1. A high-level description of the overall solution strategy:
    - Initialize queues to have values of -1
    - Input file name
    - Read first two lines for number of servers
    - Allocate memory to dynamic Server array where each index represents a server for the purpose of storing the status and idle time of each server.
    - Read first event from file as arrival and store in events min heap.
    - While occur time of the top in events is not a terminator, process top event and then pop it.
    - Processing an event involves:
        - Updating current time and time elapsed since last event.
        - If event type is an arrival
            - Find an available primary server
            - If a primary server is idle
                - Create event for leaving primary server, parsing occur time for this event (current time + time spent at primary server), time spent at primary and secondary servers, and the server that is serving them.
                - Set server to busy.
            - Else
                - Add event to primary queue.
            - Then read next line from file and create an arrival event based upon this data.
            - Add one to total people served count.
            - Add time spent at primary and secondary queues to total service time.
        - Else if event type is leaving primary server
            - Set the primary server that was serving the person who left to idle.
            - Find available secondary server.
            - If a secondary server is idle
                - Create event for leaving secondary server, parsing occur time for this event (current time + time spent at secondary server), time spent at primary and secondary servers, and the server that is serving them.
                - Set server to busy.
            - Else
                - Add event to secondary queue
            - Then, if there is a customer waiting in the primary queue, move them straight into the server that was freed during the current event.
                - Add waiting time to total time spent in primary queue (to calculate average).
                - Create event for leaving primary server, parsing occur time for this event (current time + queued event's time spent at primary server), queued event's time spent at primary and secondary servers, and the server that is serving them.
                - Dequeue event from primary queue.
                - Set server to busy.
        - Else the event type is leaving secondary server
            - Set the secondary server that was serving the person who left to idle.

            - Then, if there is a customer waiting in the secondary queue, move them straight into the server that was freed during the current event.
                - Add waiting time to total time spent in secondary queue (to calculate average).
                - Create event for leaving secondary server, parsing occur time for this event (current time + queued event's time spent at secondary server), queued event's time spent at primary and secondary servers, and the server that is serving them.
                - Dequeue event from secondary queue.
                - Set server to busy.
        - Finally, after each event has been processed, the statistics must be updated:
            - Average primary and secondary queue lengths are calculated by summing (elapsed time since last event * current length of queue).
            - If current queue lengths are larger than the maximums, update the maximums to current queue length.
            - Sum the total amount of time that each server has been busy.
- After the final event has been processed, statistics are updated:
    - Last service time = current time value, as the last event would be leave secondary server.
    - Add total waiting times for primary and secondary queues to the total service time.
    - Divide total service time, total time in primary queue and total time in secondary queue each by total people served to make them averages.
    - Add average primary queue time and average secondary queue time together to get the average overall que time.
    - Divide value of (for each event processed sum(elapsed time*(pri or sec)queue length)) by the total time elapsed to produce the average queue lengths.
    - Add average primary queue length and average secondary queue length to get the average overall queue length.
    - If max length of primary queue is larger than max length of secondary queue, the overall max queue length is max length of primary queue, other wise it is the max length of secondary queue.
    - For each server in the primary queues, subtract the total busy time from total time elapsed to get the total idle time.
- Print stats, rounding down to hundredths or tenths depending on the scale used.
- Delete dynamic memory used for storing servers.

2. A list of all of the data structures used, where they are used and the reasons for their choice.

- **Event – Used in arrays: events, priQue, secQue; and in functions: process(Event), pushEvent(Event), enPriQueue(Event), enSecQueue(Event).** Used for storing **enum EventType**{arrive, primary, secondary} in order to know how to process the data stored in the event, **occurTime** a double by which the events are sorted that marks when the event should be processed, **priTime** a double which stores the time spent at the primary server of the customer which corresponds to this event, and **secTime** a double which serves the same purpose as priTime, but for the secondary server.
- **Server – Used in dynamic arrays: priServer, secServer; and in function: process(Event).** Used for storing **bool busy,** which is the status of the server (busy or idle) and **double**

idleTime, which is used to store the sum total of busy times of each server at the end of each events loop iteration. At the end of the processing, this will be converted to total idle time.

3. A list of any standard algorithms used, where they are used and why they are used.
   - **Min Heap – Line 259**
     - **Siftup(int):** Used for adding a new Event to the events array. When a new event is created and added to the end of the heap, it is sifted up into its correct position in the min heap by swapping it with its parent when it violates the heap.
     - **Siftdown(int):** Used for removing the top event from the events array. The first event in events is overwritten by the last event in events, and that event is then sifted down to the end of the heap, moving everything up as it goes.
     - **Swap(int, int):** Swaps events where siftup and siftdown deem appropriate.
   - **Queue – Line 335**
     - **en***Queue:** Adds an event that's being processed to the corresponding queue.
     - **de***Queue:** Deletes an event that's been processed from the corresponding queue.