Design approach:
We heavily followed the implementation hints for our project. First, we have a loop that reads input from the input.txt file. Instructions will be read while the instruction arrival time is less than the next internal event time. The internal event time is calculated after each new instruction to determine if the quantum is to be interrupted by a device request. We take the head of the ready queue, and if it can experience a whole quantum, we add the quantum to the time accrued and put the process to the back of the ready queue. If a process will finish within the quantum, it will free all resources it held. The wait queue is then checked with bankers algorithm to see if any processes with requested resources can be given those resources and leave the system in a safe state. Then the hold queues are checked to see if there is enough available memory to be taken off the hold queue and onto the ready queue. Once the instruction is 'D 9999', the simulation is not to receive further instruction. Once the time is equal to 9999, the system state is printed and the average turnaround is given. If there are still lines to be read after 'D 9999', we have another simulation. All used memory is deallocated and all variables are reset to prepare for a brand-new simulation.

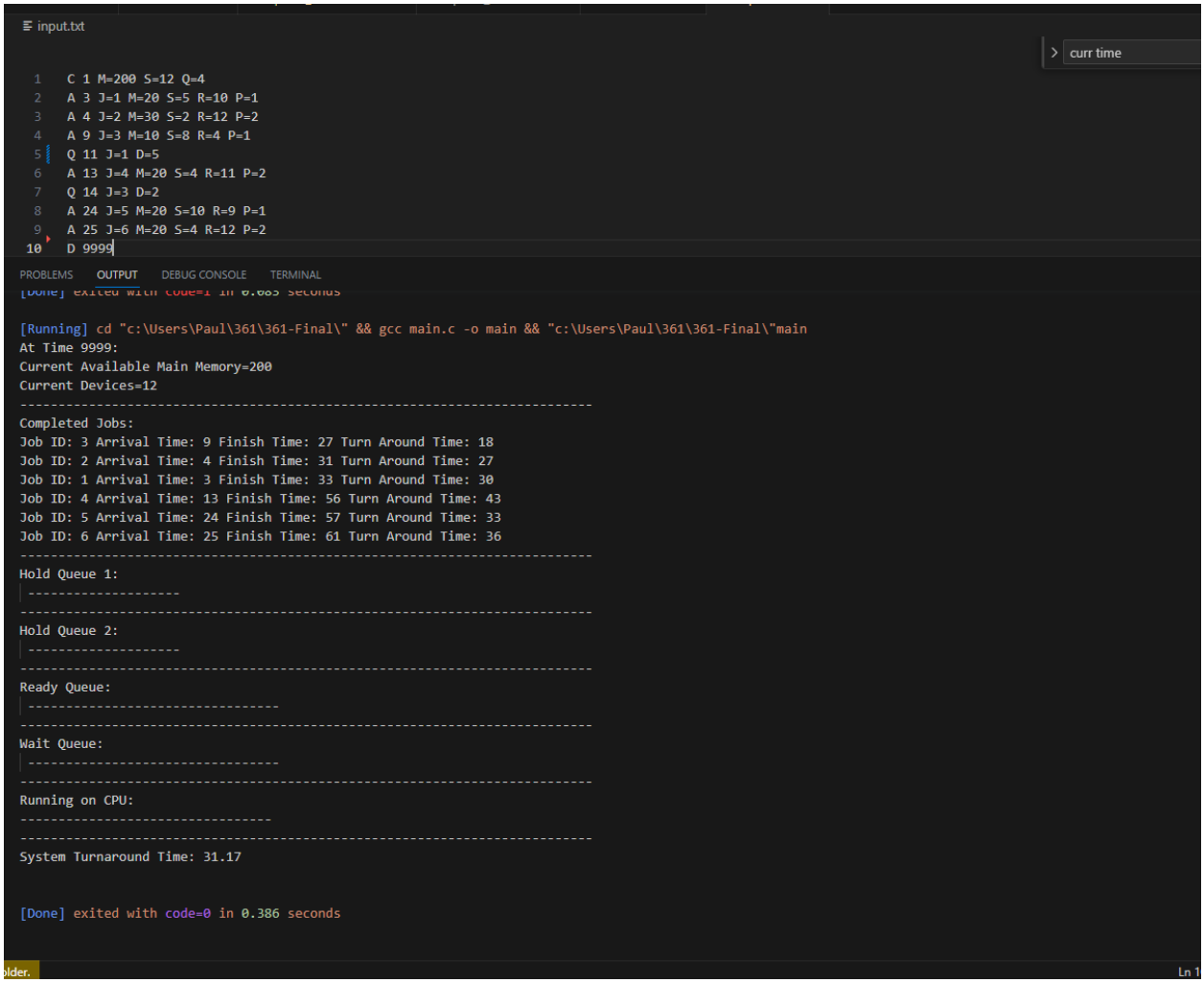
Data structures and algorithms used:

We used bankers algorithm for device allocation, and linked list for each queue. We also used a linked list for the completed jobs. Our project file has a queue_functions.c file that makes dealing with linked lists manageable.

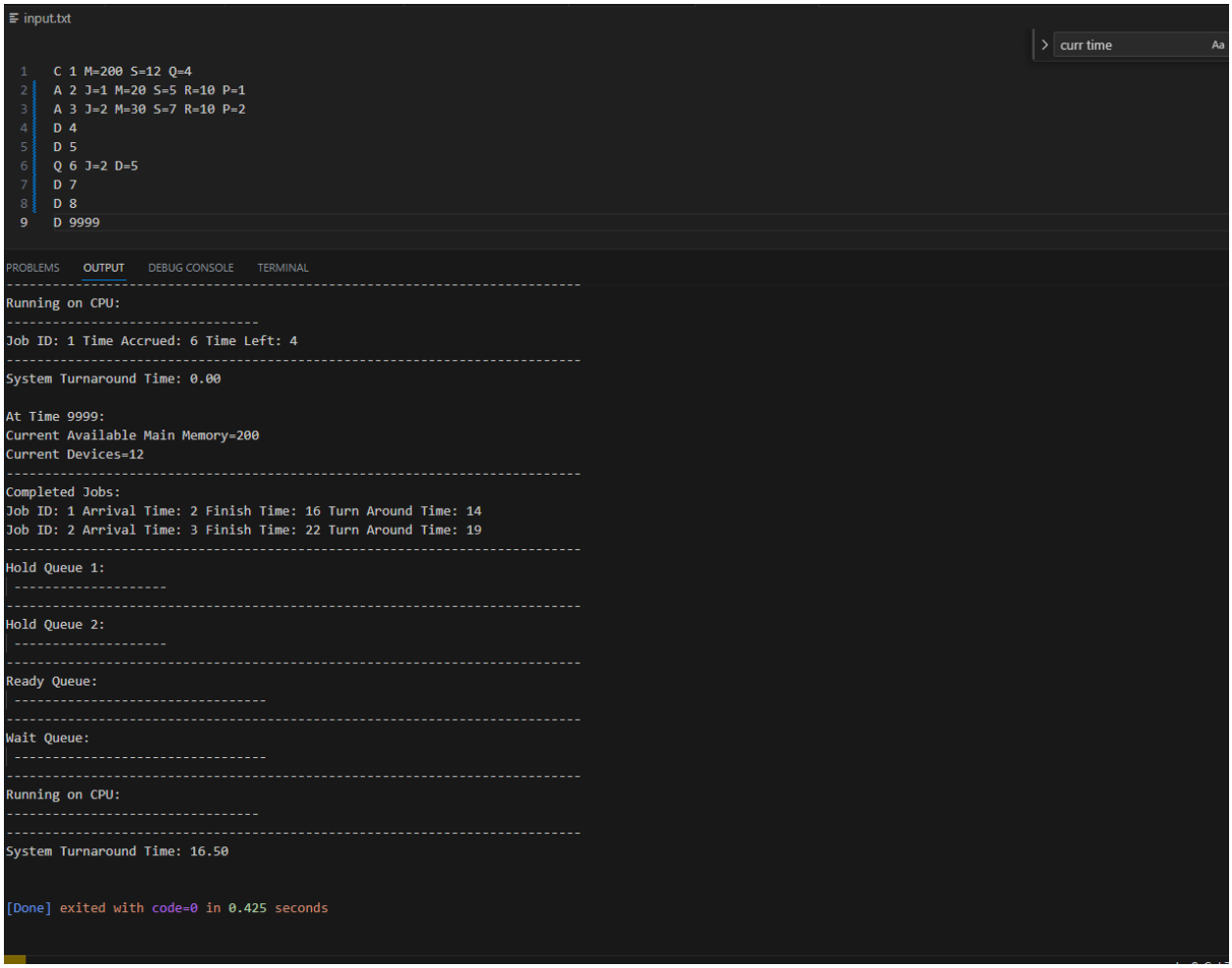
Reflection:

If we were to do this project again, we would all like to learn more about gdb for debugging. We mainly used the 'D' instruction to test our system/debug. We also learned alot about process/job scheduling and deadlock avoidance while trying to get our program to match the expected output so not only are we far more confident on how the OS interacts with processes, but we all would have a far easier time making a project similar to this one. Also, I think we should have taken the time to draw out a Gantt chart for each simulation instead of just comparing the end of the simulation to the expected end. While printing the system state often was useful, it was somewhat redundant.

Output for input 0:



Output for input 1



Output for i2:

```
C 1 M=200 S=12 Q=4
A 3 J=1 M=20 S=5 R=10 P=1
A 4 J=2 M=30 S=2 R=12 P=2
A 9 J=3 M=10 S=8 R=4 P=1
Q 12 J=1 D=5
A 13 J=4 M=20 S=4 R=11 P=2
Q 14 J=3 D=2
Q 26 J=4 D=11
L 27 J=3 D=2
D 9999
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
[Done] exited with code=0 in 0.392 seconds

[Running] cd "c:\Users\Paul\361\361-Final\" && gcc main.c -o main && "c:\Users\Paul\361\361-Final\"main
At Time 9999:
Current Available Main Memory=200
Current Devices=12
--------------------------------------------------------------------------------
Completed Jobs:
Job ID: 2 Arrival Time: 4 Finish Time: 31 Turn Around Time: 27
Job ID: 1 Arrival Time: 3 Finish Time: 32 Turn Around Time: 29
Job ID: 3 Arrival Time: 9 Finish Time: 37 Turn Around Time: 28
Job ID: 4 Arrival Time: 13 Finish Time: 40 Turn Around Time: 27
--------------------------------------------------------------------------------
Hold Queue 1:
--------------------
--------------------------------------------------------------------------------
Hold Queue 2:
--------------------
--------------------------------------------------------------------------------
Ready Queue:
---------------------------------
--------------------------------------------------------------------------------
Wait Queue:
---------------------------------
--------------------------------------------------------------------------------
Running on CPU:
---------------------------------
--------------------------------------------------------------------------------
System Turnaround Time: 27.75


[Done] exited with code=0 in 0.431 seconds
```