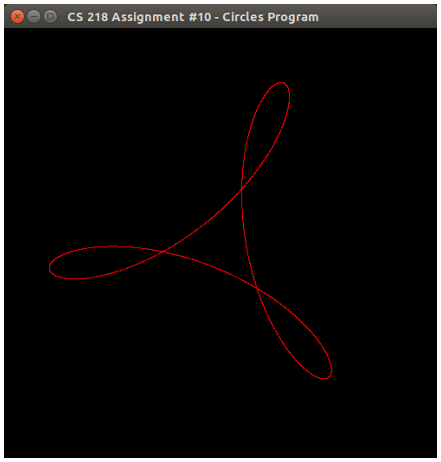
CS 218 – Assignment #10

Purpose: Become more familiar with data representation, program control instructions, procedure handling, stacks, and operating system interaction.
Points: 175

Assignment:

Write an assembly language program to repeatedly plot a circle. Each time the circle is redraw 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. For example:




```
ed-vm% ./circles -sp 1 -dc 360178b -bk 0
```

The required format for the options is: `"-sp <tridecimalNum>"`, `"-dc <tridecimalNum>"`, and `"-bk <tridecimalNum>"`. The program must verify the format, read the arguments, and ensure the arguments are valid. The provided main program calls the following routines:

- Value returning function ***getParams()*** to read the command line arguments (***sp***, ***dc***, and ***bk***). The procedure should read each argument, convert ASCII/Senary to integer, and verify the range. The draw and nackground color 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 error message and return false. The function 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.

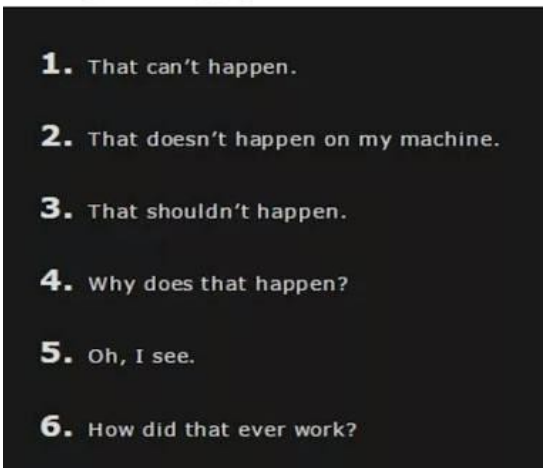
- Void function ***drawCircles()*** to plot the following equations:

$$\begin{aligned} x &= (1.0 - s)\cos(t + \pi\, s) + s\cos(2t) \\ y &= (1.0 - s)\sin(t + \pi\, s) - s\sin(2t) \end{aligned}$$

iterated $\frac{2\pi}{tStep}$ times.

The ***sStep*** is the draw speed value read from the command line divided by 10,000.0.

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 C++ main program should be linked with the assembly language functions. Only the *functions* file will be submitted. As such, the provided main file can not be altered in any way.



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.
  - Note, 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.
- Late submissions will be accepted for a period of 24 hours after the due date/time for any given lab. 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 10%.

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. Must be submitted on time for full score.

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.

Example Executions (with errors):

Below are some sample executions showing the error handling.

```
ed-vm% ./circles
Usage: circles -sp <triDecimalNum> -dc <triDecimalNum> -bk <triDecimalNum>
ed-vm%
ed-vm%
ed-vm% ./circles -s 3 -dc 360178b -bk 0
Error, speed specifier incorrect.
ed-vm%
ed-vm%
ed-vm% ./circles -sp 7 -dc 360178b -bk 0
Error, speed value must be between 1 and 4344(6).
ed-vm%
ed-vm%
ed-vm% ./circles -sp 5 dc 360178b -bk 0
Error, draw color specifier incorrect.
ed-vm%
ed-vm%
ed-vm% ./circles -sp 15 -dc 360178b -bk  360178b
Error, draw color and background color canbe the same.
ed-vm%
ed-vm%
ed-vm% ./circles -sp 3 -dc 451354104520 -bk 0
Error, draw color value must be between 0 and 1355332143(6).
ed-vm%
ed-vm%
ed-vm%  ./circles -sp 3 -dc 360178b -bkk 0
Error, background color specifier incorrect.
ed-vm%
ed-vm%
ed-vm% ./circles -sp 13 -dc 360178b -bk gray
Error, background color value must be between 0 and 1355332143(6).
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 ();
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(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(), glFlush(), and glutPostRedisplay() are as follows:

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

These function calls should not be included in the loop.

OpenGL Errors

Note, some VM's may generate error, 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 several static images of the cycling pattern are shown for reference.

```
ed-vm% ./circles -sp 1 -dc 360178b -bk 0
```