====== Description ======
This is the Readme file to explain how I solve the problem,
an overview of the functions, and how to run(use) my program
=====================
1. --system: to indicate that only the system usage should be generated. It should include cpu and memory display.

2. --user: to indicate that only the users usage should be generated.

3. --graphics: to include graphical output in the cases where a graphical outcome is possible as indicated below.

4. --sequential: to indicate that the information will be output sequentially without needing to "refresh" the screen (useful if you would like to redirect the output into a file).

5. --samples=N: if used the value N will indicate how many times the statistics are going to be collected and results will be average and reported based on the N number of repetitions. Default value is 10.

6. --tdelay=T: to indicate how frequently to sample in seconds. Default value is 1 sec.
===Some General Questions=======
1. How to read the argv?
The argc indicates how many parts(sub-command) in my command line. Thus, I can use it to for loop the whole command line and compare them with --sample=N, --tdelay=N, --sequential, --graphics/-g, --system, and --user. And by the piazza post, we can use any library we want. Therefore, I use strstr, strcmp from string.h to compare which command to make each flag to 1.

2. How to know which function should I run in my program.
Based on the user's input, after processing the arguments, I use several flags to indicate which function should the program run, for example, when user call "--user", my program will assign the flag of "--user" to 1.

3. How to solve the problem positional arguments problem?
Basically, I use strlen to get the length and then for loop it. While I am looping each character of the command, I check if it is integer by using isdigit() from ctype.h. If it is character(letters), it will automatically end this time loop and begin checking subcommand by using “continue”. If it is integer, then I just need to identify it is N or tdelay(T). Therefore, I use a variable to indicate if it is first single digit or the second

digit. First single digit is N and second single digit is delay. And then I use atoi() to convert it into real integer.(before converting, it is a string but only one char)

4. How to print at the same time and it is really fast.
I use ANSI Escape Sequence that controls test terminal. I use "\x1b[H\x1b[2J" to clear screen screen and send the cursor to home position. Therefore, I clear the output every time and print out everything updated fields and then it will look like refreshing and it is really fast.

Another relevant question is how to make cpu refresh at the same line. After I print out my cpu usage, my cursor will go next line since '\n'. And then I use \033[1A to make the cursor up 1 line. And next time loop, it will print on the same line so that the new line will cover the old line but the number is different. Therefore, it will looks like refreshing at the same line.

=================Command Function====================

1. System Information: I use uname function from sys/utsname.h to get the system information in the structure pointed to by buf. The utsname is defined in <sys/utsname.h>. And then use basic struct knowledge to get the information we want.

2. User: In order to solve this problem, I import utmp.h since the utmp file allows one to discover information about who is currently using the system. There may be more users currently using the system, because not all programs use utmp logging. I use getutent() since it return a pointer to a struct utmp on success. And then I need to check if it is Normal process. If it is, then I will access its utmp struct to get info.

3. Memory: In order to get the memory usage, I use the getrusage function from sys/resource.h. Getrusage will help me to get information about resource utilization. And then I use ru_maxrss to obtain the memory used by my program. For different parts of memory used, I use sysinfo() from sys/info.h. Sysinfo() returns certain statistics on memory and swap usage, as well as the load Average. First of all, I need to get the struct. sysinfo() gives me that struct.

Formulas: Phys.Tot = totalram
Phys.used = totalram - freeram(total - unused memory = used memory)
Virtual.Tot = totalram + totalswap(total + total swap space size)
Vir.used = totalswap - freeswap + Phys.used(total swap - unused swap = used swap size and then plus Phys.used)

After I got these formulas, it is easy to calculate the different kind of memory. Since we need to send   formatted output therefore, I use a 2D array to store them. And then I use

sprintf to send formatted output to this 2D array by string. Therefore, when we run our main, I can easily print them out in a good format.
Lastly, we need to convert to GB. Therefore, I divide 1024^3.

4. *System.* "--system" includes two parts.
  1. Memory(Above)
  2. CPU usage.

CPU: 1. Number of cores: I create a pointer of file and let it equal to /proc/stat by fopen() and and then I use fscanf to read and use strstr to identify how many cpu we have since the property of /proc/stat file.

CPU usage: As usual, first of all, I use fopen to open my file and I use fgetc to read the file   since this time I need to skip the first three or four letters(cpu or cpu#). And then I can get the number easily. For the method of getting these numbers, I use strtok to separate the string and then use strtol to get the number and use a variable called total to cumulate. To be more specific, we need to store one important value called idle. Therefore, when index = 4, I need to use a special variable to store idel.

And then sum all of the times found on the first line to get the total time and then divide the fourth column("idle") by the total time, to get the fraction of time spent being idle. And then subtract the previous fraction from 100 to get the time spent being not idle.

Formula: 1 - ((idle - prev_idle) / (total - prev_total)) * 100 =
    100 -[(idle - pre_idle) * 100] / (total -prev_total)

And then let previous_idle = current idle and let previous_total so that it is convenient for us to calculate next loop. And then use 2D array and sprintf to store formatted output.

5. Sequential: "--sequential". Basically, I use a flag to indicate if we need to do sequential. If we need to do sequential, then we need to print out the index of i in the 2D array.(string) If we loop to the index of not i, we just print "   \n" so that we can skip and achieve the function of sequential.

6. --sample=N: I use strstr and strtol to identify if it is --sample=N command and get the number(N) respectively.

7. --tdelay=T: Same as --sample=N

8. --graphics and -g:
  I have a flag belong to graphics

a) Memory: In order to keep have previous_used, I need to have a pointer to prev_used so that when I call the function, I can get the difference between current usage and previous usage. After I get the difference, I divide into 4 cases.

1. when difference == 0, which means no change. So after and before '|', nothing is here.
2. when it is the first sample, "|o 0.00(vir_used)" is put into 2D array. (I did exactly as same as in the assignment 1 description on quercus).

3. when difference > 0, I will convert the difference to 2 decimal places. And then I will divide into 2 case(based on my reasonable assumption)
    - when the difference > 0.01, I will for sure strcat * and #. And then how 0.01 do I have, I will insert how many # and then at the end, insert a '*'.
    - when the difference < 0.01(really small), then I just strcat '*' after '|'
4. when difference < 0, same as diff > 0 but this time is negative(decreasing). After I did all my assumption and then I use twice sprintf to store my formatted line into corresponding 2D array.

b) CPU: In order to keep having previous_used, I need to have a pointer to store previous idle and previous total. My assumption is the first line is just 3 bars and after the line, each bar is 0.77. And then I use a temp_string to store the bar and then use sprintf to store in a 2D array.

==

How to use my program.
(— is two "-")
gcc a1.c & ./a.out
and after ./a.out, you can add command

1. —system
2. —user
3. —sequential
4. —graphics
5. —samples=N (N represent any integer)
6. —tdelay=T (T represent any integer)
7. add any two integer. Ex: ./a.out 5 3

Here are some combinations

1. gcc a1.c && ./a.out => system + user
2. gcc a1.c && ./a.out -g(—graphics) => system + user + graphics

3. gcc a1.c && ./a.out —user => user and just system info
4. gcc a1.c && ./a.out —system => system
5. gcc a1.c && ./a.out —sequential => basic sequential(system and user)
6. gcc a1.c && ./a.out —system —sequential => sequential for system

==

Function documentation

1. system_infomation() : Get the system information by using name from <sys/utsname.h>
2. session_user: Print all the session_user by importing <utmp.h>
3. memory_usage(): Print and get the memory usage by importing <sys/resource.h>
4. memory(): Store all the formatted line related to memory into 2D array and they will be printed later.
5. get_num_cpu(): Get the number of CPU by reading "/proc/stat".
6. cpu(): Store all the formatted line related to cpu usage and they will be printed later into 2D array.
7. display_user_f(int flag, int system): Print out user info.
8. check_seq(int seq, int iter): Check if we need to do sequential.
9. display_user_f(int flag, int system): Check if we need to print user
10. display_given_val(int seq, int num, int dly): Print the number of samples and delay.