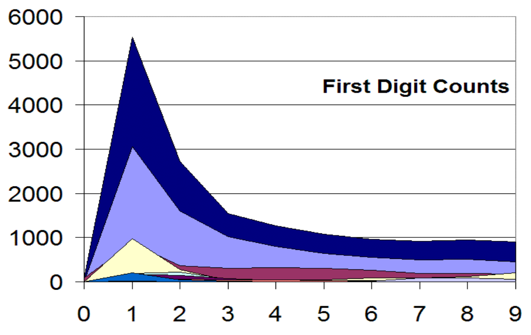
CS 218 – Assignment #11, Part A

Purpose:	Become more familiar with operating system interaction, file input/output operations, and file I/O buffering.
Due:	Monday (6/27)
Points:	250 (grading will include functionality, documentation, and coding style)

Description:

Benford's Law[1] describes a surprising pattern in the frequency of occurrence of the digits 1–9 as the first digits of natural data. You might expect each digit to occur with equal frequency in arbitrary data. In truly random data (over appropriate ranges) each digit does appear with equal frequency. However, a substantial amount of natural data from diverse sources do not exhibit a uniform distribution. Instead, 1 is more common than 2, 2 more common than 3, and so forth. We will test this law using a program to read data source files and count the leading digits.



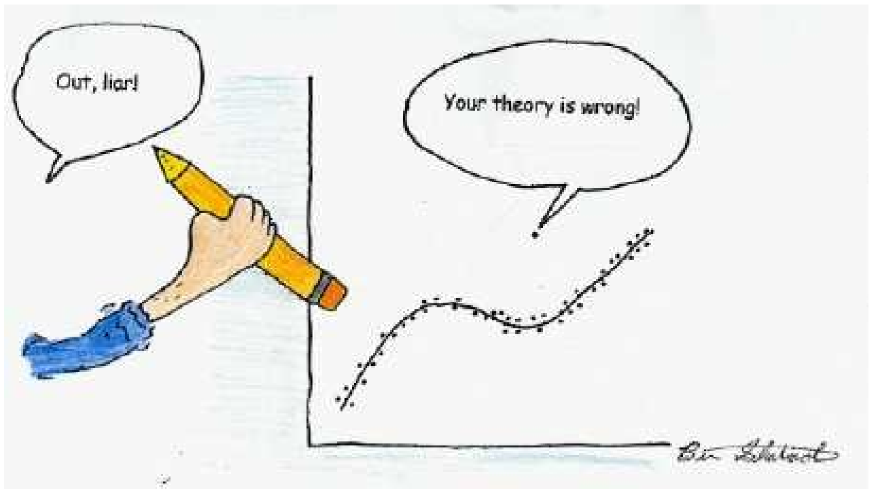
Assignment:

Write an assembly language program that will read a data source file, count the leading digits, and output a formatted text-based graph of the results. The graph will be written to the file and, optionally displayed to the screen. The program should read the screen display option (true or false) and both file names (input and output) from the command line. An example command line is:

```
ed-vm% ./benford  -i books.txt  -o results.txt [-d]
```

The [ ]'s around the **-d** means that the **-d** argument is optional. The program must perform error checking on the command line arguments.

The input files will be specially formatted. The first 5 lines will be header information, and the first 7 characters of each line will be title characters (that must be skipped). Each number will be of different size. You need only check for the first digit. You do not have to handle files that do not conform. You will be provided a series of example files for testing. Additionally, the output graph should follow the provide example (with the binary counts).



1 For more information, refer to: http://en.wikipedia.org/wiki/Benford's_law

To ensure overall efficiency, the program **must** perform buffered input and output with a buffer size of BUFF_SIZE (originally set to 500,000). *Note*, this will be changed for part B. You must use the provided main routine. ***The main program must not be changed in any way.*** All your functions must be in a separate, independently assembled source file.

Functions

The provided main program calls the following functions:

- Value returning function ***getArguments()*** to read the command line arguments. The input file must be specified as "-i <filename>" and the output file must be specified as "-o <filename>". If provided, the optional display option must be specified as "-d". If the display specifier is no provided, the display to screen boolean must be set to false. If no command line input was entered, invalid/incorrect file names were entered, or no files matching the specification were entered an appropriate error message should be generated. If an error occurs, the appropriate error message should be printed and the function should return false. If all arguments are correct, the function should return the three arguments (via reference) and return true. Refer to the sample executions for error message examples.
- Void function ***countDigits()*** to implement the chaos plotting algorithm. The I/O buffering must be fully implemented in this function.
- A void function, ***writeString()***, to write a string to an output file. The function must count the number of characters to write.
- A void function,***aBin2int()***, to convert the ASCII/binary string to an integer. The string must be returned with exactly 32 characters (1's and 0s) and NULL terminated (for a total of 33 characters).

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 (**`a11procs.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, **`a11procs.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.

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 (`a11procs.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.

Example Executions:
The following execution will read file "books.txt" and create a file named "results.txt".

```
ed-vm%
ed-vm%
ed-vm% ./benford -i books.txt -o results.txt -d
CS 218 Benfords Law
Test Results

Source File: books.txt

Total Data Points: 00000000000000000010001110110010

 0  00000000000000000000000000000000 |
 1  00000000000000000000101111110000 | ******************************
 2  00000000000000000000011001000110 | ****************
 3  00000000000000000000001111111010 | **********
 4  00000000000000000000001100100001 | ********
 5  00000000000000000000001010000000 | ******
 6  00000000000000000000001000110000 | *****
 7  00000000000000000000000111110110 | *****
 8  00000000000000000000000111110111 | *****
 9  00000000000000000000000111000100 | ****

ed-vm%
ed-vm% ./benford -i books.txt -o results.txt
ed-vm%
ed-vm% more results.txt
CS 218 Benfords Law
Test Results

Source File: books.txt

Total Data Points: 00000000000000000010001110110010

 0  00000000000000000000000000000000 |
 1  00000000000000000000101111110000 | ******************************
 2  00000000000000000000011001000110 | ****************
 3  00000000000000000000001111111010 | **********
 4  00000000000000000000001100100001 | ********
 5  00000000000000000000001010000000 | ******
 6  00000000000000000000001000110000 | *****
 7  00000000000000000000000111110110 | *****
 8  00000000000000000000000111110111 | *****
 9  00000000000000000000000111000100 | ****

ed-vm%
ed-vm%
ed-vm%
```

```
ed-vm%
ed-vm%
ed-vm% ./benford -i books.txt -o
Error, too few characters on the command line.
ed-vm%
ed-vm%
ed-vm% ./benford -i books.txt -o tmp1.txt -d -d
Error, too many characters on the command line.
ed-vm%
ed-vm% ./benford i books.txt -o tmp1.txt
Error, invalid input file specifier.
ed-vm%
ed-vm% ed-vm% ./benford -i books.txt - tmp1.txt
Error, invalid output file specifier.
ed-vm%
ed-vm% ./benford -i boks.txt -o tmp1.txt
Error, can not open input file.
ed-vm%
ed-vm% ./benford -i books.txt -o tmp1.txt -dd
Error, invalid output display specifier.
ed-vm%
ed-vm%
ed-vm% echo "hello world" > tmp1.txt
ed-vm% chmod 400 tmp1.txt
ed-vm%
ed-vm% ./benford -i test/books.txt -o tmp1.txt
Error, can not open output file.
ed-vm%
ed-vm% chmod 600 tmp1.txt
ed-vm%
ed-vm%
```