Step-by-Step Lab Guide - Variables

Important: Always Start on Paper

For each exercise in this guide, you must complete the paper portion first. Working through problems on paper helps you understand what's happening before you start coding. Let's walk through each exercise step by step.

Section 1: Finding and Fixing Errors

Exercise 1.1: Finding Syntax and Type Errors

Paper Steps:

1. Look at each line of code separately:

```
int count = "5";
double price = $19.99;
int x = 3.14;
```

2. For each line, ask yourself:
   - What type is on the left of the equals sign?
   - What type is on the right of the equals sign?
   - Can the right type fit into the left type?
3. Write down the problems you find:
   - Line 1: `"5"` is a string, but we're trying to put it in an `int`
   - Line 2: The `$` symbol isn't allowed in number literals
   - Line 3: `3.14` is a double (decimal), but we're trying to put it in an `int`
4. Write down how to fix each line:
   - Line 1: Remove the quotes → `int count = 5;`
   - Line 2: Remove the $ → `double price = 19.99;`
   - Line 3: Either round down → `int x = 3;` or explicitly convert → `int x = (int)3.14;`

Computer Steps:

1. Create a new file called `Errors1.cs`
2. First, try writing the program without System.Diagnostics:

```
public class Errors1 {
    public static void Main() {
        int count = 5;
        double price = 19.99;
        int x = 3;

        Debug.Assert(count == 5);  // This will cause an error!
    }
}
```

3. When you compile this, you'll get an error like:

```
error CS0103: The name 'Debug' does not exist in the current context
```

4. Now add System.Diagnostics at the top:

```
using System.Diagnostics;  // This lets us use Debug.Assert

public class Errors1 {
    public static void Main() {
        int count = 5;
        double price = 19.99;
        int x = 3;

        Debug.Assert(count == 5, "count should be 5");
        Debug.Assert(Math.Abs(price - 19.99) < 0.001, "price should be
19.99");
        Debug.Assert(x == 3, "x should be 3");
    }
}
```

Exercise 1.2: Creating Variables

Paper Steps:

1. For each variable you need to create, write down:

- Its name
- Its type
- Its value
- Whether it can change (is it const?)

2. Make a table like this:

Variable	Type	Value	Can Change?
MAX	int	100	No (const)
count	int	42	Yes
amount	decimal	13.75	Yes
initial	char	'R'	Yes

3. Write down the correct syntax for each:
   - `const int MAX = 100;`
   - `int count = 42;`
   - `decimal amount = 13.75m;` // Note the 'm' for decimal
   - `char initial = 'R';` // Note the single quotes

Computer Steps:

1. Create Errors2.cs
2. Remember to add System.Diagnostics first:

```
using System.Diagnostics;  // Without this, Debug.Assert won't work

public class Errors2 {
    public static void Main() {
        const int MAX = 100;
        int count = 42;
        decimal amount = 13.75m;
        char initial = 'R';

        Debug.Assert(MAX == 100, "MAX should be 100");
        Debug.Assert(count == 42, "count should be 42");
        Debug.Assert(amount == 13.75m, "amount should be 13.75");
```

```
            Debug.Assert(initial == 'R', "initial should be 'R'");
        }
    }
```

Exercise 1.3: Grade Calculation

Paper Steps:

1. Write down what you know:

- Homework is 30% of grade
- Midterm is 30% of grade
- Final exam is 40% of grade
- Passing grade is 70 or higher

2. Write down the formula:

```
finalGrade = (homework × 0.3) + (midterm × 0.3) + (finalExam × 0.4)
```

3. Test with sample numbers:
    - Homework = 85
    - Midterm = 90
    - Final = 75
4. Calculate by hand:
    - (85 × 0.3) = 25.5
    - (90 × 0.3) = 27
    - (75 × 0.4) = 30
    - Total = 82.5

Computer Steps:

1. Create Errors3.cs
2. Add System.Diagnostics and implement your solution:

```
using System.Diagnostics;  // Required for Debug.Assert
```

```
public class Errors3 {
    public static void Main() {
        int homework = 85;
        int midterm = 90;
        int finalExam = 75;

        // Always check input values first
        Debug.Assert(homework >= 0 && homework <= 100, "Invalid homework
score");
        Debug.Assert(midterm >= 0 && midterm <= 100, "Invalid midterm
score");
        Debug.Assert(finalExam >= 0 && finalExam <= 100, "Invalid final
exam score");

        // Calculate using the formula you worked out on paper
        double finalGrade = (homework * 0.3) + (midterm * 0.3) +
(finalExam * 0.4);
        bool isPass = finalGrade >= 70;

        // Check that your calculation matches your paper solution
        Debug.Assert(Math.Abs(finalGrade - 82.5) < 0.001, "Final grade
incorrect");
        Debug.Assert(isPass, "Student should pass");
    }
}
```

Section 2: Understanding Program State

Exercise 2.1: Tracking Variable Changes

In this exercise, you'll learn how to track how variables change as a program runs. This is an essential skill for understanding your code.

Paper Steps:

1. Set up your paper work:
    - Draw a line down the middle of your paper
    - On the left side, write "Code"
    - On the right side, write "State Table"
2. Copy this code to the left side of your paper:

```
int x = 5;
int y = x + 2;
x = y * 2;
bool isLarge = x > 10;
```

3. On the right side, create your state table:
    - Draw a table with 4 columns
    - Label them: "Step", "x", "y", and "isLarge"
    - Draw enough rows for each line of code plus the column headers

Your table should look like this:

Step	x	y	isLarge

1. Now you'll fill in the table one line at a time. For each line of code:
    - Write the current line of code in the "Step" column
    - Fill in the current value of each variable
    - Use "-" if a variable doesn't exist yet

Let's do this together:

First line: `int x = 5;`

- Write this line in the "Step" column
- Under "x", write 5 (the new value)
- Under "y", write "-" (doesn't exist yet)
- Under "isLarge", write "-" (doesn't exist yet)

Second line: `int y = x + 2;`

- Write this line in the "Step" column
- Look at the row above to see x is 5

- Calculate: 5 + 2 = 7
- Under "x", write 5 (unchanged)
- Under "y", write 7 (your calculation result)
- Under "isLarge", write "-" (still doesn't exist)

Third line: `x = y * 2;`

- Write this line in the "Step" column
- Look at the row above to see y is 7
- Calculate: 7 * 2 = 14
- Under "x", write 14 (your new calculation)
- Under "y", write 7 (unchanged)
- Under "isLarge", write "-" (still doesn't exist)

Fourth line: `bool isLarge = x > 10;`

- Write this line in the "Step" column
- Look at the row above to see x is 14
- Ask yourself: Is 14 greater than 10? Yes, so isLarge is true
- Under "x", write 14 (unchanged)
- Under "y", write 7 (unchanged)
- Under "isLarge", write "true"

2. Check your work. Your completed table should look exactly like this:

Step	x	y	isLarge
int x = 5	5	-	-
int y = x + 2	5	7	-
x = y * 2	14	7	-
bool isLarge = x > 10	14	7	true

If your table doesn't match, go back and check each calculation step by step.

Computer Steps:

1. Open your text editor and create a new file named State1.cs
   - Make sure to save it with the exact name: State1.cs
   - The name is case-sensitive, so capital 'S' matters!
2. Type in the basic program structure:

```
public class State1 {
    public static void Main() {

    }
}
```

3. Add the required line at the top:

```
using System.Diagnostics;  // This lets us use Debug.Assert
```

   If you forget this line, you'll get this error:

```
error CS0103: The name 'Debug' does not exist in the current context
```

4. Copy your code from the paper exercise into Main():

```
int x = 5;
int y = x + 2;
x = y * 2;
bool isLarge = x > 10;
```

5. After each line, add a Debug.Assert to check your work:

```
using System.Diagnostics;

public class State1 {
    public static void Main() {
        // First line: create x
        int x = 5;
        Debug.Assert(x == 5, "x should be 5");
```

```
        // Second line: calculate y
        int y = x + 2;
        Debug.Assert(x == 5, "x should still be 5");
        Debug.Assert(y == 7, "y should be 7");

        // Third line: update x
        x = y * 2;
        Debug.Assert(x == 14, "x should be 14");
        Debug.Assert(y == 7, "y should still be 7");

        // Fourth line: check if x is large
        bool isLarge = x > 10;
        Debug.Assert(x == 14, "x should still be 14");
        Debug.Assert(y == 7, "y should still be 7");
        Debug.Assert(isLarge, "isLarge should be true");
    }
}
```

Exercise 2.2: Fill-in-the-Blank State Table

Paper Steps:

1. You're given this state table:

Step	value	result
After int value = ___	10	-
After value += ___	15	-
After int result = ___	15	45

2. Work backwards to figure out the missing code:
    - We know value starts at 10
    - Then value becomes 15
    - So the second line must add 5 (because 10 + 5 = 15)
    - Result becomes 45 when value is 15
    - So result must be value 3 (*because* 15 3 = 45)
3. Write down the complete code:

```
int value = 10;
value += 5;
int result = value * 3;
```

Computer Steps:

1. Create State2.cs
2. Implement your solution with checks:

```
using System.Diagnostics;

public class State2 {
    public static void Main() {
        // Step 1: Initialize value
        int value = 10;
        Debug.Assert(value == 10, "value should be 10");

        // Step 2: Add 5 to value
        value += 5;
        Debug.Assert(value == 15, "value should be 15");

        // Step 3: Calculate result
        int result = value * 3;
        Debug.Assert(value == 15, "value should still be 15");
        Debug.Assert(result == 45, "result should be 45");
    }
}
```

Exercise 2.3: Creating Your Own State Table

Paper Steps:

1. Write a simple program that:
   - Creates two numbers
   - Adds them together
   - Checks if the sum is even
2. Create your code:

```
int a = 7;
int b = 3;
int sum = a + b;
bool isEven = (sum % 2) == 0;
```

3. Create and fill out a state table:

Step	a	b	sum	isEven
After int a = 7	7	-	-	-
After int b = 3	7	3	-	-
After int sum = a + b	7	3	10	-
After bool isEven = ...	7	3	10	true

Computer Steps:

1. Create State3.cs
2. Implement your solution:

```
using System.Diagnostics;

public class State3 {
    public static void Main() {
        // Step 1: Create first number
        int a = 7;
        Debug.Assert(a == 7, "a should be 7");

        // Step 2: Create second number
        int b = 3;
        Debug.Assert(a == 7, "a should still be 7");
        Debug.Assert(b == 3, "b should be 3");

        // Step 3: Calculate sum
        int sum = a + b;
        Debug.Assert(a == 7, "a should still be 7");
        Debug.Assert(b == 3, "b should still be 3");
        Debug.Assert(sum == 10, "sum should be 10");
```

```
        // Step 4: Check if sum is even
        bool isEven = (sum % 2) == 0;
        Debug.Assert(sum == 10, "sum should still be 10");
        Debug.Assert(isEven, "isEven should be true");
    }
}
```

Common Mistakes to Avoid

1. Skipping the paper step
    - Always work out the problem on paper first
    - Check your paper solution before coding
2. Forgetting System.Diagnostics
    - Without `using System.Diagnostics;`, you'll get the error:

      ```
      error CS0103: The name 'Debug' does not exist in the current context
      ```

    - This error means C# doesn't know what Debug.Assert is
3. Not checking your work
    - Use Debug.Assert to verify your calculations
    - Compare computer results with your paper calculations