**CMPUT 275 - Tangible Computing** **Winter 2020**
**Assignment #1**

---

**Restaurant Finder**

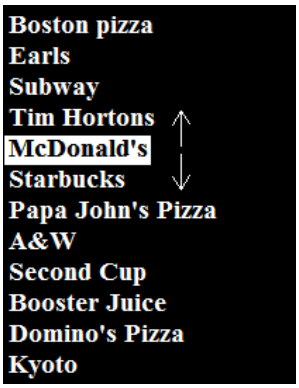
---

## Overview:

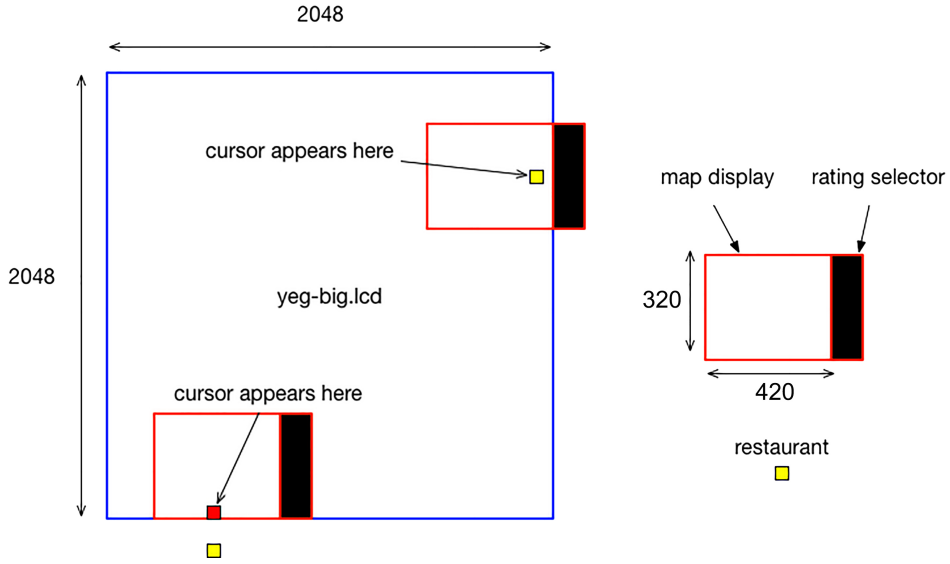
In this major assignment, you will be implementing a restaurant finder for the Edmonton area on your Arduino. The program will load an Edmonton map and show a cursor for the current location on the map. Using a joystick, you will be able to move the cursor around the map and press down the joystick to select the current location. Then, the restaurants in the order closest to that location will be displayed on the LCD. Then you will be able to select a restaurant on the screen using the joystick. Pushing the joystick again will return you to the map with the selected restaurant at the centre of the map section. A demonstration is shown below:


When your program starts up, it should display a portion of the Edmonton map (referred to as a map section) on the leftmost 320 x 420 pixels of the display. The rightmost 320 x 60 rectangle will be black for part 1, and will contain rating and sort selector buttons for part 2. The user can use the joystick to move the cursor around (the red dot in the centre). Note that the red arrows are just to indicate that the cursor is movable. Your program should not display such arrows.

When the user presses the joystick, your program will display a list of 21 restaurants using a `fontSize` of 2. They will be displayed in order based on their distances to the current location of the cursor on map: the closest should be on the top as the first entry; the second closest should be displayed as the second entry, and so on. Moreover, one of the restaurants should be highlighted to indicate user's selection. The joystick will be used again to move the highlight block as shown by the white arrows (again, these arrows are just to show you how the block could be moved, and you program should not display such arrows).




When the user presses the joystick again with restaurant selection, your program will return to the map. However, now your map section should be shifted so that the selected restaurant is located at the centre of the map display, co-located with the cursor. (Note this map is now shifted, compared to previous map) The user will be able to move the cursor and navigate the map again.




There are two boundary cases about displaying the selected restaurant (as shown above):

1. If the selected restaurant is close to the map boundary, then the cursor does not have to be placed at the centre of the screen. Rather, the screen should be centred as much as possible over the restaurant (i.e. against the edge of the Edmonton map) and the cursor should be located on the restaurant.

2. If the selected restaurant is not in the range of the map, then the cursor should be placed on the map location that is closest to the selected restaurant.

We will discuss in class the code for loading the map, drawing the cursor, handling joystick input and touchscreen input, reading SD card data and basic sorting. **Among other tasks, you will be writing the code for:**

1. Cleaning up the cursor movement (partially developed in weekly exercise 1).

2. Efficiently loading the restaurants from the SD card using a simple caching scheme (partially developed in weekly exercise 2).

3. Sorting the restaurants based on **Manhattan distance** (see below) to the current point.

4. Displaying the names of the restaurants in increasing order of distance to the cursor on the LCD screen.

5. Allowing the user to tap anywhere on the map region of the screen (the leftmost 320 x 420 pixels) to view small dots indicating the locations of all restaurants visible in the current screen.

6. **Part 2 only:** Allowing the user to select restaurant sorting algorithm (insertion sort, quicksort, or both) and printing statistics to the serial monitor.

7. **Part 2 only:** Allowing the user to set a minimum restaurant rating using the touch screen. Then, the restaurant finder will only display restaurants with a rating that is greater than or equal to the minimum rating set by the user.

### Submission Details:

The assignment is structured into two parts. The first part is worth 50% and the second part is worth 50%.

- Part 1 is due on Monday, Feb 3 at 11:55pm
- Part 2 is due on Monday, Feb 10 at 11:55pm

**You will work with a partner on this assignment using the full collaboration model. You must cite all of your sources that contributed to your assignment.**

For each part, one or both group members should submit all of the following files as a `.tar.gz` or `.zip` file:

- all files required to build your project including the `Makefile`, `lcd_image.h`, and `lcd_image.cpp`. You may create multiple source code files if it helps you organize your project better.
- your `README`, following the Code Submission Guidelines

## Part 1: Simple Version of Restaurant Finder

In the first part, you will develop a simple version of the restaurant finder. Your program will have two modes that we will call Mode 0 and Mode 1.

### Mode 0

Your program should display a scrollable version of the large map of Edmonton provided ("yeg-big.lcd" with 2048 x 2048 pixels, which will be on the card after formatting it in class). The big map is too large to be displayed in its entirety on your screen, so your program will show only a portion (a "patch") of it at a time. The user can control a cursor to move around the entire map using the joystick. You should not change the section of the map shown every time the user moves the joystick, only when the cursor reaches the edge of the screen.

When the cursor hits the edge of the map display, the map should shift over by one entire "patch" of the Edmonton map. For example, if the cursor hits the top edge of the map then the display shifts up an entire 320 pixels to show the new part of Edmonton. Of course, this "scrolling" should be clamped to the boundary of the map image itself. For example, if only 40 pixels of the Edmonton map lie above the current map display, then after nudging upwards the Edmonton map display

would only move up by 40 pixels.

After nudging the map when the cursor hits the edge of the display, you may choose to either centre the cursor in the middle of the map display again or to move the cursor to be close to the geographic position of where the cursor was before (i.e. if you nudged the screen up then the cursor could be left at the bottom of the new display close to where the old cursor was on the map of Edmonton).

The rightmost 320 x 60 pixels should be black for this entire part. You will do something with this region of the display in part 2.

### Mode 1

Whenever the user presses the joystick button in Mode 0, your program will enter Mode 1 and display a list of the restaurants nearest to the cursor's current position. For Part 1, you will only need to display the 21 restaurants nearest to the cursor on the LCD screen using the text size multiplier of 2. Using the joystick, the user should be able to move a highlight block to select a restaurant on the list.

When you push the button in Mode 1, your program will enter Mode 0 and redisplay the map, with the selected restaurant located at the centre of the screen (there are two exceptions, see above). That means you probably need to shift the map section accordingly.

## Loading the Restaurants Efficiently

You have to load in the restaurant structs from the SD card into memory. Recall from the lecture that each block is 512 bytes and each restaurant is 64 bytes.

```
struct restaurant {
    int32_t lat; // Stored in 1/100,000 degrees
    int32_t lon; // Stored in 1/100,000 degrees
    uint8_t rating; // 0-10: [2 = 1 star, 10 = 5 stars]
    char name[55]; // already null terminated on the SD card
} restaurant;
```

Due to how slow it can be loading from the SD card, you will discover that it is important to load data efficiently. As we will discuss in class (and you saw in weekly exercise 2), you will need to implement a simple rudimentary caching scheme to make help make loading data more efficient.

## Sorting the Restaurants

Sorting the restaurants will involve computing which restaurants are closest to the current position. We could use straight-line distance to the restaurant; but to be more realistic, we will instead use Manhattan distance:

$$d((x_1, y_1), (x_2, y_2)) = |x_1 - x_2| + |y_1 - y_2|$$

Though still not realistic for crossing rivers (or other impassable areas), it is slightly better than straight-line distance in many situations in a city like Edmonton especially in places where streets are arranged in a grid-like fashion.

In the actual implementation, we will want to load all the restaurants into memory for sorting, as read/write access on the SD card is too slow for sorting. However, with 64 byte structs for restaurants, we would need (64 bytes $\cdot 1066) \geq$ 64KB. So, the restaurants cannot be stored in the Arduino's 8K of memory. Instead we will use a simplified, smaller restaurant struct so that the `NUM_RESTAURANTS = 1066` restaurants can all fit in the 8K of memory. Since we are sorting them on $(x, y)$ position, we will use the following struct:

```
struct RestDist {
  uint16_t index; // index of restaurant from 0 to NUM_RESTAURANTS-1
  uint16_t dist;  // Manhatten distance to cursor position
};

RestDist rest_dist[NUM_RESTAURANTS];
```

Using this smaller struct now only requires us to store 4 bytes per restaurant; since (4 bytes $\cdot 1066) <$ 4.5K, we can now sort in memory. When loading in restaurant info, each restaurant needs to be converted into this form for sorting. If more information is needed about a restaurant after sorting (such as name or rating), it can be fetched later from the SD card with the `index` field.

### Insertion Sort

For Part 1, we will use insertion sort. In class, you practiced developing code from pseudocode. You will be expected to do the same thing for insertion sort in this major assignment.

You should base your insertion sort code off the following pseudocode (taken from Wikipedia):

**Algorithm** Insertion Sort

```
i ← 1
while i < length(A) do
    j ← i
    while j > 0 and A[j − 1] > A[j] do
        swap A[j] and A[j − 1]
        j ← j − 1
    end while
    i ← i + 1
end while
```

The idea behind this sorting algorithm is the following. At the start of each iteration $i$, the array $A[0] \ldots A[i-1]$ will be already sorted. The inner loop takes the next element $A[i]$ and swaps it back through the sorted array until it finds its place. After this, the subarray $A[0] \ldots A[i]$ is sorted and we are ready to increment $i$ and repeat.

Insertion sort runs in $O(n^2)$ time.

**Your task:**

- Translate the above pseudocode into a function called `isort()` which takes the following parameters:
  - A pointer to the array of `RestDist` structs to be sorted.
  - The length of the given array.

The `isort()` function should not return anything (should have a void signature), but instead should **modify** the given array directly.

- As part of the insertion sort algorithm, you will also need to create an additional function called `swap` (also void) to swap the locations of two given restaurant structs in the array.

**Be sure to use the distance field** of each simplified restaurant struct (see above) when comparing restaurants when you sort them.

### Displaying Text on the Screen

Here's a code snippet that displays the names of the first 21 restaurants as stored in the `rest_dist` array, which is a `RestDist` struct (see the Sorting the Restaurants section above for a definition of the RestDist struct). This is very similar to the code we developed in class. Note that

- We use the `setTextColor` function to highlight the restaurant selection.
- We have to read the name for the restaurant off the SD card because it's not stored in the `RestDist` struct.
- We will write the function `getRestaurant` together in class.

Code for displaying names was / will be developed in class on Jan 22. The following snippet shows some tips on how to integrate `displayNames` with `getRestaurant`:

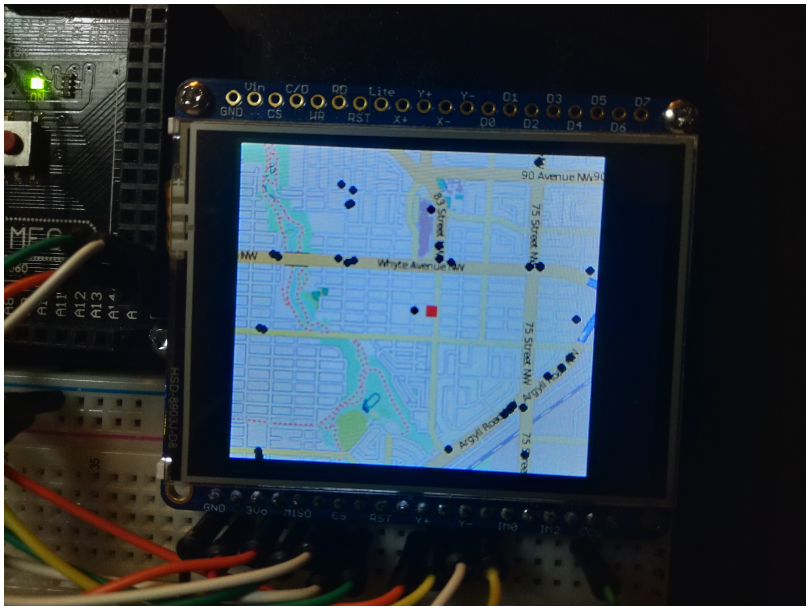
```
tft.fillScreen(0);
tft.setCursor(0, 0); // where the characters will be displayed
tft.setTextWrap(false);

selectedRest = 0; // which restaurant is selected?
for (int16_t i = 0; i < 21; i++) {
    Restaurant r;
    getRestaurant(restDist[i].index, &r);
    if (i != selectedRest) { // not highlighted
        // white characters on black background
        tft.setTextColor(0xFFFF, 0x0000);
    } else { // highlighted
        // black characters on white background
        tft.setTextColor(0x0000, 0xFFFF);
    }
    tft.print(r.name);
    tft.print("\n");
  }
tft.print("\n");
```

Moving the highlight block on the restaurant list with the joystick is very similar to the movement of the cursor on map: reprinting the whole list for each movement is too slow and will not be acceptable, so we only do partial printing with the help of the `setCursor` function. More information about these functions can be found on the documentation of the Adafruit-GFX-Library.

### Displaying Restaurant Locations on the Screen

By default, the restaurants stored on the SD card are not visible on the Edmonton map. When scrolling around the map, it would be helpful to be able to see where the nearest restaurants are located, so you can navigate to them more easily. You will implement this functionality so that touching the map draws the restaurants as dots.

**Your task:** When the user taps anywhere on the map section of the screen (the leftmost 320 x 420 pixels), your program should display a small dot in the location of each restaurant in the current view. Here is an example for part 1 (on an older TFT display model, which is not the same as the one used in class) where each nearby restaurant is drawn using a small dots:



**Implementation Details:**

- You have some flexibility in how you portray the restaurant dots. However, they must be no larger than the cursor and easily distinguishable. The indicators portraying the restaurants must be drawn entirely within the map region. It is ok (i.e. it is your choice) to not draw restaurants that are close to the boundary of the current map view, say within 3 or 4 pixels.
- If the cursor runs over a dot, it is okay to simply erase it. You do not need to worry about redrawing dots. It is also ok if a restaurant dot is initially drawn over the cursor when you first tap the screen. **Optional Challenge (not for marks):** Ensure that restaurant dots are redrawn when the cursor moves over them (but only if the user has tapped while on the current map section to enable showing restaurant dots!).
- If the cursor moves to a new map view, the new map section should be displayed with no dots. If the user wants the dots back, they have to tap the screen again.
- Once you have implemented this functionality, you can use it to check that your restaurant selection is working properly. After selecting a restaurant in Mode 1, your program should return to Mode 0 with the cursor centred on the selected restaurant. You can confirm that this worked by tapping the screen to display nearby restaurants. You should see a small restaurant dot printed directly over your cursor (representing the selected restaurant).

## Part 2: Full Version of Restaurant Finder

The full version of the restaurant finder adds three extensions:

1. **Implement Quicksort:** We will quickly review the quicksort algorithm in class, but you will be required to implement it on your own and use it to sort the restaurants.
2. **Scrollable Restaurant List:** Use the joystick to scroll through more than the restaurants than can be displayed on one screen by displaying a new page of restaurants if you scroll past the current screen (should be able to move back a page by scrolling up from the top restaurant). After scrolling past the last restaurant on the display, the first restaurant on the next page should be highlighted. Similarly, after scrolling back one page the last restaurant on the previous page should be highlighted.

   So, for example, after scrolling down past the 30th restaurant the display should change to show the **next** 30 restaurants in terms of distance to the cursor.

   It is your decision on whether to wrap around the list: you can either have scrolling up from the 1st restaurant load the last page of restaurants with the furthest restaurant selected or to make scrolling up from the 1st restaurant do nothing. Similarly for scrolling down past the last restaurant.
3. **Rating Selector:** Sort and display only those restaurants that have a rating above the threshold set by the touchscreen.
4. **Sort Selector:** Choose which sort should be used (`ISORT`, `QSORT`, and `BOTH`) to order the restaurant structs. You will also print statistics on the speed of the sort(s) used to the Serial monitor.

5. **Restaurant Dot Display:** Modify your solution to visualize only the restaurants that are above the rating threshold. If you tap the map display, adjust to a higher rating, and tap again, it is okay not to erase lower rating dots (although this is a good challenge if you want to implement it.)

### Button Column

In Part 2 of the assignment, the previously black 320 x 60 pixel area on the right hand side of the screen will become the button column. Here, you will display the following two buttons:

1. RATING SELECTOR BUTTON: pressing it cycles through ratings from 1 to 5 (so the label of the button changes).
2. SORT SELECTOR BUTTON: the button cycles through the text `ISORT`, `QSORT`, and `BOTH` when you press it. If you click the joystick when `ISORT` or `QSORT` is the mode, then it uses the corresponding algorithm to sort the restaurants. If you click the joystick in `BOTH`, then it runs both sorting methods in sequence. In all three modes, you will print the time taken for the chosen sort(s) to the serial monitor.

The important thing to note is that there are **only two buttons**, both of which cycle to display a different label depending on the mode chosen. For example, pressing the button when 1 is displayed as the rating could then display a 2.

**Implementation Details:** You can modify some details of the buttons in the button bar, but make sure that your solution:

- is intuitive
- does not draw anything over the map on the left 320 x 420 pixels of the display.

Here is an example of a possible button layout (again, using a different TFT display but the idea is the same). The first photo shows what the buttons look like on startup, while the second shows what should be displayed after each button is pressed once:


Figure 1: Appearance on Startup

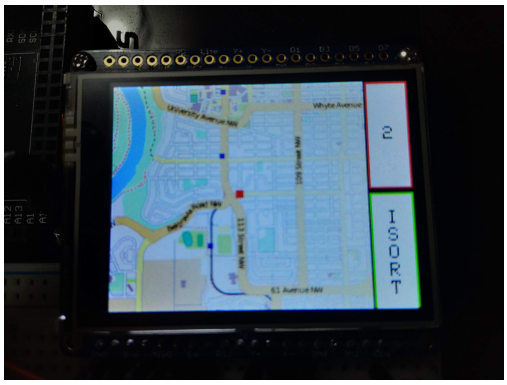



Figure 2: After Pressing Each Button Once

**Clarification:** You only need to display the buttons in the button column during Mode 0. The rating selector and sort selector are not displayed in Mode 1.

You can find more information on how these buttons work in the following two sections.

### Rating Selector

The rating selector will involve adding touchscreen controls to select the minimum rating of restaurants displayed. This allows you to "filter out" some of the more undesirable nearby restaurants. We will assume a 1 to 5 star rating system. Since the rating is stored on the scale 0-10, you can assume the natural conversion to a 5-point scale using the following formula:

$$x \in 0, 1, 2, ..., 10 : s = max(\lfloor (x+1)/2 \rfloor, 1)$$

Recall $\lfloor a \rfloor$ is the floor of $a$, which means the greatest integer that is at most $a$.

To display and change the minimum restaurant rating, you will use the touchscreen. Specifically, the button bar (formerly the "black space" on the right side of the display from part 1) will now display a visual indicator of the selected rating. You will be able to change the selected rating this by pressing the rating selector button (see the section on the Button Column above) to cycle through the various ratings from 1 to 5 with 1 being the default on startup.

Using this rating selected using the touchscreen, you will remove restaurants from the list that are below the minimum rating. For example, if you select a minimum rating of 5 stars rating, then only restaurants with a 5 star rating on the 5-point scale are displayed on the LCD, ordered still by proximity. If you select a minimum rating of 3 stars, then only restaurants with a 3, 4, or 5 star rating are displayed.

### Sort Selector

You will implement three different modes for sorting the restaurant structs. In all three modes, you should **fetch and sort only the restaurants over the rating threshold** (see Rating Selector, above) from the SD card. Note that the text `QSORT`, `ISORT`, or `BOTH` should be displayed on the sort selector button on the right hand side button column to indicate which sorting mode is currently active. You should be able to press the sort selector button to cycle through the three methods. Then, when the joystick button is pressed, you should perform the action related to current sorting mode:

1. `QSORT`: Fetch the restaurants over the rating threshold from the SD card and then use quicksort to sort the restaurant structs (calling the `qsort()` function you will develop in part 2).
2. `ISORT`: Fetch the restaurants over the rating threshold from the SD card and then use insertion sort to sort the restaurant structs (calling the `isort()` function you created in part 1).
3. `BOTH`: Perform both `QSORT` and `ISORT` as explained above. Note, this means after you run `QSORT` you should re-read the restaurants from the SD card again before calling `ISORT`. This is so the sorting algorithms are run on the **exact same initial ordering** of the restaurants with the exact same distance calculations to the cursor when it was clicked.

The default sorting mode on startup should be `QSORT`.

In all three modes, you should **record the amount of time taken by each sort used (in milliseconds)** and print the time taken to the Serial monitor in a format like the following:

`sort_algorithm` running time: `XX` ms

where `sort_algorithm` is either `Insertion sort` or `Quicksort` depending on the method used, and `XX` is the amount of time the sort took to complete in milliseconds. Record **only** the time taken to sort the restaurants, not the time taken to read from the SD card and filter restaurants.

No matter which sorting mode was chosen (`QSORT`, `ISORT`, or `BOTH`) you should then move on to Mode 1 to list the restaurants, exactly like in part 1.

**Optional Challenge (not for marks):** If you want, you may print additional information like the number of restaurants sorted (recall that you will filter out some restaurants based on the rating), the number of times this sort has been used, etc. You may find additional information like this is helpful for checking your implementation. If you decide to print additional information, make sure the running times are listed **first**.

## Further Information

We are using the same orientation convention that was used in the 1st weekly exercise, and that has been used in class to date. That is, running the program in `joy_cursor.cpp` on eClass should work naturally. **Use the same wiring**! In particular, The display rotation should be set to 1, as with other code we have seen in the class.

```
// Joystick pins:
#define JOYSTICK_VERT   A9 // Analog pin A9 should connect to pin VRx
#define JOYSTICK_HORIZ  A8 // Analog pin A8 should connect to pin VRy
#define JOYSTICK_SEL    53 // Digital pin 53 should connect to pin SW
```

### Converting Units

The map function that is part of the Arduino library will come handy. The constrain function may also be of interest.

```
// These constants are for the 2048 by 2048 map.
#define MAP_WIDTH 2048
#define MAP_HEIGHT 2048
#define LAT_NORTH 53618581
#define LAT_SOUTH 53409531
#define LON_WEST -113686521
#define LON_EAST -113334961

// These functions convert between x/y map position and lat/lon
// (and vice versa.)
int32_t x_to_lon(int16_t x) {
    return map(x, 0, MAP_WIDTH, LON_WEST, LON_EAST);
}

int32_t y_to_lat(int16_t y) {
    return map(y, 0, MAP_HEIGHT, LAT_NORTH, LAT_SOUTH);
}
```

```
int16_t lon_to_x(int32_t lon) {
    return map(lon, LON_WEST, LON_EAST, 0, MAP_WIDTH);
}

int16_t lat_to_y(int32_t lat) {
    return map(lat, LAT_NORTH, LAT_SOUTH, 0, MAP_HEIGHT);
}
```

### The Data on the SD Card

You must use an SD card that was formatted for use in this course. It contains the raw restaurant data as well as the map of Edmonton picture.

## Code Submission Guidelines

The file with the `main()` function will be `a1part1.cpp` or `a1part2.cpp`, depending on the part you are submitting. In each submission, you must include a `README` file, the Arduino `Makefile`, and any other files you used to complete the assignment. Note, you are permitted to use extra `.cpp/.h` files you created if it helps organize your code, but this is not necessary.

Submit everything as `a1partX.tar.gz` or `a1partX.zip` where `X` should be either `1` or `2`, again depending on the part. Adhere to the code submission guidelines. In particular, we should be able to compile your entire project simply by typing `make` after extracting your submission.