Programming Assignment 2 (PA2) - DisplayImage

Milestone Due: **Wednesday, May 1 @ 11:59 pm**
Final Due: **Wednesday, May 8 @ 11:59 pm**

Assignment Overview

In this assignment you will be displaying an 8x8 "image" using commands implemented with bit operations. Your program will be able to manipulate each pixel of the 8x8 image using commands (setting a pixel, clearing a pixel, toggling a pixel, etc). Furthermore, you will implement more complex commands such as scrolling the image and inverting the image. At the end of the assignment, you should be able to use the program to display and manipulate different images.

The purpose of this programming assignment is to gain more experience with ARM assembly, bitwise operations, memory loads and stores (ldr/str), and allocating local variables on the runtime stack and accessing them relative to the frame pointer (fp). You will use Standard C Library routines such as `fgets()`, `strtok()`, `isdigit()`, and `toupper()`. You will also be unit testing functions to ensure correctness of the program.

IMPORTANT NOTE for Assembly routines:

1. Make sure you **do not** use registers other than **r0-r3** as scratch registers in your assembly functions. Allocate local variables on the stack instead.
2. Only **fp, lr** are pushed to the stack.
3. Note that values in **r0-r3** will not be preserved after function calls.
4. Remember to note in the **function header** about what stack variables you used (i.e somevar -- [fp - 8] -- for calculating length, etc). Stack variables include local variables, formal parameters, and additional arguments. Refer to style guidelines for the specific way to format the header.

Grading

- **README: 10 points** - See README Requirements here and questions below
  - http://cseweb.ucsd.edu/~ricko/CSE30READMEGuidelines.pdf
- **Compiling: 5 points** - Using our Makefile; no warnings. If what you turn in does not compile with the given Makefile, you will receive 0 points for this assignment. **NO EXCEPTIONS!**
- **Style: 20 points** - See Style Requirements here
  - http://cseweb.ucsd.edu/~ricko/CSE30StyleGuidelines.pdf
- **Correctness: 65 points**
  - **Milestone (20 points)** - To be distributed across the Milestone functions (see below)
  - Make sure you have all files tracked in Git.
- **Extra Credit: 8 points** - View Extra Credit section for more information.
- **Wrong Language:** You will lose 10 points for each module in the wrong language, C vs. Assembly or vice versa.

NOTE: If what you turn in does not compile with given Makefile, you will receive 0 points for this assignment.

Getting Started

Follow these steps to acquire the starter files and prepare your Git repository.

Gathering Starter Files:

The first step is to gather all the appropriate files for this assignment. Connect to pi-cluster via ssh.

```
$ ssh cs30xyz@pi-cluster.ucsd.edu
```

Note: For this PA, you **DO NOT** need to specifically ssh into any pi-cluster node (i.e no need to specify node 001-039); you can work on this PA on any node in the pi-cluster.

Create and enter the pa2 working directory.

```
$ mkdir ~/pa2
$ cd ~/pa2
```

Copy the starter files from the public directory.

```
$ cp ~/../public/pa2StarterFiles/* ~/pa2/
```

Starter files provided:

`pa2.h`	`pa2Strings.h`	`shouldPrompt.c`
`test.h`	`testscrollHorizontal.c`	`Makefile`

You will also need to copy over your outputChar.s, myRem.s, and intervalContains.s from pa1.

```
$ cp ~/pa1/outputChar.s ~/pa2/
$ cp ~/pa1/myRem.s ~/pa2/
$ cp ~/pa1/intervalContains.s ~/pa2/
```

Preparing Git Repository:

You are required to use Git with this and all future programming assignments. Refer to the PA0 writeup for how to set up your local git repository.

Walkthrough & Example Input

Public Executable

A sample stripped executable provided for you to try and compare your output against is available in the public directory. Note that you cannot copy it to your own directory; you can only run it using the following command (where you will also pass in the command line arguments):

```
$ ~/../public/pa2test
```

NOTE:

1. The output of your program MUST match exactly as it appears in the `pa2test` output. You need to pay attention to everything in the output, from the order of the error messages to the small things like extra newlines at the end (or beginning, or middle, or everywhere)!
2. **We are not giving you any sample outputs, instead you are provided some example inputs. You are responsible for trying out all functionality of the program; the list of example inputs is not exhaustive or complete. It is important that you fully understand how the program works and you test your final solution thoroughly against the executable.**

Walkthrough

Let's try out the executable! Here is the usage message:

```
Usage: ./pa2 [on [off]]
    on  -- The character printed for pixels that are on. Defaults to @.
    off -- The character printed for pixels that are off. Defaults to -.

Both on and off must be single characters in the ASCII range [32, 126].
```

Notice that the executable can take up to two arguments. Square brackets around an argument name means that the argument is optional. The nested square brackets here mean that `on` is optional, but in order to specify `off`, you need to have specified `on`.

For simplicity, let's run the executable with no arguments (**bolded text** is user-input).

```
cs30xyz@pi-cluster-001:pa2$ ./pa2

Bit ops:
    set    part0 part1 -- Turns pixels on where part0/part1 bits are 1.
    clear  part0 part1 -- Turns pixels off where part0/part1 bits are 1.
    toggle part0 part1 -- Flips pixels where part0/part1 bits are 1.
    invert             -- Flips all pixels in the image.
Strings:
    character c -- Sets the character c into the image.
                Allowed characters: A-Z, a-z, 0-9.
    say str     -- Displays characters from str as a sequence of images.
Movement:
    scrollHorizontal offset   -- Scrolls image horizontally.
                                 positive offset scrolls right
                                 negative offset scrolls left
    scrollVertical offset  -- Scrolls image vertically.
                              positive offset scrolls down
                              negative offset scrolls up
    translate offsetH offsetV -- Scrolls image offsetH horizontally and
                                 offsetV vertically.
Misc:
    help -- Shows this help message.

Press Ctrl + D to exit.

>>>
```

Wow, that's a mouthful. This is just a helpful usage message that lists all of the available commands in the program. After the usage message, it prompts us to enter input with the three chevrons ">>>".

Let's try the `set` command.

```
>>> set 30 420
--------
--------
--------
---@@@@-
--------
--------
--------
-------@
@-@--@--
```

You can see that an 8x8 image pops up. You can think of each character as a "pixel" where each hyphen (-) is an "off pixel" and each at-sign (@) is an "on pixel". What determines which pixels are on/off is the bit-pattern of the numbers we provided (30 and 420). The bit pattern is 64 bits overall (two 32-bit unsigned integers) and is wrapped around every 8 bits to form the 8x8 image. Each integer is displayed out in **big-endian byte order.**

The `set` command in particular will always turn on pixels where the bit-pattern of the two numbers has 1 bits. Since we know that the 2's complement representation of -1 is all 1 bits (0xFFFFFFFF), let's try using the set command with -1:

```
>>> set -1 0
@@@@@@@@
@@@@@@@@
@@@@@@@@
@@@@@@@@
--------
--------
-------@
@-@--@--
```

Notice now that the whole upper half of the image is all on, since we set the first number to be -1. Since we set the second number to 0, it does nothing to the second half of the image. Remember -- `set` will only turn pixels on, not off (more on that later).

Let's clear the image so we have a blank canvas. The clear command is the opposite of `set`. It will turn off bits where the inputs (in this case, -1 and -1) have 1 bits. Since -1 is all 1s, the whole image is cleared.

```
>>> clear -1 -1
--------
--------
--------
--------
--------
--------
--------
--------
```


Back to Top

You can also specify the pattern using hexadecimal or octal values. These might be easier to work with since they have a more direct relationship with the binary representation.

```
>>> set 0xF00F0FF0 0123456
@@@@----
----@@@@
----@@@@
@@@@----
--------
--------
@-@--@@@
--@-@@@-
```

Now let's take a look at the `character` command and the movement commands.

```
>>> character f
--------
-@@@@@@-
-@------
-@@@@---
-@------
-@------
-@------
--------
>>> scrollHorizontal 3
--------
@@--@@@@
----@---
----@@@@
----@---
----@---
----@---
--------
```

The `character` command completely replaces the image with a pixel pattern of the given character. The `scrollHorizontal` command will scroll the image horizontally, wrapping pixels around the sides. The number indicates how many pixels (bits) to scroll by. A positive number will scroll the image to the right, and a negative number will scroll the image to the left.

Feel free to try the rest of the commands before pressing Ctrl+D (^D) to exit.

```
>>> ^D
cs30xyz@pi-cluster-001:pa2$
```

Example Inputs

Example input that has error output:

```
cs30xyz@pi-cluster-001:pa2$ ./pa2 a a a
cs30xyz@pi-cluster-001:pa2$ ./pa2 12 213
cs30xyz@pi-cluster-001:pa2$ ./pa2 $'\0'
```

Example input that has normal output:

```
cs30xyz@pi-cluster-001:pa2$ ./pa2 a a
cs30xyz@pi-cluster-001:pa2$ ./pa2 b
cs30xyz@pi-cluster-001:pa2$ ./pa2 a \\
```


Back to Top

Once valid input was used to run the program, it will be in user interactive mode [**NOTE: all output starting from this point will be printed to stdout]**:

Example command user inputs:

```
>>> set 12 12
>>> set 0x12345678 0x87654321
>>> clear 0xFFFFFFFF
>>> clear 0xFFFFFFFF 0xFFFFFFFF
>>> help
```

*Commands will be explained in detail in their corresponding function description.

Detailed Overview

The function prototypes for the various C and Assembly functions are as follows.

C routines:

```
int shouldPrompt( void );
void loadPatternString( unsigned int pattern[], const char * patternStr );
void scrollHorizontal( unsigned int pattern[], int offset );
int findCommand( const char * cmdString, const char * commands[] );
void say( const char * str, char on, char off );
void commandLoop( char on, char off );
int main( int argc, char * argv[] );
```

Assembly routines:

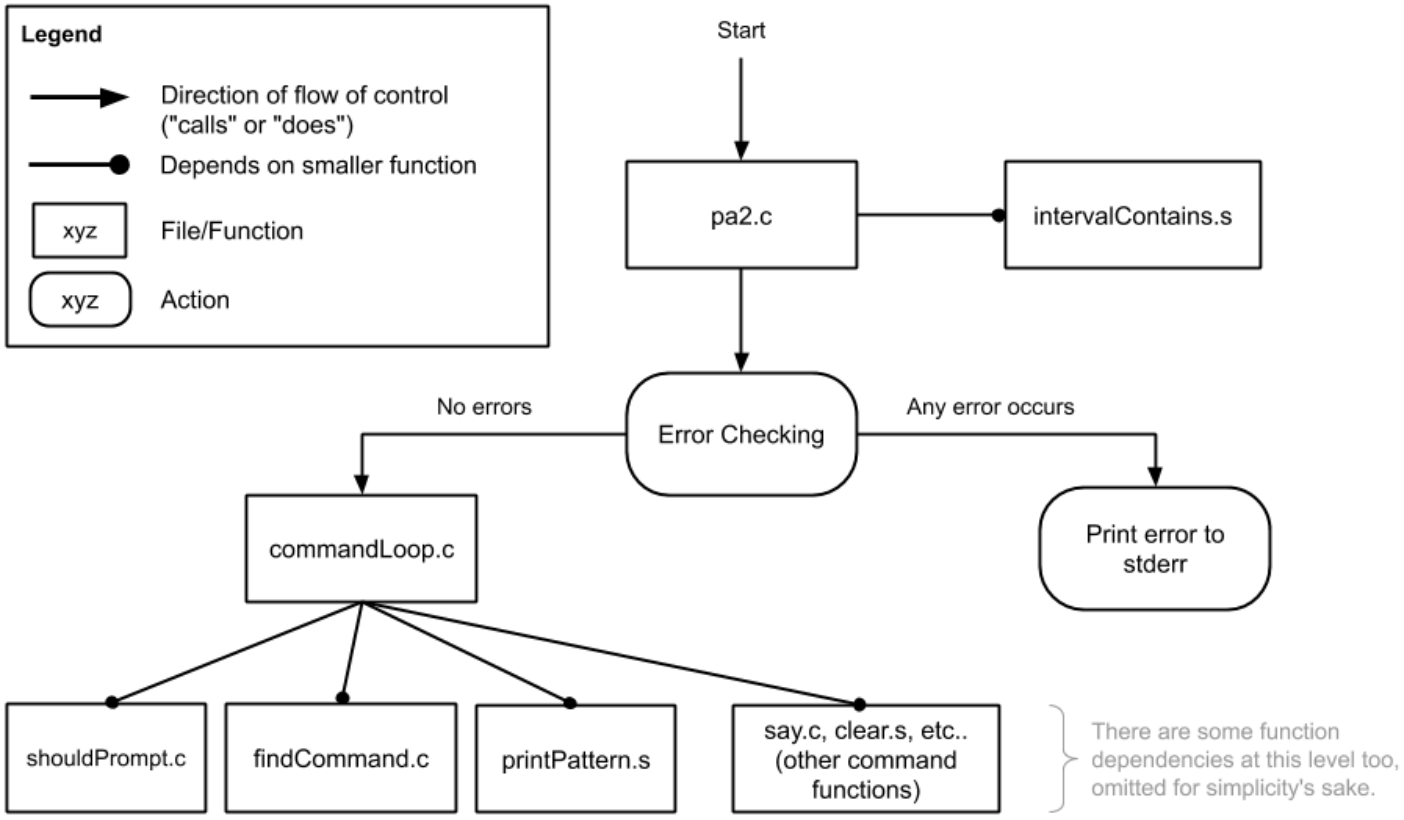
```
void outputChar( char ch );
int intervalContains( int start, int end, int value );
int myRem( int dividend, int divisor );
void set( unsigned int pattern[], unsigned int part0,
          unsigned int part1 );
void clear( unsigned int pattern[], unsigned int part0,
            unsigned int part1 );
void toggle( unsigned int pattern[], unsigned int part0,
             unsigned int part1 );
int character( unsigned int pattern[], char ch,
               const char * alphabetPatterns[], const char * digitPatterns[] );
void scrollVertical( unsigned int pattern[], int offset );
void invert( unsigned int pattern[] );
void translate( unsigned int pattern[], int hOffset, int vOffset );
void printPattern( unsigned int pattern[], char on, char off );
```

For the Milestone, you will need to complete:

`set.s`	`clear.s`	`toggle.s`
`character.s`	`loadPatternString.c`	`scrollHorizontal.c`
`scrollVertical.s`	`findCommand.c`	
`testset.c`	`testclear.c`	`testtoggle.c`
`testcharacter.c`	`testloadPatternString.c`	`testscrollHorizontal.c`
`testscrollVertical.c`	`testfindCommand.c`	

Process Overview:

The following is an explanation of the main tasks of the assignment, and how the individual functions work together to form the whole program.



The program is implemented, at a high-level, in the following steps:

1. Parse command-line arguments, noting whether the user provided an on/off character instead of the default.
    a. If any error occurs, print the corresponding error message to stderr and exit.
2. Run the command loop:
    a. Read user input line-by-line.
    b. For each line, parse the command (and print out any appropriate error messages).
    c. Execute the command and display the image.

Milestone Functions to be Written

Back to Top

Listed below are the modules to be written for the milestone.

Note: For all of the assembly milestone functions, you are not allowed to use the ARM instructions `bic` or `ror`. You are also not allowed to use `ldrb/strb` to load/store individual bytes in the pattern array. You will lose significant points if you do either.

---

set.s

```
void set( unsigned int pattern[], unsigned int part0, unsigned int part1 );
```

Turns on the specified bits in `pattern` with the bit patterns `part0` and `part1`. If a bit value in `part0` or `part1` is 1, then its corresponding bit in `pattern` should also become 1. However, if the bit value in `part0` or `part1` is 0, then its corresponding bit in `pattern` should remain the same.

For example:

Before set	`pattern[0]: 0x00100001`	`pattern[1]: 0x1F00F001`
	`part0: 0x420C5E30`	`part1: 0xF001F008`
After set	`pattern[0]: 0x421C5E31`	`pattern[1]: 0xFF01F009`

---

clear.s

```
void clear( unsigned int pattern[], unsigned int part0, unsigned int part1 );
```

Works similar to `set()`, except this function turns off the specified bits in `pattern` with the bit patterns `part0` and `part1`. If a bit value in `part0` or `part1` is 1, then its corresponding bit in `pattern` should become 0. However, if the bit value in `part0` or `part1` is 0, then its corresponding bit in `pattern` should remain the same.

For example:

Before clear	`pattern[0]: 0xFF00FF00`	`pattern[1]: 0xAAAAAAAA`
	`part0: 0xFF000000`	`part1: 0x88888888`
After clear	`pattern[0]: 0x0000FF00`	`pattern[1]: 0x22222222`

---

toggle.s

```
void toggle( unsigned int pattern[], unsigned int part0, unsigned int part1 );
```

Toggles all the bits in `pattern` specified in `part0` and `part1`. If the bit value in `part0` or `part1` is 1, then its corresponding bit in `pattern` should invert (i.e. 1 becomes 0, and 0 becomes 1). However, if the bit value in `part0` or `part1` is 0, then the corresponding bit in `pattern` should remain the same.

For example:

Before toggle	`pattern[0]: 0xCAFEBABE`	`pattern[1]: 0xDECAFBAD`
	`part0: 0x420C5E30`	`part1: 0xDEADBEEF`
After toggle	`pattern[0]: 0x88F2E48E`	`pattern[1]: 0x00674542`

---

loadPatternString.c

```
void loadPatternString( unsigned int pattern[], const char * patternStr );
```

Goes through each character in `patternStr` and, depending on its value, sets the bits in `pattern` to either be on or off. The `patternStr` is assumed to be exactly 64 characters long, where each `'@'` (`DEFAULT_ON_CHAR`) character represents an "on" bit and each `'-'` (`DEFAULT_OFF_CHAR`) character represents an "off" bit.

For example:

`patternStr: "@-@-@-@-@-@-@-@-@-@-@-@-@-@-@-@-@-@-@-@-@-@-@-@-@-@-@-@-@-@-@-@-"`
will set `pattern` to
`pattern[0]: 0xAAAAAAAA`
`pattern[1]: 0xAAAAAAAA`

---

character.s

```
int character( unsigned int pattern[], char ch,
               const char * alphabetPatterns[], const char * digitPatterns[] );
```

Fills `pattern` with the bit pattern of `ch`. If `ch` is a letter, look up its bit pattern in `alphabetPatterns` (uppercase letters A-Z). Otherwise, if `ch` is a number, then look up its bit pattern in `digitPatterns` (digits 0-9). If its bit pattern is found, use it to set the bits in `pattern`.

Note: This should be case insensitive. If `ch` is a lowercase letter, we want to fill `pattern` with the bit pattern of the uppercase version of `ch` (hint: `man -s3 toupper`).

Reasons for error:

- If `ch` is not a letter or a number, return -1.

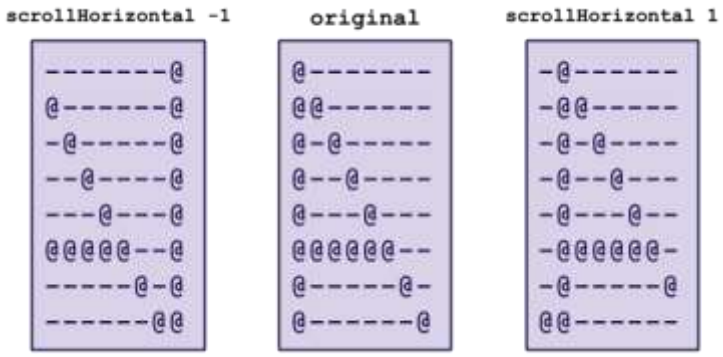
Return Value: -1 if `ch` is not a letter or a number, else return 0.

---

scrollHorizontal.c

```
void scrollHorizontal( unsigned int pattern[], int offset );
```

Scroll `pattern` horizontally based on `offset`. If `offset` is positive, scroll right and wrap the bits around to the left. If `offset` is negative, scroll left and wrap the bits around to the right.

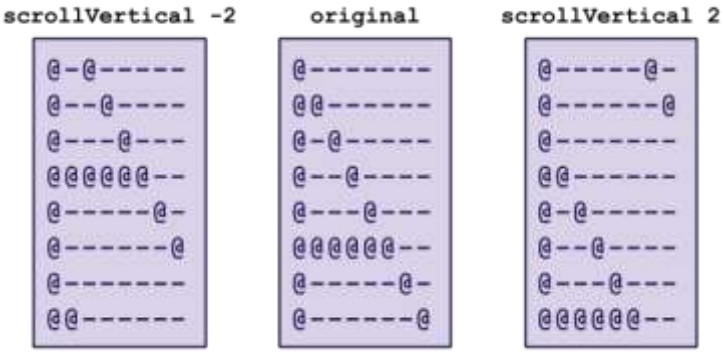


---

scrollVertical.s

```
void scrollVertical( unsigned int pattern[], int offset );
```

Scroll `pattern` vertically based on `offset`. If `offset` is positive, scroll down and wrap the bits around to the top. If `offset` is negative, scroll up and wrap the bits around to the bottom.



---

findCommand.c

```
int findCommand( const char * cmdString, const char * commands[] );
```

Look for the given `cmdString` in `commands` and return its index.

Reasons for error:

- `cmdString` is null
- `commands` is null
- `cmdString` is not in `commands`

Return Value: -1 if either argument is null or if `cmdString` is not found. Otherwise, return the index of the command.

Post-Milestone Functions to be Written

Listed below are the modules to be written after the milestone functions are complete.

---

outputChar.s

```
void outputChar( char ch );
```

Copy from PA1. No changes necessary.

---

intervalContains.s

```
int intervalContains( int start, int end, int value );
```

Copy from PA1. No changes necessary.

---

myRem.s

```
int myRem( int dividend, int divisor );
```

Copy from PA1. No changes necessary.

---

shouldPrompt.c

```
int shouldPrompt( void );
```

Given in the starter files. No need to change. Detects if input on stdin is coming from a terminal or from file redirection.

Return Value: 1 if input is from terminal, 0 otherwise.

---

invert.s

```
void invert( unsigned int pattern[] );
```

Inverts all the bits in pattern (0 to 1 and 1 to 0). Hint: you shouldn't need to write this function from scratch.

For example:

Before invert	`pattern[0]: 0x1CEB00DA`	`pattern[1]: 0xDEADBEEF`
After invert	`pattern[0]: 0xE314FF25`	`pattern[1]: 0x21524110`

---

translate.s

```
void translate( unsigned int pattern[], int hOffset, int vOffset );
```

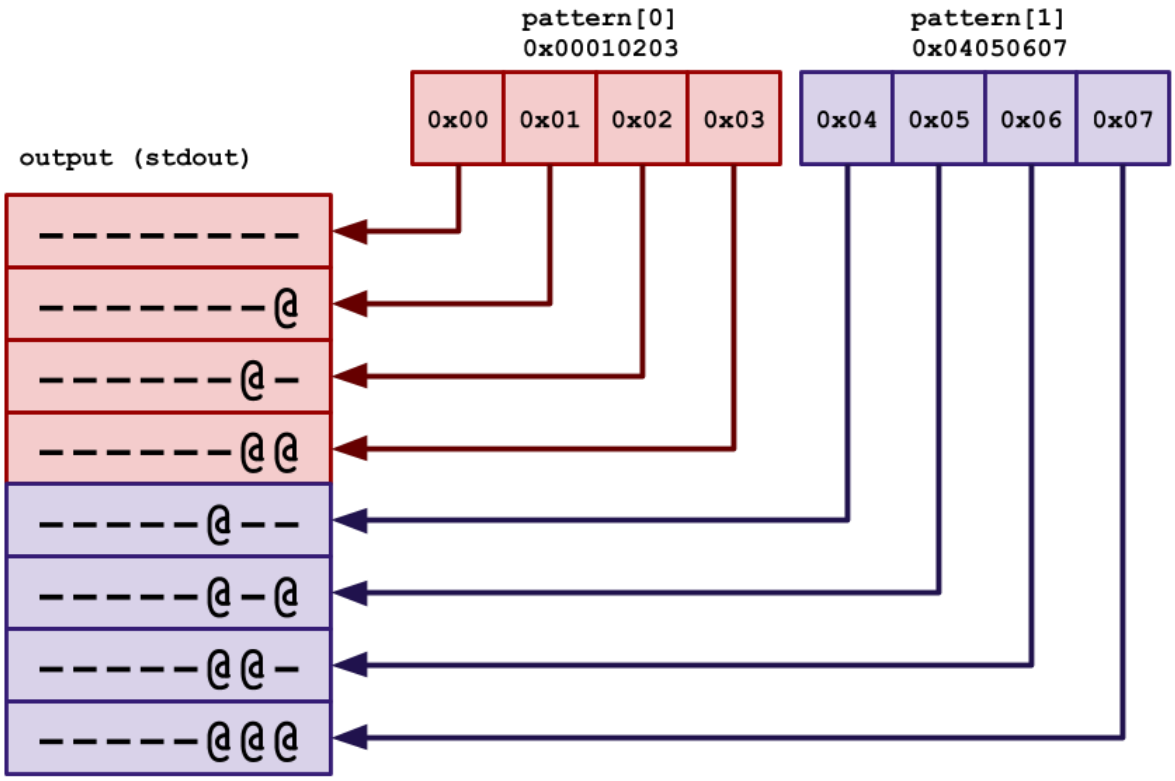
Translates `pattern` horizontally with `hOffset` and vertically with `vOffset`. A translation is the same as scrolling the image, but scrolling can go in both directions (horizontal and vertical). The same positive/negative offset and wrapping rules apply as in `scrollHorizontal` and `scrollVertical`. Hint: you shouldn't write this function from scratch.

---

printPattern.s

```
void printPattern( unsigned int pattern[], char on, char off );
```

Prints out the `pattern` as an 8x8 grid. Each bit in `pattern` will be represented by a character, with each byte being a row in the grid. If the bit is 1, print `on`, otherwise print `off`.



---

say.c

```
void say( const char * str, char on, char off );
```

Prints out each character in `str`. For each character in `str`, first look up its bit pattern in the appropriate lookup table (hint: some other function you wrote can be helpful). If it is not a valid character, print out error message (`STR_ERR_SAY_INVALID_CHAR`) to stdout and move on to the next character in `str`. Otherwise, print out the character pattern using the specified `on` and `off` bits.

Reasons for error:

- A character in `str` is not found in the lookup table. In this case, print the corresponding error message to stdout.

---

commandLoop.c

```
void commandLoop( char on, char off );
```

This function allows the user to interactively manipulate the bits in the pattern.

This function **first** prints out a list of possible commands (provided in `pa2Strings.h:STR_COMMAND_USAGE`). It should only print this usage message if `shouldPrompt()` returns 1.

The function then prompts the user (using `PRINT_PROMPT`) to enter a command. This function continues to re-prompt the user until the user enters `ctrl-D` (sends the `EOF` character). This is written in more detail below.

Note: Make sure to print to **stdout** for all output in this function.

An overview of the user-interactive process:

1. Read the command and argument(s) entered by the user (`man -s3 fgets`). Note that `fgets()` will include the newline character as part of the string read, so immediately after your call to `fgets()`, replace the newline character with the `NUL` character (`man -s3 strchr`).

   You should read user input in a for-loop with the following structure:
   `for (PRINT_PROMPT; condition; PRINT_PROMPT) { /* Parse line of input */ }`

   All the steps below should be within this for-loop structure. Note that when the following instructions say "reprompt the user" or "reprompt", it means you should print the prompt again and start parsing a new line of input. Hint: something about the way this for-loop is written will make this easier than it sounds.

2. After reading a line, parse the command that was entered by the user by using `strtok()` and the provided token separator string `DELIM`. Extract the command name portion of the user's input from what was read by `fgets()`.
   - If the string could not be tokenized (i.e. the line was empty), reprompt the user.
   - Otherwise, use `findCommand()` to determine which command was entered.
     - If the command is unrecognized, then print the `STR_ERR_BAD_COMMAND` error message and reprompt the user.

Now that you have the index of the command, you will need to do different things depending on the command entered:

3. If the "help" command was entered, print the `STR_COMMAND_USAGE` message and reprompt the user.

4. If the "set", "clear", or "toggle" commands were entered, parse the two arguments (part0/part1) that follow, using `strtok()`.
   - If there are any missing arguments, print the `STR_ERR_COMMAND_MISSING_ARG` error message, then reprompt.
   - If there are trailing arguments, print the `STR_ERR_EXTRA_ARG` error message and reprompt.
   - If no errors have occurred so far, try to convert the two arguments into **unsigned** ints (hint: `man -s3 strtoul`).
     - If an error occurs in the parsing for either of the arguments, print the `STR_ERR_PATTERN_INVALID` error message and reprompt. (hint: `man -s3 errno`)
   - If converting part0/part1 to unsigned ints was successful, then call the corresponding function (set, clear, toggle) and print the pattern.

5. If the "invert" command was entered, ensure that no arguments following the command were entered.
   - If there was an extra command, print the `STR_ERR_EXTRA_ARG` error message and reprompt.
   - Otherwise, call `invert()` and print the resulting pattern.

6. If the "character" command was entered, parse the next argument.
   - If no argument was provided, print the `STR_ERR_COMMAND_MISSING_ARG` error message and reprompt.
   - If there are trailing arguments, print the `STR_ERR_EXTRA_ARG` error message and reprompt.
   - If the argument is not a single character, print the `STR_ERR_CHAR_COMMAND_SINGLE` error message and reprompt.

    - Extract the single character from the argument, then try to call `character()` to set the pattern. If it was unsuccessful in setting the pattern, print `STR_ERR_CHAR_INVALID` and reprompt.
    - Otherwise, if everything is successful, print the pattern.

7. If the "say" command was entered, parse the next argument.
    - If no argument was provided, print the `STR_ERR_COMMAND_MISSING_ARG` error message and reprompt.
    - If there are trailing arguments, print the `STR_ERR_EXTRA_ARG` error message and reprompt.
    - Otherwise, call `say()` with the string.

8. If the "scrollHorizontal" or "scrollVertical" commands were entered, parse the next argument.
    - If no argument was provided, print the `STR_ERR_COMMAND_MISSING_ARG` error message and reprompt.
    - If there are trailing arguments, print the `STR_ERR_EXTRA_ARG` error message and reprompt.
    - If no errors have occurred so far, try to convert the argument into a **signed** int. (hint: `man -s3 strtol`).
        - If an error occurs in the parsing, print the `STR_ERR_INT_INVALID` error message and reprompt.
    - Otherwise, call the corresponding function and print the pattern.

9. If the "translate" command was entered, parse the next two arguments.
    - If there are any missing arguments, print the `STR_ERR_COMMAND_MISSING_ARG` error message and reprompt.
    - If there are trailing arguments, print the `STR_ERR_EXTRA_ARG` error message and reprompt.
    - If no errors have occurred so far, try to convert the two arguments into **signed** ints.
        - If an error has occurred for either of the arguments, print the `STR_ERR_INT_INVALID` error message and reprompt.
    - Otherwise, call `translate()` and print the pattern.

10. If `Ctrl+D` was entered, then print the `STR_END_PROMPT` before terminating the program.

Remember, if at any point you encounter an error in parsing the argument, print the corresponding error message, then reprompt the user and restart the procedure for parsing commands and arguments.

Otherwise, if you have successfully parsed the command and valid argument(s), then call on the respective function of the command the user entered with a subsequent call on `printPattern()`. After doing so, print out the prompt and repeat the parsing procedures above.

---

pa2.c

```
int main( int argc, char * argv[] );
```

This is the main driver of your program. Its main tasks are to parse any command line arguments that are passed in and start the user-interactive mode. By default, the "on" character and the "off" character should be `DEFAULT_ON_CHAR` and `DEFAULT_OFF_CHAR`, respectively.

If there are any command line arguments, you should first check to make sure the correct number of arguments was passed in. Note that there can be 1 or 2 arguments passed after the executable name. After

checking the number (if there are any), parse the first argument as the "on" character and the second argument as the "off" character. If an error occurs in parsing, immediately return after the first error, indicating failure and print the respective error message (see "Reasons for error" for more detail).

If no errors have occurred, then enter interactive mode by calling `commandLoop()`.

Reasons for error:
In this file, ***as soon as* an error is detected (in this order)**, print the appropriate error message(s) to `stderr` (see `pa2Strings.h`), and *return* `EXIT_FAILURE`.

- The number arguments, excluding the program name, was not in between `MIN_EXPECTED_ARGS` and `MAX_EXPECTED_ARGS`. (prints `STR_USAGE`)
- The first argument ("on character") was not a single character. (prints `STR_ERR_SINGLE_CHAR`)
- The first argument ("on character") was not in between `MIN_ASCII` and `MAX_ASCII`. (prints `STR_ERR_CHAR_RANGE`)
- The second argument ("off character") was not a single character. (prints `STR_ERR_SINGLE_CHAR`)
- The second argument ("off character") was not in between `MIN_ASCII` and `MAX_ASCII`. (prints `STR_ERR_CHAR_RANGE`)

Return Value: `EXIT_SUCCESS` on success, `EXIT_FAILURE` on failure.

Unit Testing

You are provided with only one basic unit test file for the Milestone function, `scrollHorizontal()`. This file only has minimal test cases and is only meant to give you an idea of how to write your own unit test files. **You must write unit test files for each of the Milestone functions, as well as add several of your own thorough test cases to all of the unit test files. You need to add as many unit tests as necessary to cover all possible use cases of each function. You will lose points if you don't do this!** You are responsible for making sure you thoroughly test your functions. Make sure you think about boundary cases, special cases, general cases, extreme limits, error cases, etc. as appropriate for each function. The Makefile includes the rules for compiling these tests. Keep in mind that your unit tests will not build until all required files for that specific unit test have been written. These test files **will be collected with the Milestone**, they must be complete before you submit your Milestone.

Unit tests you need to complete:

```
testset.c
testclear.c
testtoggle.c
testcharacter.c
testloadPatternString.c
testscrollHorizontal.c
testscrollVertical.c
testfindCommand.c
```

To compile:

```
$ make testscrollHorizontal
```

To run:

```
$ ./testscrollHorizontal
```

(Replace "`testscrollHorizontal`" with the appropriate file names to compile and run the other unit tests)

We also supply you an extra set of "optimized" Makefile targets for your tester files. These targets will compile your tester file with all optimizations turned on, similar to how your Milestone functions will be compiled for grading. Sometimes you will get (un)lucky where your program appears to work even when there is something wrong (like incorrect memory layout or modifying a register you shouldn't). Compiling with optimizations will

expose some of these hidden problems. Again, this is how the milestone will be tested during grading, so use the optimization targets to try to expose these issues.

However, compiling this way prevents you from using gdb to debug, so make sure to compile the regular way when debugging and try the optimized targets after your tests pass.

To compile with optimizations on:

```
$ make testscrollHorizontal-opt
```

To run:

```
$ ./testscrollHorizontal-opt
```

README File

Remember to follow all of the guidelines outlined in the README Guidelines. If you did the extra credit, write a program description for it in the README file as well.

Questions to Answer in your README:

1. How would you rename a file called `foo.txt` to `foobar.txt`?
2. How would you make a copy of a file called `simpleboy.txt` in a directory called `midwest` one level above your current directory?
3. What happens when you select text and then middle click in the vim editor when in insert/input mode?
4. (a) What is a .vimrc file, (b) and how do you create/edit them?
5. (a) What is the command to cut a full line of text to the clipboard in vim? (b) How do you paste it? (Both the questions refer to using the keyboard, not using the mouse).
6. How would you search for a string in vim?
7. How do you turn on line numbers in vim?
8. How can you quickly (with a single Linux command) change directory to a directory named fubar that is in your home (login) directory? Note that for this question, you cannot first change directory to your home directory and then change directory to fubar, as that would take two commands. State how you would do this with a single command no matter where your current working directory might be.
9. (a) How do you change the permissions on a file? (b) Specify the command to give write permission to the group.
10. Why are professional engineers expected to act with integrity?

Extra Credit

There are 8 points total for extra credit on this assignment.

- Early turnin: **[3 Points]** 72 hours before regular due date and time
  [2 Points] 48 hours before regular due date and time
  [1 Point] 24 hours before regular due date and time
  (it's only one of them, not more than 1)
- **[5 Points]** Implement some more displayImages commands
  - **[1 Point]** mirrorHorizontal
  - **[1 Point]** mirrorVertical
  - **[2 Points]** transpose
  - **[0.5 Point]** rotateClockwise
  - **[0.5 Point]** rotateCounterClockwise

Getting Started:

Copy over your commandLoop.c to a new file called commandLoopEC.c

```
$ cp ~/pa2/commandLoop.c ~/pa2/commandLoopEC.c
```

Compiling:

You can compile the extra credit program using the following command.

```
$ make pa2EC
```

Sample Executable:

A sample test executable is provided for the EC like the sample executable provided for the regular portion of the assignment. You can run the sample executable using the following command:

```
$ ~/../public/pa2ECtest
```

For extra credit, you will be implementing 5 new commands: `mirrorHorizontal`, `mirrorVertical`, `transpose`, `rotateClockwise`, and `rotateCounterClockwise`. All of these commands will be implemented in C, and use similar techniques to the commands from the main program. None of these commands take any arguments.

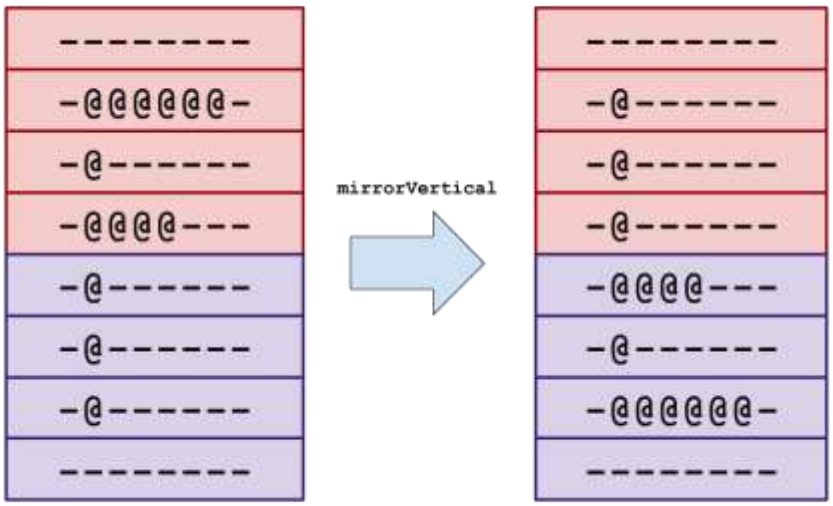
Once you've implemented the 5 new commands, modify your `commandLoopEC` so that it can handle the new commands.

---

mirrorVertical.c

```
void mirrorVertical( unsigned int pattern[] );
```

Modifies `pattern` such that the resulting image is mirrored along the x-axis:

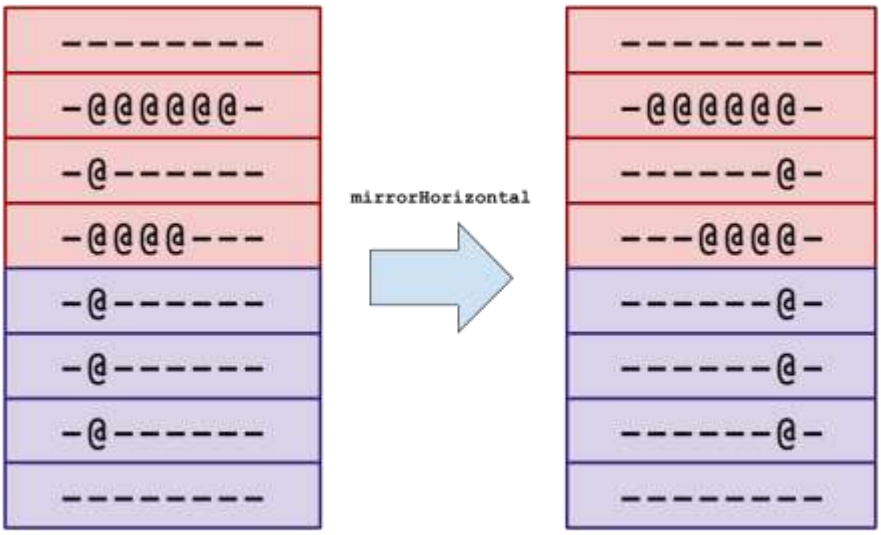


---

mirrorHorizontal.c

```
void mirrorHorizontal( unsigned int pattern[] );
```

Modifies `pattern` such that the resulting image is mirrored along the y-axis:

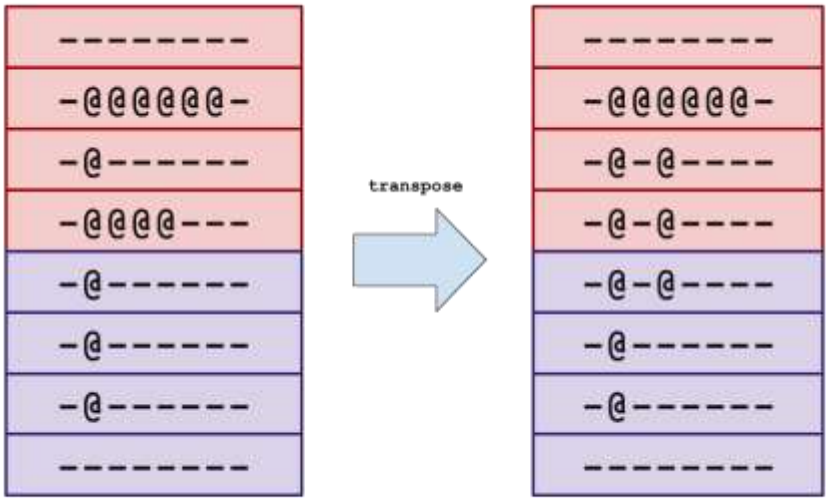


---

transpose.c

```
void transpose( unsigned int pattern[] );
```

Modifies `pattern` such that the resulting image is transposed across the diagonal of y = -x:

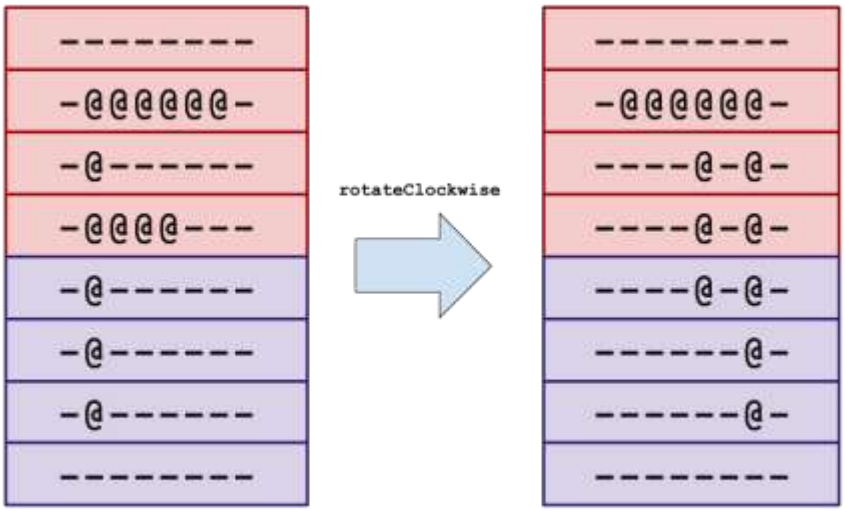


---

rotateClockwise.c

```
void rotateClockwise( unsigned int pattern[] );
```

Modifies `pattern` such that the resulting images is rotated 90° to the right:

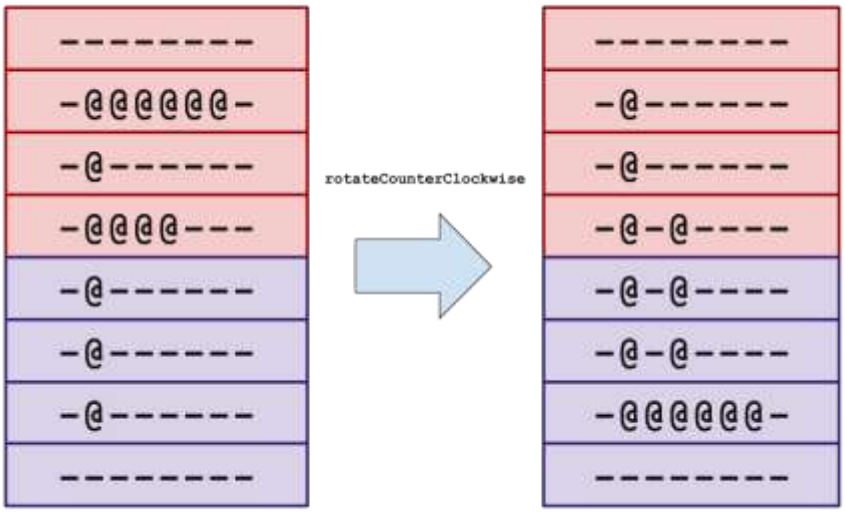


---

rotateCounterClockwise.c

```
void rotateCounterClockwise( unsigned int pattern[] );
```

Modifies `pattern` such that the resulting images is rotated 90° to the left:



---

commandLoopEC.c

```
void commandLoop( char on, char off );
```

Behaves exactly like `commandLoop.c`, except it includes the above EC commands for the user to execute, and prints the corresponding `STR_COMMAND_USAGE_EC` help message on program startup and for the "help" command.

Note: The file name is `commandLoopEC.c`, but the function is still called `commandLoop`.

Hint: None of the commands take any arguments. Which command from the main assignment took no arguments? You should implement the `commandLoop` for the 5 new EC commands like the command from the main assignment that took no arguments.

Turnin Summary

See the turnin instructions here. Your file names must match the below exactly; otherwise our Makefile will not find your files.

Milestone Turnin:
Due: Wednesday night, May 1 @ 11:59 pm

Files required for the Milestone:

set.s	clear.s	toggle.s
character.s	loadPatternString.c	scrollHorizontal.c
scrollVertical.s	findCommand.c	
testset.c	testclear.c	testtoggle.c
testcharacter.c	testloadPatternString.c	testscrollHorizontal.c
testscrollVertical.c	testfindCommand.c	

Final Turnin:
Due: Wednesday night, May 8 @ 11:59 pm

Files required for the Final Turn-in:

character.s	commandLoop.c	pa2.h
clear.s	findCommand.c	pa2Strings.h
intervalContains.s	loadPatternString.c	test.h
invert.s	pa2.c	Makefile
myRem.s	say.c	README
outputChar.s	scrollHorizontal.c	
printPattern.s	shouldPrompt.c	
scrollVertical.s	testcharacter.c	
set.s	testclear.c	
toggle.s	testfindCommand.c	
translate.s	testloadPatternString.c	
	testscrollHorizontal.c	
	testscrollVertical.c	
	testset.c	
	testtoggle.c	

Extra Credit Files:

commandLoopEC.c	mirrorHorizontal.c	mirrorVertical.c
rotateClockwise.c	rotateCounterClockwise.c	transpose.c

If there is anything in these procedures which needs clarifying, please feel free to ask any tutor, the instructor, or post on the Piazza Discussion Board.