Programming Assignment 1 (PA1) - drawTriangles

Milestone Due:  Wednesday, April 19 @ 11:59pm
Final Due:  Wednesday, April 26 @ 11:59 pm

Assignment Overview
The purpose of this assignment is to build your knowledge of the SPARC assembly language, especially branching and looping logic, calling assembly routines from within a C program, calling C functions from within assembly routines, passing parameters and returning values, using Unix command line arguments, and learning some useful Standard C Library routines.

You will be writing a program that takes 4 inputs from the command line and prints a triangle design to stdout. See man ascii for a map of the ASCII character set. This assignment will require appropriate error checking and reporting (as documented below).

Start early! Remember that you can and should use man in order to lookup information on specific C functions. For example, if you would like to know what type of parameters strtol() takes, or what strtol() does to errno, type man -s3c strtol. Also, take advantage of the tutors in the lab. They are there to help you learn more on your own and help you get through the course!

Grading
- README: 10 points - See README File section
- 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/CSE30StyleGuidlines.pdf
- Correctness: 65 points
  - Milestone (15 points) - To be distributed across the Milestone functions (see below)
  - Make sure you have all files tracked in Git.
- Extra Credit: 5 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 ieng9 via ssh (replace cs30xyz with YOUR cs30 account).

$ ssh cs30xyz@ieng9.ucsd.edu

Create and enter the pa1 working directory.

$ mkdir ~/pa1
$ cd ~/pa1
Copy the starter files from the public directory.

$ cp -r ~/../public/pa1StarterFiles/* ~/pa1/

<table>
<thead>
<tr>
<th>Starter files provided:</th>
</tr>
</thead>
<tbody>
<tr>
<td>pa1.h</td>
</tr>
</tbody>
</table>

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.

Sample Output
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/pa1test

If there is a discrepancy between the sample output in this document and the `pa1test` output, follow the `pa1test` output.

The output of your program MUST match exactly as it appears in the `pa1test` output. You need to pay attention to the order of error messages!

Important note for entering arguments on the command line:
When you enter alphanumeric and some non-alphanumeric ASCII characters on the command line, you can choose to enclose the character in single quotes ("a"), double quotes("a"), or no quotes at all.

Example:

```
./pal 6 "a" 'b' c
```

However, while this will work for all alphanumeric characters, this will not work for some non-alphanumeric characters! Some characters, such as >, #, <, and others, MUST be enclosed with either single or double quotes in order for the shell to read them properly.

Example:

```
./pal 6 "#" '<' '>'
```

In addition, some other characters, such as the newline character and the tab character, must be entered using a backslash to escape them. This also requires that you type in a $ before the string representing the escaped character in order for bash to correctly interpret it. This can be done in three different ways: using a character escape sequence such as '\n', using the ASCII code of the character in octal, or using the ASCII code of the character in hexadecimal. For more information on this see the man page for bash and search for "$'string'".

Example of using escape sequence: (newline char using escape char sequence)
```
./pal 12 "$\n" a b
```
Example of using octal code: (newline char in octal)
./pal 12 $'\12' a b

Example of using hexadecimal code: (newline char in hex)
./pal 12 $'\xA' a b

Some common escape characters:

<table>
<thead>
<tr>
<th>Oct</th>
<th>Dec</th>
<th>Hex</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0</td>
<td>00</td>
<td>NUL '\0'</td>
</tr>
<tr>
<td>007</td>
<td>7</td>
<td>07</td>
<td>BEL '\a' (bell)</td>
</tr>
<tr>
<td>010</td>
<td>8</td>
<td>08</td>
<td>BS '\b' (backspace)</td>
</tr>
<tr>
<td>011</td>
<td>9</td>
<td>09</td>
<td>HT '\t' (horizontal tab)</td>
</tr>
<tr>
<td>012</td>
<td>10</td>
<td>0A</td>
<td>LF '\n' (new line)</td>
</tr>
<tr>
<td>013</td>
<td>11</td>
<td>0B</td>
<td>VT '\v' (vertical tab)</td>
</tr>
<tr>
<td>014</td>
<td>12</td>
<td>0C</td>
<td>FF '\f' (form feed)</td>
</tr>
<tr>
<td>015</td>
<td>13</td>
<td>0D</td>
<td>CR '\r' (carriage ret)</td>
</tr>
</tbody>
</table>

Below are some brief example outputs of this program. Make sure you experiment with the public executable to further understand the program behavior. Bolded text is what you type in the terminal.

1. Command-line Parsing Errors

1.1. Too many arguments (extra operand).
[cs30xyz@ieng9]:pa1$ ./pal 5 a a a a

Usage: ./pal triWidth triChar1 triChar2 borderChar

triWidth -- an even integer representing the width of each of the triangles, border width is derived from this, but not included in it.
Must be within the limits of [2-120]

triChar1 -- the character used to draw the 1st triangle, must not be the same as the triangle 2 or border characters.
Must be within the limits of ASCII [33-126]

triChar2 -- the character used to draw the 2nd triangle, must not be the same as the triangle 1 or border characters.
Must be within the limits of ASCII [33-126]

borderChar -- the character used to draw the border.
Must be within the limits of ASCII [33-126]

[cs30xyz@ieng9]:pa1$

1.2. No arguments.
[cs30xyz@ieng9]:pa1$ ./pal

Usage: ./pal triWidth triChar1 triChar2 borderChar

triWidth -- an even integer representing the width of each of the triangles, border width is derived from this, but not included in it.
Must be within the limits of [2-120]

triChar1 -- the character used to draw the 1st triangle, must not be the same as the triangle 2 or border characters.
Must be within the limits of ASCII [33-126]

triChar2 -- the character used to draw the 2nd triangle, must not be the same as the triangle 1 or border characters.
Must be within the limits of ASCII [33-126]
borderChar -- the character used to draw the border.
    Must be within the limits of ASCII [33-126]

1.3 Invalid triWidth strtol conversion (no errno)
[cs30xyz@ieng9]:pa1$ ./pa1 5a a a a
Triangle width 5a is not a valid long.

Usage: ./pa1 triWidth triChar1 triChar2 borderChar
    triWidth -- an even integer representing the width of each of the
                triangles, border width is derived from this, but not
                included in it.
                Must be within the limits of [2-120]
    triChar1 -- the character used to draw the 1st triangle, must not be
                the same as the triangle 2 or border characters.
                Must be within the limits of ASCII [33-126]
    triChar2 -- the character used to draw the 2nd triangle, must not be
                the same as the triangle 1 or border characters.
                Must be within the limits of ASCII [33-126]
    borderChar -- the character used to draw the border.
                Must be within the limits of ASCII [33-126]

1.4 Errno set in strtol conversion
[cs30xyz@ieng9]:pa1$ ./pa1 999999999999999999999 a a a
Value 999999999999999999999 cannot be converted to long in base 10: Result too large

Usage: ./pa1 triWidth triChar1 triChar2 borderChar
    triWidth -- an even integer representing the width of each of the
                triangles, border width is derived from this, but not
                included in it.
                Must be within the limits of [2-120]
    triChar1 -- the character used to draw the 1st triangle, must not be
                the same as the triangle 2 or border characters.
                Must be within the limits of ASCII [33-126]
    triChar2 -- the character used to draw the 2nd triangle, must not be
                the same as the triangle 1 or border characters.
                Must be within the limits of ASCII [33-126]
    borderChar -- the character used to draw the border.
                Must be within the limits of ASCII [33-126]

2. Other Errors
2.1 Multiple errors: triWidth is odd, triChar1, triChar2, and borderChar are not valid single
    characters.
[cs30xyz@ieng9]:pa1$ ./pa1 5 35 aa bbb
Triangle width 5 is odd, but should be even.
Triangle 1 character 35 is not a valid single character.
Triangle 2 character aa is not a valid single character.
Border character bbb is not a valid single character.

Usage: ./pa1 triWidth triChar1 triChar2 borderChar
    triWidth -- an even integer representing the width of each of the
triangles, border width is derived from this, but not
included in it.
Must be within the limits of [2-120]
triChar1 -- the character used to draw the 1st triangle, must not be
the same as the triangle 2 or border characters.
Must be within the limits of ASCII [33-126]
triChar2 -- the character used to draw the 2nd triangle, must not be
the same as the triangle 1 or border characters.
Must be within the limits of ASCII [33-126]
borderChar -- the character used to draw the border.
Must be within the limits of ASCII [33-126]

2.2. triChar1 and borderChar are the same character.

The triangle characters must be different from each other and the border character.

Usage: ./pa1 triWidth triChar1 triChar2 borderChar

triWidth -- an even integer representing the width of each of the
triangles, border width is derived from this, but not
included in it.
Must be within the limits of [2-120]
triChar1 -- the character used to draw the 1st triangle, must not be
the same as the triangle 2 or border characters.
Must be within the limits of ASCII [33-126]
triChar2 -- the character used to draw the 2nd triangle, must not be
the same as the triangle 1 or border characters.
Must be within the limits of ASCII [33-126]
borderChar -- the character used to draw the border.
Must be within the limits of ASCII [33-126]

2.3 triChar1 and triChar2 are the same character.

The triangle characters must be different from each other and the border character.

Usage: ./pa1 triWidth triChar1 triChar2 borderChar

triWidth -- an even integer representing the width of each of the
triangles, border width is derived from this, but not
included in it.
Must be within the limits of [2-120]
triChar1 -- the character used to draw the 1st triangle, must not be
the same as the triangle 2 or border characters.
Must be within the limits of ASCII [33-126]
triChar2 -- the character used to draw the 2nd triangle, must not be
the same as the triangle 1 or border characters.
Must be within the limits of ASCII [33-126]
borderChar -- the character used to draw the border.
Must be within the limits of ASCII [33-126]
2.4 Character not within limits
[cs30xyz@ieng9]:pa1$./pa1 6 ' ' b c
Triangle 1 character 32 not within limits [33-126].

Usage: ./pa1 triWidth triChar1 triChar2 borderChar
   triWidth -- an even integer representing the width of each of the
triangles, border width is derived from this, but not
   included in it.
   Must be within the limits of [2-120]
   triChar1 -- the character used to draw the 1st triangle, must not be
   the same as the triangle 2 or border characters.
   Must be within the limits of ASCII [33-126]
   triChar2 -- the character used to draw the 2nd triangle, must not be
   the same as the triangle 1 or border characters.
   Must be within the limits of ASCII [33-126]
   borderChar -- the character used to draw the border.
   Must be within the limits of ASCII [33-126]
[cs30xyz@ieng9]:pa1$

2.5 Escaped character not within limits (newline character).
[cs30xyz@ieng9]:pa1$./pa1 6 '$\n' b c
Triangle 1 character 10 not within limits [33-126].

Usage: ./pal triWidth triChar1 triChar2 borderChar
   triWidth -- an even integer representing the width of each of the
triangles, border width is derived from this, but not
   included in it.
   Must be within the limits of [2-120]
   triChar1 -- the character used to draw the 1st triangle, must not be
   the same as the triangle 2 or border characters.
   Must be within the limits of ASCII [33-126]
   triChar2 -- the character used to draw the 2nd triangle, must not be
   the same as the triangle 1 or border characters.
   Must be within the limits of ASCII [33-126]
   borderChar -- the character used to draw the border.
   Must be within the limits of ASCII [33-126]
[cs30xyz@ieng9]:pa1$

2.6 Characters entered using octal ASCII code not within limits (newline character).
[cs30xyz@ieng9]:pa1$./pa1 6 '$\12' b c
Triangle 1 character 10 not within limits [33-126].

Usage: ./pal triWidth triChar1 triChar2 borderChar
   triWidth -- an even integer representing the width of each of the
triangles, border width is derived from this, but not
   included in it.
   Must be within the limits of [2-120]
   triChar1 -- the character used to draw the 1st triangle, must not be
   the same as the triangle 2 or border characters.
   Must be within the limits of ASCII [33-126]
   triChar2 -- the character used to draw the 2nd triangle, must not be
   the same as the triangle 1 or border characters.
   Must be within the limits of ASCII [33-126]
   borderChar -- the character used to draw the border.
   Must be within the limits of ASCII [33-126]
2.7. Characters entered using hexadecimal ASCII code not within limits (newline character).

Usage: ./pal triWidth triChar1 triChar2 borderChar
    triWidth -- an even integer representing the width of each of the
                triangles, border width is derived from this, but not
                included in it.
    Must be within the limits of [2-120]
    triChar1 -- the character used to draw the 1st triangle, must not be
               the same as the triangle 2 or border characters.
               Must be within the limits of ASCII [33-126]
    triChar2 -- the character used to draw the 2nd triangle, must not be
               the same as the triangle 1 or border characters.
               Must be within the limits of ASCII [33-126]
    borderChar -- the character used to draw the border.
               Must be within the limits of ASCII [33-126]

3. Valid Output
3.1. triWidth = 10, triChar1 = ‘a’, triChar2 = ‘w’, borderChar = ‘-’

3.2. triWidth = 8, triChar1 = ‘a’, triChar2 = ‘D’, borderChar = ‘!’
3.3. \(\text{triWidth} = 2, \text{triChar1} = ' = ', \text{triChar2} = ' + ', \text{borderChar} = ' !'\)

```
./pel 2 = + !
```

```
!!!!!!!
! = ++!
!== + !
!!!!!!!
```

3.4. Using octal and hex ascii codes for legal characters.

\(\text{triWidth} = 8, \text{triChar1} = '%', \text{triChar2} = '\"', \text{borderChar} = '#'\) (same as \./pel 8 ' % ' '' '' '#')

```
./pel 8 \$'45' \$'x22' '#'
```

```
#   % " " " " #
# % " " " " #
# % " " " " #
# % " " " " #
# % " " " " #
# % " " " " #
# % " " " " #
# % " " " " #
```

```
[cs30xyz@ieng9]:pal$
```

**Detailed Overview**

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

**C routines (to be written):**

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

**Assembly routines:**

```assembly
long withinLimits( long num, long min, long max );
long isDivisible( long num, long divisor );
long numOfDigits( long num, long base );
void printChar( char c );
void drawTriangles( long triWidth, char triChar1, char triChar2, char borderChar );
```

**For the Milestone, you will need to complete:**

```plaintext
withinLimits.s  isDivisible.s  numOfDigits.s
```
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.

This program takes 4 command line arguments:

```
$ ./pa1 triWidth triChar1 triChar2 borderChar
```

Explanation of Command Line Arguments:
- **triWidth** - an even integer representing the width of each of the triangles. Border width is derived from this, but not included in it. For example, a triangle width of 102 would have a border width of 3, since there are 3 digits in the number 102.
- **triChar1** - the character used to draw the 1st (left) triangle (red in the diagram below).
- **triChar2** - the character used to draw the 2nd (right) triangle (blue in the diagram below).
- **borderChar** - the character used to draw the border (green in the diagram below).

Drawing the triangles consists of the following steps:
1. Parse command line arguments in `main()`, where `withinLimits()` and `isDivisible()` will help with the error checking.
2. Draw the triangle pattern using `drawTriangles()`, where `numOfDigits()` will be used to calculate the border width, and `printChar()` will be used to print the individual characters.
C Functions to be Written
Listed below are the modules to be written in C.

main.c

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

The main function will drive the rest of the program. It will first perform input checking by parsing the command-line arguments and checking for errors. If all inputs are valid, it will call `drawTriangles()`. If any of the input checks fail, it will print the corresponding errors, the usage and then exit. Keep in mind that all the error strings have format specifiers, so be sure to add the appropriate arguments when printing error messages. Make sure you use your `withinLimits()` function when checking the limits of the command line arguments.

First, check that the user entered the correct number of command line arguments (use the `EXPECTED_ARGS` constant from `pal.h`). If they didn’t, print the `USAGE` and return `EXIT_FAILURE` right away.

Now we can parse the command line arguments:

1. **triWidth**: set the global variable `errno` to 0 (see `man -s2 intro`), then use `strtol()` to convert the triangle width to a long (see `man -s3c strtol`). Check for the following errors in the order they appear below.

<table>
<thead>
<tr>
<th>Error</th>
<th>How to Handle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error converting to long (<code>errno</code> was set by <code>strtol()</code>)</td>
<td>Use <code>snprintf()</code> to build the error string using <code>TOO_BIG_NUM</code> (make sure the string is null-terminated). Call <code>perror()</code>, passing this string as a parameter. Skip to step 2.</td>
</tr>
<tr>
<td>triWidth contains non-numerical characters (check <code>endptr</code>)</td>
<td>Print the <code>INVALID_LONG</code> error message. Skip to step 2.</td>
</tr>
<tr>
<td>triWidth is not within limits (not within <code>[MIN_TRI_WIDTH - MAX_TRI_WIDTH]</code>)</td>
<td>Print the <code>OUT_OF_LIMITS</code> error message (make sure to pass in <code>TRI_WIDTH</code> as part of the string). Skip to step 2.</td>
</tr>
<tr>
<td>triWidth is not even (use <code>isDivisible()</code>)</td>
<td>Print the <code>NUM_ODD</code> error message. Skip to step 2.</td>
</tr>
</tbody>
</table>

2. **triChar1**: extract the first character from the `triChar1` argument. Check for the following errors in the order they appear below.

<table>
<thead>
<tr>
<th>Error</th>
<th>How to Handle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not a valid single character (hint: <code>man -s3c strlen</code>)</td>
<td>Print the <code>INVALID_CHAR</code> error message (make sure to pass in <code>TRI1_CHAR</code> as part of the string). Skip to step 3.</td>
</tr>
<tr>
<td>char not within limits (not within <code>[LOWER_CHAR_LIMIT - UPPER_CHAR_LIMIT]</code>)</td>
<td>Print the <code>OUT_OF_LIMITS</code> error message (make sure to pass in <code>TRI1_CHAR</code> as part of the string). Skip to step 3.</td>
</tr>
</tbody>
</table>

3. **triChar2**: extract the first character from the `triChar2` argument. Check for the following errors in the order they appear below.
<table>
<thead>
<tr>
<th>Error</th>
<th>How to Handle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not a valid single character (hint: man -s3c strlen)</td>
<td>Print the INVALID_CHAR error message (make sure to pass in TRI2_CHAR as part of the string). Skip to step 4.</td>
</tr>
<tr>
<td>char not within limits (not within [LOWER_CHAR_LIMIT - UPPER_CHAR_LIMIT])</td>
<td>Print the OUT_OF_LIMITS error message (make sure to pass in TRI2_CHAR as part of the string). Skip to step 4.</td>
</tr>
</tbody>
</table>

4. **borderChar:** extract the first character from the borderChar argument. Check for the following errors in the order they appear below.

<table>
<thead>
<tr>
<th>Error</th>
<th>How to Handle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not a valid single character (hint: man -s3c strlen)</td>
<td>Print the INVALID_CHAR error message (make sure to pass in BORDER_CHAR as part of the string). Skip to step 5.</td>
</tr>
<tr>
<td>char not within limits (not within [LOWER_CHAR_LIMIT - UPPER_CHAR_LIMIT])</td>
<td>Print the OUT_OF_LIMITS error message (make sure to pass in BORDER_CHAR as part of the string). Skip to step 5.</td>
</tr>
</tbody>
</table>

5. If any errors were encountered, print the **USAGE** and return **EXIT_FAILURE**. Otherwise, make sure triChar1 and triChar2 are different from each other and the borderChar. If they are not, print the **EXCLUSIVE** error message, the **USAGE**, and return **EXIT_FAILURE**.

6. If no errors were encountered, draw the triangle pattern simply by calling the `drawTriangles()` function, passing in the appropriate parameters.

**Return Value:** If errors were encountered, return **EXIT_FAILURE**. Otherwise, return **EXIT_SUCCESS**.

---

**Assembly Functions to be Written**

Listed below are the modules to be written in Assembly.

### withinLimits.s

```assembly
long withinLimits( long num, long min, long max );
```

This assembly module will check if the `num` is within `min` and `max`, inclusive on both ends. This means if `num` is equal to `min`, it is within the limits, and if `num` is equal to `max`, it is also within the limits.

For example:

<table>
<thead>
<tr>
<th>num</th>
<th>min</th>
<th>max</th>
<th>Within limits?</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>-4</td>
<td>error</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>5</td>
<td>yes</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>5</td>
<td>no</td>
</tr>
</tbody>
</table>

**Error conditions:**

- `min` is greater than `max` → return -1

**Return Value:** If any errors occur, return -1. Otherwise, return 1 to represent true, 0 to represent false.
isDivisible.s
long isDivisible( long num, long divisor );

This function determines whether or not the num is evenly divisible by divisor. You need to check if divisor is within the limits of 2 and LONG_MAX using your withinLimits() function.

Note: Since this function is written in assembly, you do not have access to the value of LONG_MAX. Instead, declare an assembler constant which is equal to the value of LONG_MAX (0x7FFFFFFF).

For example:

<table>
<thead>
<tr>
<th>num</th>
<th>divisor</th>
<th>Evenly divisible?</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>2</td>
<td>yes</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>no</td>
</tr>
<tr>
<td>10</td>
<td>-5</td>
<td>error</td>
</tr>
</tbody>
</table>

Error conditions:
- divisor is not within [2 - LONG_MAX] → return -1

Return Value: If any errors occur, return -1. Otherwise, return 1 to represent true, 0 to represent false.

numOfDigits.s
long numOfDigits( long num, long base );

This function counts the number of digits in num in the given base. You need to check if base is within the limits of 2 and 36 (inclusive) using your withinLimits() function.

If num is negative, do NOT convert it to positive before counting the number of digits. The reason behind this is that if num is the largest negative number, this number cannot be represented as a positive number. Therefore, in the case of negative numbers, we want to just leave them negative.

Note for unit testing: To type hex values, prefix the number with 0x. To type octal values prefix them with 0. C does not allow you to type values in binary, so you will need to convert binary to either decimal, octal, or hex.

For example:

<table>
<thead>
<tr>
<th>num</th>
<th>base</th>
<th>Number of digits</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>any</td>
<td>1</td>
</tr>
<tr>
<td>-2693</td>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>0x7A2</td>
<td>16</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>4 ( (8_{10} \equiv 1000_2) )</td>
</tr>
</tbody>
</table>

Error conditions:
- base is not within [2-36] inclusive → return -1

Return Value: If any errors occur, return -1. Otherwise, return the number of digits in num of the given base.
**printChar.s**

```c
void printChar( char c );
```

This assembly module prints the character argument to stdout. This is very similar to the assembly module `printWelcome.s` given as part of PA0 -- use `printf()`. The difference is that `printChar()` just prints a single character (so think about how that might affect the format string).

**Return Value:** None.

---

**drawTriangles.s**

```c
void drawTriangles( long triWidth, char triChar1, char triChar2, char borderChar );
```

This assembly module will perform the actual outputting of individual characters (via calls to `printChar()`) such that the triangle pattern is displayed with the user-supplied values.

Here is the equivalent C version:

[http://ieng9.ucsd.edu/~cs30x/pa1/drawTriangles_template](http://ieng9.ucsd.edu/~cs30x/pa1/drawTriangles_template)

All of the assembly constructs you will be using will have been covered in lecture and can be referenced in Chapter 2 of the textbook. You are not limited to using the above algorithm, but part of the purpose of this programming assignment is to learn how to write looping/conditional constructs (branches), use the simple `.mul`, `.div`, `.rem` subroutines with parameter passing and return values, and perform simple arithmetic instructions (inc, add/sub) in assembly.

We would encourage you to use the linked algorithm for these reasons, however we do not want to suppress creative thinking - alternative solutions are welcome. You must use the "preferred" style of coding loops, backwards branching logic, as detailed in class: set up an opposite logic branch to jump over the loop body and a positive logic branch to jump backwards to the loop body. Points will be taken off for not using backwards branching logic.

**Return Value:** None

---

**Unit Testing**

You are provided with only one basic unit test file for the Milestone function, `withinLimits()`. 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 are not being collected for the Milestone and will only be collected for the final turnin (however, they should already be written by the time you turn in the Milestone because you should be using them to test your Milestone functions).
Unit tests files you need to complete:
- testwithinLimits.c
- testisDivisible.c
- testnumOfDigits.c

To compile:
$ make testwithinLimits

To run:
$ ./testwithinLimits

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

**Pro tip for unit testing:**
When you are writing your unit test files, if you get a compiler error saying “implicit function declaration,” this most likely means that the function prototype (for the function you are testing) is commented out in `pa1.h` (because it is only called from assembly routines in the main program). If this is the case, simply add the function prototype to the top of the unit test file and all the (compiler) angels will sing. Happy testing!

**README File**
Your README file for this and all assignments should contain:
- High level description of what your program does.
- How to compile it (be more specific than: just typing “make”—i.e., what directory should you be in?, where should the source files be?, etc.).
- How to run it (give an example).
- An example of normal output and where that normal output goes (stdout or a file or ???).
- An example of abnormal/error output and where that error output goes (stderr usually).
- How you tested your program (what test values you used to test normal and error states) and showing your tests covered all parts of your code (test coverage). (Be more specific than diff’ing your output with the solution output—i.e., what are some specific test cases you tried?, what different types of cases did you test?, etc.)
- Anything else that you would want/need to communicate with someone who has not read the assignment write-up but may want to compile and run your program.
- Answers to questions (if there are any).

**Questions to Answer in the README**
Start gdb with your pa1 executable, then set a breakpoint at strtol:
```plaintext
break strtol
```
Run the program (in gdb) with the following command line args:
```plaintext
run 9InchNails a b c
```
You should be at the entry point of the Std C Lib routine strtol called from main.

1. How do you print the value of the string that is the 1st arg in strtol? (The value should be "9InchNails")
2. How do you print the decimal value of the base that is the 3rd arg in strtol? (The value should be 10)
3. How do you print the hex value of `&endptr` that is the 2nd arg in strtol? (The value should be something like 0xffbe---- (a high stack address - will vary))

Go to the next high level source instruction in main. This should be the next C instruction in main after the function call to strtol. Type `next`
4. How do you print the value returned by strtol? (The value should be 9) Show two ways:
   a. Using the name of the local variable you use to hold the return value
   b. Displaying the value in the register used to return the value

5. How do you print the character endptr is pointing to? (Should be the character 'I')

6. How do you print the entire null-terminated string endptr is pointing to? (Should be “InchNails”)

7. How do you print the decimal value of the global variable errno at this point? (The value should be 0)

Continue the execution of your pa1 in gdb. Type continue

You should see the error message Triangle width 9InchNails is not a valid long. displayed by your program along with the usage statement.

Run the program again (in gdb) with the following command line args:
   run 99999999999999 a b c

You should be at the entry point of the Std C Lib routine strtol called from main. Go to the next source instruction in main. Type next

It should be the source-level instruction after the call to strtol passing in 99999999999999 to convert to an int. Print the decimal value of errno at this point. The value of errno should be 34 now which is the value of ERANGE. See the man page for errno and section 2 intro. You can continue or quit.

The following questions pertain to Git:

8. What is the Git command to show the current state of your working tree?

9. What is the Git command to discard any changes made to a file since its last commit?

10. What is the Git command to display the differences between the local version of a file and the version last committed?

Academic Integrity:

11. What was your process for completing this assignment with integrity?

Extra Credit

There are 5 points total for extra credit on this assignment.

- Early turnin: [2 Points] 48 hours before regular due date and time
  [1 Point] 24 hours before regular due date and time
  (One or the other, not both. Final turn-in only, does not apply to milestone)
- [3 Points Total, 0.5 for each nop] Eliminating nops in the sample assembly file.
  (1 nop cannot be removed - see the comment in pa1NOPS.s)

Getting Started

Copy over pa1EC.c and pa1NOPS.s from the public directory:

[cs30xyz@ieng9]:pa1$ cp ~/.public/pa1EC.c ~/pa1
[cs30xyz@ieng9]:pa1$ cp ~/.public/pa1NOPS.s ~/pa1
Overview
You will be removing up to six nops in pa1Nops.s to perform assembly optimization—do NOT modify any other files. Note that there are seven nops total in the file, and there is a comment specifying which one should not be removed. Be sure to move the instruction you intend to fill the delay slot with into where the nop instruction was, do NOT just remove the nop.

In order to do this extra credit, you need a working version of isDivisible.s and withinLimits.s. A reference executable will not be provided, so you should use the non-optimized version of pa1NOPS.s to test against your optimized version.

Compiling
Once you have a working version, you can compile the extra credit program:

```
[cs30xyz@ieng9]:pa1$ gcc -o pa1EC pa1EC.c pa1NOPS.s isDivisible.s withinLimits.s
```

Example output
This program takes in a single positive integer as input, and prints a secret number based on the input. Bolded text is what you type in the terminal.

```
[cs30xyz@ieng9]:pa1$ ./pa1EC
Usage: ./pa1EC num
       num - integer to calculate secret number for
[cs30xyz@ieng9]:pa1$ ./pa1EC 0
The super secret number for 0 is 0
[cs30xyz@ieng9]:pa1$ ./pa1EC 1
   1: 0
The super secret number for 1 is 0
[cs30xyz@ieng9]:pa1$ ./pa1EC 2
   1: 0
   2: 0
The super secret number for 2 is 0
[cs30xyz@ieng9]:pa1$ ./pa1EC 3
   1: 0
   2: 0
   3: 2
The super secret number for 3 is 2
[cs30xyz@ieng9]:pa1$ ./pa1EC 4
   1: 0
   2: 0
   3: 2
   4: 3
The super secret number for 4 is 3
[cs30xyz@ieng9]:pa1$ ./pa1EC 5
   1: 0
   2: 0
   3: 2
   4: 3
   5: 7
The super secret number for 5 is 7
[cs30xyz@ieng9]:pa1$ ./pa1EC 6
   1: 0
```
The super secret number for 6 is 17

[cs30xyz@ieng9]:pa1$ ./pa1EC 17
  1: 0
  2: 0
  3: 2
  4: 3
  5: 7
  6: 17
  7: 23
  8: 80
  9: 88
 10: 396
 11: 406
 12: 2233
 13: 2245
 14: 14592
 15: 14606
 16: 109545
 17: 109561
The super secret number for 17 is 109561

Milestone Turn-in Instructions

Milestone Turn-in - due Wednesday night, April 19 @ 11:59 pm [15 points of Correctness Section]
Before final and complete turnin of your assignment, you are required to turnin several modules for the Milestone check. Each module must pass all of our unit tests in order to receive full credit.

Files required for the Milestone:

| withinLimits.s | isDivisible.s | numOfDigits.s |

A working Makefile with all the appropriate targets and any required header files must be turned in as well. All Makefile test cases for the milestone functions must compile successfully via the commands make test***. You do not need to turn in your README with the milestone.

In order for your files to be graded for the Milestone Check, you must use the milestone specific turnin script.

$ cd ~/pa1
$ cse30_palmilestone_turnin

To verify your turn-in:

$ cse30verify palmilestone

Final Turn-in Instructions

Final Turn-in - due Wednesday night, April 26 @ 11:59 pm
Once you have checked your output, compiled, executed your code, finished your README file (see above), and double-checked your style, you are ready to turn it in. Use the following names *exactly* otherwise our Makefile will not find your files.
Files required for the Final Turn-in:

drawTriangles.s  main.c  pal.h
isDivisible.s    testisDivisible.c  test.h
numOfDigits.s    testnumOfDigits.c  Makefile
printChar.s      testwithinLimits.c  README
withinLimits.s

Extra Credit Files:
pa1EC.c  pa1NOPS.s

How to Turn in an Assignment
Before turning in, run make clean and then make to double check for any compiler errors/warnings. Then use
the following turnin script to submit your full assignment before the due date as follows:

$ cse30turnin pa1

To verify your turn-in:

$ cse30verify pa1

Up until the due date, you can re-submit your assignment via the scripts above. Note, if you turned in the
assignment early for extra credit and then turned it in again later (after the extra credit cutoff), you will no
longer receive early turn-in credit.

Failure to follow the procedures outlined here will result in your assignment not being collected properly and
will result in a loss of points. Late assignments WILL NOT be accepted.

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.