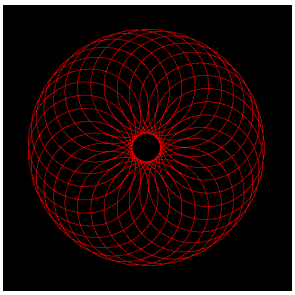
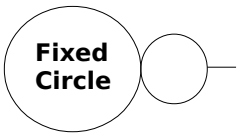
CS 218
Homework, Asst. #10

Purpose: Become more familiar with data representation, program control instructions, procedure handling, stacks, and operating system interaction.
Due: Tuesday (10/14)
Points: 175

Assignment:



Write a simple assembly language program to plot a spirograph[1] drawing. Imagine the movement of a small circle that rolls on the outside of a rigid circle. Imagine now that the small circle has an arm, rigidly atached, with a plotting pen fixed at some point. That is a epicycloid, commonly called spirographs.



The program should read radius 1 (fixed circle), radius 2 (moving circle), offset position (rigid arm length), and color from the command line. For example:

```
ed-vm% ./spiro -r1 6A -r2 8 -op 3A -cl r
```

The required format for the date options is: "-r1 <hex number>", "-r2 <hex number>", and "-op <hex number>", and "-cl <color letter> with no spaces in each specifier. The program must be able to handle one or more spaces before the between specifiers. The program must verify the format, read the arguments, and ensure the arguments are valid. The program must and ensure that the ***r1***, ***r2***, and ***op*** values are between 1 and FA_{16}. If there are any input errors, the program should display an appropriate error message and terminate. Refer to the sample executions for examples.

The provided main program calls the following routines:

- Procedure ***getRadii()*** to read the command line arguments (***r1***, ***r2***, ***op***, and ***cl***). The procedure should read each argument, convert ASCII/Hex to integer, and verify the range (1 and FA_{16}), and verify the color ("r" red, "g" green, "b" blue, or "w" white), lower case only. If there are any errors, display error message and terminate.

- Procedure ***spirograph()*** to plot the following spirograph equations:

```
for (t=0.0; t<3600.0; t+=0.1) {
        x = ( (r1+r2) * cos(t) ) + ( op * cos((r1+r2) * (t/r2)) );
        y = ( (r1+r2) * sin(t) ) + ( op * sin((r1+r2) * (t/r2)) ); }
```

  The loop will iterate 36,000 times.

All procedures ***must*** follow the standard calling convention as discussed in class. The procedures for the command line arguments and drawing functions must be in a separate assembly source file from the provided main program. The provided main program should be linked with the procedures. Only the *procedures* file should be submitted. As such, the main file can not be altered in any way.

---

1 For more information, refer to: http://en.wikipedia.org/wiki/Spirograph

Submission:
When complete, submit only the *procedures* file:

- ***A copy of the procedures source file via the class web page (assignment submission link) by 11:59:59pm. Assignments received after the allotted time will not be accepted!***

Debugging -> Command Line Arguments
When debugging a program that uses command line arguments, the command line arguments muct be entered after the debugger has been started. The debugger is started normally (ddd <program>) and once the debugger comes up, the initial breakpoint can be set. Then, when you are ready to run the program, enter the command line arguments. This can be done either from the menu (Propgram -> Run) or on the GDB Console Window (at bottom) by typing `run <commandLineArguments>` at the (gdb) prompt.

Testing
A batch file to execute the program on a series of pre-defined inputs will be provided. *Note*, please follow the I/O examples. The test script executes the program on a series of error tests (with expected output). Refer to the examples for output formatting and error handling.

Example Executions (with errors):
Below are some sample executions showing the error handling.

```
ed-vm% ./spiro -r1 999 -r2 7 -op 3A -cl r
Error, radius 1 value must be between 1 and FA(16).

ed-vm%
ed-vm% ./spiro  -r9 5F  -r2 7  -op 3A  -cl r
Error, radius 1 specifier incorrect.

ed-vm%
ed-vm% ./spiro  -r1 3A  -r2 0  -op 3A -cl r
Error, radius 2 value must be between 1 and FA(16).

ed-vm%
ed-vm% ./spiro  -r1 5F -r22 7 -op 3A -cl r
Error, radius 2 specifier incorrect.

ed-vm%
ed-vm% ./spiro  -r1 5F -r2 7  -op 3A -cl x
Error, c (color) value must be b, g, r, or w.

ed-vm%
ed-vm% ./spiro   -r1 5F  -r2 7 -op 3A -c g
Error, color specifier incorrect.

ed-vm%
```

Open GL Plotting Functions:

In order to plot points with openGl, a series of calls is required. First, the draw color must be set, the point plot mode must be turned on. Then, the points can be plotted in a loop. Once all the points hve been plotted, the plot mode can be ended and the points dispalyed.

The following are the sequence of calls required:

```
glColor3ub(r,g,b);
glBegin(GL_POINTS);

// plot calculations loop
        glVertex2d(x,y);

glEnd();
glFlush();
```

The calls must be performed at assembly level with the appropriate argument transmission. For example, to set a draw color ir red, **`glColor3ub (255, 0, 0)`**, and set point plot mode, **`glBegin(GL_POINTS)`**, the code would be as follows:

```
mov     rdi, 255
mov     rsi, 0
mov     rdx, 0
call    glColor3ub

mov     rsi, GL_POINTS
call    glBegin
```

Assuming the variables x and y are delcared as quad words and set to valid floating points values, the call to **`glVertex2d(x,y)`** would be as follows:

```
movsd   xmm0, qword [x]
movsd   xmm1, qword [y]
call    glVertex2d
```

This call would be iterated in a plot loop (unless a single point is to be plotted).

The calls for glEnd() and glFlush() are as follows:

```
call    glEnd
call    glFlush
```

These function calls should not be included in the loop.

Note 1, the template declares some local variables a double-words. These probably should be changed to quad words as required for 64-bit IEEE floating point values (e.g., C++ double).

Note 2, the template does not include a extern for glVertex2d. It must be includes with all the other extern functions at the top of the template.

Example Execution:
Below is a example execution showing the displayed output.

```
ed-vm% ./spiro -r1 af -r2 2a -op 6A -cl b
```