CSE333 Exercise 3

Out: Wednesday, January 12
Due: ~~Monday, January 17~~ Tuesday, January 18 by **11 am PST**
Rating: 2 (note)

Goals

- Define and use structs with `typedef`
- Use `malloc` and `free` to manage dynamically-allocated data without memory leaks
- Write some basic automated tests in C
- Modularize a C program across multiple files

Problem Description

Your job is to write a C program that does the following:

- Uses `typedef` to define a new structured type called `Point3d`, which contains `int32_t` fields for x, y, and z coordinates.
  - This means you should be able to refer to the struct with the following syntax: `Point3d p = ...;`
  - For the sake of the autograder, do *not* use `typedef` to define pointers to `Point3d` structs to be `Point3dPtr`; instead, use `Point3d*`.
- Defines a function called `Point3d_Allocate` that (1) accepts three `int32_t` arguments, (2) mallocs space for a `Point3d`, (3) assigns the three arguments to the x, y, and z fields, and (4) returns (a pointer to) the `malloc`'ed `Point3d`.
- Defines a function called `Point3d_Scale` that (1) accepts one `Point3d*` and one `int32_t` value as arguments, (2) scales the x, y, and z fields of the pointed-to struct by the given value, and (3) returns nothing.
  - For the sake of the autograder, do *not* return a `bool`, even though it would a reasonable design decision to do so.
- Defines a function called `Point3d_GetOrigin` that (1) accepts no arguments, (2) constructs a `Point3d` with x, y, and z equal to zero, and (3) returns a *copy* of the struct.
- Has a `main` function that runs at least one *automated test* on each of your functions.
- Is modularized nicely across three files: `Point3d.h`, `Point3d.c`, and `ex3.c`.
  - `Point3d.h` should contain the struct definition and function declarations related to `Point3d`.
  - `Point3d.c` should contain the function definitions.
  - `ex3.c` should contain the `main` that tests your functions.

Implementation Notes

Error Handling & Robustness

As part of this exercise, you will have to make design decisions about how to handle errors. Be sure to take some time and reason about what arguments might be problematic for each function. Be sure to handle other types of errors, like the case in which malloc fails. Whatever decisions you make, be sure to document them appropriately.

- You may ignore any possible integer overflow errors.

Memory Cleanup

Make sure your `main` frees any memory that was dynamically allocated, even when handling edge cases. We will be using valgrind (`valgrind --leak-check=full ./ex3`) to check for memory leaks and other memory issues.

Testing

For the purposes of this exercise, it is enough for you to write code that makes use of all of the functions that you wrote and verifies that the struct members contain the correct values at each step. When an error is encountered during testing, you should immediately return an appropriate status code to indicate failure in addition to printing an error message. This signals an error to the parent process (this is what makes it *automated*) and is easier to detect than having to manually scan over your program output, especially if you have a lot of tests.

- `main` is considered testing code for your `Point3d` module. Testing code is the only place where the use of `assert()` is allowed, though you do not need to use it at all if you don't want to.

Style Focus

Don't forget that all of the good practices from previous exercises still apply!

General

For the sake of our autograder, make sure that your function and type names match the specifications *exactly*, including capitalization. You should write comments explaining the behaviour and purpose of the struct and functions you define. Be sure that the comments you write document how errors are handled.

Typedef

This is supposed to make your life easier! Make sure that your syntax is correct and that you're taking advantage of the new alias.

Multiple files

Make sure to include header guards, where appropriate, and that you place the function block comments in the appropriate place.

Submission

You will submit: `ex3.c` , `Point3d.h` , and `Point3d.c` .

Your code must:

- Compile without errors or warnings on CSE Linux machines (lab workstations, `attu` , or CSE home VM).
- Have no runtime errors, memory leaks, or memory errors (`gcc` and `valgrind`).
- Be contained in the three files described above that compile into an executable with the following commands:

```
bash$ gcc -Wall -g -std=c17 -c -o Point3d.o Point3d.c
bash$ gcc -Wall -g -std=c17 -c -o ex3.o ex3.c
bash$ gcc -Wall -g -std=c17 -o ex3 ex3.o Point3d.o
```

- Have a comment at the top of your `.c` file with your name(s) and CSE or UW email address(es).
- Be pretty: the formatting, modularization, variable and function names, commenting, and so on should be consistent with class style guidelines. Additionally, the linter shouldn't have any complaints about your code (`clint.py`).
- Be robust: your code should deal with hard-to-handle/edge cases and bogus user input (if there are any) gracefully.

Submit your code on Gradescope (https://www.gradescope.com). Don't forget to add your partner if you have one.