Alexander Williams
V01042108
CSC 370, Summer 2024

1. How many threads will be used, and for what?
   This program will use two types of threads.
   - Clerk threads, one for each clerk, to process each customer. These are terminated after all customers are served.
   - Customer threads, which exist to ensure each customer enters the queue at the correct time.
      Once the customer enters the queue, the thread terminates.

2. Do the threads work independently, or is there a controller thread?
   There is no controller thread, all customers and clerks are independent

3. How many mutexes will be used? what does each one guard?
   This program will use 2 mutexes
   - mtxQ, which guards all operations involving adding or removing customers to the queue, along with adjusting values like queue head and tail positions.
     Both customers and clerks use this mutex
   - mtxTotal, which guards the values for total wait time for each queue.
     Only clerks use this mutex

4. Will the main thread be idle?
   Once all the threads are created, the main thread will be idle until it has rejoined all the queues.
    However, once all customers and clerks have joined, it will be in charge of calculating averages

5. How will customers be represented? What type of data structure will be used?
   Customers will be represented in a struct, which is stored in an array that acts like a queue

```
struct Customer
{
    int id;
    int class;
    int arrivalTime;
    int serviceTime;
};
```

6. How will you ensure data structures won't be modified concurrently?
   This will be done using the two previously mentioned mutexes. At the start of each critical section,
    the thread will call pthread_mutex_lock(&mutex), and then call pthread_mutex_unlock(&mutex)
    once it leaves the critical section

7. How many convars will be used?
   This program will only use one convar
  a) what condition will it represent?
   this convar, called cvQ, represents when there are no customers in the queues.
  b) which mutex is associated with this convar?
   the related mutex is mtxQ.
   it will only be used by clerks, as they don't need to grab a customer when there are no customers.
  c) what operations should be performed after pthread_cond_wait()?

```
pthread_mutex_lock(&mtxQ);
while(!Condition)
{
    pthread_cond_wait(&cvQ, &mtxQ);
}
// CRITICAL SECTION
clerk pulls customer from queue
clerk adjusts qHead for that queue
if(Condition)
{
    pthread_cond_signal(&cvQ);
}
pthread_mutex_unlock(&mtxQ);
// END CRITICAL SECTION
```

8. Give a brief sketch of the overall algorithm used

```
int main()
{
   obtain file

   set up global variables
   set up mutexes and convars

   create clerk threads on function ClerkThreads

   create customers on function CustomerThreads, read line from file into each one, customer thread processes it

   join customers

   join clerks
```

```
    calculate averages

    destroy mutexes and convars, free data
}

void* CustomerThreads(char* data)
{
    read data into Customer struct

    sleep until ready to enter queue

    // CRITICAL SECTION
    pthread_mutex_lock(&mtxQ);
    add customer into queue
    adjust qTail
    pthread_cond_signal(&cvQ);
    pthread_mutex_unlock(&mtxQ);
    // CRITICAL SECTION OVER

    pthread_exit(0);
}

void* ClerkThreads(int* id)
{
    while(!done)
    {
        // CRITICAL SECTION WITH CONVAR
        pthread_mutex_lock(&mtxQ);
        while(!Condition)
        {
            pthread_cond_wait(&cvQ, &mtxQ);
        }
        // CRITICAL SECTION
        clerk pulls customer from queue
        clerk adjusts qHead for that queue
        if(Condition)
        {
            pthread_cond_signal(&cvQ);
        }
        pthread_mutex_unlock(&mtxQ);
        // END CRITICAL SECTION

        // CRITICAL SECTION AGAIN
        pthread_mutex_lock(&mtxTotal);
```

```
        add time customer sent waiting in queue to total
        if(total customers served == total number of customers)
        {
            done = 1;
        }
        pthread_mutex_unlock(&mtxTotal);
        // CRITICAL SECTION OVER

        sleep for service time of customer
    }
}
```