EGRE 246 Advanced Engineering Programming
Homework #2 – Structures and Functions

This homework must be your own (individual) work as defined in the course syllabus and discussed in class.

1) Write a set of C functions that manipulate a representation of complex numbers. A complex number is a representation of a number line on a 2-dimensional plane. The horizontal axis on the plane is the real part of the number and the vertical axis on the plane is the imaginary part of the number. Complex numbers can be represented as the real (*re*) and imaginary (*im*) parts in a Cartesian coordinate system:

$$z = re + im\, i$$

Or they can be represented as a magnitude and an angle in a polar coordinate system:

$$z = mag \angle ang$$

In this assignment, you will use a C struct to hold the complex numbers. The struct has three members, two doubles, to hold either **re** and **im** or **mag** and **ang**, and a single character **coord** that specifies the coordinate system for the individual complex number. This struct **must** be declared in a *hw2.h* file, along with some useful definitions, as shown below:

```
// coordinate system representations
#define RECT       'r'
#define POLAR      'p'

// short-hand for real and imaginary parts
#define RE                  p1
#define IM                  p2

// short-hand for magnitude and angle parts
#define MAG                 p1
#define ANG                 p2

struct complex {
         double p1;  // complex "part1" - either the real part (Cartesian) or mag (polar)
         double p2;  // complex "part2" - either the imaginary part or angle (degrees)
         char coord; // which coordinate system this complex number is in
};
```

You must write a set of functions that manipulate the *complex* struct to enter or read new complex numbers, print the value of complex numbers, convert from one coordinate system to another, and perform various arithmetic functions on complex numbers.

The function prototypes must be contained in a *hw2_functions.h* file as shown below:

```
void enter_point(struct complex *point);
void print_point(struct complex point);
void convert_point(struct complex *point);
int read_point(FILE *infile, struct complex *point);

double real(struct complex point);
double imag(struct complex point);
double mag(struct complex point);
double ang(struct complex point);

void add(struct complex *result, struct complex a, struct complex b);
void sub(struct complex *result, struct complex a, struct complex b);
void mult(struct complex *result, struct complex a, struct complex b);
void div(struct complex *result, struct complex a, struct complex b);
```

The `enter_point()` function should allow the user to enter the values for a complex point in either rectangular or polar coordinates. Calls to the `enter_point()` function should result in the following (example) at the command line:

```
C:\Temp\egre245\c_code\hw_solutions>hw2
Enter complex number.
Choose Rectangular (r) or Polar (p) coordinates:r
Enter real and imaginary values
Real value:3
Imaginary value:4
Enter complex number.
Choose Rectangular (r) or Polar (p) coordinates:p
Enter magnitude and angle values
Magnitude value:4
Angle value (degrees):45
```

The `print_point()` function should print out a complex number. Calls to `print_point()` with the two complex numbers entered above would look like this:

```
z = 3.0000 + 4.0000i
z = 4.0000 /_ 45.00
```

The `convert_point()` function takes a pointer to a complex structure and converts the values in it from rectangular to polar or polar to rectangular, as appropriate.

The `read_point()` function takes a pointer to a file that has been opened for reading and a pointer to complex structure and reads in the values for the complex number from the file. The result of the `fscanf()` call used to read the values from the file are returned to the caller of the `read_point()` function so that it can be used to detect errors and the end of the input file. The values in the file are arranged on a single line for an individual complex number and include the **re** or **mag** part, the **im** or **ang** part, and the **coord** coordinate system specification.

Thus, the entries in the file corresponding to the two points entered in the example above would look like this in the file:

```
3.0000          4.0000      r
4.0000          45.00       p
```

The `real()`, `imag()`, `mag()`, and `ang()` functions take a complex point and return the real, imaginary, magnitude, or angle (degrees) of the complex number. Each of the functions will work correctly regardless of the coordinate system of the point sent to them as an argument.

The `add()`, `sub()`, `mult()`, and `div()` functions perform the appropriate mathematical functions on the pair of complex numbers sent to them in the last two arguments. The result of the operation is stored in the first argument, which is sent to the function as a pointer. These functions all work correctly regardless of the coordinate systems of either of the arguments sent to them. The coordinate system passed to the function in first argument where the result is stored is used to determine the coordinate system of the result. If the coordinate system in the first argument is not set, then the result is stored in Cartesian coordinates. Note that it is possible, and permissible, to use the simpler functions to implement the more complex functions.

In addition to the functions, you must write a `main()` program to thoroughly test **all** of your complex number functions. You should create an input file that has the proper examples in it to test all of the functions and you should also construct several examples that can be used to test the `enter_point()` function as well.

For this assignment, you must turn in a zip file that contains the following files:

hw2.h – header file that contains the structure definition and the other define statements shown above

hw2_functions.h – header file that contains the function declarations shown above

hw2_functions.c – c source code file that contains the implementation of all of the functions declared in hw2_functions.h

hw2.c – c source code file that contains the implementation of your main() program that tests all of the functions

Makefile – a working makefile for the GNU tools that compiles and links your complete homework solution

test.txt – input text file that you constructed to test your read_point() function reading complex structure values from a file

Turn in your assignment by attaching the zip file to the assignment submission page.

Remember the class policy on late submissions – no late submissions are allowed unless prior arrangement is made with the instructor.