CS 111 - Homework 10

Deadline

11:59 pm on **Friday, November 21, 2025**

Purpose

To practice designing more C++ functions, including *some* using **`switch`** statements, and to practice a bit with **local variables** and **interactive input**.

How to submit

You complete **Problems 1-4** 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 4** onward, in your files **`111hw10.cpp`**, **`111hw10-out.txt`**, and **`111hw10-prob8.cpp`**, on the course Canvas site.

(So, NOTE that, THIS time, you will be creating **TWO `.cpp`** files to turn in, for the remaining problems!)

Turn in versions of your files **early** and **often**!

- Each time you submit a version of your **`111hw10.cpp`**, IF that version currently compiles, **also** submit a copy of the example output from running that latest version in file **`111hw10-out.txt`**.
- Be careful that each submitted **`111hw10-out.txt`** was created by running the compiled version of the **`111hw10.cpp`** file submitted along with it.
- (You are **NOT** submitting a **`111hw10-prob8-out.txt`** file, for the same reason you were not asked to submit a file **`lab12-out.txt`** for the Week 12 Lab Exercise!).

Important notes - 14 points

- **NOTE:** if you are just adding statements **to a `main` function**, the usual design recipe steps are **NOT** required. (They are, of course, required for **ALL (non-`main`) functions** that you design/define.)
- IF you would like: FEEL FREE to include additional **`cout`**s of **`endl`** or spacing or headings between testing calls of different problems if you would like to have more-readable program output!
- Be careful to follow class style standards, including required class indentation, especially with **`if`** and **`switch`** 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
  - the statements for each `case` are also indented under that `case` 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 (non-`main`) **functions** that you design/define.
  - Remember the C++ "graphic design recipe helper" 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.

- Please let me know if you have any questions or concerns about the above requirements.

Problem 1 - 8 points

Problem 1 is correctly answering the "HW 10 - Problem 1 - Short-answer questions on `switch` statement syntax" on the course Canvas site.

Problem 2 - 8 points

Problem 2 is correctly answering the "HW 10 - Problem 2 - Short-answer questions on when you can use a `switch`" on the course Canvas site.

Problem 3 - 6 points

Problem 3 is correctly answering the "HW 10 - Problem 3 - Short-answer questions on `switch`, `cout`, and `return`" on the course Canvas site.

Problem 4 - 4 points

Problem 4 is correctly answering the "HW 10 - Problem 4 - Short-answer questions focusing on assignment statements" on the course Canvas site.

Homework Program Setup for Problems 5 onward

For **EACH** of the **TWO** programs involved in this homework:

- **Copy** the contents of the **`111template.cpp`**, posted on the course Canvas site and on the public course web site, into a file within the CS50 IDE (at https:/cs50.dev/) named as specified in Problem 5 and Problem 8.
- See the comment that has `by:` and `last modified:` ?
  - START that comment with: `CS 111 - HW 10`
  - Then put your name after `by:` , and today's date after `last modified:` .
  - For example:

```
/*---
    CS 111 - HW 10
    by: Your Name
    last modified: 2025-11-17
---*/
```

Problem 5 - function `coin_worth` - 15 points

Problems 5 through 7 will all be in a single file named **`111hw10.cpp`**.

In the "first `main.cpp` template" you pasted into your **`111hw10.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 5
===*/
```

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

The purpose of this problem is to provide more practice with **`switch`** statements.

Assume that coins are represented as follows:

- 'Q' or 'q' -- quarter
- 'D' or 'd' -- dime
- 'N' or 'n' -- nickel
- 'C' or 'c' -- cent

Use the design recipe to design a C++ function **`coin_worth`** that expects a character representing a coin, **and uses a C++ `switch` statement** to return the decimal worth of that coin (for example, a cent is worth `0.01`). If it receives any character besides those noted above, it should return a worth of `0.0`.

- Remember to include a **signature**, **purpose**, **function header**, **tests**, and then completed **function body** for **`coin_worth`**.
- 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 **`coin_worth`**?
- 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 - function `compute_it` - 15 points

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

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

This problem's purpose is to provide still more practice with the C++ **`switch`** statement.

Fun fact: the C++ `cmath` library has a function `pow` that expects two `double` arguments and returns the result of raising the first argument to the power given by the second argument. That is, `pow(2.0, 3.0)` gives you the result of raising 2.0 to the power 3.0, and so results in 2.0 * 2.0 * 2.0 == 8.0.

Consider: the `char` expression `'+'` cannot be used to add two numbers together in C++. But -- if you were given that `char` expression, and two numbers, you *could* write expressions and statements that would see if the given `char` expression was a '+', and if that is so, then it *could* add those numbers together using a *proper* + operator.

(And for a non-existent operator? You *could* write expressions and statements that would see if the given `char` expression was that character, and if that is so, then it *could* perform the desired computation for those numbers using an appropriate compound expression, for example using a function call.)

Use the design recipe to develop a C++ function **`compute_it`** that indeed expects an operator expressed as a `char` expression and two numbers, **and uses a C++ `switch` statement** to return the result of performing the computation with the operator corresponding to that `char` expression to those two numbers. It should be able to support at least:

`'+'` -- add the two numbers

`'-'` -- subtract the two numbers

`'*'` -- multiply the two numbers

`'/'` -- divide the two numbers

`'^'` -- raise the first number to the power of the second number (**No**, C++ does **NOT** have a ^ operator. But it *does* have that `pow` function mentioned earlier.)

`compute_it` should simply return 0.0 if called with any **unsupported/unexpected** `char` expression as the operator `char` expression.

For example:

```
compute_it('+', 3.4, 1.6) == (3.4 + 1.6)

compute_it('-', 5, 2) == (5 - 2)

compute_it('?', 5.6, 8) == 0.0
```

(although, if necessary,

```
abs(compute_it('+', 3.4, 1.6) - 5.0) < 0.01

abs(compute_it('-', 5, 2) - 3.0) < 0.01
```

)

- **OPTIONAL VARIATION:**

  You may **ADD additional** operator-`char`-expressions *IF* you would like. Note that you should add **additional** tests as needed for your variation.

- Remember to include a **signature**, **purpose**, **function header**, **tests**, and then completed **function body** for **`compute_it`**.
- 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 7 - function `piggify_it` - 15 points

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

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

So you don't forget -- let's add a bit more practice with:

- `if` statement logic
- `string` class methods
- a function that calls another function

(So, note that this function does **NOT** need to use a `switch` statement!)

Step 1

Copy the opening comment with the signature, purpose, and `bool` test expressions and the function definition for the Week 11 Lab Exercise's function **`is_vowel`**.

- Note: if you did not do the Week 11 Lab Exercise or you are not confident in your version of **`is_vowel`**, you can e-mail me and ask for a version of **`is_vowel`**.

Now that **`is_vowel`** is in your **`111hw10.cpp`** file, it can be used by another function that follows it in this file.

Onward!

Consider:

- The now-available function **`is_vowel`**.
- The `string` method **`at`**, which expects a desired (0-based) position within a string, and returns a `char` whose value is the character at that position in the calling string.
  - If you had a parameter named `word` whose type is `string`, then this expression:

    `is_vowel( word.at(0) ) == true`

    ...SHOULD indeed be `true` if `word` begins with a vowel, right?
- And, consider the `string` method **`substr`** -- how can you use it to get a string containing all EXCEPT the first letter of a given `string`?

Use the above and the design recipe to design a function **`piggify_it`** which is meant to be a **SIMPLIFIED** variation on pig latin. Function **`piggify_it`** expects a word containing at least one character, and:

- IF it starts with a vowel, it returns a string that is that word with `-ay` added to its end
- OTHERWISE, it returns a string that is that ALL BUT the first character in that word with `-` and its first letter and `ay` added to its end.
- for example, `piggify_it("orange") == "orange-ay"`

  `piggify_it("moo") == "oo-may"`

  `piggify_it("Harold") == "arold-Hay"`

  `piggify_it("I don't read directions") == "I don't read directions-ay"`
- **OPTIONAL VARIATION:**

  You may design a **more complex** variation on this if you would like, as long as it still appropriately uses function **`is_vowel`**. Note that you should add additional tests as needed for your variation.

- Remember to include a **signature**, **purpose**, **function header**, **tests**, and then completed **function body** for **`piggify_it`**.
- 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 8 - an interactive front-end for a function - 15 points

We have mentioned in class that not all `main` functions are used just for testing other functions. Sometimes they simply "control" a desired program.

Trying this out will be less awkward if it is done in a separate C++ program (with its own `main` function).

Copy the contents of the **`111template.cpp`**, posted on the course Canvas site and on the public course web site, into a file named **`111hw10-prob8.cpp`** within the CS50 IDE (at https:/cs50.dev/).

This program will contain a program whose `main` function JUST serves as an interactive front end for previously-designed function(s) (for example, as `lab12.cpp`'s `main` function does).

CHOOSE ONE or MORE of your functions from Problems 5, 6, or 7 -- one or more of the functions **`coin_worth`**, **`compute_it`**, or **`piggify_it`**.

- In **`111hw10-prob8.cpp`**, paste in COPIES of your signature, purpose, `bool` test expressions, and function definition for your chosen function(s).
  - (In this case, do **NOT** copy over the tests from its testing `main` in `111hw10.cpp` -- but **DO** copy over the tests in its opening comment, after its purpose statement.)
  - If you choose **`piggify_it`**, **also** copy over the signature, purpose, `bool` test expressions, and function definition for **`is_vowel`**.
- Then, in project **`111hw10-prob8.cpp`**'s `main` function, do the following:
  - Declare a local variable for each parameter of the function(s) you chose.

    For example:

    – if you choose **`coin_worth`**, you would declare **one** local variable able to hold a `char` coin character

    – if you choose **`compute_it`**, you would declare **three** local variables, one able to hold a `char` operator and two able to hold `double` values

    – if you choose **`piggify_it`**, you would declare **one** local variable able to hold a `string` word to piggify
  - For each parameter of the function(s) chosen, use `cout` to ask the user to enter in one of the values, and read the entered quantity using `cin` and >> into the appropriate local variable.
  - Then call your chosen function(s) appropriately, with the now-set local variables as its arguments, such that something appropriate will be printed to the screen.

OPTIONAL VARIATIONS:

- Would you like to do more than the minimum requirements above? As long as you **at least** do what's described above, you can add more statements to this `main` function.

- Would you like to call your chosen function(s) **repeatedly**? You may do so if you would like. How many times will you repeat? How will you decide when to stop?

Remember to submit your files-in-progress **`111hw10.cpp`**, **`111hw10-out.txt`**, and **`111hw10-prob8.cpp`** early and often!