CS 1313 010: Programming for Non-majors in C, Spring 2025
Programming Project #4: Fast Food Restaurant
Due by Wednesday March 26 2025 9:50am Central Time

This fourth project will give you experience writing a program that involves branching (`if`). This project will use the same development process as in Programming Projects #2 & #3, and will be subject to the same rules and grading criteria, with some new criteria added. This specification is less detailed than for previous programming projects. **YOU ARE EXPECTED TO KNOW HOW TO PERFORM BASIC TASKS WITHOUT HAVING TO BE TOLD EXPLICITLY.** For example, **YOU** should choose the filenames for your C source file and your script file, **BASED ON THE COMPARABLE FILENAMES IN PREVIOUS PROGRAMMING PROJECTS.**

IMPORTANT IMPORTANT IMPORTANT IMPORTANT IMPORTANT IMPORTANT!!!
This programming project specification contains many small code examples. In most cases, these code examples will be extremely useful in your actual PP#4. **WE URGE YOU TO USE THEM.**

I. PROJECT DESCRIPTION

You've just opened a fast food restaurant.

Each meal at your fast food restaurant can consist of an entree, a side dish, and a drink.

Your fast food restaurant serves several entrees (burger, chicken, fish), several side dishes (fries, rice, onion rings), and several drinks (coffee, ice tea, soda).

A customer can order **AT MOST** one item from each category; for example, they can order burger as their entree, fries as their side dish, and coffee as their drink. But, they **AREN'T REQUIRED** to order one of each category; for example, they can order no entree and no side dish at all, just a cup of coffee. They can even choose to order nothing at all. The prices are:

Item	Price	Item	Price
Burger	$7.25	Coffee	$2.25
Chicken sandwich	$8.50	Ice Tea	$2.50
Fish sandwich	$9.00	Soda	$2.00
Fries	$5.00		
Rice	$3.25		
Onion Rings	$6.50		

Your community's sales tax rate is 8.75%.

NOTE: Except where and as explicitly permitted in writing (for example, in a Programming Project specification), you are **ABSOLUTELY FORBIDDEN** to **COPY** **EVEN A SINGLE CHARACTER** from, or to have **ANY** shared code with, **ANY** other entity, whether a human being (regardless of whether in CS1313 or not), a text resource, a computing resource or anything else, whether in person, on a local computer, online or anywhere else. It's **INCREDIBLY EASY** for us to detect such copying, so **DON'T EVEN THINK ABOUT IT!**

You **ARE** allowed to use the code examples in this document and in "How to Comment if Blocks."

NOTE: In PP#4, you are **ABSOLUTELY FORBIDDEN** to use loops, arrays or user defined functions, which we haven't covered in lecture yet. (See the Grading Criteria for more detail.) And you will **ALWAYS** be **ABSOLUTELY FORBIDDEN** to use `goto` statements.

II. WHAT TO DO FIRST

Add the new program into your `makefile` in the usual way, as well as the example program (see below).

III. WHAT TO DO SECOND

For the example program in "if Lesson 1," slides #37-40:

Type in, compile and run that example program, using the input values on slides #41-43 of the same lecture slide packet.

Then, comment that example program, and compile and run it again, with the same inputs.

Then, create a script file for it, named

```
pp4_example.txt
```

IV. EXTRA PREPROCESSOR DIRECTIVE

Your C source file **MUST** start with the following **TWO** preprocessor directives, in this order:

```
#include <stdio.h>
#include <stdlib.h>
```

V. STRUCTURE OF THE PROGRAM

1. **Greeting subsection:** Greet the customer (user).
2. **Input subsection:** Prompt for and input their order, one item at a time, **IDIOTPROOFING EACH VALUE AS SOON AS IT IS INPUT**, with the idiotproof **IN ITS OWN `if` BLOCK, IMMEDIATELY AFTER THE `scanf` STATEMENT.**
3. **Calculation subsection:** Determine the price of each item, the subtotal, the tax amount, and the grand total.
4. **Output subsection:** Output their bill in receipt form.

Please note that you are **ABSOLUTELY FORBIDDEN** to have:

- **ANY** executable statements in your declaration section;
- **ANY** declarations in your execution section (body);
- **ANY** inputs or calculations in your greeting subsection;
- **ANY** calculations, or outputs other than prompts and idiotproofing error messages, in your input subsection;
- **ANY** inputs or outputs in your calculation subsection;
- **ANY** inputs or calculations in your output subsection.

That is, the subsections **MUST BE COMPLETELY SEPARATE,** and **MUST BE CLEARLY LABELED WITH COMMENTS.**

For this programming project, `if` blocks are not considered to be inputs, nor calculations, nor outputs; that is, in principle you may have an `if` block in **ANY** subsection of the program body. However, statements **inside** the clauses of an `if` block **MUST** follow the rules above.

VI. IMPLEMENTATION ORDER

Because the program will be complicated, you are **STRONGLY** advised to **IMPLEMENT ONE PART AT A TIME,** thoroughly test and debug it, and then go on to the next part. Also, it would probably be best to implement the subsections, **NOT** in the order in which they appear in the program (as described above), but rather in the following order:

1. **Greeting subsection.**
2. **Input subsection** (developed one item at a time). **NOTE:** If you're unclear on how to idiotproof, then you can skip the idiotproofing during initial implementation and then develop the idiotproofing code later.
3. **Output subsection** (developed one item at a time). Note that, at this stage, some or all of your outputs will be garbage, because you haven't yet written the calculation subsection.
4. **Calculation subsection** (developed one item at a time), located between the input and output subsections.

VII. DETAILS OF THE PROGRAM STRUCTURE

IMPORTANT IMPORTANT IMPORTANT IMPORTANT IMPORTANT IMPORTANT!!!
ADVICE: Avoid using ambiguous names for variables and named constants. Specifically, in this project, **DON'T** use names such as `drink` or `coffee`. Instead, use names that **CLEARLY** state the **ROLE** of the variable or named constant, such as `drink_item_code` or `coffee_price`.

Your program should have the following components, in the following order.

A. Greet the Customer

Welcome the customer to the restaurant.

B. Input the Customer's Order

1. **Ask (prompt) the customer for their entree item choice,** giving them a list of entree items to choose from. You can use integer-valued codes to represent the entree items, and you may choose any **REASONABLE** values for coding these items. So, when the program prompts the customer for their entree item choice, the **OUTPUT** might be something like:

   ```
   What entree item would you like?
   Please enter:
     0 for no entree
     1 for burger
     2 for chicken sandwich
     3 for fish sandwich
   ```

2. **Input** the customer's entree item choice.
3. **IDIOTPROOF** the entree item choice, to ensure that the value that the user has input is one of the values listed in the prompt. (See the Grading Criteria for details.)

4. **Ask (prompt) the customer for their side dish item choice,** giving them a list of side dish items to choose from. Again, you can use integer-valued codes to represent the side dish items, and you may choose any **REASONABLE** values for coding these side dish items. So, when the program prompts the customer for their side dish item, the **OUTPUT** might be something like:

```
What side dish would you like?
Please enter:
  0 for no side dish
  1 for fries
  2 for rice
  3 for onion rings
```

5. **Input** the customer's side dish item choice.
6. **IDIOTPROOF** the side dish item choice, to ensure that the value that the user has input is one of the values listed in the prompt. (See the Grading Criteria for details.)
7. **Ask (prompt) the customer for their drink item choice.** Again, you can use integer-valued codes to represent the drink item choices, and you may choose any **REASONABLE** values for coding these items. So, when the program prompts the customer for their drink item choice, the **OUTPUT** might be something like:

```
What drink item would you like?
Please enter:
  0 for no drink
  1 for coffee
  2 for ice tea
  3 for soda
```

8. **Input** the customer's drink item choice.
9. **IDIOTPROOF** the drink item choice, to ensure that the value that the user has input is one of the values listed in the prompt. (See the Grading Criteria for details.)

NOTE: YOU MUST FULLY IDIOTPROOF EVERY INPUT THAT NEEDS IDIOTPROOFING. YOU ARE RESPONSIBLE FOR DETERMINING ALL POSSIBLE FORMS OF IDIOCY. Idiotproofing error messages **MUST** be **HELPFUL** and sufficiently detailed that even an idiot could figure out **SPECIFICALLY** what they've done wrong.

Each idiotproof should be **IN ITS OWN `if` BLOCK, IMMEDIATELY AFTER THE ASSOCIATED `scanf` STATEMENT.** You are **ABSOLUTELY FORBIDDEN** to have **ANY** other statement(s) between the `scanf` statement and its associated idiotproofing `if` block (though you will have comments between them).

If you haven't yet learned how to idiotproof (we'll get to it in lecture while you're working on PP#4), then work on the rest of your program, and come back to the idiotproofing once you've learned how to idiotproof.

NOTE: If the customer chooses to buy nothing — no entree, no side dish, no drink — then thank them and **EXIT** the program with a return code of zero (using an appropriate named constant), rather than calculating and printing an empty bill.

C. Calculate the Bill

1. **Entree price:** This value isn’t calculated as such; it’s obtained from the entree item choice.
2. **Side dish price:** This value isn’t calculated as such; it’s obtained from the side dish item choice.
3. **Drink price:** This value isn’t calculated as such; it’s obtained from the drink item choice.
4. **Subtotal:** Calculate the subtotal amount of the food and drink.
5. **Tax amount:** Calculate the amount of tax on the food and drink. Both food and drink are taxed at the same tax rate (8.75%).
6. **Grand Total:** The grand total bill is the sum of the entree price, the side dish price, the drink price, the tax amount.

D. Print the Bill

1. Present an itemized bill in receipt form, using the placeholder below. For example:

   ```
   ---------------------------------------------------
   Henry’s Fast Food -- Receipt
   ---------------------------------------------------

     Chicken Sandwich:           $ 8.50
     Rice:                       $ 3.25
     Ice Tea:                    $ 2.50
     ----------------------------------
     Food Total:                 $14.25
     Tax:                        $ 1.25
     ----------------------------------
     Grand Total:                $15.50

   Thank you for visiting Henry’s Fast Food!
   ---------------------------------------------------
   ```

   (Substitute the name of your restaurant at the top and bottom.)
2. For all of the outputs in the itemized list (from the entree through the total), use the `printf` placeholder `%5.2f`, like so:

   ```
   printf("  Steak:              $%5.2f\n",
   ```

   WE URGE YOU TO USE THE CODE JUST ABOVE IN YOUR PP#4!

   The *conversion format* in the placeholder tells the compiler that the `printf` statement will output some literal text, followed by a floating point number that takes up at least five spaces, two of which are to the right of the decimal point.
3. Item names **MUST** line up on the left side of the bill, dollar signs **MUST** line up, and the `printf` placeholder will cause prices to be flush to the right of the line. For the conversion format to work, **all dollar amounts MUST** be `float`.
4. List the bill entries in the order shown, using the name of each menu choice (table, page 1).
5. In some cases, the exact tax amount will have more than two digits to the right of the decimal point. Because of rounding, we will accept results within five cents of exactly correct.

VIII. RUNS

In your script, run the program 5 times, using the following inputs, in the following order:

1. no entree, no side dish, no drink
2. burger, fries, coffee
3. chicken sandwich, rice, ice tea
4. fish sandwich, onion rings, soda
5. no entree, no side dish, coffee

In addition, **RUN THE PROGRAM ONCE FOR EACH POSSIBLE CASE OF IDIOCY** that a user might exhibit; that is, you **MUST** have runs that **COMPLETELY TEST EACH AND EVERY IDIOTPROOF CHECK. YOU ARE RESPONSIBLE FOR DETERMINING ALL POSSIBLE FORMS OF IDIOCY.** In your script file, the idiotproof test runs **MUST** occur **AFTER** the runs listed above.

ADVICE: Calculate each (non-idiotproof) run's result by hand, then compare your hand-calculated values to the output of the program, to determine whether the program is running correctly.

IX. ADDITIONAL GRADING CRITERIA

NOTE: ALL grading criteria introduced in any Programming Project apply to **ALL** future PPs, unless explicitly stated otherwise.

A. SUBJECTIVE GRADING OF COMMENTS IN THE PROGRAM BODY

In previous programming projects, one of the grading criteria for comments in the program body has been that **EVERY** statement in the program body had to be preceded by a clear, helpful explanatory comment.

- For PP#4 and beyond, you may choose to write fewer comments than this (though still in the format described in the PP#4 specification), in which case **YOU AGREE TO ACCEPT WITHOUT ARGUMENT** the graders' **SUBJECTIVE** opinion on whether the amount and nature of your comments is sufficient.
- Alternatively, you may choose to continue to comply with the old criterion, preceding **EVERY** statement in the program body with a clear, helpful explanatory comment, in which case you are guaranteed to get full credit for this aspect of documentation in the program body (assuming that your comments comply with the original grading criteria for comments in the PP#4 specification).

B. NEW GRADING CRITERIA

1. **Format** of `if` statements, `else if` statements and `else` statements:
   For each `if` statement, the `if` keyword **MUST** be followed by a blank space and then the open parenthesis that begins the condition. After the close parenthesis that ends the condition, there **MUST** be a blank space, followed by the block open.
   For each `else if` statement, the same.
   For each `else` statement, there **MUST** be a single blank space between the `else` keyword and the block open. For example:

   ```
   if (drink_item_code == no_item_code) {
       drink_price = no_item_price;
   } /* if (drink_item_code == no_item_code) */
   else if (drink_item_code == coffee_item_code) {
       drink_price = coffee_price;
   } /* if (drink_item_code == coffee_item_code) */
   else if (drink_item_code == ice_tea_item_code) {
       drink_price = ice_tea_price;
   } /* if (drink_item_code == ice_tea_item_code) */
   else if (drink_item_code == soda_item_code) {
       drink_price = soda_price;
   } /* if (drink_item_code == soda_item_code) */
   ```

   WE URGE YOU TO USE THE CODE JUST ABOVE IN YOUR PP#4!
2. **Block open**: No source code text on the same line as, and after, a block open.
3. **Block close**: Only comment text on same line after a block close (see “Commenting `if` Blocks”).
4. **Format** of `if` conditions and `else if` conditions:
   For each `if` statement and each `else if` statement:
   (a) In the condition, each binary operator — including relational operators such as `==` and Boolean operators such as `&&` — **MUST** be surrounded by one or more blank spaces on each side.
   (b) In the condition, each unary operator such as `!` **MUST** have either a blank space or an open parenthesis to its left, but **NO** blank space nor parenthesis to its right.
   (c) If the condition has multiple subexpressions, then each subexpression **MUST** be on a line by itself, and the subexpressions **MUST** line up with each other.

   For example:

   ```
   if ((drink_item_code != no_item_code)      &&
       (drink_item_code != coffee_item_code)  &&
       (drink_item_code != ice_tea_item_code) &&
       (drink_item_code != soda_item_code)) {
       printf("ERROR: unknown drink item code %d.\n",
           drink_item_code);
       exit(program_failure_code);
   } /* if ((drink_item_code != no_item_code) && ...) */
   ```

   WE URGE YOU TO USE THE CODE JUST ABOVE IN YOUR PP#4!

5. **Indenting OF `if` blocks:**
   For a given `if` block, the `if` statement, all `else if` statements (if any), the `else` statement (if any) and the block closes associated with these statements **MUST** all line up, both with each other and with other statements as appropriate. For example:

```
scanf("%d", &side_dish_item_code);
if ((side_dish_item_code != no_item_code)    &&
    (side_dish_item_code != fries_item_code) &&
    (side_dish_item_code != rice_item_code)  &&
    (side_dish_item_code != onion_rings_item_code)) {
    printf("ERROR: unknown side dish item code %d.\n",
        side_dish_item_code);
    exit(program_failure_code);
} /* if ((side_dish_item_code != no_item_code) && ...) */
printf("What drink item would you like?\n");
```

   WE URGE YOU TO USE THE CODE JUST ABOVE IN YOUR PP#4!

6. **Indenting INSIDE `if` blocks:**
   For a given `if` block, all statements **INSIDE** any clause of the `if` block **MUST** be indented **FOUR SPACES** farther than the associated `if` statement. This applies especially to `if` blocks nested inside other `if` blocks. For example:

```
if (drink_item_code == coffee_item_code) {
    printf("  Coffee:             $%5.2f\n",
        drink_price);
} /* if (drink_item_code == coffee_item_code) */
else if (drink_item_code == ice_tea_item_code) {
    printf("  Ice Tea:            $%5.2f\n",
        drink_price);
} /* if (drink_item_code == ice_tea_item_code) */
else if (drink_item_code == soda_item_code) {
    printf("  Soda:               $%5.2f\n",
        drink_price);
} /* if (drink_item_code == soda_item_code) */
```

   WE URGE YOU TO USE THE CODE JUST ABOVE IN YOUR PP#4!

7. **Commenting `if` blocks:** You **MUST** follow all of the rules that are specified in the packet titled “Commenting `if` Blocks,” which will be posted on the course website soon.

8. **Idiotproofing**
   (a) **ALL** inputs **MUST** include idiotproofing checks. Here's a typical idiotproofing check:

   ```
   if ((drink_item_code != no_item_code)       &&
        (drink_item_code != coffee_item_code)   &&
        (drink_item_code != ice_tea_item_code)  &&
        (drink_item_code != soda_item_code)) {
        printf("ERROR: unknown drink item code %d.\n",
            drink_item_code);
        exit(program_failure_code);
   } /* if ((drink_item_code != no_item_code) && ...) */
   ```

   WE URGE YOU TO USE THE CODE JUST ABOVE IN YOUR PP#4!

   Notice the `exit` statement, which causes the program to immediately halt. **ALL** idiotproofing checks **MUST** include an `exit` statement. The `exit` statement is **INSIDE** the `if` block, and should be indented appropriately (see above). Also, your program **MUST** have a named constant `program_failure_code` whose value is `-1`.

   (b) Each idiotproofing check **MUST occur IMMEDIATELY after the associated `scanf`**. **ALL IDIOTPROOFING MUST BE COMPLETED BEFORE ANY CALCULATIONS ARE PERFORMED**; that is, idiotproofing belongs in the input subsection.

   (c) Idiotproof error messages **MUST CLEARLY AND UNAMBIGUOUSLY** state the nature of the error. Thus, no two error messages should be the same.

   (d) Some idiotproof checks could depend on whether the item to be idiotproofed was actually input. You **MUSTN'T** idiotproof variables that haven't been input.

   (e) Each idiotproof should be **IN ITS OWN `if` BLOCK, IMMEDIATELY AFTER THE ASSOCIATED `scanf` STATEMENT.** You are **ABSOLUTELY FORBIDDEN** to have **ANY** other statement(s) between the `scanf` statement and its associated idiotproofing `if` block (though you will have comments between them).

9. For PP#4, you are allowed to use **ONLY** the following kinds of programming constructs in your C source file, but **NO OTHER KINDS OF PROGRAMMING CONSTRUCTS**:

   - all programming constructs allowed in PP#2 and PP#3;
   - `if` blocks, including `if` clauses, `else if` clauses and `else` clauses;
   - exit statements.

   Use of any other kind of programming constructs in your C source file might result in **SEVERE PENALTIES, UP TO 50% OFF BEFORE ANY OTHER DEDUCTIONS ARE APPLIED,** at the instructor's sole discretion.

X. WHAT TO SUBMIT

Upload to Canvas in the usual way, in the usual style and format, the summary essay, example script file, C source file and script file.

HELP SESSION BONUS EXTRA CREDIT

You can receive an extra credit bonus of as much as 5% of the total value of PP#4 as follows:

1. Attend at least one regularly scheduled CS1313 help session for at least 30 minutes, through Wed March 26.
2. During the regularly scheduled help session that you attend, work on CS1313 assignments (ideally PP#4, but any CS1313 assignment is acceptable). **YOU CANNOT GET EXTRA CREDIT IF YOU DON'T WORK ON CS1313 ASSIGNMENTS DURING THE HELP SESSION.**

BONUS VALUE NOTICE: Through Tue March 11, the extra credit bonus will be worth **5%** of the total value of PP#4; from Mon March 24 through Wed March 26, the extra credit bonus will be worth **only 2.5%** of the total value of PP#4. That is, **YOU'LL GET TWICE AS MUCH EXTRA CREDIT DURING THE FIRST TWO WEEKS AS DURING THE FINAL WEEK.**

NOTE: This extra credit bonus **WON'T** be available on any other programming project unless explicitly stated so in the project's specification.

PP#4 CHECKLIST

- □ Example program: I typed in, compiled, ran, comments, recompiled, reran, and created a script for, the example program, (as described in the PP#4 specification, page 2, section III).
- □ `#include` directives: I used the correct `#include` directives, in the correct order (as described in the PP#4 specification, page 2, section IV).
- □ Named constants: I declared several named constants, for prices, item codes and tax rate (as described in the PP#4 specification, page 1).
- □ Named constant names and variable names: My named constant names and variable names are sufficiently specific that I can easily tell what they refer to which are prices, which are item codes and so on, for example `coffee_price` and `drink_item_code` (as described in the PP#4 specification, page 3, section VII, the **ADVICE**).
- □ Program structure: I used the correct program structure (as described in the PP#4 specification, page 2, section V).
- □ Implementation order: I implemented the subsections of my progam's execution section (body) in the correct order (as described in the PP#4 specification, page 3, section VI).
- □ Greeting: I wrote an appropriate greeting (as described in the PP#4 specification, page 3, item VII.A).
- □ Entree item prompt: I wrote an appropriate prompt for the entree item code (as described in the PP#4 specification, page 3, item VII.B.1).
- □ Entree item input: I wrote an appropriate input for the entree item code (as described in the PP#4 specification, page 3, item VII.B.2).
- □ Entree item idiotproof: I wrote an appropriate idiotproof for the entree item code (as described in the PP#4 specification, page 3, item VII.B.3, and page 9, grading criterion 8a).
- □ Side dish item prompt: I wrote an appropriate prompt for the side dish item code (as described in the PP#4 specification, page 4, item VII.B.4).
- □ Side dish item input: I wrote an appropriate input for the side dish item code (as described in the PP#4 specification, page 4, item VII.B.5).
- □ Side dish item idiotproof: I wrote an appropriate idiotproof for the side dish item code (as described in the PP#4 specification, page 4, item VII.B.6).
- □ Drink item prompt: I wrote an appropriate prompt for the drink item code (as described in the PP#4 specification, page 4, item VII.B.7).
- □ Drink item input: I wrote an appropriate input for the drink item code (as described in the PP#4 specification, page 4, item VII.B.8).
- □ Drink item idiotproof: I wrote an appropriate idiotproof for the drink item code (as described in the PP#4 specification, page 4, item VII.B.9, and page 9, grading criterion #8a).
- □ Test for buying nothing: I wrote appropriate code to test whether the user ordered nothing, in which case they would be thanked, and the program would exit without printing a bill (as described in the PP#4 specification, the bottom of page 4, the **NOTE** just before section VII.C).
- □ Determine entree price: I wrote appropriate code to determine the entree price (as described in the PP#4 specification, page 5, item VII.C.1).
- □ Determine side dish price: I wrote appropriate code to determine the side dish price (as described in the PP#4 specification, page 5, item VII.C.2).

□ Determine drink price: I wrote appropriate code to determine the drink price (as described in the PP#4 specification, page 5, item VII.C.3, and the example on page 7, grading criterion 1).

□ Calculate subtotal: I wrote appropriate code to calculate the subtotal (as described in the PP#4 specification, page 5, item VII.C.4).

□ Calculate tax amount: I wrote appropriate code to calculate the tax amount (as described in the PP#4 specification, page 5, item VII.C.5).

□ Calculate grand total: I wrote appropriate code to calculate the grand total (as described in the PP#4 specification, page 5, item VII.C.6).

□ Output bill header: I wrote appropriate code to output the header of the bill (as described in the PP#4 specification, page 5, item VII.D.1).

□ Output entree name and its price: I wrote appropriate code to output the entree name and its price (as described in the PP#4 specification, page 5, item VII.D.1).

□ Output side dish name and its price: I wrote appropriate code to output the side dish name and its price (as described in the PP#4 specification, page 5, item VII.D.1). and the example on page 8, grading criterion 3).

□ Output drink name and its price: I wrote appropriate code to output the drink name and its price (as described in the PP#4 specification, page 5, item VII.D.1).

□ Output subtotal: I wrote appropriate code to output the subtotal (as described in the PP#4 specification, page 5, item VII.D.1).

□ Output tax amount: I wrote appropriate code to output the tax amount (as described in the PP#4 specification, page 5, item VII.D.1).

□ Output grand total: I wrote appropriate code to output the grand total (as described in the PP#4 specification, page 5, item VII.D.1).

□ Output bill footer: I wrote appropriate code to output the footer of the bill (as described in the PP#4 specification, page 5, item VII.D.1).

□ Format of dollar figures in output: In my outputs of various dollar amounts, I used the appropriate conversion format in my placeholder (as described in the PP#4 specification, page 5, item VII.D.2).

□ Dollar figures in output line up: In my outputs of various dollar amounts, my dollar figures line up properly (as described in the PP#4 specification, page 5, item VII.D.3).

□ Order of outputs: In my outputs of various dollar amounts, I output them in the appropriate order (as described in the PP#4 specification, page 5, item VII.D.4).

□ Accuracy of outputs: In my outputs of various dollar amounts, my outputs are correct to within 5 cents (as described in the PP#4 specification, page 5, item VII.D.5).

□ Runs: In my script file, I did the correct runs in the correct order (as described in the PP#4 specification, page 6, section VIII).

□ Idiotproof runs: In my script file, I did all appropriate idiotproof runs, which all come after my regular runs (as described in the PP#4 specification, page 6, section VIII).

□ Calculation by hand: To verify that my program is correct, I calculated by hand the correct bill for each test run, and checked that result against my runs (as described in the PP#4 specification, page 6, section VIII).

□ Format of `if` statements: In my program, my `if` statements have the correct format (as described in the PP#4 specification, page 7, grading criterion 1).

□ No source code text on same line after a block open: In my program, my `if` statements have no source code text on the same line after the block open (as described in the PP#4 specification, page 7, grading criterion 2).

□ Only comment text on same line after a block close: In my program, my `if` block closes have no source code text on the same line after the block close except the comment that labels the block close (as described in the PP#4 specification, page 7, grading criterion 3).

□ Format of `if` conditions and `else if` conditions: In my program, my `if` conditions and my `else if` conditions have the correct format (as described in the PP#4 specification, page 7, grading criterion 4).

□ Indenting OF `if` blocks: In my program, my `if` blocks are properly indented (as described in the PP#4 specification, page 8, grading criterion 5).

□ Indenting INSIDE `if` blocks: In my program, statements inside my `if` blocks are properly indented (as described in the PP#4 specification, page 8, grading criterion 6).

□ Commenting `if` blocks: In my program, the block closes of my `if` blocks are properly labeled with comments on the same line (as described in "Commenting `if` Blocks").

□ Idiotproofing of all inputs: In my program, every input has an idiotproof (as described in the PP#4 specification, page 9, grading criterion 8a).

□ Idiotproofing `exit` statement: In my program, every idiotproof has an `exit` statement (as described in the PP#4 specification, page 9, grading criterion 8a).

□ Idiotproofing `exit` statement inside `if` block: In my program, every idiotproof's `exit` statement is inside the idiotproof `if` block (as described in the PP#4 specification, page 9, grading criterion 8a).

□ Idiotproofing `exit` statement indented properly: In my program, every idiotproof's `exit` statement is indented properly (as described in the PP#4 specification, page 9, grading criterion 8a).

□ Idiotproofing `exit` statement uses `program_failure_code`: In my program, every idiotproof's `exit` statement takes the argument `program_failure_code`, which is initialized to the correct value (as described in the PP#4 specification, page 9, grading criterion 8a).

□ Idiotproofing immediately after input: In my program, every idiotproof occurs immediately after the associated input (as described in the PP#4 specification, page 9, grading criterion 8b).

□ Idiotproofing error messages: In my program, every idiotproof error message is unique and unambiguous (as described in the PP#4 specification, page 9, grading criterion 8c).

□ Idiotproofing only variables that have been input: In my program, I only idiotproof variables that have been input (as described in the PP#4 specification, page 9, grading criterion 8d).

□ Each idiotproof in its own `if` block: In my program, I only idiotproof variables that have been input (as described in the PP#4 specification, page 9, grading criterion 8e).

□ Only approved programming constructs: In my program, I only only used the approved kinds of programming constructs (as described in the PP#4 specification, page 9, grading criterion 9).

□ Uploads: I've uploaded the correct files to the Canvas PP#4 dropbox (as described in the PP#4 specification, page 9, section X).