This fourth project will give you experience writing programs that involve branching (if). This project will use the same development process as in Programming Projects #1, #2 and #3, and will be subject to the same rules and grading criteria, with a few new criteria added.

This specification will be less detailed than for previous programming projects. **YOU ARE EXPECTED TO KNOW HOW TO PERFORM BASIC TASKS** without having to be told explicitly, based on your experience with previous programming projects.

I. PROJECT DESCRIPTION

You’ve just opened the “Imperial Family” Chinese restaurant.

This Chinese restaurant has a menu that features several appetizers (pot stickers, spring roll, fried wonton), several entrees (stir fried beef, stir fried chicken and stir fried tofu), several starches (steamed rice, fried rice and noodles) and several drinks (soda, tea and coffee).

Each meal can consist of an appetizer, an entree, a starch and a drink.

A patron can order **AT MOST** one item from each category; for example, they can order a spring roll as their appetizer, chicken as their entree, steamed rice as their starch and tea as their drink.

The prices are:

<table>
<thead>
<tr>
<th>Item</th>
<th>Price</th>
<th>Item</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pot Stickers</td>
<td>$3.00</td>
<td>Soda</td>
<td>$2.00</td>
</tr>
<tr>
<td>Spring Roll</td>
<td>$2.50</td>
<td>Tea</td>
<td>$1.50</td>
</tr>
<tr>
<td>Fried Wonton</td>
<td>$3.25</td>
<td>Coffee</td>
<td>$1.75</td>
</tr>
<tr>
<td>Stir Fried Beef</td>
<td>$5.00</td>
<td>Steamed Rice</td>
<td>$2.00</td>
</tr>
<tr>
<td>Stir Fried Chicken</td>
<td>$5.00</td>
<td>Fried Rice</td>
<td>$2.50</td>
</tr>
<tr>
<td>Stir Fried Tofu</td>
<td>$3.00</td>
<td>Noodles</td>
<td>$3.00</td>
</tr>
</tbody>
</table>

Your community’s sales tax rate is 7.5%.

**NOTE:**

You can not substitute other items, i.e. you must use the items given.

Write a program that simulates greeting a customer, taking their order, calculating the bill and producing the check. The computer will be the wait staff, and the user will be the customer.
II. STRUCTURE OF THE PROGRAM

The program body **MUST** be broken into four subsections:

1. **Greeting subsection:** Greet the customer.
2. **Input subsection:** Take (input) their order, **IDIOTPROOFING EACH VALUE AS SOON AS IT IS INPUT.**
3. **Calculation subsection:** Calculate their bill.
4. **Output subsection:** Output their bill in receipt form.

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

- **ANY** declarations in your program 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.**

For purposes of this programming project, *if* blocks are not considered to be inputs, nor calculations, nor outputs; that is, in principle you can 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.

III. 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).
IV. DETAILED DESCRIPTION

A. Greet the Customer
Welcome the customer to the restaurant.

B. Take the Customer’s Order

1. **Ask the customer for their appetizer choice**, giving them a list of items to choose from.
   You must use the given integer values for coding these choices. When the program prompts them for their appetizer order, it should **OUTPUT** the following:
   
   What appetizer would you like?
   Please enter:
   1 for pot stickers
   2 for spring roll
   3 for fried wonton

2. **IDIOTPROOF** the appetizer 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.)

3. **Ask the customer for their entree choice**, giving them a list of items to choose from.
   You must use the given integer values for coding these choices.
   
   When the program prompts them for their entree order, it should **OUTPUT** the following:
   
   What entree would you like?
   Please enter:
   1 for stir fried beef
   2 for stir fried chicken
   3 for stir fried tofu

4. **IDIOTPROOF** the entree 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.)

5. **Ask the customer for their starch choice**, giving them a list of items to choose from.
   You must use the given integer values for coding these choices.
   
   When the program prompts them for their starch order, it should **OUTPUT** the following:
   
   What starch would you like?
   Please enter:
   1 for steamed rice
   2 for fried rice
   3 for noodles

6. **IDIOTPROOF** the starch 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 the customer for their drink choice**, giving them a list of items to choose from.
   You must use the given integer values for coding these choices.
   
   When the program prompts them for their drink order, it should **OUTPUT** the following:
What drink would you like?  
Please enter:
  1 for soda
  2 for tea
  3 for coffee

8. **IDIOTPROOF** the drink 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. 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. (See the Grading Criteria for details.)

**ADVICE:** Avoid using ambiguous names for variables and named constants. Specifically, in this project, DON’T use names such as tofu or tea. Instead, use names that clearly specify the ROLE that the variable or named constant plays, such as tofu code or tea price.

**C. Calculate the Bill**

1. **Appetizer price:** This value is not calculated as such, but is obtained from the appetizer choice.
2. **Entree price:** This value is not calculated as such, but is obtained from the entree choice.
3. **Starch price:** This value is not calculated as such, but is obtained from the starch choice.
4. **Drink price:** This value is not calculated as such, but is obtained from the drink choice.
5. **Sales tax amount:**
   - (a) Food is taxed at the sales tax rate (7.5%).
   - (b) Drinks are also taxed at the sales tax rate (7.5%).
6. **Tip amount:** The restaurant automatically adds a tip of 15% of the total food and drink bill, *excluding tax* — that is, the tip is calculated based on the sum of the appetizer price, the entree price, the starch price, and the drink price, **NOT** on the sum of the appetizer price, the entree price, the drink price and the sales tax amount.
7. **Total:** The total bill is the sum of the food price, the drink price, the tax amount, and the tip amount.
D. Print the Bill

1. Present the customer with an itemized bill in receipt form. For example:


Imperial Family Bill

Your order was:
Pot Sticker: $ 3.00
Stir Fried Chicken: $ 5.00
Noodles: $ 3.00
Coffee: $ 1.75

Food Total: $12.75
Tax: $ 0.96
Tip: $ 1.91

Grand Total: $15.62

Thank you for visiting Imperial Family!

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(" Stir fried chicken: $%5.2f\n", entree_price);

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 characters’ worth of width, two of which are to the right of the decimal point.

3. Item names MUST line up on the left side of the bill. 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 and tip amounts will have more than two digits to the right of the decimal point. We will accept results that are within five cents of the exactly correct bill.
V. RUNS

In your script, perform each of the following runs:
1. spring roll, stir fried chicken, steamed rice, tea
2. fried wonton, stir fried beef, fried rice, 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.

VI. ADDITIONAL 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.

2. Format of if conditions and else if conditions:
   For each if or else if statement, in the condition, any binary operators — including relational operators such as == and Boolean operators such as && — MUST be surrounded by one or more blank spaces on each side. Unary operators such as ! MUSTN’T be surrounded by blank spaces.

3. Indenting of if blocks:
   For each if block, the if statement, all else if statements (if any), the else statement (if any) and the associated block closes MUST all line up, both with each other and with other statements as appropriate. For example:

   ```c
   scanf("%d", &meal_code);
   if ((meal_code != lunch_code) &&
       (meal_code != dinner_code)) {
       printf("ERROR: unrecognized meal code %d.\n",
               meal_code);
       exit(program_failure_code);
   } /* if ((meal_code != child) && ... */
   printf("What appetizer would you like?\n");
   ```

4. Indenting inside if blocks:
   For each if block, all statements INSIDE the if block MUST BE INDENTED FOUR SPACES farther than the associated if statement. See the example above.

5. Commenting if blocks: You MUST follow all of the rules that are specified in the packet titled “Commenting if Blocks,” which is posted on the course website.

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

```c
if ((meal_code != lunch_code) &&
    (meal_code != dinner_code)) {
    printf("ERROR: unrecognized meal code %d.\n", meal_code);
    exit(program_failure_code);
} /* if ((meal_code != child) && ... */
```

Notice the `exit` statement, which causes the program to immediately terminate. **ALL** idiotproofing checks **MUST** include an `exit` statement. Therefore, 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` statement. **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.

**VII. WHAT TO SUBMIT**
Submit cover, summary essay, and script in the usual style, format and order, and upload your source file to Desire2Learn in the usual way.