University of Pittsburgh
Department of Electrical and Computer Engineering ECE 0301 – Spring 2021

In-Class Assignment #3

Programming concepts:

- Branching instructions: `if/else`, `switch/case`
- Input validation
- Numerical calculations with functions from the math library

ECE concepts:

- Vectors in $\mathbb{R}^2$ and complex numbers in $\mathbb{C}$
- Cartesian and polar coordinates
- Arithmetic with vectors and complex numbers

Develop software according to the following specifications and submit whatever portion you have completed to the class repository before midnight tonight. **For all in-class assignments, you must include line-by-line comments to explain what your does and why! You must also choose meaningful names for all variables so that it will be easy for a reader to understand your code.**

1. Write a computer program that will display the following introductory message to standard output:

   ```
   ECE 0301 – Vectors in R2 and Complex Numbers
   Please enter two numbers, separated by a space,
   that will represent a vector or a complex number.
   ```

   Don't forget to include comments! Type them in your code as you complete each item.

2. In step 1, you gave a prompt for the user to enter two numbers. Use the `cin` stream to read the user response from standard input, and store the values in double-precision floating-point variables. Print a line of text to standard output to report the values back to the user. Use the `setprecision` and `fixed` manipulators, so that the values are displayed in fixed-point notation, with three digits to the right of the decimal point. For example, for the case when the user enters 1.5 and 2, the correct output is:

   ```
   You entered 1.500 and 2.000.
   ```

   No input validation will be required here. If the user input does not conform to your method of reading it, such as when the user enters something that is not a number, you are not required to detect this or exit the program. Test your program with valid and invalid data. What values are read from standard input if the user enters non-numeric data?

3. The numbers entered by the user can be used to represent a vector or a complex number in Cartesian coordinates, i.e. as x and y to represent the vector $v = (x, y) \in \mathbb{R}^2$ or the complex number $z = x + jy \in \mathbb{C}$. These two numbers can also be viewed as the representation of a vector or complex number in *polar coordinates*, i.e. R and θ to represent a vector in $\mathbb{R}^2$ at distance R from the origin and angle θ from the positive x-axis, measured counterclockwise, or the complex number $z = Re^{j\theta} \in \mathbb{C}$.

   Prompt the user to indicate whether they have entered their numbers as Cartesian or polar coordinates, and store the response in a variable from the `string` class. Instruct the user to enter a single letter to represent their choice: `c` or `p`. If the user response is not one of {`c`, `C`, `p`, `P`}, or if the user enters more than one character, then the program should print the following error message and return from `main()` with a value of -1:

```
ERROR! Invalid selection, exiting.
```

   Test your program with valid and invalid selections to make sure this feature works correctly.

4. If the user enters one of {`c`, `C`, `p`, `P`}, then the program should declare new variables to store the coordinate representation that the user *did not* choose, and then compute values for this representation. So for example, if the user entered Cartesian coordinates, the program should compute polar coordinates, and vice-versa. The conversion formulas are:

$$x = R\cos\theta \qquad y = R\sin\theta$$

$$R = \sqrt{x^2 + y^2} \qquad \theta = \begin{cases} \tan^{-1}\left(\dfrac{y}{x}\right) & x \geq 0 \\ \tan^{-1}\left(\dfrac{y}{x}\right) + \pi & x < 0, y \geq 0 \\ \tan^{-1}\left(\dfrac{y}{x}\right) - \pi & x < 0, y < 0. \end{cases}$$

   Print messages to standard output to report the coordinates entered by the user, and the equivalent coordinates in the other representation. Suppose that the user entered the numbers 1.5 and 2, and selected Cartesian coordinates. Then the correct output message is:

```
You entered Cartesian coordinates.
The equivalent polar coordinates are:
R = 2.500, theta = 0.927
```

5. If the user entered their values in polar coordinates, then R must be non-negative. Also, we will force the user to enter the principle angle for θ, so that $-\pi \leq \theta \leq \pi$. Modify your program so that, if the user enters $R < 0$, or an angle outside of $[-\pi, \pi]$, the program will print the following error message, and return from `main()` with a value of -1.

```
ERROR! Invalid polar coordinates, exiting.
```

6. Prompt the user to enter a second pair of numbers, read them from standard input, and report these values back to the user, as in step 2. Repeat all parts of steps 3, 4 and 5 for the second

pair of numbers. It must be possible for the user to enter one pair of numbers in Cartesian form and the other in polar form, if they choose. Make sure to print a line that reports back the numbers to the user, as you did in step 2.

Test your program thoroughly, and verify that it produces the correct output for all cases of valid and invalid data entered by the user. When you are certain your program is correct, save it in a file named:

`ece0301_ICA03_step06.cpp`

Submit this file to the class repository.

7. Ask the user if they wish to treat the two pairs of number as vectors or complex numbers by entering a single letter: `v` or `c`. Read the user response, and validate the input as in step 3, by printing the error message shown below, and returning from `main()` with a value of -1, if the response is anything other than `v` , `V` , `c` or `C`.

```
ERROR! Invalid selection, exiting.
```

8. If the user responds with `v` or `V`, treat the pairs of numbers as vector coordinates, and display them in both Cartesian and polar forms, formatted appropriately for vectors. For example,

```
Cartesian:   v1 = (-4.000, 3.000)     v2 = (-1.000, -1.000)
Polar:       v1 = (5.000, 2.498)      v2 = (1.414, -2.356)
```

If the user responds with `c` or `C`, treat the pairs of numbers as the real and imaginary parts of complex numbers, and display them in both Cartesian and polar forms, as appropriate for complex numbers. For example,

```
Cartesian:   z1 = -4.000 + j 3.000    z2 = -1.000 – j 1.000
Polar:       z1 = 5.000 exp(j 2.498)  z2 = 1.414 exp(-j 2.356)
```

9. If the user chose `v` for vectors, prompt the user with a menu to choose which of the following vector sums or differences they would like to compute:

   (1) $v_{\text{sum}} = v_1 + v_2$

   (2) $v_{\text{sum}} = v_1 - v_2$

   (3) $v_{\text{sum}} = v_2 - v_1$

   (4) $v_{\text{sum}} = -v_1 - v_2$

   Read the user response to an `int` variable. If the user response is outside the range 1-4, the program must print an error message and return from `main()` with a value of -1. Otherwise, use a `switch/case` statement to compute the selected sum or difference. Then display the result in both Cartesian and polar coordinates, using the notation for vectors.

10. If the user chose `c` for complex numbers, prompt the user with a menu to choose which of the following sums or differences they would like to compute:

    (1) $z_{sum} = z_1 + z_2$

    (2) $z_{sum} = z_1 - z_2$

    (3) $z_{sum} = z_2 - z_1$

    (4) $z_{sum} = -z_1 - z_2$

    Read the user response to an `int` variable. If the user response is outside the range 1-4, the program must print an error message and return from `main()` with a value of -1. Otherwise, use a switch/case statement to compute the selected sum or difference. Then display the result in both Cartesian and polar coordinates, using the notation for complex numbers.

    Test your program thoroughly, and verify that it produces the correct output for all cases of valid and invalid data entered by the user. When you are certain your program is correct, save it in a file named:

    `ece0301_ICA03_step10.cpp`

    Submit this file to the class repository.

11. For any two vectors, the inner product or dot product is a scalar quantity that is equal to the product of the lengths of the vectors, multiplied by the cosine of the angle between them. Using the notation for polar coordinates from part 6, the dot product is

$$v_1 . v_2 = R_1 R_2 \cos(\theta_1 - \theta_2).$$

    It is also possible to compute the dot product as the sum of the pairwise products of the Cartesian coordinates. For vectors in the plane, i.e. $(x, y) \in \mathbb{R}^2$, this is simply

$$v_1 . v_2 = x_1 x_2 + y_1 y_2.$$

    If the user selected vectors, use both methods to compute the dot product of the vectors, and report both results to the user, to confirm that they are consistent.

12. There are many similarities between 2-vectors and complex numbers, because they both take their values in a plane, and because the adding and subtracting operations are performed by adding or subtracting the Cartesian coordinates or the real and imaginary parts,

$$v_1 + v_2 = (x_1 + x_2, y_1 + y_2) \qquad \text{and} \qquad z_1 + z_2 = (x_1 + x_2) + j(y_1 + y_2).$$

    However, complex numbers are different from 2-vectors, because of the unit imaginary number $j = \sqrt{-1}$ that relates the real and imaginary parts. In particular, the multiplication operation is defined differently for vectors and complex numbers; the product of two complex numbers is another complex number,

$$z_1 z_2 = R_1 R_2 e^{j(\theta_1 + \theta_2)},$$

    as compared to the inner product of two vectors which is a single real number. We can also compute the product of two complex numbers directly from the real and imaginary parts of z_1 and z_2,

$$z_1 z_2 = (x_1 + jy_1)(x_2 + jy_2) = (x_1 x_2 - y_1 y_2) + j(x_1 y_2 + y_1 x_2).$$

The first method gives the product in polar coordinates, and the second method gives the product in rectangular coordinates. Use both methods to compute the product of the complex numbers, and report both products to the user. Finally, convert the polar coordinates for the product to Cartesian, to confirm that they are consistent.

Test your program thoroughly, and verify that it produces the correct output for all cases of valid and invalid data entered by the user. When you are certain your program is correct, save it in a file named:

`ece0301_ICA03_step12.cpp`

Submit this file to the class repository.

Don't forget to include comments! You will lose credit if you leave them out, even if your code functions as directed!