Final Project Report
Patrick Feeney, William Bernard

Design
Before anything else, the doors, boxes, and robots are randomly generated onto the grid. This is done through the “randomlyGeneratePositions()” function.

The robots within the application move through the “moveRobot()” function, which take the “robot” (an unsigned int) and the direction to move (a char) as arguments. The “moveRobot()” function moves a robot only **one square** within the grid.

“solveRobot()” performs the path planning algorithm, as discussed in *final.pdf.* Perform any movements are performed, the function calculates two things:

(1) The amount the robot should move to reach its box (dxRob, dyRob - representing the total distances that the robot must move)
(2) The amount the box should move to reach its door (dxBox, dyBox - representing the total distances the box must move)

Once these distances are calculated, two things occur:

(1) Within one while-loop, until the robot reaches its correct x-position, move the robot one-by-one via the moveRobot() function. After this, using a seperate while-loop, until the robot reaches its correct y-position, move the robot one-by-one via the moveRobot() function. Please note: *the robot must move until the distance between the “robot” and the “box” is **1**, as it must appear, visually, that the robot is pushing the box.*
(2) Within one while-loop, the box is pushed in the x-direction, Then, in another while-loop, the box is pushed in the y-direction. Eventually, this is done until the box is pushed into the door.

The code was modified to adhere to each version’s requirements. In V1, no threads or mutexes are used. In V2, each robot is implemented as a seperate thread, and the output file is protected by a mutex. In V3, each grid element is protected by a mutex. The initialization for these mutexes can be seen below:

```
posLocks = malloc(numRows * sizeof(pthread_mutex_t *));
    for(int i = 0; i < numRows; i++)
    {
        posLocks[i] = malloc(numCols * sizeof(pthread_mutex_t));
        for(int j = 0; j < numCols; j++)
            pthread_mutex_init(posLocks[i] + j, NULL);
    }
}
```

Limitations of the Program

(1) There are some certain instances where the program has difficulty exiting, specifically when the final robot moves. This was an issue I was able to reproduce on Ubuntu, but not on macOS.
(2) Deadlock when two robots want to access the same grid.
(3) Program cannot run when the grid is too small -- specifically, when there are more objects than the total amount of grid spaces.

Project Difficulties

In terms of difficulty, randomly generating the objects on the grid took a great deal of time. While generating a random array of numbers is one problem, generating a random array of numbers that are **each unique** is more difficult. In order to give each object a unique position, we do the following:

(1) Assign a random door-ID to each position. This simply uses the C random number generator, as robots can share the same door-ID.
(2) Create random, **unique,** positions for each robot, box, and door in the array.
    (a) For each object, generate a coordinate.
    (b) If this coordinate has already been generated, try again.