Homework

0

C++ Mechanics

Here is our problem for your first homework assignment.

> *A metric ton is 35,273.92 ounces. Write a program that will read the weight of a package of breakfast cereal in ounces and output the weight in metric tons as well as the number of boxes needed to yield one metric ton of cereal.* --Savitch, Absolute C++ 5th Edition, Chapter 2

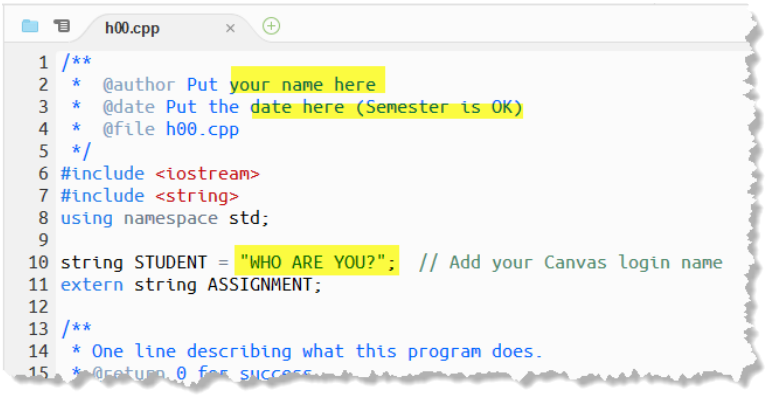
This is an **interactive IPO** (*Input*, *Processing*, *Output*) programs. You'll type the input at the keyboard in response to a **prompt**, and the output will be displayed on your monitor. If you find this "old fashioned", rest assured that the concepts you learn will remain the same for even the most sophisticated program.

1. Identify Yourself

Add **identification** to every project you build.

- In the **file documentation comment** at the top of the program, add your name after the `@author` tag and the date after the `@date` tag. You can just use the semester for the date if you like, but tell me what section you are in.

- Find the `STUDENT` variable and fill in in with your **Canvas login ID**. (This is your student email without the @student.cccd.edu part). This is a sorting key when processing the assignments; if you do this wrong, you will not receive any credit.



2. Design a Solution

All programming **starts with planning and design**. Before we can write a program, we have to spend time thinking about the inputs, processing and outputs.

- **Always** design your programs **before** you start writing code.
- What are the **inputs**, **outputs**, **algorithms** and **assumptions**?
- **Write it in English** before you ever start writing in C++.

> The sooner you start writing code, the longer it will take to finish

A Preliminary Solution

Here is some information that we can discover from the problem description:

- **Input**: weight of a box of cereal in ounces
- **Output**: weight of box in metric tons *and* number of boxes in a metric ton.
- **Given**: metric ton is `35,273.92` ounces
- **Calculation**: the weight in metric tons is equal to the weight of the box in ounces divided by the number of ounces per one metric ton.
- **Calculation**: the number of boxes per metric ton is equal to one divided by the weight of a single box in metric tons.

While you shouldn't jump in and start coding, you **can** put your design information into a **program comment** before you begin programming.

Internal Documentation

Internal documentation (such as that needed by the programmer to implement the solution), should appear in an **implementation comment** inside the function.

Place that in the `run()` function, like this:

```
//  Input: weight of a box of cereal in ounces
//  Output: weight of box in metric tons, boxes per ton.
//  Given: metric ton is 35,273.92 ounces
//  Calculation: weight tons<-weight in oz divided by oz per metric ton.
//  Calculation: boxes<- 1 divided by weight of box in tons.
```

> *Using single-line comments is easier than using paired comments, because your IDE has a shortcut key to generate them (**Shift+/**).*

What is the run Function?

As you know from the Course Reader, every C++ program begins with a **function** named `main()`. When you scroll though `h00.cpp` you won't find `main()` but a **function** named `run()` instead. In CS 150 we'll use `run()` for our programs to **simplify the process of testing**. The "real" `main()` function is inside the **library file** `libh00.a`.

3. Interaction Mock Up

The easiest way to start coding an IPO program is by **mocking up the interaction**, using plain output, substituting both input and output values with **literals**.

The C++ **standard library output object** is named `cout` (like `System.out` in Java). You use it along with the **insertion** (or output) **operator** `<<` like this:

```
25
26      cout << STUDENT << "-" << ASSIGNMENT << ": ";  (1)
27      cout << "Cereal Box Calculator" << endl;
28      cout << string(50, '-') << endl;  (2)
29      // Input
30      cout << "Enter ounces per box of cereal: " << 10 << endl;  (3)
31      cout << "Weight in metric tons, boxes per ton: ["
32          << 0.000283496 << ", " << 3527.39 << "]" << endl;  (4)
33
34      return 0;
35  }
```

The **arguments** sent to `cout` are printed from left to right, each separated from the others by the insertion operator. Place these statements inside the `run()` function.

Let's look at each of the sections:

1. Line 26 has **four arguments**: your student ID and the assignment number, (stored in two **variables** that have been previously defined), and two **string literals**, one containing a hyphen and the other containing a colon followed by a space. A **string literal** is text inside of **double quotes**.

   Each output line ends in a **semicolon**, the C++ **statement terminator**. Each line (except the first), also ends with `endl`, pronounced "end-ell", which represents the **newline** (or end-line) **character** on your platform. That means the first two source code lines will produce **one line of output**.

2. The second section (line 28 in the code shown here), creates a C++ `string` **object** by using a **constructor**. In C++ `string` objects are a different type

than string **literals**. This `string` will consist of 50 hyphens. To use the C++ `string` type, the starter code includes the `<string>` header.

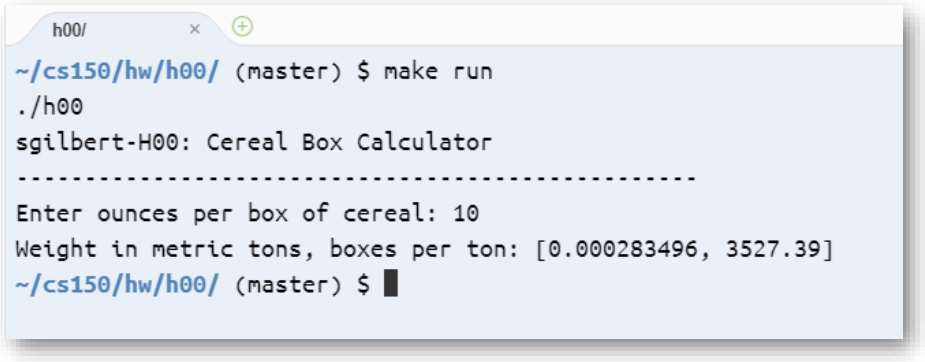
3. The third section (line 30) is the **input section**. The **prompt** (`"Enter …"`) will be printed as literal text which will not change when the program runs. The number **10** is highlighted to indicate that you expect to **receive this value as input from the user**. There are no quotation marks around the number.
4. The last section (lines 31-32) are the **program's output**. The highlighted numbers will be replaced with the **values calculated by processing**. Separating variable data from fixed data makes subsequent development a little easier since you won't have to change the fixed text.

4. Compile, Link & Run

Here are the instructions for compiling, linking and running your program:

- Enter **and save** your source code in the `.cpp` file.
- **Build** the program by switching to the correct directory in the terminal and then typing `make` followed by `ENTER`.
- **Run** the program by typing `make run`, followed by `ENTER`.

If you have no errors, then you should see something like this.



Of course it's possible that your code **didn't** compile and run successfully. If you have any **syntax errors**, the build process will fail, and your program will not run. If your program runs, it may still not produce the expected output. Check both of these.

5. Input, Processing and Output

Your program now looks like the mock-up so you can turn to **input and processing**.

- Create **variables** to hold the **input** and the **results** of our calculations.
- Use **`cin`** (*see-in*) with the **extraction** operator **`>>`** to read the input.
- Write **expressions** to calculate the output, storing that in variables.
- **Display** the output and then **test** to see that the output is correct.

Reading Input

Here's the completed input section.

```
29    cout << string(50, '-') << endl;
30
31    // Input
32    cout << "Enter ounces per box of cereal: "; // prompt
33    double ouncesPerBox;                        // store the input
34    cin >> ouncesPerBox;                        // read the input
35
36    // Processing section
37    cout << "Weight in metric tons, boxes per ton: ["
```

We need one input value: the number of ounces per box. Cereal boxes often contain a portion of an ounce, so we'll use the data-type called **`double`**. You may use either the **CamelCase** style for variable names, or all lowercase with underscores. Don't start your variable names with a capital letter, however.

To **read input** from the user, first **prompt for the information** which the user is expected to enter, and then read the input from the **`cin`** object which is the **standard input stream**. By default, this is the keyboard.

Prompting and Converting

The "mockup" data previously appearing on the prompt line has been removed, as well as the newline (**`endl`**) appearing after the prompt. When you display a prompt for input, you generally omit the **`endl`** at the end of the line.

In addition, make sure that the prompt **ends in a single space**, so that it displays the prompt but leaves the console cursor–the blinking vertical bar or square that marks the current input position at the end of the line–waiting for the user's response.

The final statement in the marked section reads a sequence of characters typed by the user at the keyboard, and stores the results in the variable **`ouncesPerBox`**. Because this variable was declared as a **`double`**, the **`>>`** operator **automatically converts** the characters typed by the user into a floating-point value.

Processing and Output

Create variables to hold the output values. There are two outputs, so we create two variables: **`weightInMetricTons`** and **`boxesPerMetricTon`**. You should **initialize** these variables using the expressions which you discovered during the planning phase.

> ***Note**: In C++, variables are not initially given a value; instead, they use whatever random value happens to be in memory at that time. (This is different than Java which prohibits assigning to uninitialized variables). Instead of first creating the variables and then assigning a value, create variables **only when you can calculate an initial value**.*

Finally, in the output section, replace the literals with the new variables, like this.

h00.cpp

```
26
27      cout << STUDENT << "-" << ASSIGNMENT << ": ";
28      cout << "Cereal Box Calculator" << endl;
29      cout << string(50, '-') << endl;
30
31      // Input
32      cout << "Enter ounces per box of cereal: "; // prompt
33      double ouncesPerBox;                        // store the input
34      cin >> ouncesPerBox;                        // read the input
35
36      // Processing section
37      double weightInTons = ouncesPerBox / 35273.92;
38      double boxesPerTon = 1.0 / weightInTons;
39
40      // Output section
41      cout << "Weight in metric tons, boxes per ton: ["
42          << weightInTons << ", " << boxesPerTon << "]" << endl;
43
44      return 0;
45  }
```

A Small Improvement: Constants

Our program looks fine, but has one small flaw. We were **given** the number of ounces in a metric ton using the literal value **`35,273.92`**. However, our calculations **should not** use such **"magic numbers"**; they are too easy to mistype and they make code more confusing. Instead, **store all "given" values in named constants**.

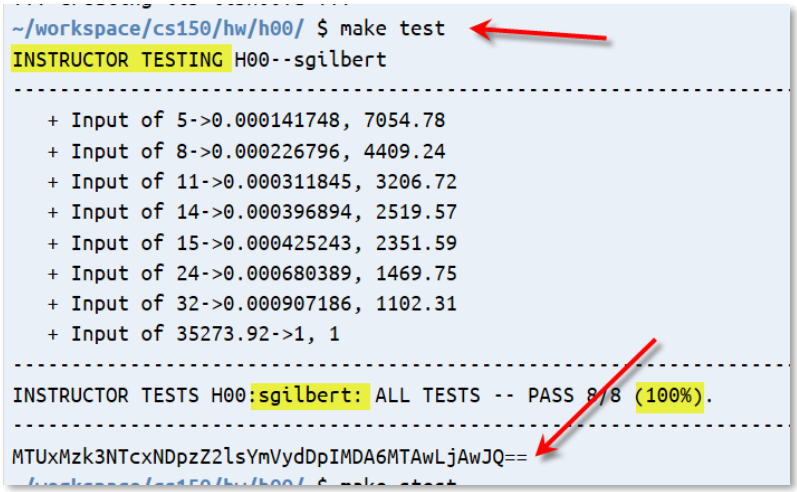
```
// Processing section
const double OUNCES_PER_TON = 35273.92;
double weightInTons = ouncesPerBox / OUNCES_PER_TON;
double boxesPerTon = 1.0 / weightInTons;
```

6. Testing & Submitting

To see if your program is correct, you need to **test** it. Your instructor has written one set of tests, and below, you'll see how you can write your own.

To run the instructor tests, type **`make test`** in the terminal. The **`run()`** function is called with different input values, and checked to see if you have the correct output.

The tests for this assignment calculated boxes from **5** to **32** ounces using Google Docs, and then adjusted the decimal places so that they match the default used by C++. The last entry was set to **`35273.92`** (the number of ounces in a metric ton), to double-check that you got **1** as the output for each calculation.



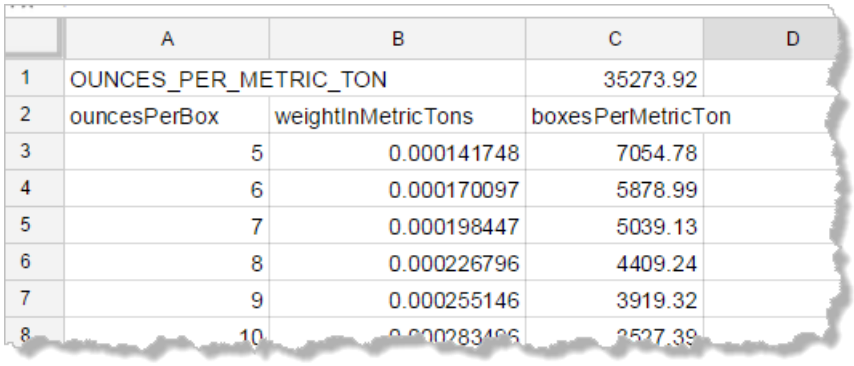
At the bottom of the run is the score. Make sure that **your ID** correctly appears, and that the assignment displayed is also correct. The final line is the completion code.

Submitting

To submit your work, type **`make submit`** from the console. You'll receive a confirmation if your submission is accepted. The CS 150 Homework Console allows you to check your scores and see about future deadlines. **Make sure that every time you submit, your score is recorded.** If you have difficulties, make sure you ask your questions on the Canvas Discussion Board.

Optional: Student Testing

If you want, you can also run your own tests. To do this, you need to supply several input values, and then figure out **exactly what the expected output should be**. The easiest way to do that is to use Excel or Google Docs like this:



	A	B	C	D
1	OUNCES_PER_METRIC_TON		35273.92	
2	ouncesPerBox	weightInMetricTons	boxesPerMetricTon	
3	5	0.000141748	7054.78	
4	6	0.000170097	5878.99	
5	7	0.000198447	5039.13	
6	8	0.000226796	4409.24	
7	9	0.000255146	3919.32	
8	10	0.000283496	3527.39	

Since our program hasn't formatted any of the output, so you might have to adjust the number of decimal places for each portion.

Adding the Tests

The **CS 150 framework** has a simple student testing scheme for IPO programs. Here's how it works.

- For each new input you want to test, **add a new line** to the file **`h00.tests`** (which you'll find in the folder with your starter code). If there are multiple inputs, then separate them with a space or a newline (`\n`).
- Add a vertical bar (`|`) to **separate the input from the expected output**, and then type the output that you want to check.
- The values being checked appear between square brackets (`[ ]`) in your program. If you have multiple outputs, they must all appear between a single set of brackets. **Do not** put the square brackets in your test file, however.
- To run the student tests, type **`make stest`**

When run like this, instead of reading the input from the keyboard, input will be read from the text file, and each line of input will be compared to the expected output.