CS/240/Extra/Credit/Assignment

Objective

The goal of this assignment is for you to learn basic concepts of function pointers in structures. Recall that a structure is a collection of variables that are possibly of different types. One key difference between classes in Java and structs in C is that structs don't contain member functions. But with function pointers as its members, we can simulate some basic features of classes with a struct. In this assignment you are going to do the following:

- Declare functions which can be referenced in a struct through pointers to the functions. These function pointers provide access to the struct's *member functions*.
- Define constructor functions that allocate memory for new *instances* of the struct, and initialize the fields of the new structs including the member function pointers.
- Interact with the created struct instances through a number of user commands.

Building Blocks

The struct you are going to use is an abstract class called *sea_animal* representing a *generic* sea animal. The class defines some characteristics of any sea animal and is defined as follows:

```
struct sea_animal {
  char name[20];
  int age;
  double weight;
  void (*eat)(struct sea_animal*);
  void (*swim)(struct sea_animal*, int);
};
```

Notice that the generic sea animal has a name, age, and weight; it can also perform two activities called "eat" and "swim." However, specific kinds of sea animal will eat and swim differently. To support these different behaviors, we will "specialize" the generic eat and swim functions by having them point to different actual implementations of the functions according to the sea animal type. In this assignment, you will deal with the sea animal types of whales and sharks. The eat and swim functions of a whale are defined as:

```
void whale_eat(struct sea_animal* this);
void whale_swim(struct sea_animal* this, int hours);
```

For sharks, the corresponding declarations are:

```
void shark_eat(struct sea_animal* this);
void shark_swim(struct sea_animal* this, int hours);
```

For the eat functions, we assume how much a shark or a whale eats is determined by its weight. A shark eats food that is 10 percent of its weight at a time, while a whale eats food that is 5 percent of its weight. After each meal, a shark's weight increases by 1 percent but never exceeds 2,000 pounds. Similarly, a whale's weight increases by 5 percent after each meal and the maximum weight is 400,000 pounds. The swimming speed is a linear function of the age and weight. A shark swims at 10 * age * (1000 / weight) miles per hour and a whale swims at 3 * age * (200000 / weight) miles per hour. Obviously, these functions are oversimplifications, but they give the idea that as animals eat, their weights increase and as the weights increase, they swim slower.

Also, notice that member functions (e.g., eat in our case) may change the state variable (e.g., weight) of the struct. In general, therefore, a member function needs access to the calling struct variable for the state update. For example, the eat function will need to know the whale/shark that is eating. For Java, member functions have access to the built-in *this* variable for this purpose. For C, we need to explicitly pass a pointer to the calling struct as a parameter of the member function. That's why all the eat/swim functions defined above have a `struct sea_animal *` defined as their first parameter. That way, if you want to set the calling struct's age to be 2, you will simply write `this->age = 2` in the member function.

Finally, before the user can manipulate an object represented as a struct, the program will have to first create the struct. This is similar to *constructor methods* in Java, and you will also have to implement similar constructor functions for your whale/shark. These constructors functions do not need to be defined inside the structs. Instead, standalone functions can be used, and they should be defined as follows:

```
struct sea_animal* whale_new(char *name, int age, double weight);
struct sea_animal* shark_new(char *name, int age, double weight);
```

Both functions take 3 parameters and carry out the following tasks:

1. Allocate memory for the sea_animal struct to be created.
2. Initialize the name, age, and weight fields of the new struct to be the three input parameters, in that order.
3. Make the eat/swim member function pointers point to the correct functions.
4. Return the address of the newly created struct if the call was successful, otherwise return NULL.

Problem and Hints

You will write a program that a user can use to create and interact with a virtual pet that is a sea animal (whale or shark). According to the user input, your program creates a new whale or a new shark and sets the pet's name, age, and weight. The user can then either feed his/her pet (through the eat function) or "walk" the pet (through the swim function) multiple times until he/she quits the program.

An example execution scenario is as follows:

```
Create your new pet, whale or shark? (w/s): s
Pet's name: Ginger
Age: 1
Weight: 300
Pet created!
Have fun with your new pet Ginger:
1. Feed
2. Walk
3. Play later

Input: 1
```

```
Ginger happily ate 30.00 pounds of food.  Its weight is now 303.00 pounds.

Have fun with your new pet Ginger:
1. Feed
2. Walk
3. Play later

Input: 2
For how many hours?: 1
Ginger happily swam 33.00 miles.

Have fun with your new pet Ginger:
1. Feed
2. Walk
3. Play later

Input: 1
Ginger happily ate 30.30 pounds of food. Its weight is now 306.03 pounds.

Have fun with your new pet Ginger:
1. Feed
2. Walk
3. Play later

Input: 3
Quit.
```

The age, weight, and hour input are integers. Certainly the input should be within some valid ranges, but you don't have to worry about that in your program. You can just assume that they are correct.

Except for the constructor functions (`shark_new` and `whale_new`) at the beginning, you should write only *one* copy of code for both sharks and whales for the remaining interactions with the user. That means, in order to feed or walk the pet, you *must* use the struct's function pointers to call the correct shark/whale functions, but not call those functions directly. In particular, *explicit use of if (or other control) statements to determine which version (shark vs. whale) of the eat/swim functions to call is not allowed in this assignment.*

For this assignment, you are allowed to use `getchar`, `putchar`, `printf`, and the C library functions given below. You are not allowed to use any other C library function. To read a string, you can simply use the `void readline(char c[])` function given below. To read an integer or floating point number from the keyboard, you can first call the `void readline(char c[])` function, and then use the standard library function `int atoi(char c[])` or double `atof(char c[])` to convert a string to an integer or a double. For how to use a library function, use the Unix command man (e.g., `man atoi`).

```
void readline(char c[]) // read a line of text from stdin to char array c
{
    int i=0;
    while((c[i++]=getchar())!='\n');
    c[i-1]='\0';
}
```

Turning in

For your submission, pack your files as follows:

```
tar zcf turnin.tgz Makefile src1.c  ...
```

This assignment will be accepted until Wednesday, May 2nd before midnight.

The autograder will be set up to accept submissions, but will not attempt to grade them. All assignments will be graded by hand.