CS 111 - Homework 9

Deadline

11:59 pm on **Friday, November 7, 2025**

Purpose

To practice designing more C++ functions, including one using a `string` class method and *some* with `if` statements.

How to submit

You complete **Problems 1 and 2** on the course Canvas site (short-answer questions on various C++-related topics), so that you can see if you are on the right track.

Then, you will submit your work for **Problems 3** onward, in your files **`111hw9.cpp`** and **`111hw9-out.txt`**, on the course Canvas site.

Submit your **`111hw9.cpp`** file-in-progress early and often!

- Each time you submit a version of your **`111hw9.cpp`**, IF that version currently compiles, also submit a copy of the example output from running that latest version in file **`111hw9-out.txt`**.
- Be careful that each submitted **`111hw9-out.txt`** was created by running the compiled version of the **`111hw9.cpp`** file submitted along with it.

Important notes - 14 points

- Be careful to follow class style standards, including required class indentation, especially with `if` statements involved; for example,
  - curly braces each on their own line
  - each set of `{` and `}` lined up with each other, with each `{` lined up with the *first* character of the previous line as shown in posted class examples
  - each statement within curly braces is indented by at least 3 spaces

  ...and if you are not sure what is meant by any of the above, see the posted class examples!
- You are still expected to follow the Design Recipe for all **functions** that you design/define.
  - Remember that a C++ "graphic design recipe helper" has been posted on the course Canvas site and on the public course web site, "translating" the design recipe steps into C++ syntax.
  - Remember, you will receive **significant** credit for the signature, purpose, header, and tests/test expressions portions of your functions.
  - Typically you'll get at least half-credit for a correct signature, purpose, header, and tests/test expressions, even if your function body is not correct.
  - (and, you'll **lose at least half-credit** if you omit these or do them poorly, even if your function body is correct).
- Be especially careful to include at least two tests/test expressions for every function, including at least one specific test/test expression for each "kind"/category of data, and (when there *are* boundaries) for boundaries between data. You can lose credit for not doing so.

And, remember that tests should be:

- written as `bool` expressions within a non-`main` function's opening comment, after its purpose statement, **AND**
- written within parentheses `( )` within a `cout` in the testing `main` function.

- Remember: in the CS50 IDE at https://cs50.dev:
  - each time you want to **compile** your program-in-progress:

    In a CS50 terminal, open to the folder CONTAINING `111hw9.cpp`, (**"Open in Integrated Terminal"**), type:

    ```
    g++ 111hw9.cpp -o 111hw9
    ```
  - when it has successfully compiled and created an executable program result, each time you want to run your program-in-progress:

    In that same CS50 terminal, open to the folder CONTAINING `111hw9.cpp`, type:

    ```
    ./111hw9
    ```
  - when you are satisfied with the program's output for one of your functions, you can create an example output from running your program at that point to submit to Canvas by running :

    ```
    ./111hw9 > 111hw9-out.txt
    ```
- Please let me know if you have any questions or concerns about the above requirements.

Problem 1 - 7 points

Problem 1 is correctly answering the "HW 9 - Problem 1 - Short-answer questions on `string` methods" on the course Canvas site.

Problem 2 - 6 points

Problem 2 is correctly answering the "HW 9 - Problem 2 - Short-answer questions *mostly* on reading `if` statements" on the course Canvas site.

Homework Program Setup for Problems 3 onward

- **Copy** the contents of the file **`111template.cpp`**, posted on the course Canvas site and on the public course web site, into a file named **`111hw9.cpp`** within the CS50 IDE (at https://cs50.dev/).
- See the comment that has `by:` and `last modified:` ?
  - START that comment with: `CS 111 - HW 9`
  - Then put your name after `by:` , and today's date after `last modified:` .
  - For example:

```
/*---
    CS 111 - HW 9
    by: Your Name
    last modified: 2025-11-03
---*/
```

Problem 3 - 13 points

In the "first `main.cpp` template" you pasted into your **`111hw9.cpp`**, find the comment:

```
/*--- PUT YOUR SIGNATURES, PURPOSES, TESTS, and FUNCTION DEFINITIONS HERE ---*/
```

AFTER this comment -- but **BEFORE** the function header for the function named `main` -- type a blank link, and then type the comment:

```
/*===
    Problem 3
===*/
```

(You can now delete the `"...PUT YOUR SIGNATURES,..."` comment if you wish, or leave it -- your choice!)

Recall, from the Week 10 Lecture 2 and Problem 1's short-answer questions, that the C++ `string` class includes two substring methods named `substr`, one of whose versions:

- expects a starting position and a length, and
- returns the `string` starting at that position and going that many characters in the calling `string` instance (or until the end of that calling `string` instance, whichever comes first).
- And, interestingly, the position of the 1st character in a `string` is 0!

That is, for:

```
const string CHEER = "hip, Hip, HOORAY!";
```

...this is a `true` expression:

```
CHEER.substr(5, 3) == "Hip"
```

Your function for Problem 3

Use the design recipe to design a C++ function `monogram` that expects a first name, middle name, and last name, and returns an appropriate "monogram" for that name (a string with tasteful angle brackets surrounding the initials of that first, middle, and last name). For example,

`monogram("Ava", "Marie", "DuVernay")` would return `"<AMD>"`, and

`monogram("Grace", "Murray", "Hopper")` would return `"<GMH>"`.

OPTIONAL VARIATION:

- You may design a **more-elaborate** resulting monogram, as long as it includes initials of the given first, middle, *and* last name.

- Remember to include a **signature**, **purpose**, **function header**, **tests**, and then completed **function body** for `monogram`.
  - (HINT: Note that, for this particular function, you should use the `string` class's **`substr`** method, which returns a `string`, instead of the `string` class's `at` method, which returns a `char`.)
- Be sure to include your tests BOTH in a comment after your purpose statement, AND in `main`, as we have done in class.
  - **IMPORTANT**: be careful with your **expected value** expression in `monogram`'s test expressions; **you cannot use + between two `char*` expressions, nor between a `char` and a `char*` expression**.

    This is a case where the expected value expression likely needs to be just the hoped-for resulting literal expression, without any operations.

    ASK ME if you have questions about this or run into problems related to this.

- **IF YOU WOULD LIKE:** Feel free to add EXAMPLE CALLS to your `main` function calling `monogram` after the above tests if you would like, putting the call INSIDE of a `cout` statement as follows:

  ```
  cout << (monogram("Grace", "Murray", "Hopper")) << endl;
  ```

Problem 4 - 19 points

After your function for Problem 3, type a blank link, and then type the comment:

```
/*===
    Problem 4
===*/
```

Now for a problem using `if` statements, along with some named constants.

Recall the function from Homework 4, Problem 4 that gives discounts to frequent shoppers based on their frequent-shopper level. As a reminder:

A store gives discounts to frequent shoppers based on their past level of purchases; they are either "bronze" level, "silver" level, or "gold" level. Bronze level frequent shoppers receive a 10% discount, silver level frequent shoppers receive a 15% discount, and gold level frequent shoppers receive a 20% discount. All other shoppers receive no discount.

Use the design recipe to develop a C++ function `get_freq_disc` that expects a string representing the level of frequent shopper, and returns the appropriate discount for that level written as a decimal fraction. It should return a discount of 0 if the shopper level is not one of those noted above.

(Note that you can find the Racket example solution for this on the course Canvas site, under "Selected solutions - Homeworks".)

- THIS function also uses named constants -- three of them! Here's how I declared them in the Homework 4 posted example solutions:

  ```
  (define BRONZE-DISC 0.1)

  (define SILVER-DISC 0.15)

  (define GOLD-DISC 0.2)
  ```

  - Write **C++ named constant declaration statements** for these, declaring:

    `BRONZE_DISC`

    `SILVER_DISC`

    `GOLD_DISC`

    ...giving them the correct values.
  - PLACE these declaration statements right *before* the function header for `get_freq_disc` (*after* the comment with the function's signature, purpose, and test `bool` expressions).
- Remember to include a **signature**, **purpose**, **function header**, **tests**, and then completed **function body** for `get_freq_disc`.
- For full credit: within your function body, make sure that you:
  - use your named constants appropriately
  - appropriately use a chained `if-else-if` pattern
- Be sure to include your tests BOTH in a comment after your purpose statement, AND in `main`, as we have

done in class.

  - At least how many tests, covering at least which cases, are needed for `get_freq_disc`?

- IF you would like, you can also include one or more `cout` statements that include JUST an example call of your function **after** these tests, so that you see the value those call(s) return.

Problem 5 - 13 points

NO `if`-statements here -- this problem's purpose is to practice writing a C++ function that calls another C++ function.

After your function for Problem 4, type a blank link, and then type the comment:

```
/*===
    Problem 5
===*/
```

Recall the function from Homework 4, Problem 5 that used Homework 4, Problem 4's function to determine the calculated discounted total for a purchase by a frequent shopper.

That is, you will write a function that happens to *use* Problem 4's `get_freq_disc` function.

Use the design recipe to develop a C++ function `total_incl_disc` that expects a string representing the level of frequent shopper *and* the total of a purchase *before* discount, and returns the calculated discounted total for that purchase after applying the appropriate discount, as provided by the `get_freq_disc` function.

(Hint: you should **not** need an `if` statement in this particular function, thanks to `get_freq_disc.`)

(Note that you can find the Racket example solution for this on the course Canvas site, under "Selected solutions - Homeworks".)

- **OPTIONAL VARIATION**:
  - But, would you *like* some more optional `if` practice here? If so, you may write this so that, in addition to the frequent shopper discount, it calculates an **additional** discount based on the purchase total before discount (for example, they might receive an additional 5% discount if their total is more than 50 dollars).
  - If you do this optional variation, make sure you describe this in your function's purpose statement.
  - If you do this optional variation, note that you will need more than two tests (the number of tests needed will depend on the rules you decide to use of the additional discount).
- Remember to include a **signature**, **purpose**, **function header**, **tests**, and then completed **function body** for `total_incl_disc`.
- For full credit: within `total_incl_disc`'s function body, make sure that you:
  - call `get_freq_disc` appropriately
- Be sure to include your tests BOTH in a comment after your purpose statement, AND in `main`, as we have done in class.
- IF you would like, you can also include one or more `cout` statements that include JUST an example call of your function **after** these tests, so that you see the value those call(s) return.

Problem 6 - 14.5 points

After your function for Problem 5, type a blank link, and then type the comment:

```
/*===
    Problem 6
===*/
```

Recall the function from Homework 4, Problem 6 that recommends what outerwear to wear on a given day given the predicted high temperature in Fahrenheit degrees, based on the following:

```
<= 32 - "down jacket"
(32, 48] - "sweater"
(48, 65] - "sweatshirt"
> 65 - "no outerwear today"
```

Use the design recipe to develop a C++ function `sugg_outerwear` that expects the predicted high temperature in Fahrenheit, and returns the recommended outerwear for that day.

(Note that you can find the Racket example solution for this on the course Canvas site, under "Selected solutions - Homeworks".)

- Remember to include a **signature**, **purpose**, **function header**, **tests**, and then completed **function body** for `sugg_outerwear`.
- For full credit: within your function body, make sure that you:
  - appropriately use a chained `if-else-if` pattern
- Be sure to include your tests BOTH in a comment after your purpose statement, AND in `main`, as we have done in class.
  - For full credit, make sure that you include at least the minimum required tests for this data (hint: including boundary cases!).
- IF you would like, you can also include one or more `cout` statements that include JUST an example call of your function **after** these tests, so that you see the value those call(s) return.

Problem 7 - 13.5 points

After your function for Problem 6, type a blank link, and then type the comment:

```
/*===
    Problem 7
===*/
```

Recall the function from Homework 4, Problem 7 regarding pizza consumption and exercise.

Use the design recipe to develop a C++ function `workout_hrs` that expects a number that represents daily pizza consumption, in slices, and returns a number, in hours, that represents the amount of exercise time that you need.

For a daily intake of :	You need to work out for :
`0` slices	1/2 hour
`(0, 3]` slices	1 hour
`> 3` slices	1 hour + (1/2 hour per slice above 3)

(Note that you can find the Racket example solution for this on the course Canvas site, under "Selected solutions - Homeworks".)

- Remember to include a **signature**, **purpose**, **function header**, **tests**, and then completed **function body** for `workout_hrs`.
- For full credit: within your function body, make sure that you:
  - appropriately use a chained `if-else-if` pattern
- Be sure to include your tests BOTH in a comment after your purpose statement, AND in `main`, as we have done in class.
  - For full credit, make sure that you include at least 4 well-chosen, appropriate tests. You could certainly follow the example of the tests from the Racket example solution for `workout-hrs`.
- IF you would like, you can also include one or more `cout` statements that include JUST an example call of your function **after** these tests, so that you see the value those call(s) return.

Remember to submit your files-in-progress **`111hw9.cpp`** and **`111hw9-out.txt`** early and often!