COMP2401 - Assignment #3

(Due: **Tuesday, February 28th, 2023 @ 11pm**)

In this assignment, you will gain experience in using **structs**, **pointers**, **malloc()** and **free()**. Make sure to put comments in your code … especially ones to describe your functions, definitions and important parts of your code. You must also write your code as cleanly & efficiently as possible.

PART1:

You will write code that simulates people hiring employees from a pool of applicants. Call the program **hiringSimulator.c**. In the program, **Hirers** will try to select **Applicants** and then they will keep some that they like on a selections list, while the ones that they don't like will be put back into the **Pool**. In our simulation, a **Hirer** will be allowed to select at most **8 Applicants** (this should be properly hardcoded by using a defined constant). The **Pool** of applicants itself will have a maximum capacity of **15** (also to be hardcoded using a defined constant). Follow the instructions below.

- Create a **typedef** called **Applicant** to represent an applicant that will either be in a pool or on a hirer's selections list. It should store:
  - gender (a char) (e.g., 'M', 'F', or '?')
  - height (in **feet** as an float)
  - age (in **lbs** as an unsigned char)

- Create a **typedef** called **Hirer** to represent a person who will select applicants from a pool and then keep them in a list. It should store:
  - name (as a char *)
  - desiredAge representing the minimum age of an applicant that this person is willing to select.
  - desiredHeight representing the minimum height of an applicant that this person is willing to select.
  - selections (an array that can hold a maximum of **10** applicants). This should NOT be an array of pointers to applicants, but it should reserve static space to hold all the applicants.
  - numSelected representing the number of applicants currently in the selections list.

- Create a **typedef** called **Pool** to represent a pool with a maximum number of **15** applicants. It should store:
  - applicants (an array that can hold a maximum of **15** applicants). This should NOT be an array of pointers to applicants, but it should reserve static space to hold all the applicants.
  - numApplicants representing the number of applicants currently in the pool.

You will now need to write the following 6 functions. You MUST implement these functions as specified with the exact same parameters and return type. You MUST not alter them.

`int addApplicant(Pool *p, char gender, float height, unsigned char age)`

- This code should add a new applicant to pool **p** with the **gender**, **height** and **age** specified in the parameters. If **p** was already at capacity, then **0** should be returned and the applicant should NOT be added to **p**, otherwise **1** should be returned.

`void listApplicants(Applicant *arrayOfApplicants, int n)`

- This code should list the first **n** applicants of the given **arrayOfApplicants**. Each applicant should be printed on a separate line … showing the age, height and gender of each one in this format: **14 year old 5.33 foot Female** where the gender is either "Male", "Female" or "Unknown".

`char likes(Hirer *h, Applicant *a)`

- This code should return a positive non-zero value if hirer **h** *likes* applicant **a**, otherwise a **0** should be returned. A hirer *likes* an applicant if he/she is at least as tall (i.e., >=) as the hirer's desired height and if the applicant is within 1 year of the hirer's desired age.

`int keep(Hirer *h, Applicant *a)`

- This code should cause hirer **h** to keep applicant **a** by placing him/her on their selection list. However, an applicant should only be kept if the hirer's selection list has NOT reached capacity AND if adding the applicant to the hirer's selection list will not cause the list to have more than 50% (of capacity) of one type of gender in it. The code should return **1** if the applicant is kept and **0** otherwise.

`int selectApplicant(Hirer *h, Pool *p)`

- This code should simulate a hirer delving into pool **p** to look for an applicant. If there are no applicants left in the pool, **0** should be returned … otherwise **1** should be returned at the end of the function. The code should **choose a random applicant** from the pool. If the applicant is then *liked* by the hirer, he/she should attempt to be kept by the hirer (i.e., **keep()** function) and removed from the pool. When removing an applicant from the pool, the last applicant in the pool should be moved/inserted into the position that the applicant was removed from so that there are no gaps in the pool's array.

`void giveAwayApplicants(Hirer *h1, Hirer *h2, Pool *p)`

- This code should cause ALL of hirer **h1**'s applicants to be given to hirer **h2** to keep in their selection list. If **h2** does not *like* any of the applicants or cannot *keep* them for whatever reason, they should be returned to pool **p**. But if the pool capacity has been reached, they applicants should be discarded altogether.

To test your code, you should write a **main()** function that does the following (in order):

1) Create two hirers … one called "Fred" who is willing to keep applicants of age **18** and height **5.6** or more … the other called "Suzy" who is willing to keep applicants of age **17** and height **5.3** or more.

2) Create a pool with 15 applicants by calling your **addApplicant()** function 15 times and then call your **listApplicant()** function to list the applicants in the pool. You should add the following applicants in this order:

```
14 year old 5.33 foot Female
17 year old 6.12 foot Male
15 year old 5.20 foot Female
18 year old 5.82 foot Male
```

```
18 year old 5.39 foot Male
19 year old 5.25 foot Female
16 year old 6.04 foot Male
19 year old 5.96 foot Female
18 year old 5.75 foot Male
16 year old 5.37 foot Female
16 year old 5.28 foot Male
16 year old 5.81 foot ?
18 year old 6.23 foot Female
17 year old 6.49 foot Male
18 year old 5.57 foot Female
```

3) Simulate "Fred" attempting to select an applicant (i.e., call **selectApplicant()**) from the pool **8** times, then list his selected applicants. Simulate "Suzy" attempting to select applicants from the pool **8** times, then list her selected applicants. Note that in both these cases … some of the 8 randomly chosen applicants may not satisfy the conditions to be selected by the hirer. So out of the 8 attempts, sometimes just 2 or 3 will be selected.

4) List all the remaining applicants in the pool again.

5) Have "Fred" give away all his applicants to "Suzy" (i.e., call **giveAwayApplicants()**).

6) List "Fred's" remaining selected applicants, "Suzy's" remaining selected applicants and then the applicants remaining in the pool.

Although the chosen applicants should be random each time, here is a sample of output that you may have:

```
Here are the applicants in the pool:
14 year old 5.33 foot Female
17 year old 6.12 foot Male
15 year old 5.20 foot Female
18 year old 5.82 foot Male
18 year old 5.39 foot Male
19 year old 5.25 foot Female
16 year old 6.04 foot Male
19 year old 5.96 foot Female
18 year old 5.75 foot Male
16 year old 5.37 foot Female
16 year old 5.28 foot Male
16 year old 5.81 foot Unknown
18 year old 6.23 foot Female
17 year old 6.49 foot Male
18 year old 5.57 foot Female

First Fred attempts to select 8 applicants from the pool ...
Fred's selected applicants:
17 year old 6.49 foot Male
18 year old 6.23 foot Female
18 year old 5.75 foot Male
18 year old 5.82 foot Male

Suzy now attempts to select 8 applicants from the pool ...
Suzy's selected applicants:
18 year old 5.39 foot Male
18 year old 5.57 foot Female
16 year old 5.81 foot Unknown
16 year old 5.37 foot Female
17 year old 6.12 foot Male

Here is what is left of the pool ...
14 year old 5.33 foot Female
```

```
16 year old 6.04 foot Male
15 year old 5.20 foot Female
19 year old 5.96 foot Female
16 year old 5.28 foot Male
19 year old 5.25 foot Female

Fred gives his selected applicants to Suzy ...

Fred's selected applicants:

Suzy's selected applicants:
18 year old 5.39 foot Male
18 year old 5.57 foot Female
16 year old 5.81 foot Unknown
16 year old 5.37 foot Female
17 year old 6.12 foot Male
17 year old 6.49 foot Male
18 year old 6.23 foot Female
18 year old 5.75 foot Male

Here are the applicants now in the pool:
14 year old 5.33 foot Female
16 year old 6.04 foot Male
15 year old 5.20 foot Female
19 year old 5.96 foot Female
16 year old 5.28 foot Male
19 year old 5.25 foot Female
18 year old 5.82 foot Male
```

PART 2:

Copy your program to another file called **dynamicVersion.c**. You will edit this code now to make it dynamically allocate the applicants in the arrays (i.e., using **malloc**()), as opposed to statically allocating them.

1) Adjust the code so that the types **Hirer** and **Pool** both keep arrays of pointers to **Applicants** instead of arrays of **Applicants**. The capacity of the arrays should remain the same.

2) Adjust the **addApplicant()** function so that it dynamically allocates the new applicant and adds it to the pool.

3) Adjust the **listApplicants()** function as needed.

4) Adjust the **keep()**, **selectApplicant()** and **giveAwayApplicants()** functions as needed. The code in these functions MUST be simpler now when moving applicants around in the array since you need not copy the size/species values but should just copy over the **Applicant** pointers.

5) Write code in the **main()** function to ensure that your allocated applicants are all freed. Use the following to test to make sure that you freed all allocated memory:

```
valgrind --leak-check=yes ./dynamicVersion
```

IMPORTANT SUBMISSION INSTRUCTIONS:

Submit all of your **c source code** files (DO NOT **TAR** YOUR FILES):

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.

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).