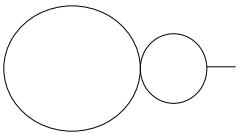
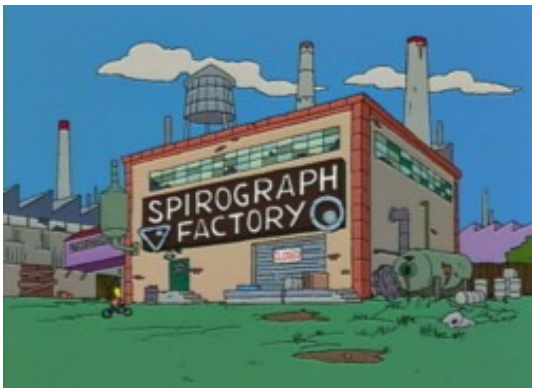
CS 218 – Assignment #10

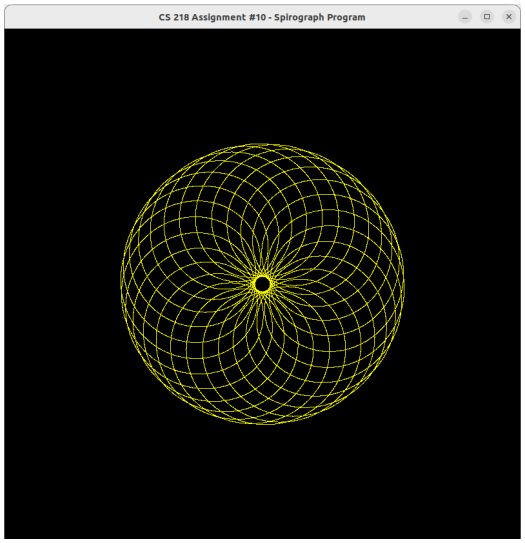
Purpose: Become more familiar with data representation, program control instructions, function handling, stacks, floating point operations, and operating system interaction.
Points: 150

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 attached, with a plotting pen fixed at some point. That is a epicycloid[2], often referred to as Spirographs. Use the provided main program which includes the appropriate OpenGL initializations. You will need to add your code (two functions) in the provided functions template.







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

```
./spirograph -r1 252 -r2 20 -op 252 -sp 2 -cl y
```

The required format for the date options is: "-r1 <senary number>", "-r2 <senary number>", "-op <senary number>", "-sp < senary number>", and "-cl <color letter> with no spaces in each specifier. The program must verify the format, read the arguments, and ensure the arguments are valid. The program must and ensure that the ***r1***, ***r2***, ***op***, and ***sp*** values are between the specified ranges (provided constants). If there are any command line errors, the program should display an appropriate error message and terminate. Refer to the sample executions for examples.

All functions ***must*** follow the standard calling convention as discussed in class. The functions for the command line arguments and drawing function 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 *functions* file should be submitted. As such, the provided main file can not be altered in any way.

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

Functions

The provided main program calls the following routines:

- Function ***getRadii()*** to read and check the command line arguments (***r1***, ***r2***, ***op***, ***sp***, and ***cl***). The function must read each argument, convert each ASCII/Tridecimal string to integer, and verify the ranges. The range for radius 1 (***r1***) is 0 and 1054_6 (inclusive). The range for radius 2 (***r2***) and offset position (***op***) is 1 to 1054_6 (inclusive). The range for speed (***sp***) is 1 to 244_6 (inclusive). The allowed colors values (***cl***) are "**r**" red, "**g**" green, "**b**" blue, "**p**" purple, "**y**" yellow, or "**w**" white), lower case only. If all arguments are correct, return the values (via reference) and return to the main with a value of TRUE. If there are any errors, display the appropriate error message and return to the main with a value of FALSE.
- Function ***drawSpiro()*** to plot the Spirograph functions noted below. The functions should be iterated and will generate a series of (***x***,***y***) values which must be plotted.

$$x = \left[(r_1+r_2) \times \cos(t) \right] + \left[offPos \times \cos\left((r_1+r_2) \times \frac{(t+s)}{r_2} \right) \right]$$

$$y = \left[(r_1+r_2) \times \sin(t) \right] + \left[offPos \times \sin\left((r_1+r_2) \times \frac{(t+s)}{r_2} \right) \right]$$

The ***t*** value should range between 0.0 and 360.0 and be incremented by ***tStep*** (predefined) each iteration. As such, it will require 360.0 / ***tStep*** iterations. Before leaving the function, the ***s*** value should be incremented by ***sStep*** which must be set as follows;

$$sStep = \frac{speed}{scale}$$

The function is called repeatedly which generates the animation (based on the changing ***s*** value between successive calls). The template already includes some of the applicable openGL initialization calls (which must not be removed).

OpenGL Installation

For this assignment, we will be using the openGL (graphics library) to provide some basic windowing capabilities. As such, the OpenGL development libraries must be installed. This can be done via the command line with the following commands.

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install binutils-gold
sudo apt-get install libgl1-mesa-dev
sudo apt-get install freeglut3 freeglut3-dev
```

It will take a few minutes to install each. You must be connected to the Internet during the installation. After the installation, a re-install of Virtual Box Guest Additions may be required.

RGB Color

RGB stands for Red, Green, Blue. It is a color model used in digital imaging and computer graphics to represent colors. In the RGB model, colors are created by mixing different intensities of red, green, and blue light. Each color is represented by a value between 0 and 255, where 0 represents the absence of that color and 255 represents the maximum amount of that color. By combining different values of red, green, and blue, it is possible to create a wide range of colors. For example, red is represented by [255,0,0], green is represented by [0,255,0], yellow is represented by [255,255,0], purple is represented by [255,0,255], blue is represented by [0,0,255], and white is represented by [255,255,255].

OpenGL Calls

The basic OpenGL library calls are included in the provided main and functions template. In order to set the draw color and plot the point, the following OpenGL functions will be required.

```
call  glColor3ub(red, green, blue);          // set color
call  glVertex2f(x, y);                      // plot point (float)
```

The ***red***, ***blue***, ***green*** variables are unsigned bytes. The ***x*** and ***y*** variables are double floating point values. These calls follow the standard calling convention.

Assemble and Linking Instructions

You will be provided a main function (**`spirograph.cpp`**) that calls the functions. Your functions should be in a separate file (**`a10procs.asm`**). The files will be assembled individually and linked together.

When assembling, and linking the files for assignment #10, use the provided **makefile** to assemble, and link. *Note*, **only** the functions file, **`a10procs.asm`**, will be submitted. The submitted functions file will be assembled and linked with the provided main. As such, do not alter the provided main.

Debugging -> Command Line Arguments

When debugging a program that uses command line arguments, the command line arguments must 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 (bottom window).

openGL Warning (possible)

Based on the specific hardware configuration and/or virtual box configuration, the following warning message may be displayed.

```
OpenGL Warning: Failed to connect to host. Make sure 3D acceleration is enabled
for this VM.
```

This warning message can be ignored. *Note*, some hardware configurations using virtual box may not be able to use openGL. An *openGLtest* program is provided to verify correction functional of the openGL installation.

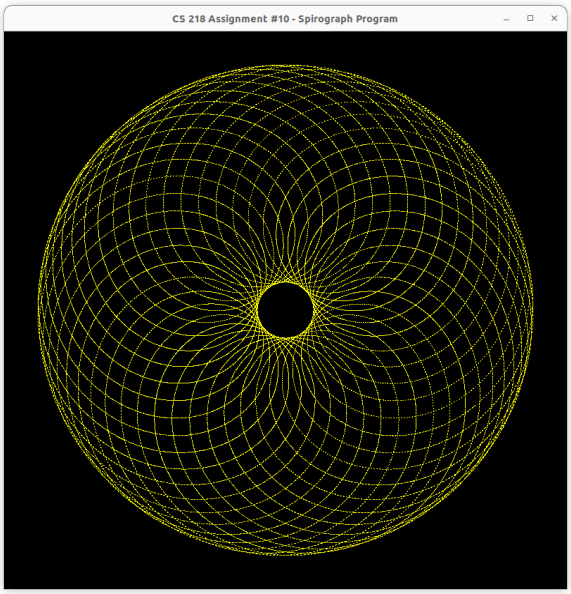
Example Executions (with errors):
Below are some example executions with errors in the command line. The program should provide an appropriate error message (as shown) and terminate. *Note*, the **ed@vm%** is the prompt.

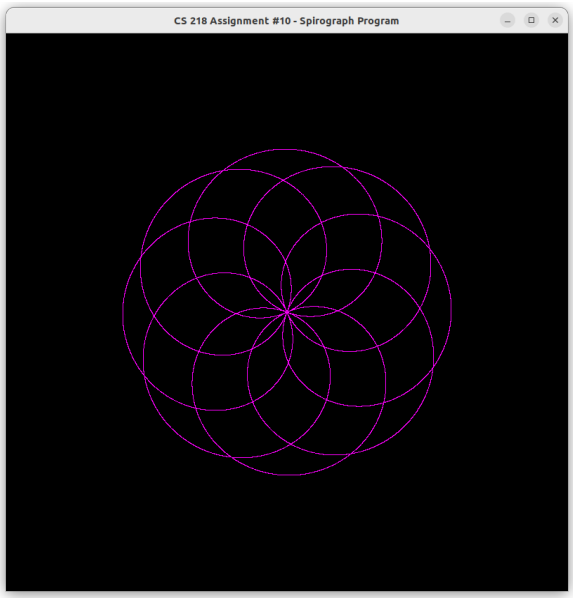
```
ed-vm% ./spirograph
Usage:  ./spirograph -r1 <senary number> -r2 <senary number>
 -op <senary number> -sp <senary number> -cl <b/g/r/y/p/w>
ed-vm%
ed-vm% ./spirograph -r1 252
Error, invalid or incomplete command line arguments.
ed-vm%
ed-vm% ./spirograph -r1 2522 -r2 21 -op 252 -sp 5 -cl p
Error, radius 1 value must be between 0 and 1054(6).
ed-vm%
ed-vm% ./spirograph -r1 252 -r2 252 -op 252d -sp 5 -cl p
Error, offset position value must be between 1 and 1054(6).
ed-vm%
ed-vm% ./spirograph -r1 252 -r2 20 -op 252 -sp 5 -cl blue
Error, color value must be b, g, r, p, or w.
ed-vm%
ed-vm% ./spirograph -r 252 -r2 20 -op 252 -sp 3 -cl b
Error, radius 1 specifier incorrect.
ed-vm%
```

Example Execution:
Below is an example execution showing the resulting image.

```
ed-vm% ./spirograph -r1 420 -r2 104 -op 420 -sp 2 -cl y      (left image)
ed-vm% ./spirograph -r1 252 -r2 21 -op 313 -sp 3 -cl p       (right image)
```

When functioning, the image will rotate based on the speed value (which will vary between different machines).





Submission:

- All source files must assemble and execute on Ubuntu with **`yasm`**.
- Submit source files
  - Submit a copy of the program source file via the on-line submission.
  - Only the functions file (**`a10procs.asm`**) will be submitted.
- Once you submit, the system will score the project and provide feedback.
  - If you do not get full score, you can (and should) correct and resubmit.
  - You can re-submit an unlimited number of times before the due date/time (at a maximum rate of 5 submissions per hour).
- Late submissions will be accepted for a period of 24 hours after the due date/time for any given assignment. Late submissions will be subject to a ~2% reduction in points per an hour late. If you submit 1 minute - 1 hour late -2%, 1-2 hours late -4%, … , 23-24 hours late -50%. This means after 24 hours late submissions will receive an automatic 0.

Program Header Block

All source files must include your name, section number, assignment, NSHE number, and program description. The required format is as follows:

```
;  Name: <your name>
;  NSHE ID: <your id>
;  Section: <section>
;  Assignment: <assignment number>
;  Description: <short description of program goes here>
```

Failure to include your name in this format will result in a loss of up to 3%.

Scoring Rubric

Scoring will include functionality, code quality, and documentation. Below is a summary of the scoring rubric for this assignment.

Criteria	Weight	Summary
Assemble	-	Failure to assemble will result in a score of 0.
Program Header	3%	Must include header block in the required format (see above).
General Comments	7%	Must include an appropriate level of program documentation.
Program Functionality (and on-time)	90%	Program must meet the functional requirements as outlined in the assignment (based on the following breakdown): • Command Line (45%) • Execution with output (5%) • Fully correct output (40%) Must be submitted on time for full score.

Note, the final output (40%) will be manually scored. As such, the initail codeGrade score will be incomplete and potentially misleading.