Assignment 2

Deadline:	Hand in by 5pm on Friday 22nd May 2020
Evaluation:	10 marks – which is 10% of your final grade
Late Submission:	1 mark off per day late
Work:	This assignment is to be done **individually** – your submission may be checked for plagiarism against other assignments and against Internet repositories. If you adapt material from the internet you must acknowledge your source.
Purpose:	Solving a nontrivial synchronisation problem.

Problem to solve:

An incomplete C implementation of a **lift simulator** is available (for both Windows and UNIX) on the course stream site (`a2_handout.c`). This program simulates a number of people moving between different floors in a building using a number of lifts. Each person and each lift are controlled by a different thread.

Currently there are a number of missing sections of code (marked by `---`) and variety of race-conditions, you must complete the code then identify and protect against these errors.

Description:

The program will create a thread for each lift and for each person in the building. Lifts go up and down picking people up and taking them to where they want to go.

Each person starts on a certain floor (`from`) and thinks of a random floor they want to go to (`to`). Then they wait for a lift going in the correct direction by waiting on a semaphore (`up_arrow` or `down_arrow`) associated with the floor they are on (`from`). When a lift signals this semaphore, the person will get into the lift, possibly press the button for the floor they want to go to and wait for the lift to arrive at their destination floor (`to`) by waiting on another semaphore. When they reach their destination they will get out, wait for a while and then start another journey.

There are two semaphores for each floor, one for people going up and one for people going down. In each lift there is a semaphore for each floor. The people in the lift can wait on these before getting in or out of a lift.

Each **floor** in the building is described by the following data structure with two semaphores that people can wait on – `up_arrow` and `down_arrow`.

```
typedef struct {
   int waitingtogoup;    // The number of people waiting to go up
   int waitingtogodown;  // The number of people waiting to go down
   semaphore up_arrow;   // People going up wait on this
   semaphore down_arrow; // People going down wait on this
} floor_info;

floor_info floors[NFLOORS];
```

(continued over)

Each **lift** in the building are described by the following data structure which has one semaphore for each floor that the passengers inside can wait on.

```
typedef struct {
   int no;                     // The lift number (id)
   int position;               // The floor it is on
   int direction;              // Which direction it is going (UP/DOWN)
   int peopleinlift;           // The number of people in the lift
   int stops[NFLOORS];         // How many people are going to each floor
   semaphore stopsem[NFLOORS]; // People in the lift wait on one of these
} lift_info;
```

The algorithm that each **person** should follow is:

```
while(true) {
   wait for a while (do some work)
   pick a different floor to go to
   if(going up)
      press up arrow and wait
   otherwise
      press down arrow and wait
   get into lift
   press button for floor to go to
   wait for lift to reach destination
   get out of lift
}
```

The algorithm that each **lift** should follow is:

```
while(true) {
   drop off all passengers waiting to get out at this floor
   if going up or empty
      direction is up
      pick up as many passengers on this floor going up as possible
   if going down or empty
      direction is down
      pick up as many passengers on this floor going down as possible
   move lift
   if on top or ground floor
      reverse direction
}
```

(continued over)

Requirements:

You must complete the program by adding semaphores (and semaphore calls) to synchronise between the threads.

The lines marked with a `---` need to be completed before the program will compile, most just need wait or signal calls. You will need to use a global variable (of type `lift_info*`) to tell people which lift to get into.

Once these lines have been completed, the program should work in some form but occasionally the screen will not be drawn properly (parts of the building may be overwritten). This is because the threads are all printing to the screen at the same time.

Add a mutual exclusion semaphore to prevent threads printing at the same time.

Try changing the settings `LIFTSPEED`, `GETINSPEED` and `GETOUTSPEED` to `10`. This may work reasonably well for a while but eventually the lifts will stop picking people up and the program may even crash. This is caused by threads accessing the same shared data.

Try changing the settings `LIFTSPEED`, `GETINSPEED` and `GETOUTSPEED` to `0` and `PEOPLESPEED` to `1`. This will make things go at top speed and make your program even more unreliable.

Add the necessary semaphores to the program to allow it to work reliably at top speed. Think about everywhere the program accesses shared data.

The threads in your assignment should not hold a mutex lock around a `Sleep()` call (with the possible exception of the print function).

Details:

There are some types and functions defined to help you implement your program in the file `lift.h`. These types/functions are designed to make the program work correctly on different operating systems. The ones you will need to use are:

```
semaphore                                  - semaphore type
semaphore_wait(semaphore *s)               - wait on semaphore s
semaphore_signal(semaphore *s)             - signal semaphore s
semaphore_create(semaphore *s, int value) - initialise semaphore s with value v.
```

Take note: you need to pass the address of the semaphores to these functions.

Compilation:

You should not need any special flags to compile this program on the lab machines. On macOS or linux you will need to link to the pthreads library:

```
gcc assignment2_handout.c –o assignment2 –lpthread
```

You **must** follow the next two specifications in **each and every assignment for this course**

1. Place the following comments at the top of your program code and **provide the appropriate information**:

```
/* Family Name, Given Name, Student ID, Assignment number, 159.341
*/
/* explain what the program is doing . . . */
```

2. Ensure that your `main` function uses `printf` to print this information to the console. You might use code like:

```
int main( int argc, char * argv[] ){
   printf( "----------------------------------------" );
   printf( "  159.341 Assignment 2 Semester 1 2020  " );
   printf( "  Submitted by: Rick Deckard, 20191187  " );
   printf( "----------------------------------------" );
   ...
}
```

Hand-in: Submit **your program and documentation (a zip file is acceptable)** electronically through the form on the stream site.
Marks will be allocated for: correctness, fitness of purpose, sensible use of data structures and algorithms, utility, style, use of sensible **comments and program documentation**, and general elegance. Good comments will help me to award you marks even if your code is not quite perfect.

If you have any questions about this assignment, please ask the lecturer.