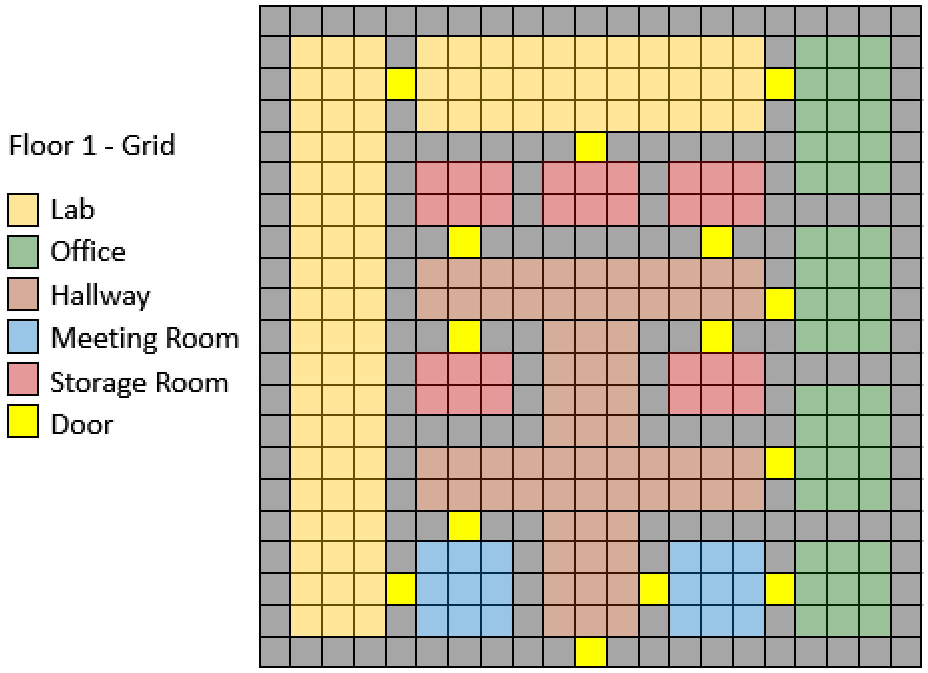
COMP2401 - Assignment #4

(Due: **Sun. Mar 15, 2020 @ 6pm**)

In this assignment, you will gain practice dynamically allocating/freeing memory as well as working with pointers to allocated structures. You will also use a makefile to compile the code. You will also get to practice your recursion … I hope that you remember how to do that.

Consider a simplified building floorplan that shows walls, doors and various rooms that are defined in a grid as follows:



We would like to write an "artificially-intelligent" program that takes arbitrary floor plans arranged in a grid (defined as a 2D array) and processes the plan to automatically identify and categorize the types and sizes of rooms in a multi-floor building.

The following files have been written for you already … you ARE NOT ALLOWED to alter them in any way, nor can you add anything to these files:

- **building.h** – header file with struct definitions and defined constants
- **buildingTest.c** – the main program that creates a building and tests your code

You will need to create a file called **buildingPlan.c** which contains all the necessary functions as described below. You will also need to create a **makefile** that compiles the program by combining the two source code files and the header file. The **makefile** must have an option for a **make clean** command which will remove all object files as well as the executable file. The TA MUST be able to simply type **make** in the terminal window to compile your code. The executable MUST be called **buildingTest**.

To get the program to work, you MUST write the following procedures in the way that is described below. You may write additional functions/procedures, but you MAY NOT alter the procedure/function signatures from what is described below:

- **findDoors**(**FloorPlan** *fplan)
    - This procedure takes a pointer to a **FloorPlan** structure as a parameter and finds all the doors for that floor. It should first iterate through the grid and count the number of doors, then store the **numDoors** total in the floor plan. (Hint: if you look at the grids defined in the main program, you will notice that walls are marked with a '1' char, doors with a '2' char and open space with a '0' char). Once the doors have been counted, you MUST allocate memory for an array that will store exactly that many doors. The array should be stored in the floor plan's **doors** pointer. You should then go through the grid again and fill in this array by setting the **row** & **col** of the door in the grid.

- **findRooms**(**Building** *build, **int** floorNum)
    - This procedure assumes that all the doors have been found previously. It takes a building pointer and a floor number as parameters. It must then go through the floorPlan's grid for that floor number to identify its rooms. You MUST go through the grid from left to right (starting at the top row) … to search for a grid location that is '0'. This will be a starting point for tracing out a room. When you find a '0', you should dynamically-allocate a new room with 0 **size** and 0 **numDoors**. You should add this room to the floor plan's linked-list of rooms (i.e., **roomList**). You MUST then call a recursive procedure (of your own creation) that will trace out the remainder of the room. To trace out a room recursively, start at some (row, col) location (i.e., the one that you found the '0' at). Then recursively traverse the grid up, down, left and right. The recursion stopping case is when you are at a non-'0' location. Make sure to mark visited locations with a character that is not '0', '1' or '2'. As you trace out the room, increase the counter for the room size and so that upon completion, the recursive function has set the room size to the correct value. Also, when you encounter a door, you should:
        - increase the door count for the room, and
        - find the door in the building at this location and set this room as one of its two rooms. That is, each door keeps track of the two rooms that it joins (i.e., **openFromRoom** and **openToRoom**). You should find out which of those two values is NULL (both may be at first) and then set that to be this room. It makes sense to use a FOR loop here to check ALL of the doors in the building to determine which one matches the location you are at while tracing. When the tracing procedure has completed, all doors on that floor should have their **openFromRoom** and **openToRoom** set accordingly. It does not matter which room is the **openToRoom** or **openFromRoom** … the order is unimportant. Note that there may be some doors that are on the edge of the building. These doors will have only one of these set … it doesn't matter which one you set.

      You MUST NOT allocated any memory while tracing out the rooms nor can you define any new arrays in the recursive function.

- **sortRooms**(**Building** *build)
    - This procedure assumes that all the doors and rooms have been found. It takes a building pointer as a parameter. It then goes through all pre-computed rooms on all floors and sorts the rooms as being an office, meeting room, lab, storage room or hallway. It MUST go through the **roomList** lists of each floor plan and build up the 5 linked lists (initially each **NULL**) of the building (i.e., the **offices**, **meetingRooms**, **labs**, **storageRooms** and **hallways** linked-lists of the building structure). The idea is to go through each of the floor plan **roomLists** and move the room into one of the five room type lists for the building. You should follow these rules to decide which list to add to:
        - If the room has size <=6, it is a storage room.
        - If the room has exactly 1 door (and size > 6), it is an office.

        - If the room has > 1 door but < 4, and has size >= 30, it is a Lab.
        - If the room has > 1 door but < 4, and has size < 30, it is a Meeting Room.
        - If the room has >= 4 doors, it is a hallway.

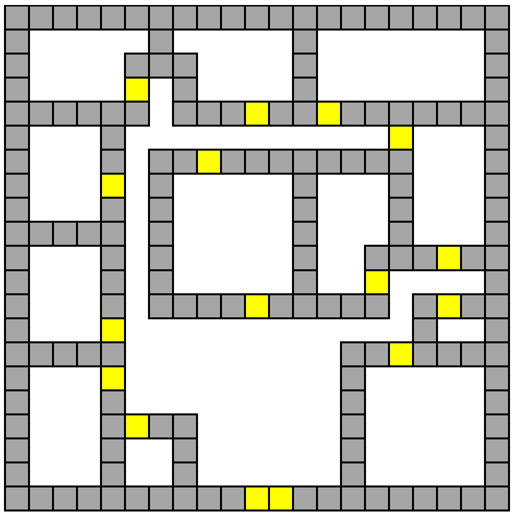
    When the procedure completes, the 5 linked lists of the building should be filled up with all rooms in the building and each floor plan's linked list of rooms should be **NULL**.

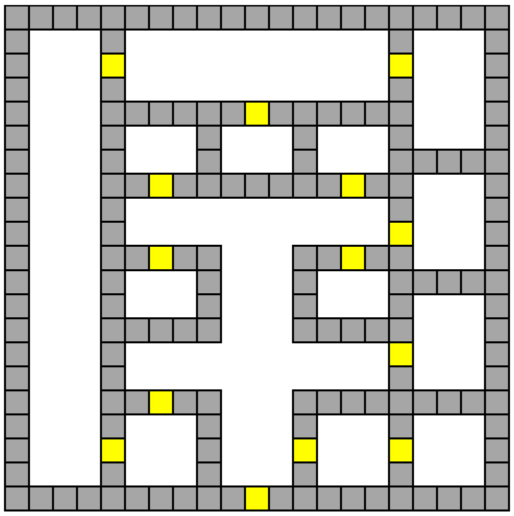
- **printRoomList**(**Room** *roomList, **char** *roomType)
    - This procedure should take a linked list of rooms and a string indicating a room type (e.g., "Office") and then display all the rooms in that list … one room at a time showing the room type, floor number, square footage (i.e., 4 times the size) and number of doors … in this format:

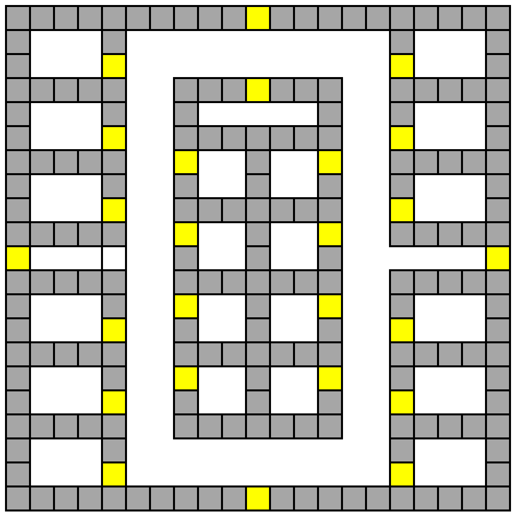
```
Meeting Room (on floor 0) with 60 square feet and 2 doors
Meeting Room (on floor 0) with 100 square feet and 2 doors
Meeting Room (on floor 1) with 36 square feet and 2 doors
Meeting Room (on floor 1) with 36 square feet and 2 doors
Meeting Room (on floor 3) with 36 square feet and 2 doors
… etc …
```

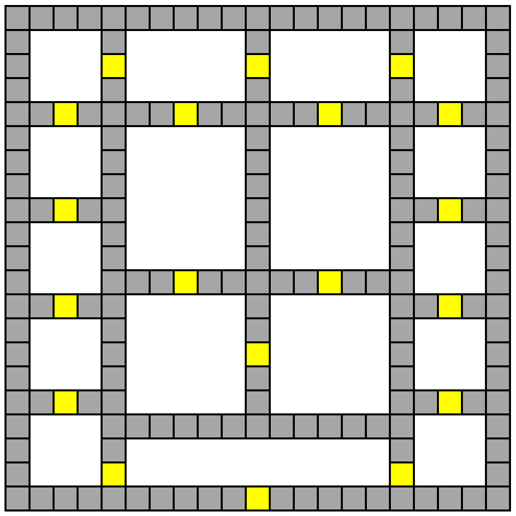
- **freeEverything**(**Building** *build)
    - You guessed it … this procedure should free up all dynamically-allocated memory so that **valgrind** shows no memory leaks nor errors.

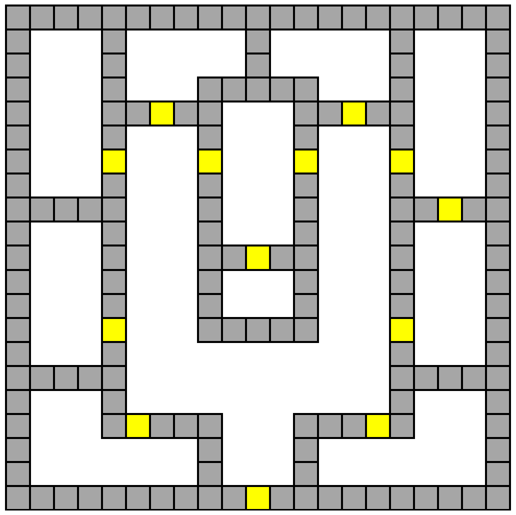
Here are the floors being used in the test program:


Floor 0


Floor 1


Floor 2


Floor 3


Floor 4

Here is the result that you should get from the program if you followed the directions properly:

```
Office (on floor 0) with 52 square feet and 1 doors
Office (on floor 0) with 52 square feet and 1 doors
Office (on floor 0) with 84 square feet and 1 doors
Office (on floor 0) with 48 square feet and 1 doors
Office (on floor 0) with 52 square feet and 1 doors
Office (on floor 0) with 48 square feet and 1 doors
Office (on floor 0) with 60 square feet and 1 doors
Office (on floor 0) with 100 square feet and 1 doors
Office (on floor 1) with 48 square feet and 1 doors
Office (on floor 1) with 48 square feet and 1 doors
Office (on floor 1) with 36 square feet and 1 doors
Office (on floor 4) with 84 square feet and 1 doors
Office (on floor 4) with 52 square feet and 1 doors
Office (on floor 4) with 52 square feet and 1 doors
Office (on floor 4) with 72 square feet and 1 doors
Office (on floor 4) with 80 square feet and 1 doors
Office (on floor 4) with 80 square feet and 1 doors
Meeting Room (on floor 0) with 60 square feet and 2 doors
Meeting Room (on floor 0) with 100 square feet and 2 doors
Meeting Room (on floor 1) with 36 square feet and 2 doors
Meeting Room (on floor 1) with 36 square feet and 2 doors
Meeting Room (on floor 3) with 36 square feet and 2 doors
Meeting Room (on floor 3) with 60 square feet and 3 doors
Meeting Room (on floor 3) with 60 square feet and 3 doors
Meeting Room (on floor 3) with 36 square feet and 2 doors
Meeting Room (on floor 3) with 36 square feet and 2 doors
Meeting Room (on floor 3) with 36 square feet and 2 doors
Meeting Room (on floor 3) with 36 square feet and 2 doors
Meeting Room (on floor 3) with 36 square feet and 2 doors
Meeting Room (on floor 3) with 100 square feet and 2 doors
Meeting Room (on floor 3) with 100 square feet and 2 doors
Meeting Room (on floor 3) with 36 square feet and 2 doors
Meeting Room (on floor 3) with 36 square feet and 2 doors
Meeting Room (on floor 3) with 36 square feet and 2 doors
Meeting Room (on floor 3) with 36 square feet and 2 doors
Meeting Room (on floor 3) with 88 square feet and 3 doors
Meeting Room (on floor 4) with 84 square feet and 2 doors
Meeting Room (on floor 4) with 72 square feet and 3 doors
Meeting Room (on floor 4) with 72 square feet and 2 doors
Lab (on floor 1) with 228 square feet and 2 doors
Lab (on floor 1) with 132 square feet and 3 doors
Lab (on floor 3) with 120 square feet and 2 doors
Lab (on floor 3) with 120 square feet and 2 doors
Storage Room (on floor 0) with 8 square feet and 1 doors
Storage Room (on floor 0) with 16 square feet and 1 doors
Storage Room (on floor 1) with 24 square feet and 2 doors
Storage Room (on floor 1) with 24 square feet and 1 doors
Storage Room (on floor 1) with 24 square feet and 1 doors
Storage Room (on floor 1) with 24 square feet and 1 doors
Storage Room (on floor 1) with 24 square feet and 1 doors
Storage Room (on floor 1) with 24 square feet and 1 doors
Storage Room (on floor 1) with 24 square feet and 1 doors
Storage Room (on floor 2) with 24 square feet and 1 doors
Storage Room (on floor 2) with 24 square feet and 1 doors
Storage Room (on floor 2) with 24 square feet and 1 doors
Storage Room (on floor 2) with 20 square feet and 1 doors
Storage Room (on floor 2) with 24 square feet and 1 doors
Storage Room (on floor 2) with 16 square feet and 1 doors
Storage Room (on floor 2) with 16 square feet and 1 doors
Storage Room (on floor 2) with 24 square feet and 1 doors
Storage Room (on floor 2) with 24 square feet and 1 doors
Storage Room (on floor 2) with 16 square feet and 1 doors
Storage Room (on floor 2) with 16 square feet and 1 doors
Storage Room (on floor 2) with 24 square feet and 1 doors
Storage Room (on floor 2) with 16 square feet and 1 doors
Storage Room (on floor 2) with 16 square feet and 1 doors
Storage Room (on floor 2) with 24 square feet and 1 doors
Storage Room (on floor 2) with 24 square feet and 1 doors
Storage Room (on floor 2) with 16 square feet and 1 doors
Storage Room (on floor 2) with 16 square feet and 1 doors
Storage Room (on floor 2) with 24 square feet and 1 doors
Storage Room (on floor 2) with 24 square feet and 1 doors
Storage Room (on floor 2) with 24 square feet and 1 doors
Storage Room (on floor 4) with 24 square feet and 1 doors
Hallway (on floor 0) with 328 square feet and 16 doors
Hallway (on floor 1) with 272 square feet and 9 doors
Hallway (on floor 2) with 448 square feet and 24 doors
Hallway (on floor 4) with 384 square feet and 11 doors
```

IMPORTANT SUBMISSION INSTRUCTIONS:

Submit all of your **c source code** files as a single **tar** file containing:

1. A **Readme** text file containing
   - your name and studentNumber
   - a list of source files submitted
   - any specific instructions for compiling and/or running your code
2. All of your **.c source** files and all other files needed for testing/running your programs.
3. Any output files required, if there are any.

4. Makefile

The code **MUST** compile and run on the course VM.

- If your internet connection at home is down or does not work, we will not accept this as a reason for handing in an assignment late ... so make sure to submit the assignment WELL BEFORE it is due !

- You WILL lose marks on this assignment if any of your files are missing. So, make sure that you hand in the correct files and version of your assignment. You will also lose marks if your code is not **written neatly with proper indentation and containing a reasonable number of comments.** See course notes for examples of what is proper indentation, writing style and reasonable commenting).
