CS 240: Programming in C
Fall 2020
Homework 4

Prof. Turkstra

Due September 30, 2020 9:00 PM

1 Goals

The purpose of this homework is to give you experience in using `typedef` and structure declarations as well as defining variables of those types. We'll also do some computations using the mathematical library.

2 Getting Started

1. Create your Homework 4 directory by using git to clone your repository from the course account.
2. For this assignment, we will provide you with a program that generates and displays the Mandelbrot set. You can use this to get some idea of how on track you are. In addition to the usual Makefile targets such as hw4_main and hw4_test, we provide a hw4_view target:

   ```
   $ make hw4_view
   ```

   This creates the Mandelbrot viewer executable, which you can then run.

3 Special Rule

You are not permitted to include `complex.h` nor rely on any built-in support for complex math provided by the C language for this assignment.

4 The Big Idea

Structure declarations are a means of expressing, in your program, an intent to collect a group of data into one name. Usually, this is done when the data that you want to represent in your program has a complex representation. For instance, if you wanted to represent information about thousands of computers, you would want to be able to refer to a single computer using one name. After that, you might want to do something with its specific sub-data. E.g., how many hard drives does it have? How many CPUs does it have? How much memory does it have?

For this homework, we're going to create a structure that will be used to represent a complex number. This structure will have two elements named x and y that are both double precision floating point numbers.

A complex number then takes the form of $x + y * i$, where i is the square root of -1.

To make life easier, we can think of x and y as being values on the coordinate plane.

We'll call this structure `struct complex` and we will create a `typedef` for it called `complex_t`. Thereafter, you can simply use the type `complex_t` to describe the type. Examine "hw4.h" carefully to see how this is done.

5 The Assignment

You are to write a series of functions that deal with arguments of type `struct complex`.

5.1 Functions You Will Write

You will write the following functions:

`complex_t add_complex(complex_t, complex_t);`

Add the corresponding fields of two complex variables together and return a complex value that represents the sum. For instance, if `add_complex()` is invoked with the values $(1.0, 2.0)$ and $(3.0, 4.0)$, it will return the value $(4.0, 6.0)$.

`complex_t neg_complex(complex_t);`

Returns the negative form of the argument. E.g., -1.0 times the value of each field.

`complex_t sub_complex(complex_t, complex_t);`

`sub_complex()` is similar to `add_complex()` except that it subtracts the second argument from the first. You should implement this function with something like:
`add_complex(arg1, neg_complex(arg2))`

`double dot_complex(complex_t, complex_t);`

Return the dot product (aka the scalar product or inner product) of the complex arguments. This is the sum of the products of the corresponding terms. E.g. the dot product of $(2, 3)$ and $(4, 5)$ is 23.

`complex_t inv_complex(complex_t);`

Returns the reciprocal of the argument. I.e., $\frac{1}{argument}$. The complex reciprocal is determined by dividing each of its components by the square of the magnitude of the complex number and negating the imaginary part.

The magnitude (aka absolute value or modulus) of a complex number is determined by taking the square root of the sum of the square of each term.

E.g. the reciprocal of $(1, 2)$ is $(0.2, -0.4)$ and the reciprocal of $(2, 4)$ is $(0.1, -0.2)$

`complex_t mul_complex(complex_t, complex_t);`

Return the complex product of the two complex arguments. Given two complex numbers: $x_1 + y_1 * i$ and $x_2 + y_2 * i$, the complex product is given by: $(x_1 * x_2 y_1 * y_2) + (x_1 * y_2 + y_1 * x_2) * i$ So, for example, multiplying $(2, 4)$ by $(3, 5)$ would result in: $(-14, 22)$.

`complex_t div_complex(complex_t, complex_t);`

Return the complex division of the first argument by the second. Implement this by using the two functions above. I.e., `mul_complex(arg1, inv_complex(arg2))`

`complex_t exp_complex(complex_t);`

Compute the complex exponential function e^n where the argument is complex. Consider the equation:

$$e^{iy} = \cos(y) + i \sin(y)$$

Computing e^n is not much harder for complex numbers. E.g.:

$$e^{x+i*y} = e^x e^{iy} = e^x(\cos(y) + i \sin(y))$$

So, for instance, $(2, 4)$ yields approximately: $(-4.83, -5.59)$

It is worth noting that C's math library includes $\sin(), \cos(), and \exp()$ functions.

`int mandelbrot(complex_t);`

This function should initialize a temporary variable call it Z—to zero and then compute the operation:

$$Z = Z^2 + C$$

Over and over until the absolute magnitude of Z is larger than 2.0. Here, C is the argument that was passed to the function. The `mandelbrot()` function should return the number of re-calculations for Z that it took to make its magnitude greater than 2.0.

Note that checking that the absolute magnitude of Z is less than 2 is the same thing as checking that the dot product of Z with itself is less than 4.0. This is because

$$|Z| = \sqrt{Z \cdot Z}$$

The `mandelbrot()` function is useful for generating the Mandelbrot set. What! You've never heard of the Mandelbrot Set!

`https://www.google.com/search?q=mandelbrot+set`

Note that some values for C will cause the Mandelbrot set calculation to iterate forever (or, at least, for a very long time). We've provided a constant for you named `MAX_MANDELBROT` that is the largest value you should ever return from the `mandelbrot()` function. You should use that as a stopping point if the calculation continues too long.

5.2 Input Files

We'll take a vacation from input files for this homework.

5.3 Header Files

We provide a header file, "hw4.h", for you. It contains prototypes for each of the functions that you will write as well as #definitions for the constants. You should not alter this file. We will replace it with the original when grading.

5.4 Error Codes

There are no error codes for any of the functions in this assignment. (Some of you might be curious what happens when you take the inverse of $(0, 0)$. You'll just have to try it yourself. :-)

Hints

- Note that for hw4.c you don't need to say `#include <stdio.h>`, do you?
- We have included a fun program that leverages your `mandelbrot()` function called `hw4_view`. This program allows you to visualize the Mandelbrot set. You must run it locally on a CS Linux system, or enable X forwarding for your SSH client.
  - Linux users can use "`ssh -Y`".
  - Windows users will have to download an X server like Xming and configure PuTTY accordingly...
    `https://sourceforge.net/projects/xming/`
  - Mac users should download and install XQuartz.

These Standard Rules Apply

- You may add any `#include`s you need to the top of your hw4.c file;
- You may not create any global variables other than those that are provided for you. Creation of additional global variables will impact your style grade;
- Do not look at anyone else's source code. Do not work with any other students.

Submission

You can submit your solution as often as you wish. We encourage you to submit your code as often as possible. Only your final submission will be graded.

6 Grading

The operation of your functions will be graded out of 100 points. The point breakdown will be determined by the test program.

The test program will be run many times when grading. It is your job to do the same. The lowest score will be your final grade.

This homework will also have a style grade based on 20 points, with 2 point deducted for each code standard violation found.

Your code must compile successfully using `-Wall -Werror -std=c99` to receive any credit. Code that does not compile will be assigned an automatic score of 0.

If your program crashes (e.g., segmentation fault) at any point during the testing process, your score will also be an automatic 0.