100 XP

Exercise - Build and test a loop for entering new pet data

20 minutes

In this exercise, you develop code that controls the input of new `ourAnimals` array data. You calculate the initial values of your loop control variables and construct the loop that collects user specified data for the animals. The detailed tasks that you complete during this exercise are:

1. Calculate petCount: write code that counts the number of pets in the `ourAnimals` array that have assigned data.
2. Conditional messages: write code to display message output when `petCount` is less than `maxPets`.
3. Outer loop: build a loop structure that will be used for entering new `ourAnimals` array data.
4. Exit criteria: write code that evaluates the exit condition for the "enter new ourAnimals array data" loop.
5. Verification test: perform verification tests for the code you develop in this exercise.

Important

You must complete the previous exercise in this module before starting this exercise.

Count the number of pets in the ourAnimals array

In this task, you establish the exit criteria for your data entry loop and you create a `for` loop that can be used to count the number of pets in `ourAnimals` that have assigned data.

1. Ensure that Visual Studio Code is open, and that your Program.cs file is visible in the Editor.

2. Locate the `switch(menuSelection)` statement, and then find the `case "2":` code line.

3. Locate the `Console.WriteLine()` statement that displays the "coming soon" message, and then replace it with a blank code line.

4. On the blank code line that you created, to declare the `anotherPet` and `petCount` variables, enter the following code:

```c#
string anotherPet = "y";
int petCount = 0;
```

These two variables control the iteration of a `while` loop that's used to enter new pet data. You initialize both variables as part of the declaration.

- `anotherPet` is initialized with a value of `y` prior to the start of the `while` loop. It will receive a user assigned value, either `y` or `n` inside the `while` loop.
- `petCount` represents the number of animals with assigned pet characteristics. It will be assigned a calculated value outside of your `while` loop, and will be incremented by `1` inside the `while` loop each time a new animal is added to the `ourAnimals` array.

Important

The scope of your variables should always be as narrow as possible. In the Contoso Pets application, you could scope `petCount` at the application level rather than scoping to the `case "2":` code block. The larger scope would enable you to access `petCount` from anywhere in the application. If `petCount` was scoped at the application level, you could assign it a value when you create the sample data and programmatically manage its value throughout the remainder of the application. For example, when you find a home for a pet and remove the pet from the `ourAnimals` array, you could reduce `petCount` by `1`. The question is, at what level should you scope a variable when you're unsure whether it will be used in other parts of your application? In this case, it's tempting to scope `petCount` at the application level even though you aren't using it anywhere else. After all, scoping `petCount` at the application level ensures that it's available if you do decide to use it elsewhere. Maybe you could scope other variables at the application level as well. That way, your variables are always in scope and accessible. So why not scope variables at the application level when you think they might be used later in the application? Scoping variables at a higher level than necessary can lead to problems. Elevated scope inflates the resource requirements of your application and may expose your application to unnecessary security risks. As your applications grow larger and more complex, they require more resources. Phones and computers allocate memory for these resources when they're in scope. As your applications become more "real-world", they become more accessible. Applications are often accessible from the cloud or other applications. Compounding these issues, applications are often left running when they aren't being used. It's important to keep an application's resource requirements under control and the security footprint as small as possible. Although today's operating systems do a great job of managing resources and securing applications, it's still best practice to keep your variables scoped to the level where they are actually needed. In your Contoso Pets app, if you decide to use `petCount` more broadly within the application, you can update your code to scope `petCount` at a higher level. Remember to keep your variables scoped as narrowly as possible, and only increase their scope when it becomes necessary.

5. On the code line below your variable declarations, to create a loop that iterates through the animals in the `ourAnimals` array, enter the following code:

   c#

   ```
   for (int i = 0; i < maxPets; i++)
   {
   }
   ```

   This code should look familiar. You'll use this `for` loop each time you iterate through the `ourAnimals` array.

6. Inside the code block of our `for` loop, to check whether pet characteristics data has been assigned to an animal, enter the following code:

   c#

   ```
   if (ourAnimals[i, 0] != "ID #: ")
   {
   }
   ```

   Again, this code should look familiar. You'll use this `if` statement each time you check whether pet characteristics have been assigned.

7. Inside the code block of the `if` that you created, to increment `petCount` by 1, enter the following code:

   c#

```
petCount += 1;
```

8. Take a minute to examine your completed `for` loop.

   Your completed `for` loop should look like the following code:

   ```c#
   for (int i = 0; i < maxPets; i++)
   {
       if (ourAnimals[i, 0] != "ID #: ")
       {
           petCount += 1;
       }
   }
   ```

   This code will loop through the `ourAnimals` array checking for assigned data. When it finds an animal with data assigned, it increments `petCounter`.

9. On the Visual Studio Code **File** menu, select **Save**.

10. Open the Integrated Terminal panel in Visual Studio Code and enter the command to Build your program.

    To open the Integrate Terminal from the EXPLORER view, right-click **Starter**, and then select **Open in Integrated Terminal**. You can also use the **View** or **Terminal** menu to open the Integrated Terminal panel.

    To Build your program, enter the `dotnet build` command from the Terminal command prompt.

11. Fix any Build errors that you see reported before continuing.

    Note

    For now, you can ignore the Warning message about `anotherPet` being assigned but never used. You'll add code that uses `anotherPet` later in this exercise.

    If you have any build errors, remember that the Build error and warning messages tell you what the issue is and where you can find it. If you update your code, remember to save your changes before you rebuild.

12. Close the Terminal panel.

Display message output when petCount is less than maxPets

In this task, you check to see if `petCount` is less than `maxPets` and if it is, you display a message for the user.

1. On a blank code line below the `for` loop that you created, to see if `petCount` is less than `maxPets`, enter the following code:

   c#

   ```
   if (petCount < maxPets)
   {
   }
   ```

2. Inside the code block of the `if` statement, to display a message to the user, enter the following code:

   c#

   ```
   Console.WriteLine($"We currently have {petCount} pets that need homes. We can
   manage {(maxPets - petCount)} more.");
   ```

   Application users are about to enter pet characteristics. This message provides important context.

3. Take a minute to review the `case "2":` code branch of your `switch` statement.

   At this point, your `case "2":` code branch should look like the following code:

   c#

   ```
   case "2":
       // Add a new animal friend to the ourAnimals array
       string anotherPet = "y";
       int petCount = 0;
       for (int i = 0; i < maxPets; i++)
       {
   ```

```
            if (ourAnimals[i, 0] != "ID #: ")
            {
                    petCount += 1;
            }

        }

        if (petCount < maxPets)
        {
            Console.WriteLine($"We currently have {petCount} pets that need homes.
    We can manage {(maxPets - petCount)} more.");
        }

        Console.WriteLine("Press the Enter key to continue.");
        readResult = Console.ReadLine();
        break;
```

4. On the Visual Studio Code **File** menu, select **Save**.

5. Open the Integrated Terminal panel in Visual Studio Code and enter the command to Build your program.

6. Fix any Build errors that you see reported before continuing.

   Again, you can ignore the Warning message about `anotherPet` being assigned but never used. In the next task, you'll start building the `while` loop that is used to enter the data for one or more pets. The expression that you create for the `while` loop will use `anotherPet` and this Warning message will go away.

   Remember, warning messages are things that you should be concerned about, but they won't prevent you from running your program.

7. At the Terminal command prompt, enter the command to run your program.

   Enter the `dotnet run` command at the Terminal command prompt to run your program code.

   As long as your code doesn't generate a runtime error, the main menu of the app should now be displayed in the Terminal panel.

8. At the Terminal command prompt, enter **2**

   This value corresponds to your `case "2":` code branch.

9. Verify that the following message is displayed in the Terminal.

Output

```
We currently have 4 pets that need homes. We can manage 4 more.
Press the Enter key to continue.
```

If you don't see the expected message displayed, review your code to identify and fix the issue. Save your changes, rebuild, and run the app again. Be sure to get code working as expected before you continue.

10. At the Terminal command prompt, press Enter to continue running your application.

11. Exit the application, and then close the Terminal panel.

Build a loop structure that will be used for entering new ourAnimals array data

In this task, you create a `while` loop that continues to iterate as long as `anotherPet` is equal to `y` and `petCount` is less than `maxPets`.

1. In the code Editor, create a blank code line below your `if (petCount < maxPets)` code block.

2. To begin the process of creating your new `while` loop, enter the following code:

C#

```csharp
while (anotherPet == "y" && petCount < maxPets)
{
}
```

3. On the Visual Studio Code **File** menu, select **Save**.

4. Open the Integrated Terminal panel in Visual Studio Code and enter the command to Build your program.

5. Notice that you no longer receive the Warning message about `anotherPet` not being used.

   If any Build errors or warnings were reported, fix the issues before continuing.

Check exit condition for new pets loop

In this task, you update the `while (anotherPet == "y" && petCount < maxPets)` code block. The new code increments `petCount` and then checks whether `petCount` is less than `maxPets`. If `petCount` is less than `maxPets`, you ask the user if they want to enter information for another pet, and ensure the response is either `y` or `n`. After the `while (anotherPet == "y" && petCount < maxPets)` code block, you check the value of `petCount`. If `petCount` is equal to `maxPets`, you inform the user that no more pets can be added.

Note

The code used to enter pet data is developed in the next exercise. For now, `petCount` is incremented as if data were being entered and saved to the `ourAnimals` array. This enables you to finish developing the code logic associated with the `while` loop.

1. Create a blank code line inside the code block of the `while (anotherPet == "y" && petCount < maxPets)` loop that you created in the previous task.

2. To increment `petCount`, enter the following code:

```c#
// increment petCount (the array is zero-based, so we increment the counter af-
ter adding to the array)
petCount = petCount + 1;
```

3. To check whether `petCount` is less than `maxPets`, enter the following code:

```c#
// check maxPet limit
if (petCount < maxPets)
{
}
```

4. Inside the code block of the `if` statement that you created, to ask the user whether they want to add another pet, enter the following code:

```c#
// another pet?
```

```
Console.WriteLine("Do you want to enter info for another pet (y/n)");
```

5. Below the `WriteLine()` message that you entered, to read the user response and ensure that the user entered "y" or "n", enter the following code:

```c#
do
{
    readResult = Console.ReadLine();
    if (readResult != null)
    {
        anotherPet = readResult.ToLower();
    }

} while (anotherPet != "y" && anotherPet != "n");
```

6. Locate the `break` statement that separates `case "2";` from `case "3";` in your `switch` statement.

7. Notice the `Console.WriteLine()` and `Console.ReadLine()` statements at the end of our `case "2";` code.

   This code displays a message to the user and then pauses the application.

8. To enclose the `Console.WriteLine()` and `Console.ReadLine()` statements inside an `if` statement, update your code as follows:

```c#

if (petCount >= maxPets)
{
    Console.WriteLine("Press the Enter key to continue.");
    readResult = Console.ReadLine();
}

break;

case "3":
```

   The value of `petCount` is incremented inside the `while` loop. If `petCount` is equal to `maxPets`, no more pets can be added to the `ourAnimals` array. You should let the user know when this occurs.

9. To inform the user that Contoso Pets has reached their capacity, update your code as follows:

```c#


    if (petCount >= maxPets)
    {
        Console.WriteLine("We have reached our limit on the number of pets that we can manage.");
        Console.WriteLine("Press the Enter key to continue.");
        readResult = Console.ReadLine();
    }

    break;

case "3":
```

10. Take a minute to review the code in your `while` loop and the user message that you've created.

    Your `while (anotherPet == "y" && petCount < maxPets)` loop and code that displays the user message should look like the following code:

```c#
while (anotherPet == "y" && petCount < maxPets)
{
    // increment petCount (the array is zero-based, so we increment the counter after adding to the array)
    petCount = petCount + 1;

    // check maxPet limit
    if (petCount < maxPets)
    {
        // another pet?
        Console.WriteLine("Do you want to enter info for another pet (y/n)");
        do
        {
            readResult = Console.ReadLine();
            if (readResult != null)
            {
                anotherPet = readResult.ToLower();
            }

        } while (anotherPet != "y" && anotherPet != "n");
    }
}

if (petCount >= maxPets)
{
    Console.WriteLine("We have reached our limit on the number of pets that we can manage.");
    Console.WriteLine("Press the Enter key to continue.");
    readResult = Console.ReadLine();
}
```

11. On the Visual Studio Code **File** menu, select **Save**.
12. Open the Integrated Terminal panel in Visual Studio Code and enter the command to Build your program.
13. Fix any Build errors or warnings that you see reported before continuing.

Check your work

In this task, you run our application from the Integrated Terminal and verify that the looping and branching logic that you've created works as expected.

1. If necessary, open Visual Studio Code's Integrated Terminal panel.
2. At the Terminal command prompt, enter **dotnet run**
3. At the Terminal command prompt, enter **2**
4. Verify that you see the following messages:

Output

```
We currently have 4 pets that need homes. We can manage 4 more.
Do you want to enter info for another pet (y/n)
```

5. At the Terminal command prompt, enter **n**
6. Verify that your code exits the loop for entering new pets when you entered "n".

   If your code logic is working as expected, you should see the main menu displayed in the Terminal.

If your code doesn't exit the loop when expected, press **Ctrl + C** in the Terminal to force execution to stop. You need to step through your code manually and trace the values of the exit criteria variables. Update your code if necessary to ensure that you exit the `while` loop when the user enters "n". Save your changes, rebuild your program, and run through the verification test to arrive back at this point.

7. At the Terminal command prompt, enter **2**

   Once again, you'll see the following messages displayed:

   Output

   ```
   We currently have 4 pets that need homes. We can manage 4 more.
   Do you want to enter info for another pet (y/n)
   ```

8. At the Terminal command prompt, enter **y**

9. Take a minute to consider how `petCount` is used in your code.

   You need to understand your code logic before you can validate your code.

   In this case, your code logic relies on the relationship between `petCount` and `maxPets`. You know that `maxPets` is assigned a value of `8`, but what about `petCount`? The following items help to evaluate the logic you've implemented:

   - You know that `petCount` is `4` when you enter the first iteration of the `while` loop.
   - You know that `petCount` is incremented each time the `while` loop iterates.
   - You know that the value assigned to `petCount` and the way that `petCount` is incremented affect how data is stored in the `ourAnimals` array. The following items explain the relationship between `petCount` and the data stored in `ourAnimals`:
     - The application adds four pets to the `ourAnimals` array when it creates the sample data.
     - The application stores new data to the `ourAnimals` array when the value of `petCount` is `4`. This isn't a bug. The code makes sense when you recall that array elements are zero-based. For example, `ourAnimals[0,0]` contains the pet ID for animal `1` and `ourAnimals[3,0]` contains the pet ID for animal `4`. Therefore, when `petCount` is `4` you're storing data for the fifth pet.
     - The application will store pet data to the array before it increments `petCount`.

   - The application increments `petCount` before it prompts the user about adding another pet.
   - When the application displays the **Do you want to enter info for another pet (y/n)** prompt for the first time, `petCount` is already set to `5`.

- If the user enters **y** at the first **Do you want to enter info for another pet (y/n)** prompt, you know that:
   - The `while (anotherPet == "y" && petCount < maxPets)` loop will iterate. You know the loop will iterate because `anotherPet == "y"` and `petCount < maxPets`.
   - The value assigned to `petCount` will be incremented (when the `while` loop iterates).
   - The value assigned to `petCount` will be `6` (after the user enters **y** the first time).

Keep this analysis of the code logic in mind as you continue testing the application.

10. Notice that the Terminal panel updates with the same "another pet?" message, but your code doesn't display an updated `petCount`.

    The Terminal panel should now show the following output:

    Output

    ```
    We currently have 4 pets that need homes. We can manage 4 more.
    Do you want to enter info for another pet (y/n)
    y
    Do you want to enter info for another pet (y/n)
    ```

11. At the Terminal command prompt, enter **y**

    When you enter `y` a second time, `petCount` is incremented to `7`. So `petCount` is still less than `maxPets`

12. At the Terminal command prompt, enter **y**

    When you enter `y` a third time, `petCount` is incremented to `8`. So `petCount` is now equal to `maxPets`

13. Verify that your code exits the `while` loop when you enter **y** the third time.

    The Terminal panel should now show the following output:

    Output

```
We currently have 4 pets that need homes. We can manage 4 more.
Do you want to enter info for another pet (y/n)
y
Do you want to enter info for another pet (y/n)
y
Do you want to enter info for another pet (y/n)
y
We have reached our limit on the number of pets that we can manage.
Press the Enter key to continue.
```

If your code doesn't exit the loop when expected, step through your code manually and trace the values of the exit criteria variables. Update your code to ensure that you exit the loop when `petCount` reaches a value equal to `maxPets`. Keep answering "y" until you know that `petCount` is equal to `maxPets`, which has a default value of `8`.

14. At the Terminal command prompt, press Enter to continue running your application.

15. Exit the application, and then close the Terminal panel.

Next unit: Exercise - Write code to read and save new ourAnimals array data

Continue >