CSE1222 Programming & Algorithms for Engineers **Autumn 2016**

Programming Assignment 4: Functions

1 Setting up the Programming Environment

1. Create a new directory (folder) called `hw4` and move to that directory.

2. Copy the files from the directory `/class/cse1222/9643/hw4` into the current directory by typing:

```
cp /class/cse1222/9643/hw4/vector2D_solution.exe .
cp /class/cse1222/9643/hw4/vector2D_template.cpp vector2D.cpp
```

2 Work by Yourself

All lab and programming assignments are to be done by yourself. You may discuss labs or assignments with other students in the class but DO NOT LOOK AT ANYONE'S CODE OTHER THAN YOUR OWN. Needless to say, you should not share or copy anyone else's code.

3 Program Requirements

Effective commenting and tabbing will affect your grade. The "style" of your program should follow the style of the sample programs in the course notes. Your program should have the file name, your name, creation and last modification dates and a brief description of the program in the comments at the top of the program. The declaration of every variable should have a comment.

A 2D vector (u, v) has its base at the origin in the cartesian coordinate system, i.e. the x-y axes, and its tip (or arrow) at $x = u$ and $y = v$. Thus, a 2D vector can be represented simply as an (x, y) point in the cartesian coordinate system. Write a program that reads in a pair of 2D vectors and a scalar value and then applies the following vector operations: *addition*, *subtraction*, *scalar multiplication*, and *perpendicularity*. The `main` routine of the program is already provided in `vector2D_template.cpp` (which you copied into `vector2D.cpp`). Your task is to add eight functions `read_vector()`, `vector_length()`, `write_vector()`, `vector_add()`, `vector_subtract()`, `scalar_mult()`, `normalize()`, and `perpendicular()` so that the program produces the desired results.

1. Write your code incrementally. I HIGHLY SUGGEST that you write each function one at a time as described in the following steps. After you have written a solution to

a particular function, you should `compile`, `run`, and `test` your code before moving to the next step.

2. DO NOT MODIFY ANY OF THE CODE in procedure `main`.

3. DO NOT change the global constant called `EPSILON` defined at the top.

4. You will use the global constant `EPSILON` whenever you need to determine the equality of two values of type `double` (Please study the lecture notes before you continue). For example, if a and b are values of type `double` you can use the following condition to determine that they are equal: $abs(a-b) <$ `EPSILON`, where $abs()$ is the absolute value function.

5. A function prototype should be written for each function and placed BEFORE the main procedure.

6. All functions should be written AFTER the main procedure.

7. Each function should have a comment explaining what it does.

8. Each function parameter should have a comment explaining the parameter.

9. Write a function `read_vector()` which inputs from the user a 2D vector by reading its x- and y-coordinates. Define the function so that it does not return a value and has three input parameters:

   - a `string` to hold the prompt displayed to the user. Define the string as a *constant* parameter and use *pass by reference*. See the lecture notes on how to do this.
   - a number x of type `double` representing the first coordinate of the 2D vector. Define this parameter as *pass by reference*.
   - a number y of type `double` representing the second coordinate of the 2D vector. Define this parameter as *pass by reference*.

10. Write a function `vector_length()` which returns the length of a 2D vector. The length of vector (x, y) is $\sqrt{x^2 + y^2}$. Input to the function are the coordinates x and y, which are *passed by value*. The function returns the length as type `double`.

11. Write a function `write_vector()` which outputs a 2D vector and its length. Call the function `vector_length()` to compute the length. Define the function so that it does not return a value and has three input parameters:

    - a `string` to hold the message to be displayed before the coordinates of the vector. Define the string as a *constant* parameter and use *pass by reference*.
    - a number x of type `double` representing the first coordinate of the 2D vector. Define this parameter as *pass by value*.

- a number y of type `double` representing the second coordinate of the 2D vector. Define this parameter as *pass by value*.

12. Write a function `vector_add()` which adds two 2D vectors and results in a new 2D vector. To add two vectors $(x1, y1)$ and $(x2, y2)$ you will simply add the corresponding coordinate values, i.e. $(x1, y1) + (x2, y2) = (x3, y3)$ where $x3 = x1 + x2$ and $y3 = y1 + y2$. Define the function so that it does not return a value and has six input parameters:

    - a number $x1$ of type `double` representing the first coordinate of the first 2D vector. Define this parameter as *pass by value*.
    - a number $y1$ of type `double` representing the second coordinate of the first 2D vector. Define this parameter as *pass by value*.
    - a number $x2$ of type `double` representing the first coordinate of the second 2D vector. Define this parameter as *pass by value*.
    - a number $y2$ of type `double` representing the second coordinate of the second 2D vector. Define this parameter as *pass by value*.
    - a number $x3$ of type `double` representing the first coordinate of the resultant 2D vector. Define this parameter as *pass by reference*.
    - a number $y3$ of type `double` representing the second coordinate of the resultant 2D vector. Define this parameter as *pass by reference*.

13. Write a function `vector_subtract()` which subracts two 2D vectors and results in a new 2D vector. To subract use the formula $(x1, y1) - (x2, y2) = (x3, y3)$ where $x3 = x1 - x2$ and $y3 = y1 - y2$. Like function `vector_add`, this function also returns no value and uses the same input parameters (see instructions above).

14. Write a function `scalar_mult()` which applies *scalar multiplication* to a two 2D vector and results in a new 2D vector. *Scalar multiplication* is defined as $s \times (x1, y1) = (x2, y2)$, where s is a scalar value, $(x2, y2)$ is the resultant vector, and $x2 = s \times x1$ and $y2 = s \times y1$. Define the function so that it does not return a value and has five input parameters:

    - a number $x1$ of type `double` representing the first coordinate of the first 2D vector. Define this parameter as *pass by value*.
    - a number $y1$ of type `double` representing the second coordinate of the first 2D vector. Define this parameter as *pass by value*.
    - a number s of type `double` representing the scalar multiplier. Define this parameter as *pass by value*.
    - a number $x2$ of type `double` representing the first coordinate of the resultant 2D vector. Define this parameter as *pass by reference*.

- a number $y2$ of type `double` representing the second coordinate of the resultant 2D vector. Define this parameter as *pass by reference*.

15. Write a function `normalize()` which normalizes vector (x, y) by dividing by its length. *Normalization* results in a new vector that has the same direction but has a length of 1, called a *unit vector*. Use the following formula to compute the resuling vector:

$$x = \frac{x}{\sqrt{x^2+y^2}}$$

$$y = \frac{y}{\sqrt{x^2+y^2}}$$

Call the function `vector_length()` to compute the length. If the length $\neq 0$, then use the formula above. If length $= 0$, then set $x = 0$ and $y = 0$ to avoid a division by zero. Since the length is type `double` you will need to determine its equality with zero using `EPSILON` as shown in step 4 above. Let a be the length and b be zero. Define the function so that it does not return a value and has two parameters that are both *passed by reference*:

- a number x of type `double` representing the first coordinate of the 2D vector;
- a number y of type `double` representing the second coordinate of the 2D vector.

16. Write a function `perpendicular()` which determines whether two 2D vectors are perpendicular to each other. In order to make this determination, we will compute two perpendicular vectors from a given vector (x, y). The first perpendicular vector, $(px1, py1)$, is defined as $px1 = -y$ and $py1 = x$. The second perpendicular vector, $(px2, py2)$, is defined as $px2 = -px1$ and $py2 = -py1$. For example if $(x, y) = (1, 2)$ then $(px1, py1) = (-2, 1)$ and $(px2, py2) = (2, -1)$. Thus, $(px1, py1)$ and $(px2, py2)$ are perpendicular vectors to (x, y).

    You must implement the following *algorithm* to determine if two vectors, $(x1, y1)$ and $(x2, y2)$, are perpendicular:

    (a) Normalize $(x1, y1)$ and $(x2, y2)$; Let's call the new vectors $v1 = (vx1, vy1)$ and $v2 = (vx2, vy2)$, respectively (use better variable names). Call the `normalize()` function to compute these. Note that $v1$ and $v2$ are *unit vectors*.

    (b) Compute two perpendicular vectors to $v1$ (see instructions above). Let's call the two new vectors $p1$ and $p2$.

    (c) Check whether $v2$ is the same as either $p1$ or $p2$. If so, then output "Vectors are PERPENDICULAR." to the screen. Otherwise, output "Vectors are NOT PERPENDICULAR." See next step for more details.

    (d) Let $p1 = (px1, py1)$, and $p2 = (px2, py2)$.

i. To determine if $v2$ is the same vector as $p1$, check that both $vx2 = px1$ and $vy2 = py1$. Use `EPSILON` as described in step 4 above to determine each equality.

ii. To determine if $v2$ is the same vector as $p2$, check that both $vx2 = px2$ and $vy2 = py2$. Use `EPSILON` as described in step 4 above to determine each equality.

Define the function so that it does not return a value and has four parameters that are all *passed by value*:

- a number $x1$ of type `double` representing the x coordinate of the first 2D vector.
- a number $y1$ of type `double` representing the y coordinate of the first 2D vector.
- a number $x2$ of type `double` representing the x coordinate of the second 2D vector;
- a number $y2$ of type `double` representing the y coordinate of the second 2D vector.

4 Program Submission

Submit your file `vector2D.cpp` in the hw4 drop box on Carmen. DO NOT submit the file `a.out`.

If you do not submit your program, you will receive zero credit for the homework.

If your program does not compile and run you will receive zero credit for the homework.