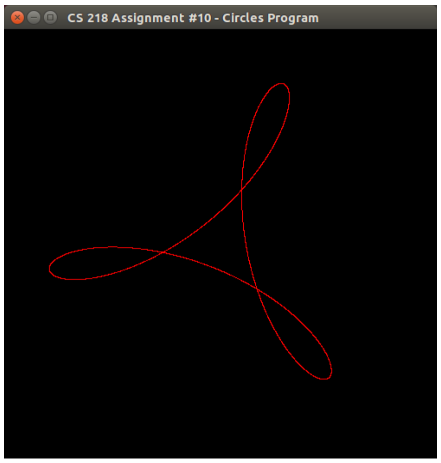
CS 218
Homework, Asst. #10

Purpose:	Become more familiar with data representation, program control instructions, function handling, standard calling convention, stacks, and operating system interaction.
Due:	Tuesday (3/12)
Points:	175

<u>Assignment:</u>

Write an assembly language program to repeatedly plot a circle. Each time the circle is redrawn it will be deformed which will appear in a cycling pattern. The program should read the draw speed, draw color, and background color from the command line (in that order). For example:

```
./circles -sp 35 -dc 1354104520 -bk 0
```



The required format for the options is: "**`-sp <dozenalNumber>`**", "**`-dc <dozenalNumber>`**", and "**`-bk <dozenalNumber>`**". The program must verify the format, read the arguments, and ensure the arguments are valid. The program must and ensure that the ***sp***, ***dc***, and ***bk*** values are between a minimum and maximum (defined constants). 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:

- Boolean value returning function ***getParams()*** to read the command line arguments (***sp***, ***dc***, and ***bk***). The function should read each argument, convert ASCII/Dozenal to integer, and verify the range. The draw and background colors may not be the same value. If all parameters are correct, they should be returned via the passed arguments and true returned. If there are any errors, display the applicable error message and return false.

- Void function ***drawCircles()*** to plot the following equations:

$$\begin{aligned} x &= (1.0 - s)\cos(t + \pi\, s) + s\cos(2t) \\ y &= (1.0 - s)\sin(t + \pi\, s) - s\sin(2t) \end{aligned} \quad \text{iterated} \quad \frac{2\pi}{tStep} \quad \text{times.}$$

  After the circle is plotted and drawn, the ***s*** value (originally ininitialized to 0.0) is incremented by (***sStep*** which is ***drawSpeed***/***scale***). The ***drawSpeed*** is the speed value read from the command line, converted to float, and then divided by the scale factor (10,000.0). When the ***s*** value becomes > 1.0, reset the ***s*** value to 0.0. In this manner, the ***s*** value is updated for the next call to the ***drawCircles()*** function.

All functions ***must*** follow the standard calling convention as discussed in class. The functions 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 functions. Only the *functions* file should be submitted. As such, the provided main file can not be altered in any way.

Submission:

When complete, submit only the *functions* file:

- ***A copy of the functions source file via the class web page (assignment submission link) by 11:55 PM. 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 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, you must enter the command line arguments. This can be done either from the menu (Propgram -> Run which will display a window for the arguments) or in the GDB Console Window (at bottom) by typing **`run <commandLineArguments>`** at the (gdb) prompt.

Testing

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

Example Executions (with errors):

Below are some sample executions showing the error handling.

```
ed-vm% ./circles
Usage: circles -sp <dozenalNumber> -dc <dozenalNumber> -bk <dozenalNumber>
ed-vm%
ed-vm% ./circles -sp fast -dc 5750x80 -bk 0
Error, speed value must be between 1 and 6E4(12).
ed-vm%
ed-vm% ./circles -sp 12 -dc 57q0x80 -bk 0
Error, draw color value must be between 0 and 5751053(12).
ed-vm%
ed-vm% ./circles -sp fast -dc 5750x80 -bk -3
Error, speed value must be between 1 and 6E4(12).
ed-vm%
ed-vm% ./circles -sp 9 -dc 57 50x80 -bk 0
Error, invalid or incomplete command line argument.
ed-vm%
ed-vm% ./circles -sp 1x -dc 5750x80 -bk 5750x80
Error, draw color and background color can not be the same.
ed-vm%
ed-vm% ./circles -spp 7 -dc 5750x80 -bk 0
Error, speed specifier incorrect.
ed-vm%
ed-vm% ./circles -sp 7 dc 5750x80 -bk 0
Error, draw color specifier incorrect.
ed-vm%
ed-vm% ./circles -sp 7 -dc 5750x80 -b 0
Error, background 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 have 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 ();
glutPostRedisplay();
```

The calls must be performed at assembly level with the appropriate argument transmission. For example, to set a draw color of 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     rdi, 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(xPnt,yPnt)`** would be as follows:

```
movsd   xmm0, qword [xPnt]
movsd   xmm1, qword [yPnt]
call    glVertex2d
```

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

The calls for **`glEnd()`**, **`glFlush()`**, and **`glutPostRedisplay()`** are as follows:

```
call    glEnd
call    glFlush
call    glutPostRedisplay
```

These function calls should not be included in the loop. They tell openGL to call the draw function again (which will re-draw the circle with sligtly dififerent parameters).

OpenGL Errors

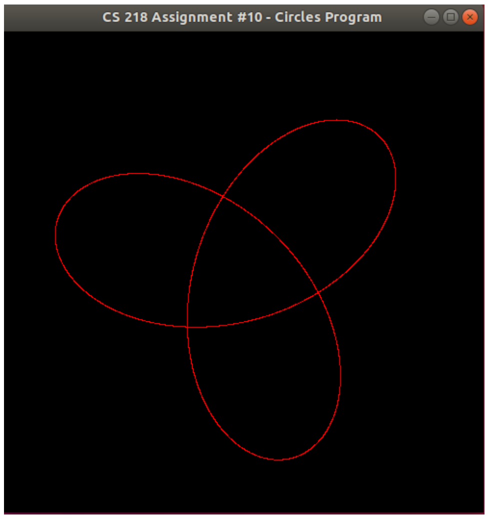
Note, some VM's may generate errors, similar to the below, which can be ignored:

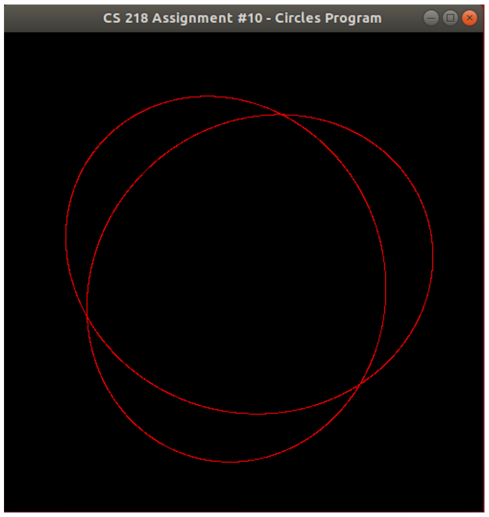
```
libGL error: pci id for fd 4: 80ee:beef, driver (null)
OpenGL Warning: Failed to connect to host. Make sure 3D acceleration is enabled for this VM.
libGL error: core dri or dri2 extension not found
libGL error: failed to load driver: vboxvideo
```

Example Execution:

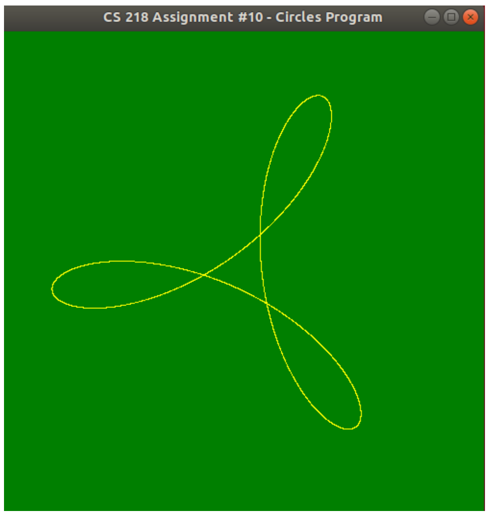
Below is a example execution showing the displayed output. The pattern will cycle and a static image(s) of the cycling pattern is shown for reference. The '**`ed-vm%`**' is the prompt on my machine.

```
ed-vm% ./circles -sp 7 -dc 571e140 -bk 0
```





```
ed-vm% ./circles -sp 7 -dc 5750x80 -bk 16994
```


CS 218 Assignment #10 - Circles Program