Enel 487 – Fall 2015 – Assign 1

A Complex Number Calculator
Handed Out: 2015-9-17
Due: 2015-9-29 by 23h59

Write a C or C++ program (in either case your program must *compile under g++-4.8 or higher*) which performs complex number arithmetic. The program is to parse a line of user console input and perform the addition, subtraction, multiplication or division of two complex numbers. The first component in the line is either `A`, `S`, `M`, or `D` which indicates which operation (addition, subtraction, multiplication or division) to perform. Lowercase characters must also work. Following the operation should be two pairs of floating point numbers, each pair representing a complex number in rectangular form. In addition the character `Q` or `q` should quit the program. The following example causes the complex numbers (45.67,-170) and (9.2,15) to be added together and the result printed.

```
A 45.67 -17e1 9.2 15
```

The calculator prints out the answer in the form

```
real + j imag
```

In the above example the output will be:

```
54.87 - j 155
```

Note that your program must support standard exponential notation, like `6.022e23`. Luckily, the standard library functions will parse doubles or floats in all standard notations, so you *don't have to*.

If the imaginary part is negative then the `+` sign should be replaced with `-`. If you feel comfortable using the standard C library, consider using the `fgets()` function to read the input as a string in one fell swoop, `sscanf()` to parse it, and the `fprintf()` function to print the output. Similar techniques exist in the standard C++ library, see `cplusplus.com` for a good reference.

- You must define your own `Complex` data type. You may be aware that the standard C and C++ libraries both provide implementations of complex numbers and arithmetic. *Don't use these!* Use four separate subroutines, which perform the 4 complex arithmetic functions. For example a prototype of the add routine might be:

```
void add(Complex z1, Complex z2, Complex* result);
```

I want your program to work in two different ways:

1. Interactive mode
2. Batch mode

In both modes, your program reads from the standard input and writes to the standard output. The difference lies in how the input is provided; whether the input is redirected from a file or not. By default if you run the program at the command line with no arguments it will use the keyboard as the standard input and the screen as the standard output. Here's an example of interactive behaviour, where the stuff typed interactively is in red, and the standard output is in blue. Note that extra stuff like the prompt and the startup help messsage go to `stderr`.

```
$ ./assign1
Complex calculator

 Type a letter to specify the arithmetic operator (A, S, M, D)
 followed by two complex numbers expressed as pairs of doubles.
 Type Q to quit.

Enter exp: m   -5.90  -7.35    -4.22   0.385
27.7278 + j 28.7455
Enter exp: a 9.2 -7.9 1.5 -8.5
10.7 - j 16.4
Enter exp: d -7.3 1.5 -7.8 8.2
0.5406 + j 0.376015
Enter exp: d 7.2 3.3 9.8 -2.6
0.602918 + j 0.496693
Enter exp: q
$
```

The input data can instead be typed into a text file (named, say, `data.txt`)

```
m   -5.90  -7.35    -4.22   0.385
a 9.2 -7.9 1.5 -8.5
d -7.3 1.5 -7.8 8.2
d 7.2 3.3 9.8 -2.6
q
```

Then the following command:

```
./assign1 < data.txt > output.txt
```

will read from `data.txt` and write to `output.txt`.

After this the contents of `output.txt` will be

```
27.7278 + j 28.7455
10.7 - j 16.4
0.5406 + j 0.376015
0.602918 + j 0.496693
```

Here's another larger example of set of input commands that could be read from a file:

```
A -0.474 0.367 -0.432 0.229
S 0.425 0.477 0.174 0.082
M 0.133 0.155 0.195 -0.498
D 0.397 -0.490 0.043 0.355

A -13.7 -6.86 4.7 3.14
S -6.37 4.42 4.85 1.51
M -3.33 4.91 -1.01 -1.71
D 3.17 -1.76 3.21 -3.31

A 5.88e-11 -2.57e-10 3.37e-10 -2.54e-10
S 1.11e-10 -1.45e-10 5.86e-11 -3.36e-10
M 4.19e-10 2.51e-10 3.14e-10 -4.53e-10
D 3.19e-10 3.67e-10 7.12e-11 -1.97e-10

A 0.313 -0.152 0 0
S 0.463 -0.150 0 0
M -0.224 0.234 0 0
D 0.430 0.339 0 0

A 0 0 0.270 -0.244
S 0 0 -0.284 -0.424
M 0 0 0.073 0.169
D 0 0 0.466 -0.148

A 0 0 0 0
S 0 0 0 0
M 0 0 0 0
D 0 0 0 0
```

```
q

```

And the output sent to a file should be:

```
-0.906 + j 0.596
0.251 + j 0.395
0.103125 - j 0.036009
-1.22682 - j 1.26691

-9 - j 3.72
-11.22 + j 2.91
11.7594 + j 0.7352
0.752641 + j 0.227801

3.958e-10 - j 5.11e-10
5.24e-11 + j 1.91e-10
2.45269e-19 - j 1.10993e-19
-1.13008 + j 2.02772

0.313 - j 0.152
0.463 - j 0.15
-0 + j 0
nan + j nan

0.27 - j 0.244
0.284 + j 0.424
0 + j 0
0 + j 0

0 + j 0
0 + j 0
0 + j 0
nan + j nan
```

Notice that if you divide by zero the output may be `nan` (Not a Number) or $\pm\infty$. It depends on how you implement the division routine.

- All required output should be sent to stdout, while all error messages, user prompts, etc. should be sent to stderr.
- Write the program so it can be compiled under g++, *even if your program is pure C*. Of course any compliant C++ compiler should give exactly the same behaviour.

- Include a `readme.txt` file that briefly discusses your program, its use, its limitations, how it handles erroneous input, *etc.* Also provide information on how your program is to be built. For example if you use make, include the `Makefile`.
- When I mark your assignments, I'll be using a python test script (included with the handout) to run your executable, supply test input and check the output. So your program input and output HAS to follow the specified format. If it can't work with my test program then you will lose significant marks.

Submit a link to your `git` repository for your solution.
Note: Late assignments will be rejected and given a grade of zero.