

Member ID: \_\_\_\_\_

Time: \_\_\_\_\_

Rank: \_\_\_\_\_



# C# PROGRAMMING

## (330)

## NATIONAL 2025

**PRODUCTION:**

National\_CSharp

\_\_\_\_\_ (715 points)

**INDUSTRY CERTIFICATION (90 minutes)**

IT Specialist Software Development

**Test Time: 90 minutes**

## GENERAL GUIDELINES:

*Failure to adhere to any of the following rules will result in disqualification:*

1. Member must hand in this test booklet and all printouts if any. Failure to do so will result in disqualification.
2. No equipment, supplies, or materials other than those specified for this event are allowed in the testing area. No previous BPA tests and/or sample tests (handwritten, photocopied, or keyed) are allowed in the testing area.
3. Electronic devices will be monitored according to ACT standards.

### **PART ONE Instructions: Production**

Once you have completed listed below, upload all competition materials (including project files, etc.) to the National BPA Pre-submit System at [presubmit.bpa.org](https://presubmit.bpa.org).

Compress all files in a single archive (zip file) and use the following naming convention for your documents:

#### **CS-MemberID.zip**

##### **Development Standards**

- Standard name prefixes must be utilized for variables.
- All subroutines, functions, and methods must be documented with comments explaining the purpose of the method, the input parameters (if any), and the output (if any).

### **PART TWO Instructions: Industry Certification**

This exam requires IT Specialist Software Development Certification. Please do not leave your seat or exit the room until you have completed the certification. After the certification has been administered, you are permitted to leave.

Your name and/or school name should *not* appear on work you submit for grading.

1. Create a folder on the flash drive provided using your contestant number as the name of the folder.
2. Copy your entire solution/project into this folder. The project folder for you has already been provided: National\_CSharp
3. Submit your entire solution/project so that the graders may open your project to review the source code.
4. Ensure that the files required to run your program are present and will execute on the flash drive provided.
5. You will need to use Visual Studios 2019 or greater to complete this exam.

\*Note that the flash drive letter may *not* be the same when the program is graded as it was when you created the program.

\*It is recommended that you use relative paths rather than absolute paths to ensure that the program will run regardless of the flash drive letter. It is **HIGHLY** recommended that you place all of the files into one folder.

The graders will *not* compile or alter your source code to correct for this.  
Submissions that do *not* contain source code will *not* be graded.

#### **Assumptions to make when taking this assessment:**

- The goal of this assessment is to create a Windows Form App that will allow the user to generate a list from specific text files in a folder. The txt files will not need to be adjusted.
- The program will have to be able to only select files that have the word “Text” somewhere in the file name. There are txt files in the folder that do not meet this requirement, your program should ignore these files.
- The txt file import will require creating two separate objects based upon a flag that precedes the fields for each record. The delimiter for the fields will be commas ( , ).
- A Visual Studio project folder with the main project file is provided in the contest folder (RestaurantSupplier\_National.sln).
- All of the object classes and helper classes have been created for you; however, you will still be required to program certain actions in those classes. In addition, the form design has been already created; however, there might be an instance where you must adjust the properties and actions of the objects on the forms.
- All interactive objects on the form will be text boxes and labels. You will be making adjustments to the panel on the form.
- The user will be allowed to type non case sensitive input into the text boxes to cause the form to perform actions. Data entry controls will be a priority.
- All dollar values (US Dollars) and percentages will need to be formatted appropriately.
- The estimated value of the entire restaurant supplier company is a sum of all the Total Value calculations for each food-product type. The **Total Value = price \* total kilograms**. The estimated value and Total Value are represented by \$’s in US Currency.
- Once the list is displayed, it will stay active on the form until the form is closed.

### Development Standards:

- Your Code must use a consistent variable naming convention.
- All subroutines (if any), functions (if any), and methods (if any) must be documented with comments explaining the purpose of the method, the input parameters (if any), and the output (if any). Readability is a goal of good code.

### Commenting for Source Code Review (see the rubric):

- Certain sections of your code will be graded. These gradable blocks of code can range from creating data structures, method algorithms, exception handling, and class construction.
- The grading rubric contains a section called Source Code Review: in this section are listed a description all of the graded programming concepts.
- Each gradable item must have a comment listed at its beginning, and the comment must be prefixed with the comment flag. The flag helps the graders easily locate the code to increase the effectiveness of grading.
- The flag will always use this naming convention: **SC#** (NOTE: the # symbol will be replaced with sequential numbering, i.e. **SC1**, **SC2**, **SC3**, etc.
- No explanation in the comment with the flag is required, only the comment flag; however, any information placed in the comment could help the grader better understand and avoid any costly errors.
- The comment flag needs to be placed in close proximity to the block of code it represents.
- If a comment flag is not present, you will not receive credit.
- In this example the Source Code Review has a gradable section of code for printing to the console (Remember these are non-related examples):
  - SC12: **WriteLine** method in the **Form1** class is printing the correct object \_\_\_\_\_ 10 pts
  - The user will place the code above the method call:

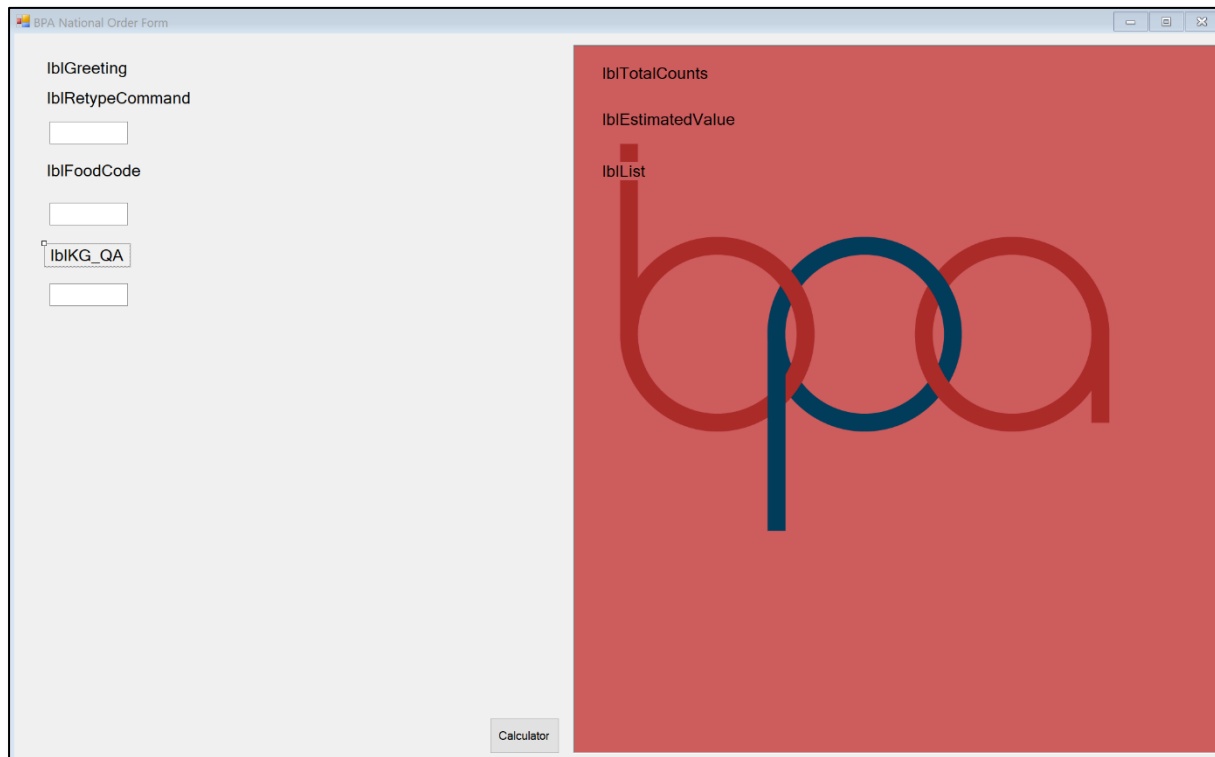
```
//SC12 printing a sarcastic comment  
Console.WriteLine("Get ready for some fun!");
```

### National\_CSharp

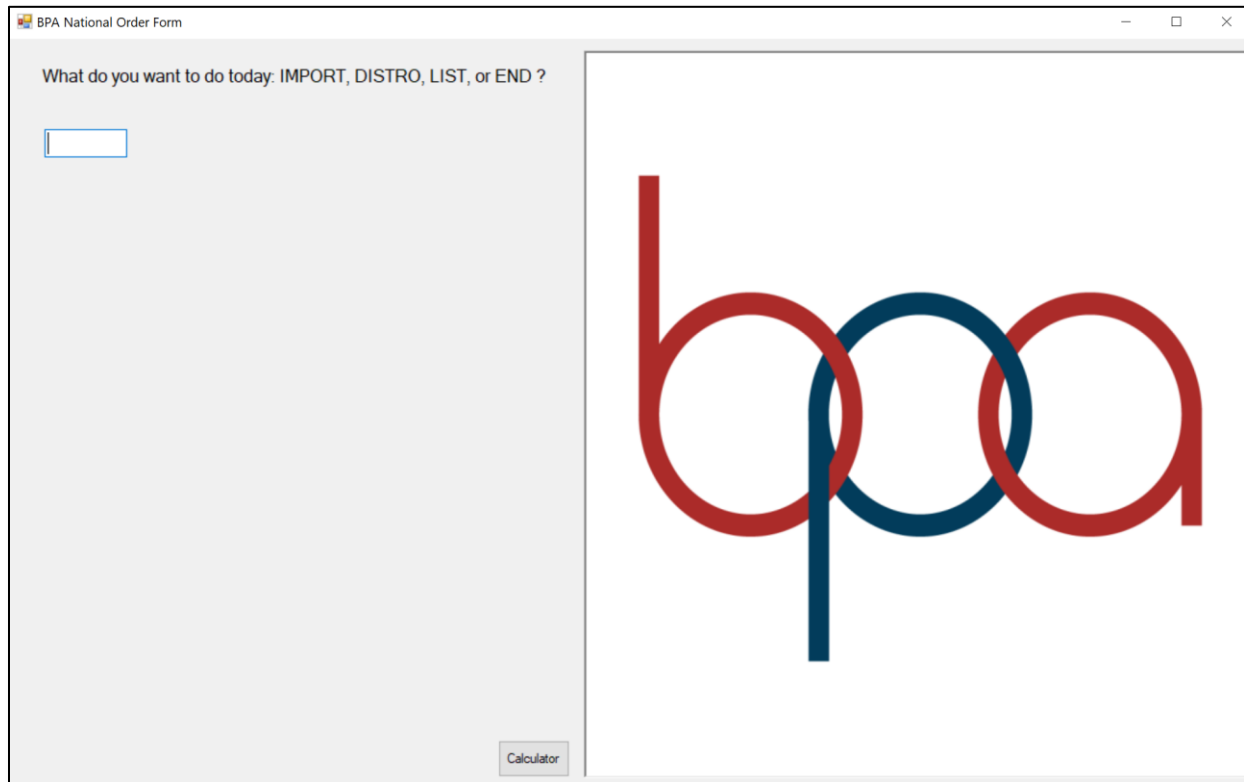
You will be developing a test software for a restaurant supplier; the program will be importing specific text files that have the word "Text" in the actual file name. There are other text files in the folder so your program will need to ignore those. Next, your program will read the text file and separate the records into either produce and dairy (NOTE: in the text files there are no food names, we will just use a naming convention of letters with increasing numbers so it will be easier to identify which records got imported. For example, one of the text files has AA1 to AA6 for each SKU and the product names are A\_One to A\_Six. The text file to be imported will contain record fields delimited by commas. Each record will begin with a flag ('P' for Produce or 'D' for Dairy). The program will continue iterating through all steps until the user instructs it to terminate, a process that can only occur at STEP 1.

## Form Design

Upon opening the provided project file, you'll find the form design already completed, as depicted in the accompanying image below. Labels starting with "lbl" are intended to convey information or commands to the user. Positioned in the left section of the form are three rectangles designated as text box objects (in the code, prefixed with "txt"). Moving to the right portion of the form, within a rectangular panel shaded in a light red, you'll encounter additional labels (lblTotalCounts, lblEstimatedValue, lblList) which serve to display outputs resulting from file reading and calculations. For a preview of the list generated by your program, refer to the Output section on the subsequent page. There is also a button at the bottom called "Calculator". This button will be explained if you can reach the calculator challenge. Until then, ignore it.



The form's appearance upon program initiation is exemplified below. Note: The list on the right solely manifests upon the user inputting the "List" command. Visibility of labels and text boxes during program runtime is flexible. They can either be visible from the program's onset or become visible as the user progresses through inputting commands, food symbols, and kilograms (kg). The list at the bottom should only emerge upon entering the list command or successfully inputting a food for importing (adding to inventory) or distribution (subtracting from inventory). Labels must accurately depict required actions. Should all text boxes appear simultaneously, it suggests users will input information in the correct sequence.



## Input

The application will accept basic user input via four primary commands.

- **IMPORT:** Allows the user to specify the amount of produce to be gathered and the quantity to harvest, enhancing agricultural productivity. It adds to the inventory.
- **DISTRO:** Utilizing this directive permits the user to select the type of foodstuff and the quantity to exchange within the market. It subtracts from the inventory.
- **LIST:** This directive provides the user with a comprehensive view of the food products held by the company, alongside detailed data.
- **END:** Employing this directive enables the user to terminate the application form.

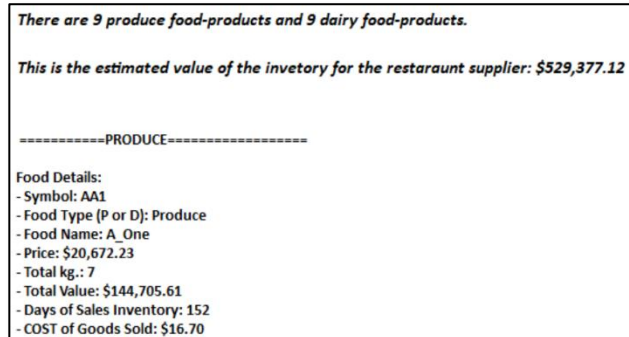
The other input will be an automatic process of reading the text files. To the right is an excerpt from the text file. Note that entries may be uppercase or lowercase, with no spaces between them. There are three usable text files, six entries per file, thus 18 total records to read. The program will utilize delimiters to accurately segregate the information fields for proper storage. The sequence of fields is as follows: P or D flag, SKU, Name, Price, Tons. In this instance, the word "STOP" denotes the termination flag, signaling the program to cease reading the file.

P,AA6,A\_Six,1230.31,7,STOP

## List Output

The ultimate result will be a list of all the foods read from the text file along with all of the calculations. Pay attention to the formatting (see image below). The top two lines of text are dynamic. The first line will display the actual count of each type of food (hint there are 6 dairy and 6 produce) read from the text files.

The second line consists of a dynamic calculation that adds up all the inventory calculations from the food-by-food listing. There is also a record below (all 18 will be similar layout but with different data). Additionally, the record will also be amended by adding three calculations of each items Total Value, Days of Sales Inventory, and COST of Goods Sold (this is all caps so it will stand out). The formulas for the calculations will be provided later, and do not try to make sense of the numbers in a real world context because we are just using random data values.



There are 9 produce food-products and 9 dairy food-products.

This is the estimated value of the inventory for the restaraunt supplier: \$529,377.12

=====PRODUCE=====

Food Details:  
- Symbol: AA1  
- Food Type (P or D): Produce  
- Food Name: A\_One  
- Price: \$20,672.23  
- Total kg.: 7  
- Total Value: \$144,705.61  
- Days of Sales Inventory: 152  
- COST of Goods Sold: \$16.70

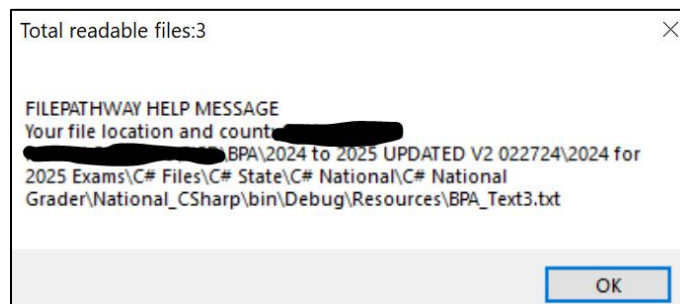
## Formulas

NOTE: these formulas are written algebraically.

- Total Value = price \* kg
- Days of Sales Inventory = ((kg / COST of Goods Sold) \* 365)
- COST of Goods Sold = (random\_double \* (product\_kg \* 10% - 1) + 1) \* 20

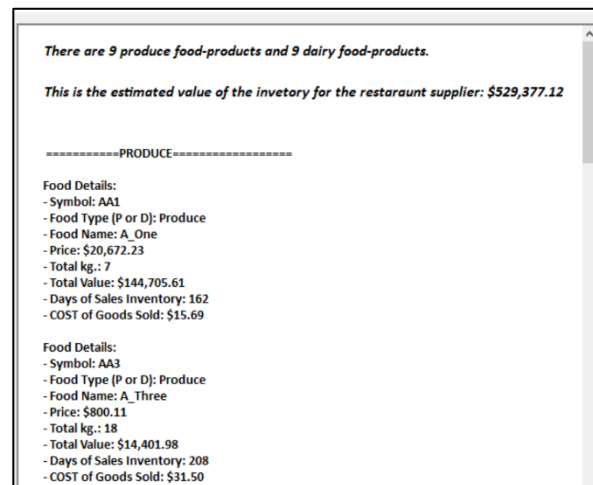
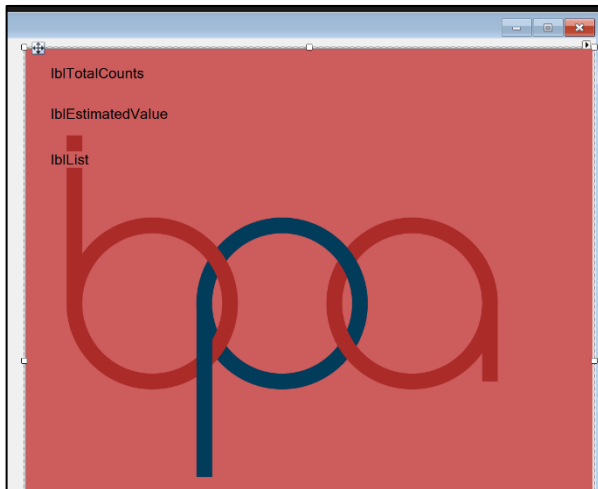
## Program Start: Message Box

When the program starts you will see a message box appear on the screen. It will show you the URL location of the files and the total number of files in that folder that are readable. You will program this action. NOTE: part the blacked out portions have been put there purposely for security purposes.



## Program Start: Dynamic Panel with List

When the program starts the panel or dynamically change period to begin it's background color will automatically be set to the color of your window on your computer. This will be different for every user. Additionally you will notice that the panel has a scroll bar activated to allow us to view all of the records that are imported. Again this is done through code. Finally you will notice that the BPA logo in the background disappears once the program starts. You will need a program all of these actions.



## Output Steps No Errors

NOTE: The IMPORT and DAIRY commands have similar output so only one will be displayed. If DISTRO is chosen, in the output provided below, all instances of the word "produce" will be substituted with the word "sell".

STEP 1 & 2: the user can enter the commands in either upper or lower case (image to the right).  
STEP 2: import or distro is entered, the program will ask the user to enter in a three-letter symbol for which food-product they wish to perform the action upon.

import

Which food-product do you want to procure?

AA1

STEP 3: (image to the right) the program will ask the user how many kg they want to import or distro. When importing, the user cannot enter more than 1000; when selling the user cannot enter more than the amount in the inventory.

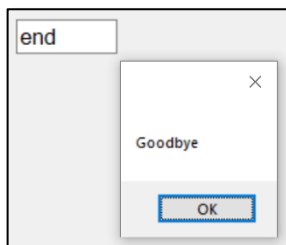
How many kg do you want to procure?

333



STEP 4: the program will display the list automatically (image to the right). Once the list is displayed, it will remain on the form panel until the program closes. NOTE: LIST command will cause the same list to appear. The program cursor focus will return to the initial prompt asking for one of the four commands.

STEP 1: Remember that the program will keep repeating STEPS 1-4 until the user enters the “End” command on STEP 1. A message box appears and once “OK” button is clicked the program exits. (See image below.



```

There are 9 produce food-products and 9 dairy food-products.

This is the estimated value of the inventory for the restaunt supplier: $7,413,229.71

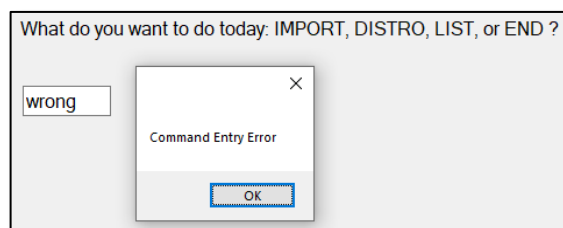
=====PRODUCE=====

Food Details:
- Symbol: AA1
- Food Type (P or D): Produce
- Food Name: A_One
- Price: $20,672.23
- Total kg.: 340
- Total Value: $7,028,558.20
- Days of Sales Inventory: 273
- COST of Goods Sold: $453.06

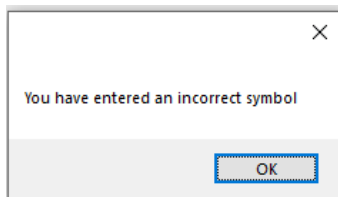
Food Details:
- Symbol: AA3
- Food Type (P or D): Produce
- Food Name: A_Three
- Price: $800.11
- Total kg.: 18
- Total Value: $14,401.98
- Days of Sales Inventory: 215
- COST of Goods Sold: $30.50
  
```

### Output IMPORT/DISTRO Command (With Errors STEP 1)

If the user enters in the wrong command (STEP 1), a message box appears indicating it is wrong.



After the message box is closed, the cursor focus should return to the command entry in the text box.

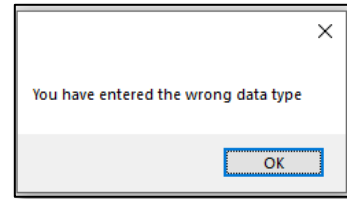


### Output Entering Food-Product Symbol (With Errors STEP 2)

If the user inputs an incorrect food (STEP 2), a message box will pop up, indicating that an incorrect food has been entered. Once the message box is closed, the text box will be cleared, and the cursor focus will revert back to the command entry in the text box.

### Output Entering Tonnage (With Errors STEP 3a)

The consumer inputs letters or symbols (STEP 3), triggering a prompt indicating an incorrect data type. Once the prompt is dismissed, the text box ought to reset, with the cursor refocusing on the command entry within the text box.



### Output Entering KG (With Errors STEP 3b)

If selling and the user enters a value greater than the inventory or 0 (or negative value) a message box appears indicating *"Incorrect Data Entry. Reenter a value greater than 0, and less-or-equal to amount in the inventory"*. After the message box is closed, the text box should clear, and the cursor focus should return to the command entry in the text box.

### Output Entering Tonnage (With Errors STEP 3c)

If importing and the user enters a value greater than 1000 or 0 (or negative value) a message box appears indicating *"Incorrect Data Entry. Reenter a value greater than 0 and not greater than 1000"*. After the message box is closed, the text box should clear, and the cursor focus should return to the command entry in the text box.

## Requirements/Explanations

1. You must use the provided Visual Studio project called “**RestaurantSupplier\_National**” in the **National\_CSharp** folder. This project will contain a “Form1.cs[design]” that has been already created in the organization and naming conventions.
2. No additional objects can be added to the form design; HOWEVER(!), you can add additional methods to any of the provided classes. If you get to the Calculator Challenge section then you will be adding a new form to the project which will require you to add objects to the new form (Form2.cs).
3. The following class files have already been provided: **Form1.cs**, **Food.cs**, **Inventory.cs**, and the text files that are already stored in the project folder at this location:  
National\_CSharp\bin\Debug\Resources
4. Your contestant number must appear as a comment at the top of the Form1.cs file.
5. You will be programming methods in **Form1.cs**, **Inventory.cs** (helper class), and **Dairy.cs** and **Produce.cs** (object classes you will need to create).
6. The **Food.cs** class is a parent class that has generic methods already programmed. You will need to use inheritance for the **Dairy** and **Produce** children classes. Most of these methods can be copied and pasted.
7. NOTE: the **Food.cs** parent class does not have the formulas programmed; you will need to create these formulas in the **Dairy** and **Produce** children classes...or you can program these formulas in the parent class and use the powers of inheritance. You will also need to complete all of the methods and constructors for these classes.
8. IMPORTANT: The **Food.cs** parent class lacks implementation of the ToString method; it is necessary to devise it for the **Dairy** and **Produce** subclasses...or alternatively, incorporate it within the parent class. Ensure that all financial figures are formatted to US currency (\$0.00). Refer to the provided sample listing for precise sequencing and expected outcomes of the computed values. While your calculations may vary due to randomization, the resulting values should remain consistent.
9. Provided Form Objects (NOTE: the output text is provided in the code, too):
  - a. **lblGreeting**: asks the user “*What do you want to do today: IMPORT, DISTRO, LIST, or END*”. Appears at initial start of the program.
  - b. **lblRetypeCommand**: prompts the user “*Please enter in your next command.*” Appears after the user has completed all steps for IMPORT and DISTRO, or LIST.
  - c. **txtCommand**: user enters in one of four commands here.
  - d. **lblFoodCode**: asks user “*Which food-product do you want to sell?*” or asks, “*Which food-product do you want to procure?*”
  - e. **txtSymbol**: user enters in one of the 18 food codes (they are 3 digits each...look at the text files for exact names, there are six and they all start with AA1, BB1, of CC1).
  - f. **lblKG\_QA**: asks user “*How many kg do you want to sell?*” or asks, “*How many kilograms do you want to mine?*”
  - g. **txtKG\_QA**: user enters in a positive integer for how many kg to distro/import.
  - h. **lblTotalCounts**: displays “*There are # produce food-products and # dairy food-products.*” In your program there will be 9 produce and 9 dairy.

- i. ***lblEstimatedValue:*** displays the sum of the Total Values of all the food-products; *“This is the estimated value of inventory for the restaurant supplier: \$0.00”*. In your program the estimated value will be \$529,377.12.
  - j. ***lblList:*** This label will display all of the food-products read from the text files plus any calculations performed in the objects (see the formula sections on how to calculate).
10. The sourcecode files in the project provide comments on all included methods and other variables you will need to program.

<b>Solution and Project (There is NO partial credit) (NOTE: UC represents uppercase and LC represents lowercase)</b>		
The VS project file is present on the flash drive in a single folder with your contest ID		10 points
<b>Program Execution (If the program does not execute, then the remaining items in this section receive a score of zero)</b>		
Message Box appears showing the file pathway to where the text files are located and displays the number of files in the folder.		30 points
Form displays and the BPA logo is removed from panel.		10 points
STEP 1: Enter LIST command causing list to be generated, vertical scrollbar appears and allows user to view all records.		10 points
Generated list has 9 Dairy and 9 Produce products		50 points
Format of the displayed list matches sample output in terms of the order of printed information: first the statement with number of food-products, statement with estimated value, and the data fields for each record (all must be present and in proper order).		30 points
In the list: <b>estimated value</b> , <b>Total Value</b> , <b>COST of Goods Sold</b> are in US currency; <b>Days of Sales Inventory</b> is an integer (no double values).		20 points
STEP 1: Wrong command is entered in <b>txtCommand</b> and message box with statement “Command Entry Error” appears. Closing message box returns cursor focus command text entry box and clears previous entry		20 points
STEP 1: “ <b>Distro</b> ” command (UC or LC) prompts next label to appears asking which food-product to sell and <b>txtSymbol</b> text box appears with the cursor focus moves to STEP		10 points
STEP 2: Food-Product symbol (UC or LC) is entered prompts next label to appears asking which food-product to sell and <b>txtKG_QA</b> text box appears with the cursor focus moves		10 points
STEP 3: Kilograms are entered for a food-products to sell, and the list changes the information for the food-product dynamically adjusts (estimated value, Total kg, Total Value, Days of Sales Inventory, COST of Goods Sold) moves to STEP 4		10 points
STEP 1: “ <b>Import</b> ” command (UC or LC) prompts next label to appears asking which food-product to sell and <b>txtSymbol</b> text box appears with the cursor focus moves to STEP		10 points
STEP 2: Food-Product symbol (UC or LC) is entered prompts next label to appears asking which food-product to mine and <b>txtKG_QA</b> text box appears with the cursor focus moves		10 points
STEP 3: Kilograms are entered for a food-product to distro, and the list changes the information for the food-product dynamically adjusts (estimated value, Total kg, Total Value, Days of Sales Inventory, COST of Goods Sold) moves to STEP 4		10 points
STEP 2: Wrong symbol is entered in <b>txtSymbol</b> and message box with statement “You have entered incorrect symbol” appears. Closing message box returns cursor focus command text entry box and clears previous entry		20 points
STEP 3: When importing or distributing: characters are entered in <b>txtKG_QA</b> a message box with statement “You have entered wrong data type” appears. Closing message box returns cursor focus command text entry box and clears previous entry		30 points
STEP 3: When selling: 0 or more than the number of kilograms are entered in <b>txtKG_QA</b> a message box appears indicating wrong amount. Closing message box returns cursor focus command text entry box and clears previous entry		20 points

STEP 3: When importing: 0, or more than 1000 kilograms are entered in <b>txtKG_QA</b> a message box appears indicating wrong amount. Closing message box returns cursor focus command text entry box and clears previous entry		20 points
<b>Solution and Project Subtotal</b>		<b>/330 Points</b>

<b>Source Code Review (There is NO partial credit)</b> <i>NOTE: you must place the comment flag in front of the comment in your code in order to get credit. The comment flag will precede the explanation. For example, if the flag is SC1, your comment must read as “//SC1...” in front of the part of the code being reviewed. Code must work to get credit.</i>		
A comment containing the contestant number is present at the top of the Form1.cs file		10 points
SC1A-C: <b>Form1.cs</b> class <b>Form1_Load( )</b> Place a SC1 comment by these four items: SC1A by the code that demonstrates the activation of the scroll; SC1B code that shows setting the fonts of all the labels to the Calibri font with the size of 12; SC1C how the background image of the panel was removed; SC1D how the panel gets its color from the background color of the windows setting.		20 points
SC2: <b>Form1.cs</b> class <b>readFile( )</b> Place a in code where the file names are read dynamically to only read text files they have the capital word “Text” in them.		40 points
SC3: <b>Form1.cs</b> class <b>readFile( )</b> Place in code where the message box is created and is given a path location to one of the files that will be read.		30 points
SC4A-D: <b>Form1.cs</b> class <b>readFile( )</b> Place a comment by these four sections: <b>SC4A</b> code demonstrates reading of the file, recognizing the flags P or D; <b>SC4B</b> Code that recognizes the word “stop”; <b>SC4C</b> code that handles the construction of the objects; <b>SC4D</b> code for separating the fields from reading the file.. No partial credit		40 points
SC5: <b>Form1.cs</b> class <b>txtCommand_KeyPress ( )</b> code showing user input being gathered and passing control to the <b>Inventory getCommand( )</b>		10 points
SC6: <b>Form1.cs</b> class <b>txtSymbol_KeyPress ( )</b> code showing user input being gathered and passing control to the <b>Inventory getCommand( )</b>		10 points
SC7: <b>Form1.cs</b> class <b>txtTonnage_KeyPress ( )</b> code showing user input being gathered and passing control to the <b>Inventory getCommand( )</b>		10 points
SC8: <b>Inventory.cs</b> class <b>getInventoryTotalValue( )</b> code shows how all of the food-products Total Value are sum to show the estimated value of farming operation		20 points
SC9: <b>Inventory.cs</b> class <b>getInventoryList( )</b> code shows how all of the food-products are printed as a list to the <b>lblList</b> object. Must show how estimated value is formatted into currency.		20 points
SC10: <b>Inventory.cs</b> class <b>supplySKUVerification( )</b> code shows how all of the food-product symbols are checked against user input		20 points
SC11: <b>Inventory.cs</b> class <b>dataEntryVerification( )</b> code shows how all of the food-product ton entries are checked against parameters of selling vs. importing.		20 points
SC12: <b>Inventory.cs</b> class <b>procureInventory( )</b> demonstrates how the data entered into <b>txtKG_QA</b> is checked for nonnumerical entries. Must use a <b>try catch</b> exception handler.		20 points
SC13: <b>Dairy.cs</b> and <b>Produce.cs</b> classes calculate Total Value with <b>getValue( )</b> . Place a SC11 comment by these two sections		10 points
SC14: <b>Dairy.cs</b> and <b>Produce.cs</b> classes calculate the Days of Sales Inventory (no hard coded data allowed, variables only) with <b>DaysSalesOfInventory( )</b>		10 points

SC15: <b>Dairy.cs</b> and <b>Produce.cs</b> classes calculate the Cost of Goods Sold (no hard coded data allowed, variables only) with <i>costOfGoodsSold()</i> .		10 points
SC16: <b>Dairy.cs</b> and <b>Produce.cs</b> classes <i>ToString()</i> method uses string interpolation to print all 8 of the required data values.		30 points
<b>Source Code Review Subtotal</b>		<b>/330 Points</b>
<b>Combined Sub Total Points</b>		<b>/660 Points</b>

## Calculator Challenge

You will create a basic four-function calculator using your C# in a Windows Form Application with .NET framework project (this will be done your current project). Instructions: Your task is to design and implement a simple five-function (includes modulus) calculator with the following functionalities:

- User Interface Design:
  - Add a new form to the project, it needs to be called Form2.
  - Design the layout of the calculator interface with buttons and labels.
- Display Feature:
  - Include a display feature to show the input and output of the calculator operations.
- Basic Operations:
  - Implement addition, subtraction, multiplication, and division operations.
  - Include buttons for each operation (+, -, \*, /, %).
  - Ensure the calculator performs these operations accurately.
- Numeric Buttons:
  - Include buttons for digits 0-9 to input numbers.
  - Users should be able to input multi-digit numbers.
- Clear Button:
  - Implement a clear button to reset the calculator display and any ongoing calculation.
- Equal Button:
  - Include an equal button (=) to perform the calculation and display the result.
- Error Handling:
  - Implement error handling for invalid input or operations (e.g., division by zero).
  - Program must give the error message "Divide by Zero"
- Starting the Form2
  - Form2 will start when you press the calculator button that is on the bottom of Form1. This is the only way that Form2 should load. You should be able to close Form2 and return to Form1 at any point in time.

<b>Calculator Challenge</b>		
Form2 loads when the btnCalcuLaunch is pressed		5 points
Form2 UI Calculator has all required buttons		5 points
Form2 when pressing any numeric key the numbers appear in the display		5 points
Form2 Calculator's adding function works		5 points
Form2 Calculator's subtracting function works		5 points
Form2 Calculator's multiplication function works		5 points
Form2 Calculator's division function works		5 points
Form2 Calculator's modulus function works		5 points
Form2 Calculator's equal function works		5 points
Form2 Calculator's clear function works		5 points
Form2 when attempting to divide by zero the calculator will not allow action. You must provide a user with an "Divide by Zero" message		5 points
<b>Calculator Challenge</b>		<b>/55 Points</b>
<b>Solution and Project</b>		<b>/330 Points</b>
<b>Source Code Review</b>		<b>/330 Points</b>
<b>Grand Total</b>		<b>/715 TOTAL</b>