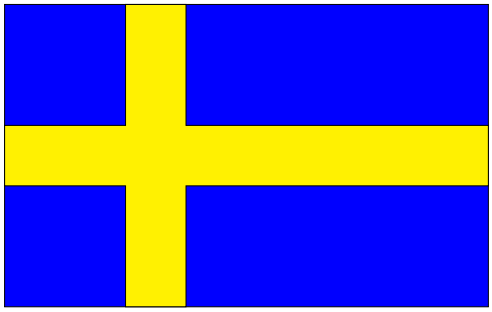
```
BBBBBBYYYYYYRRRRRR
BBBBBBYYYYYYRRRRRR
BBBBBBYYYYYYRRRRRR
BBBBBBYYYYYYRRRRRR
BBBBBBYYYYYYRRRRRR
BBBBBBYYYYYYRRRRRR
BBBBBBYYYYYYRRRRRR
BBBBBBYYYYYYRRRRRR
BBBBBBYYYYYYRRRRRR
BBBBBBYYYYYYRRRRRR
BBBBBBYYYYYYRRRRRR
BBBBBBYYYYYYRRRRRR
```

### 3.3 Flag of Sweden

The flag of Sweden is all blue, with an off–center yellow cross. To be valid, the height of the Swedish flag must be a multiple of 5. The ratio of its height to its width will be 5:8. For example, if the flag is to be created with a height of 15, its width would be 24, as in the example shown on the right.

The height or thickness of the vertical arm of the yellow cross in the center is 1/5 of the flag's total height, as is the thickness of the horizontal arm (the thickness of the horizontal arm is also 1/8 of the flag's width). The horizontal arm should be centered, but the vertical arm should be positioned starting 1/4 from the left. Note that the areas above and below the cross are each 2/5 of the flag's total height, while the area to the left of the cross is 1/4 of the flag's total width, and the area to the right of the cross is 5/8 of the flag's total width.

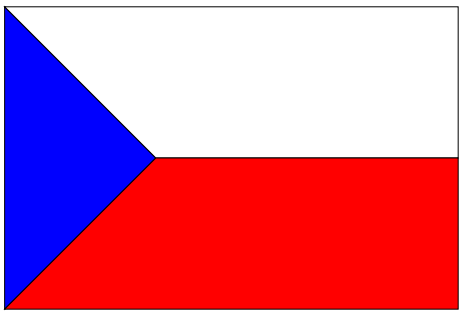


```
BBBBBBYYYBBBBBBBBBBBBB
BBBBBBYYYBBBBBBBBBBBBB
BBBBBBYYYBBBBBBBBBBBBB
BBBBBBYYYBBBBBBBBBBBBB
BBBBBBYYYBBBBBBBBBBBBB
BBBBBBYYYBBBBBBBBBBBBB
YYYYYYYYYYYYYYYYYYYYYYY
YYYYYYYYYYYYYYYYYYYYYYY
YYYYYYYYYYYYYYYYYYYYYYY
BBBBBBYYYBBBBBBBBBBBBB
BBBBBBYYYBBBBBBBBBBBBB
BBBBBBYYYBBBBBBBBBBBBB
BBBBBBYYYBBBBBBBBBBBBB
BBBBBBYYYBBBBBBBBBBBBB
BBBBBBYYYBBBBBBBBBBBBB
```

### 3.4 Flag of the Czech Republic

The flag of the Czech Republic has a blue triangle on the left, and the top half of the part to the right of it is white, while the bottom half of the part to the right of it is red. To be valid, the height of the Czech flag must be a multiple of 6. The ratio of its height to its width will be 2:3. For example, if the flag is to be created with a height of 12, its width would be 18, as in the example on the right below.

The blue triangle is situated so that its hypotenuse is along the left edge of the flag, meaning the hypotenuse is the same size as the flag's height. Note that the widest rows of the triangle are half of the flag's height, as are its base and height. The longest rows of both of the white and red regions are one less than the flag's total width (which is half again as large as its height). The narrowest rows of both of the white and red regions are the same as the flag's height. In the example below, where the flag's height is 12, the blue triangle's hypotenuse is 12, the same as the flag's height, and its base and height, and its two longest rows, are all of length 6. The longest row of the white and red regions are both of length 17, while their shortest rows are both of length 12.



```
BWWWWWWWWWWWWWWWWW
BBWWWWWWWWWWWWWWWW
BBBWWWWWWWWWWWWWWW
BBBBWWWWWWWWWWWWWW
BBBBBWWWWWWWWWWWWW
BBBBBBWWWWWWWWWWWW
BBBBBBBRRRRRRRRRRR
BBBBBBRRRRRRRRRRRR
BBBBBRRRRRRRRRRRRR
BBBBRRRRRRRRRRRRRR
BBBRRRRRRRRRRRRRRR
BBRRRRRRRRRRRRRRRR
BRRRRRRRRRRRRRRRRR
```

## 4 Functions to be written

The functions you are to write have the following prototypes (this is actually the contents of a header file `flags.h` that your code will use):

```
#define MAX 80

int create_luxembourg(char flag[][MAX], int height);
int create_romania(char flag[][MAX], int height);
int create_sweden(char flag[][MAX], int height);
int create_czechoslovakia(char flag[][MAX], int height);
int print_flag(char flag[][MAX], int height, int width);
int compare_flags(char flag1[][MAX], char flag2[][MAX], int height,
                  int width);
```

As you can see, each has a two–dimensional `char` array as a parameter; this is what your functions should modify. `MAX` is a symbolic constant that defines the maximum size of any flag; in this project no flag can have a width or a height greater than the value of `MAX`.

### 4.1 The four flag creation functions

The four flag creation functions `create_luxembourg()`, `create_romania()` `create_sweden()`, and `create_czechoslovakia()` should fill in their array parameter `flag` with the characters representing the flags of their respective countries.

- If the value of `height` is valid, the function should fill in the elements of `flag` with the characters that will "draw" that country's flag, and return 1.
- If the value of `height` is invalid, for any reason, the function should **not** modify `flag` at all, and simply return 0.
  - `height` would be invalid if it did not satisfy the restrictions mentioned above for each country's flag, for instance for the flag of Luxembourg `height` must be a multiple of 3, and for Romania it must be a multiple of 2.
  - `height` would also be invalid if it was greater (strictly greater) than `MAX`, or if the width of a country's flag would be greater (strictly greater) than `MAX`, given the ratio of height to width defined above for each flag type. For example, if the height of the Swedish flag was 60 its width would be 96, which is larger than `MAX`.
  - In this project a flag cannot have a height of zero (nonexistent flags are disallowed). Consequently, each country's flag has a specific minimum possible size, based upon the restrictions of the flag dimensions given above.

Also note the following:

- Your functions should **only** modify the region of the flag determined by the value of `height` and the ratio of the width to height of each country's flag. Array elements outside that range should **not** be altered.
- Your functions **cannot** assume that the elements of `flag` have any particular initial values. They may have any characters at all in any particular execution.
- Your functions may assume that `flag` is a valid array with `MAX` columns, and at least `height` rows.

The flag creation functions can be written in different ways, so you may write them in any way that you like, subject to any restrictions (such as style, or allowable language features) mentioned or referred to in the appendices below.

### 4.2 The `print_flag()` function

This function should print a flag, of any country. It would not be ideal to have to write four different print functions to print the various countries' flags, but it would also be undesirable to have to repeat the logic in this function that determines the width of each flag based upon its height. Therefore this function will just

be given a height and width, and should print the region of its parameter `flag` indicated by them (the first `width` elements of the first `height` rows).

If the values of `height` or `width` are invalid, the function should not print anything at all, and just return 0. If their values are valid and the flag can be printed, the function should do so, and return 1. `height` or width would be invalid if either is greater than `MAX` or less than 1.

Of course the characters of `flag` should be printed in order by increasing row subscript, and in order by increasing column subscript within each row. Each row printed will have exactly the same number of characters. A newline must be printed at the end of each row's characters, including the last row's. No blank spaces should be printed at all.

The function may assume that `flag` is a valid array with `MAX` columns, and at least `height` rows.

### 4.3 The `compare_flags()` function

This function should compare two flags. As with `print_flags()`, one comparison function will compare flags of any country (which are not necessarily even those of one of the four countries whose flags your functions will create); the values of `height` and `width` tell it the size of the flags to be compared.

The function should return 1 if the values of `height` and `width` are valid and all of the flags' elements **within** the range indicated by the values of `height` and `width` are identical, and 0 otherwise. Note that elements **beyond** the first `height` rows and `width` columns should **not** be compared, and 1 can be returned even if those elements differ. `height` or width would be invalid if either is greater than `MAX` or less than 1.

The function may assume that `flag1` and `flag2` are valid arrays with `MAX` columns, and at least `height` rows.

## A Development procedure

### A.1 Obtaining the project files

You can obtain the necessary project files by logging into one of the Grace machines, copying a tarfile from the public class directory to your course disk space, and untarring it, using commands like:

```
cd ~/216
cp ~/216public/project1/project1.tgz .
tar -zxvf project1.tgz
```

The tarfile contains the header file `flags.h` that contains prototypes of the functions that you must write, the hidden file `.submit` that will allow you to submit your project, and the public tests. When untarred, the tarfile will create a directory named `project1` containing these files. You **must** do all of your programming project in your course disk space (using the `cd` command above) for this class, otherwise we **will not accept** your submission.

After extracting the files from the tarfile, `cd` to the `project1` directory, and create a file named `flags.c` **(its name must be spelled and capitalized exactly as shown**, otherwise your program will not compile on the submit server) which will contain the implementation of the functions you are to write. `flags.c` should `#include` the `flags.h` header file. Then write the functions in `flags.c`.

### A.2 Compiling your code on the public tests

The public tests are just C programs, with `main()` functions that just call your functions in various ways. To compile your code (in `flags.c`) and run it with for example the first public test, which is in a file named `public1.c`, use the following command:

```
gcc public1.c flags.c -o public1.x
```

This will compile the program and place the resulting executable program in the file `public1.x` (assuming no problems are detected during compilation); use a different executable filename if you like. Of course, modify the command to use `public2.c` instead to compile your code for the second public test, etc.

This command will work right only if you set up your account properly in discussion section. If you did, then an alias was created for `gcc` that adds the options `-ansi`, `-pedantic-errors`, `-Wall`, and `-Werror` to the command; their meanings were explained in the recent exercise given in discussion section. These options are required for projects in this course. As mentioned, if your account was set up right they will be added to the compilation command above automatically. If your account isn't set up, so these options aren't added, your code may fail to work when submitted, even if it works fine when you run things yourself.

### A.3 Checking your results

The submit server can check that your output is correct, but as you will see below you are discouraged from submitting your project until after you have thoroughly tested it yourself. The expected outputs for all the public tests are also included in the project tarfile. Tor example, the public test `public1.c` has an associated output file named `public1.output`. It would be tedious and error–prone to have to compare your program's output to the expected results manually (meaning visually), however a UNIX utility named `diff` can perform this comparison automatically.

To verify your code passes a public test use a command like the following:

```
public1.x | diff -u - public1.output
```

This will send the output of the `public1.x` executable into the `diff` command, which will then compare it against the file `public1.output`– if there are any differences between them they will be displayed. If no differences exist between your output and the corect output diff will produce no output, and your code passed the test. (The `-u` option just controls the format of the output produced by `diff` if any differences are detected.)

## B Submitting your program

The project may not be set up on the submit server immediately when it is assigned, but if not it will be soon.

**Before** you submit your project, you must first check to make sure you have passed all the public tests, by running it yourself. Refer to the previous section for details on how to test for yourself that you have passed a test. Once you have verified that your code passes the public tests, submit by executing, in your project directory, the single command:

```
submit
```

This will prompt you for your UMD directory ID (not your nine–digit UID) and password, submit your files, and inform you if the submission succeeded. You **must** then log onto the submit server at the link below (there is also a link to the submit server from ELMS), and check whether your program worked right on the public tests there also!

https://submit.cs.umd.edu

(Note that unless you have versions of all six required functions that will at least compile, your program will fail to compile at all on the submit server.)

Do **not** submit programming assignments using the submit server's mechanism for uploading a jarfile or zipfile or individual files. You must use the command above, or for reasons described in the project grading policy, you may end up losing significant credit on the project.

Do not wait until the last minute to submit your program! The submission server enforces deadlines to the exact second. Being even one second beyond the exact time at the top of the first page means that your program will be late.

Project extensions will not be given to individual students as a result of hardware problems, network problems, power outages, etc., so you are urged to finish each assignment **early** so any such situations will not affect you even if they do arise. Finishing an assignment at least one day before its deadline will allow time to reread the assignment and any clarifications or corrections to it, and ensure you have not missed anything that could cause you to lose credit, or to make any necessary changes if you have.

# C Grading criteria

Your project grade will be determined according to the following weights:

| | |
|---|---|
| Results of public tests | 40% |
| Results of secret tests | 45% |
| Code style grading | 15% |

Secret tests, and their results, will not be visible on the submit server or released until some time after the project's late deadline has passed. As for all projects in this class, the small set of public tests will not be comprehensive, and you will need to do testing on your own to ensure the correctness of your functions. The results of the secret tests for this project, and subsequent ones, will be a significant part of your project grade.

## C.1 Style grading

Style grading will take place after the late deadline for this project has passed. The only file that will be graded for style is `flags.c`. The course project grading policy mentioned below will have full details about the criteria that will be used for style grading, so be sure to read it carefully.

# D Project–specific requirements

Be sure to **carefully read** the project grading policy that will be posted on ELMS shortly. It will describe in detail how projects are graded, which project submission will be graded if you submit more than once, what good programming style is considered to consist of, etc. This section explains any differences for this project compared to the general requirements in that document.

- You **cannot** modify the declaration of anything in the header file `flags.h` or add anything to `flags.h`. Since your submission will be compiled on the submit server using our version of this file, if you change it your program will likely not compile when graded.

  Your code **may not** comprise any source (.c) files other than `flags.c`, so all your code **must** be in `flags.c`.

- You should use only the features of C that have been covered so far in class (and in the corresponding chapters of the text).

- If you can't submit your project, for any reason, something must be set up incorrectly in your account (the settings that were done during discussion section).

- Do **not** use the submit server to keep backups of your project code. Having a large number of unnecessary submissions just slows the submit server down for everyone. Instead use the UNIX commands (see below) to frequently make copies of your project files, with different filenames or in different directories. Similarly, do **not** use the submit server to check if your program compiles, or to check its results on the public tests. You should be compiling and testing your code yourself, as described above, before submitting.

- For this project you will **lose one point** from your final project score for every submission that you make in excess of 15 submissions. You will also **lose one point** for every submission that does not compile, in excess of four noncompiling submissions. Therefore be sure to compile, run, and test your project's results **before** submitting it. We hope everyone will check their code themselves carefully, and no one will incur these penalties.

## E Other notes

- Be sure to keep backup copies of your project and any other important files (like your own tests) in a different directory than where your project is located. It's a good idea to save backup copies every time you log in or log out, if not more frequently. Recall from the first discussion sections and the UNIX tutorial that the `-r` option to `cp` will make a copy of a whole directory and all its contents, so a command like "`cp -r project1 project1.bak`" would make a new directory containing a copy of everything in the `project1` directory.

  Furthermore, it's a good practice to stop every few minutes, or even after typing every statement, and save your file in Emacs, so that an unexpected power outage or lost network connection would only cause a tiny amount of work to be lost.

- If you get an error that says "Segmentation fault" when running your program this is an indication of a fatal error. For the moment, the best way to debug this or any other type of error is to add debug `printf` statements to your functions, to print the values of the variables that are being used. Note that for technical reasons to be mentioned later, all debug `printf`s should end in a newline character to ensure that their results appear correctly.

- Write your own tests, and test each function as you write it, before going on!

  If you have a problem with your code and have to come to the TAs' office hours, you **must** come with tests you have written that illustrate the problem (tests that you wrote in addition to the public tests). In particular you will need to show the smallest test you were able to write that illustrates the problem, so whatever the cause is can be narrowed down as much as possible before the TAs even start helping you.

- Recall that the course syllabus says that all your projects must work on **at least half of the public tests** (by the end of the semester) in order for you to be eligible to pass the course. See the project grading policy for full details. (The compilation penalties mentioned above only apply to submissions made before the end of the project late submission period.)

## F Academic integrity

Please **carefully read** the academic honesty section of the course syllabus. **Any evidence** of impermissible cooperation on projects, use of disallowed materials or resources, or unauthorized use of computer accounts, **will be submitted** to the Student Honor Council, which could result in an XF for the course, or suspension or expulsion from the University. Be sure you understand what you are and what you are not permitted to do in regards to academic integrity when it comes to project assignments. These policies apply to all students, and the Student Honor Council does not consider lack of knowledge of the policies to be a defense for violating them. Full information is found in the course syllabus– please review it at this time.