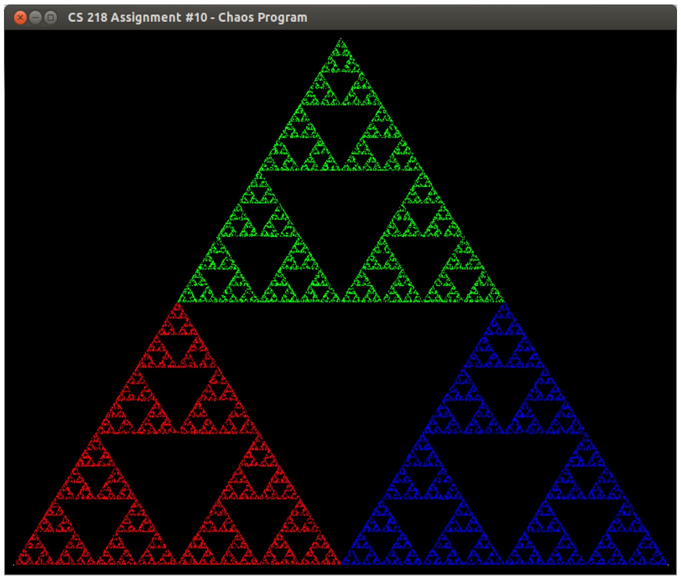
CS 218 – Assignment #10, Chaos

Purpose: Become more familiar with data representation issues, program control instructions, function handling, and operating system interaction.
Due: Friday (6/24)
Points: 225 (grading will include functionality, documentation, and coding style)

Assignment:

Write a simple assembly language program to plot a series of points on the screen using the provided algorithm. The number of points to plot must be read from command line (as binary). For example:

```
ed-vm% ./chaos -it 111111111111111 -rs 101
```



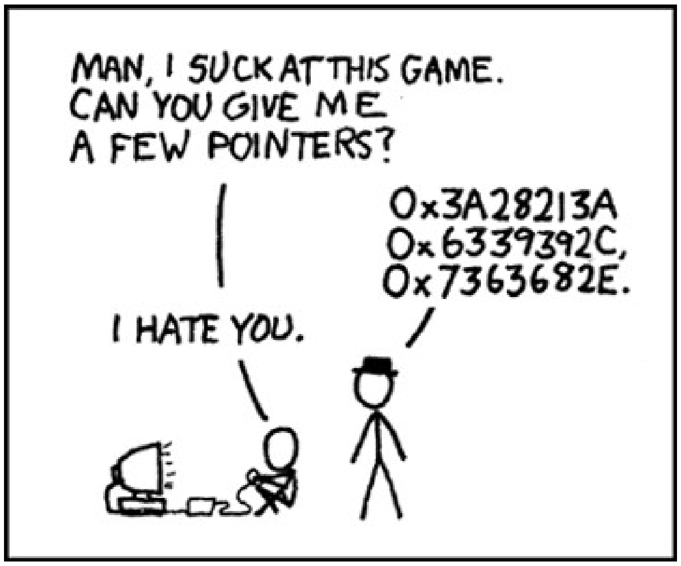
The format for the command line is "**`-it <binaryNumber>`**" and "**`-rs <binaryNumber>`**" (in that order). The program must read the iterations specifier (-it) and the rotation speed (-rs) as binary numbers, and ensure the numbers are valid. Additionally, the program must ensure that thes numbers with the specified range (provided defined constants), inclusive. The values are unsigned. If there are any errors, the program should display an appropriate error message (pre-defined) and terminate.

The provided C++ main program calls routines to check and read the command line arguments. The openGL system will call a user-written routine, ***drawChaos()***, to plot the points. All functions must follow the standard calling convention as discussed in class.

Compile, Assemble, and Linking Instructions

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

When compiling, 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.


MAN, I SUCK AT THIS GAME. CAN YOU GIVE ME A FEW POINTERS?
0x3A28213A 0x6339392C, 0x7363682E.
I HATE YOU.

Functions

The provided main program calls the following functions:

- Value returning function ***getIterations()*** to read the command line arguments (***it***, and ***rs***). The function should read each argument, convert ASCII/Binary to integer, check validity, and verify the ranges. 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 ***it***, and ***rs*** values are between a minimum and maximum (provided defined constants). If there are any input errors, the program should display an appropriate provided error message and return false. Refer to the sample executions for examples.
- Void function ***drawChaos()*** to implement the chaos plotting algorithm.
- The ***aBin2int()*** function to convert the ASCII/binary string to an integer.

Chaos Point Plotting Algorithm

Implement the following algorithm to generate the initial three points (***x,y***), where ***i***=0 to 2, positions for plotting:

$$initX[i] = \sin\left(\left(rSpeed + (i * dStep)\right) * \frac{pi}{180}\right) * scale$$

$$initY[i] = \cos\left(\left(rSpeed + (i * dStep)\right) * \frac{pi}{180}\right) * scale$$

Implement the following algorithm to generate (***x,y***) positions for plotting:

```
seed = 987123
for i = 1 to iterations do
     s = rand(seed)
     seed = s
     n = s mod 3
     x = x + ( (init_x[n] - x) / 2 )
     y = y + ( (init_y[n] - y) / 2 )
     set color (r,g,b)
     plot (x, y)
end_for
```

Note, for the color 0=>red (255,0,0), 1=>green (0,255,0), and 2=>blue (0,0,255).

Rotation Speed

Before the plotting is performed, the ***rStep*** value should be set as follows;

$$rStep = \frac{rotateSpeed}{rScale}$$

Before leaving the function, the ***rSpeed*** value should be incremented by ***rStep***.

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

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 (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 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 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. Must be submitted on time for full score.

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 update
sudo apt upgrade
sudo apt install libgl1-mesa-dev
sudo apt install freeglut3 freeglut3-dev
```

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

Random Number Generation

To generate a pseudo random number, use the *linear congruential generator* method. It is fast, simple, and (if instantiated with the right constants) gives reasonable pseudo-random numbers. The next random number is generated from the previous one by:

$$R_{n+1} = (A \times R_n + B) \bmod 2^{16}$$

The initial random number R_n (on which the rest are based on is referred to as the "seed"). The value for A must be a prime number. For our purposes, set A=9421, B=1, and SEED to 987123. *Note*, to provide a random number between 0 and 2, the plot points algorithm uses the "mod 3" function (remainder after division).

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 (bottom window). Do ***not*** type the program name after the **run**, just the command line arguments.

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 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.

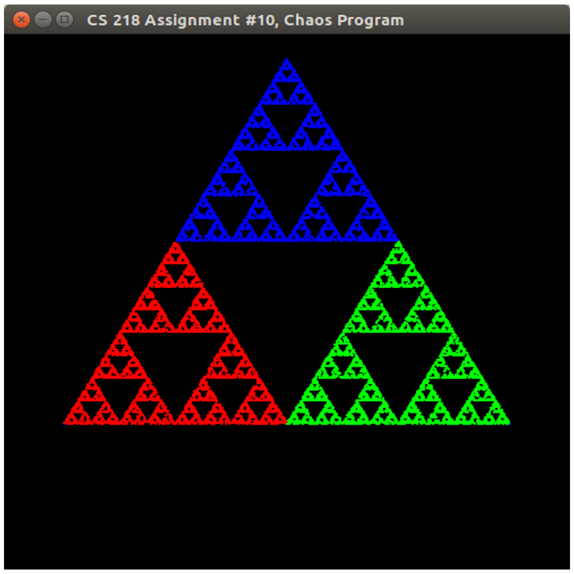
```
ed-vm% ./chaos
Usage: chaos -it <binaryNumber> -rs <binaryNumber>
ed-vm%
ed-vm% ./chaos -it 1111111
Error, invalid or incomplete command line argument.
ed-vm%
ed-vm% ./chaos -ot 1111111 -rs 101
Error, iterations specifier incorrect.
ed-vm%
ed-vm% ./chaos -it 11111211111111 -rs 101
Error, invalid iterations value.
ed-vm%
ed-vm%
```

```
ed-vm%
ed-vm% ./chaos -it 11111111111111 rs 101
Error, rotation speed specifier incorrect.
ed-vm%
ed-vm% ./chaos -it 11111111111111 -rs 10x1
Error, invalid rotation speed value.
ed-vm%
```

<u>Example I/O:</u>

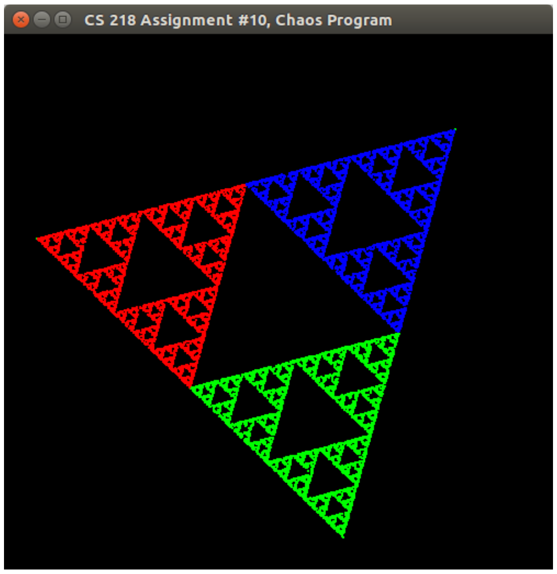
A correct output will appear similar to the following:

```
ed-vm% ./chaos -it 111111111111111 -rs 101
```



Note, the window can be terminated by typing 'q' or 'x' (while the mouse is in the window) or clicking on the **x** in the upper left corner.

Note, the image should rotate at a rate based on the used provided rotate speed.


CS 218 Assignment #10, Chaos Program