COMP2401 - Assignment #3

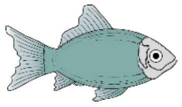
(Due: **Sunday, Oct 20 @ 6pm**)

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:

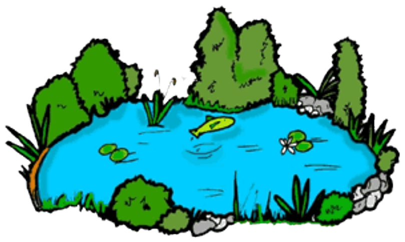
You will write code that simulates people fishing in a pond. Call the program **fishingSimulator.c**. In the program, fishers will try to catch fish and then they will keep some that they like in a bucket, while the ones that they don't like will be put back into the pond. In our simulation, a fisher will be allowed to catch at most **10** fish (this should be properly hardcoded by using a defined constant). The pond itself will have a maximum capacity of **15** fish (also to be hardcoded using a defined constant). Follow the instructions below.

- Create a **typedef** called **Fish** to represent a fish that will either be in a pond or in a fisher's bucket. It should store:
  - size (in **cm** as an unsigned char)
  - species as a char * (e.g., "Sunfish", "Perch", "Bass", "Pike").



- Create a **typedef** called **Fisher** to represent a person who will fish in a pond and then keep some of the fish he/she catches in a bucket. It should store:
  - name (as a char *)
  - keepSize (an unsigned char) representing the minimum size of a fish (in cm) that this person is willing to keep.
  - bucket (an array that can hold a maximum of **10** Fish). This should NOT be an array of pointers to Fish, but it should reserve static space to hold all the Fish.
  - numFish (an unsigned char) representing the number of fish currently in the bucket.

- Create a **typedef** called **Pond** to represent a pond that holds a maximum number of **15** fish. It should store:
  - fish (an array that can hold a maximum of **15** Fish). This should NOT be an array of pointers to Fish, but it should reserve static space to hold all the Fish.
  - numFish (an unsigned char) representing the number of fish currently in the pond.



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  addFish(Pond *p, unsigned char size, char *species)`
  - This code should add a new fish to pond **p** with the **size** and **species** specified in the parameters. If **p** was already at capacity, then 0 should be returned and the fish should NOT be added to **p**. Otherwise 1 should be returned.

- `void listFish(Fish *arrayOfFish, int n)`
  - This code should list the first **n** fish of the given **arrayOfFish**. Each fish should be printed on a separate line … showing the size (in cm) and species of each one.

- `char likes(Fisher *fisher, Fish *f)`
  - This code should return a positive non-zero value if **fisher** *likes* fish **f**, otherwise a **0** should be returned. A fisher *likes* a fish if it is at least as big (i.e., >=) as the fisher's preferred **keepSize** and if the fish is NOT a "Sunfish".

- `int keep(Fisher *fisher, Fish *f)`
  - This code should cause **fisher** to keep fish **f** by placing it in his/her bucket. However, a fish should only be kept if the fisher's bucket has NOT reached capacity. The code should return **1** if the fish is kept and **0** otherwise.

- `int  castLine(Fisher *fisher, Pond *p)`
  - This code should simulate a **fisher** casting his/her line into pond **p** to try and catch a fish. If there are no fish left in the pond, **0** should be returned … otherwise **1** should be returned at the end of the function. The code should choose a random fish. If the fish is then *liked* by the fisher, it should be kept in his/her bucket (if not past the limit) and removed from the pond. When removing a fish from the pond, the last fish in the pond should be moved/inserted into the position that the fish was removed from so that there are no gaps.

- `void giveAwayFish(Fisher *fisher1, Fisher *fisher2, Pond *p)`
  - This code should cause all of **fisher1**'s fish to be given to **fisher2** to keep (if not past the limit). If **fisher2** does not *like* any of the fish or cannot keep them, they should be returned to pond **p**.

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

1) Create two fishers … one called Fred who is willing to keep fish of size 15 cm or more … the other called Suzy who is willing to keep fish of size 10 cm or more.

2) Create a pond with 15 fish by calling your **addFish()** function 15 times and then call your **listFish()** function to list the fish in the pond. You should add these fish in this order:

```
 4 cm Sunfish
25 cm Pike
20 cm Bass
30 cm Perch
14 cm Sunfish
15 cm Pike
 9 cm Pike
12 cm Bass
 5 cm Sunfish
12 cm Sunfish
10 cm Bass
 2 cm Bass
16 cm Perch
30 cm Sunfish
 7 cm Perch
```

3) Simulate Fred casting his fishing line (i.e., call **castLine()**) in the pond 10 times, then list the fish in his bucket. Simulate Suzy casting her fishing line in the pond 10 times, then listing the fish in her bucket.
4) List all the remaining fish in the pond again.
5) Have Suzy give away all her fish to Fred (i.e., call **giveAwayFish()**).
6) List Fred's remaining fish in his bucket, Suzy's remaining fish in her bucket and then the fish remaining in the Pond.

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

```
Here are the fish in the pond:
A 4 cm Sunfish
A 25 cm Pike
A 20 cm Bass
A 30 cm Perch
A 14 cm Sunfish
A 15 cm Pike
A 9 cm Pike
A 12 cm Bass
A 5 cm Sunfish
A 12 cm Sunfish
A 10 cm Bass
A 2 cm Bass
A 16 cm Perch
A 30 cm Sunfish
A 7 cm Perch

First Fred attempts to catch 10 fish in the big pond ...
Fred's fish:
A 16 cm Perch
A 25 cm Pike

Suzy now attempts to catch 10 fish in the big pond ...
Suzy's fish:
A 12 cm Bass
A 20 cm Bass

Here is what is left of the pond ...
A 4 cm Sunfish
A 30 cm Sunfish
A 2 cm Bass
A 30 cm Perch
A 14 cm Sunfish
A 15 cm Pike
A 9 cm Pike
A 7 cm Perch
A 5 cm Sunfish
A 12 cm Sunfish
A 10 cm Bass

Fred's fish:
A 16 cm Perch
A 25 cm Pike
A 20 cm Bass

Suzy's fish:

Here are the fish now in the pond:
A 4 cm Sunfish
A 30 cm Sunfish
A 2 cm Bass
```

```
A 30 cm Perch
A 14 cm Sunfish
A 15 cm Pike
A 9 cm Pike
A 7 cm Perch
A 5 cm Sunfish
A 12 cm Sunfish
A 10 cm Bass
A 12 cm Bass
```

PART 2:

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

1) Adjust the code so that the types **Fisher** and **Pond** both keep arrays of pointers to **Fish** instead of arrays of **Fish**.

2) Adjust the **addFish()** function so that it dynamically allocates the new fish and adds it to the pond.

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

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

5) Write code to ensure that your allocated fish 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 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.

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).

- Once you submitted the assignment test that the submitted tar file contains the correct files by: 1. creating an empty directory; 2. Downloading the tar file from culearn; 3. Extract the files from the tar file; and 4. Compile and test the submitted code files.