p5 Getting Started

Introduction: In this project, students are given some executables whose source codes is not provided. Hence, we need to use a disassembler (objdump) and a debugger (gdb) to figure out what they are actually doing.

Prerequisite:
- Knowledge of basic assembly code (review: lecture notes, videos, and have or make your own x86 cheatsheet)
- Knowledge of gdb (review: follow any basic or introductory gdb tutorial, and have or make your own cheatsheet)

Steps:
(1) Disassemble and save the output for each of the executables. Example: `objdump -d ./s1 > s1.txt`

(2) Open your dis-assembled assembly file in a text editor of your choice.

(3) Search for the "main" function -- it will be labeled as "main".

(4) In main, try to understand what each statement or group of statements is doing.

(5) Now, add comments to each line of assembly code. Or in some cases it makes more sense to add comment to a multiple line of assembly codes -- for some higher level concepts. For example, in lines (s1) 0x782 to 0x786, there are push statements. These are saving some registers. That's basically just saving the callee saved registers -- they are not that related to the result -- not very helpful in terms of actually helping us to get our answer later, but, they are still helpful to know that it doesn't have much to do with the results.

(6) If you encounter function calls, you can simply comment as function call for now. And later on when you run "gdb" if those function calls seem important then you can come back and analyze and add comments.

(7) You should specially pay attention to statements which call "fail" and "success" functions. Those instructions and preceding instructions are result of or set up conditional branches.

(8) Make sure you understand the logic before and when it calls "fail" and "success" because that's where our main focus is.

(9) The program asks for 5 inputs. As the program asks for the first input.

- If you provide wrong input it will call fail and print failure message and exit.
- Otherwise, it succeeds and control transfer to next iteration of the loop to check all value.

(10) If all 5 inputs are correctly entered only then it will call success which is your end goal. The "main" should contain 5 loop iterations for this. So, you should be able to figure out where the loop is

and when and how are fail and success called.

(11) After understanding the overall assembly code now you should try to understand in high level what the code is doing, and you might make note on the side on what you believe it is doing and what should be done to get to "success".

(12) Now, run gdb, step by step. At the same time you should also look at your commented source code on the side to verify and understand each steps.

(13) Set break-points at locations which you think are important depending upon your commenting from previous steps. For example: main, fail, success, start of loop, condition checks, before inputs, before outputs etc.
(14) When it asks for input that's where you have to be extremely careful in analyzing what it expects. You can first try with any random input and continue step by step and from your previous commenting and analysis of the code, you might be able to figure out what input will result in calling success. At any point you can stop and restart gdb again until you figure out the correct inputs.

(15) After you find out all correct inputs, please save each input in its own separate line as mentioned in the project requirements. Do this for each one and submit when completed as described by the project. Don't forget to follow the project specification line-by-line and word-by-word.

Summary of helpful "gdb" commands:

Common gdb commands

Purpose	Code	Comments/Examples in ASM
Set breakpoint at main	`b main`	
Set breakpoint at an address (say. 0x56555856)	`b *address`	`b *0x56555856`
Start and break (stop) at first line in main	start	
Run/Rerun the program	`run`	
Step	`s`	step i
Next	`n`	next i
Continue (until next breakpoint)	`c`	
Finish (run till the end of current function)	`finish`	
Display or print any expression or	`print`	`print $eax` `print *(0x50 + $ebx + 4*$esi)`

variable	`expression`	
Examine memory	`x /nfu address`	`x /5sb *$ebx`
Display all register values	`info registers`	
Display flags	`print $eflags`	
Display next instruction	`x/i $pc`	
Display assembly language instructions	`disass`	shows runtime assembly language addresses

- As mentioned above, for other useful commands please reference the gdb manual or have a gdb cheat-sheet ready.
- Please also note that runtime-addresses will be different from the addresses that appear on the objdump output.

Once you have your bearings (can find your way around), you are ready to try a new interface option for gdb. The CSL and likely other gdb versions support a text based user interface that can be run remotely. You can get started with these commands:

GDB with Text User Interface Option

Purpose	**Code**	**Use**
Launch gdb with tui option	gdbtui <program>	May have to install -tui before use
Add a split window to see asm and command window	layout split	
Add view of registers to your interface	layout reg	Not all registers shown are necessary for cs354 students to know.