How I solved the problem:

Part 1) Fetching the data:

—I asked many people, and also searched online to see where to find each section of the information.

Part 2) Presenting the data

How My Program Works:

—**Taking in command line arguments**: I run a for-loop over argc to check over each string in argv to see if that string matches any of the acceptable command line arguments. If one of the command line arguments doesn't match, then I print "Invalid argument" and terminate the program.

—**Positioinal arguments:** I run a positional arguments counter, which works as follows: the first instance it sees a number standing alone, it interprets that as the sample number, and the second time it sees a number standing alone, it interprets that as the delay time.

—**Invalid Arguments:**

–if you do "–samples=invalid number", my program will assume –samples=10, similarly for tdelay

–but if you do anything that's totally invalid (i.e. not one of my if statement checks), then I print "Invalid Argument" and terminate the program.

–ensure that your spellings are correct in the command line arguments

—**Refreshing the screen:** in the main function, I run a for-loop that loops over samples, the number of times to sample, and in each iteration, I call the function perform_loop, which takes in all the different flags, as well as two arrays, one for storing cpu time, another for storing formatted strings that represent physical and virtual memory usage:

–on the ith iteration,

–I clear the screen and move the cursor to the top left corner

–i print out the header lines, that is, number of samples: N, every t seconds, \n memory: x kilobytes

–I print out the first i lines of the #memory# section.

–I re-calculate cpu usage percentage

–I call the print_user_info on each iteration

–If the sequential flag is activated, then I print them without clearing the screen

This basically works as a "refresh".

Function Documentation:

1. `print_mem_use`

Purpose: Prints the current and previous memory usage information.

Parameters:

- `int currentLine`**: The current iteration number.**
- `int totalLines`**: The total number of iterations specified by the user.**
- `bool sequential_flag`**: A flag indicating whether the output should be sequential.**
- `char mem_usage_lines[MAX_SAMPLES][MAX_MEM_LENGTH]`**: A 2D array for storing formatted strings of memory usage information.**

Behavior:

- **Fetches the current memory usage information.**
- **Stores the formatted string in the appropriate slot of** `mem_usage_lines`**.**
- **Prints all lines up to the current iteration in non-sequential mode or prints only the current line in sequential mode.**
- **Handles memory allocation and ensures proper freeing of allocated resources.**

---

2. `print_user_info`

Purpose: Prints the information about current user sessions.

Parameters: None.

Behavior:

- Opens and reads the `utmp` **file to fetch session/user data.**
- **Prints user, terminal, and host information for each active user session. Checks if a user is active by comparing** `current_record.ut_type == USER_PROCESS`

---

3. `print_header_line`

Purpose: Prints the header line indicating the program's memory usage.

Parameters: None.

Behavior:

- Uses `getrusage` **to get the memory usage of the program itself.**
- **Prints the memory usage in kilobytes.**

---

4. `print_cpu_usage`

Purpose: Calculates and prints the CPU usage percentage.

Parameters:

- `int* cpu_time_array`**: An array (on the heap) holding the last idle and last total CPU time to calculate the difference in subsequent calls.**

Returns:

- `int*`**: The updated** `cpu_time_array` **with the latest idle and total CPU time.**

Behavior:

- **Reads the CPU times from** `/proc/stat`**.**
- **Calculates the CPU usage based on the difference from the last read times.**
- **Prints the calculated CPU usage percentage.**
- **Updates the** `cpu_time_array` **with the current idle and total CPU time for the next calculation.**
- **Here's my algorithm for the exact computation (pasted from https://rosettacode.org/wiki/Linux_CPU_utilization):**
  - **read the first line of** `/proc/stat`
  - **discard the first word of that first line (it's always** `cpu`**)**
  - **sum all of the times found on that first line to get the total time**
  - **divide the fourth column ("idle") by the total time, to get the fraction of time spent being idle**
  - **subtract the previous fraction from 1.0 to get the time spent being *not* idle**
  - **multiple by 100 to get a percentage**

---

5. `print_system_stats`

Purpose: Prints general system information.

Parameters: None.

Behavior:

- Uses `uname` **to fetch and print system information such as system name, node (machine) name, release, version, and architecture.**
- **Uses** `sysinfo` **to fetch and print system uptime information.**
- **Uses modular arithmetic to convert the days, hours, minutes, and seconds appropriately.**

---

6. `perform_loop`

Purpose: Orchestrates the printing of system information based on the specified flags and iteration count.

Parameters:

- `int i`**: The current iteration number.**
- `int samples`**: The total number of iterations specified by the user.**
- `int tdelay`**: The delay between iterations.**
- `bool system_flag`**: A flag indicating whether to print system information.**
- `bool user_flag`**: A flag indicating whether to print user information.**
- `bool sequential_flag`**: A flag indicating whether the output should be sequential.**
- `int* cpu_time_array`**: An array holding the last idle and total CPU time.**
- `char mem_usage_lines[MAX_SAMPLES][MAX_MEM_LENGTH]`**: A 2D array for storing formatted strings of memory usage information.**

Behavior:

- **Calls the appropriate functions to print memory, user, and CPU usage information based on the specified flags.**

- **Handles the appropriate sections to print based on the various input boolean flags**
- **Handles the sequential and non-sequential output modes.**

---

7. `main`

Purpose: Entry point of the program. Processes command-line arguments and initializes program execution.

Parameters:

- `int argc`**: The number of command-line arguments.**
- `char* argv[]`**: An array of command-line argument strings.**

Behavior:

- **Parses command-line arguments to configure the program's behavior (number of samples, delay, sequential flag, system flag, user flag).**
- **Allocates memory for CPU time tracking.**
- **Initiates the loop for printing system information.**
- **Frees allocated resources and prints system statistics at the end.**