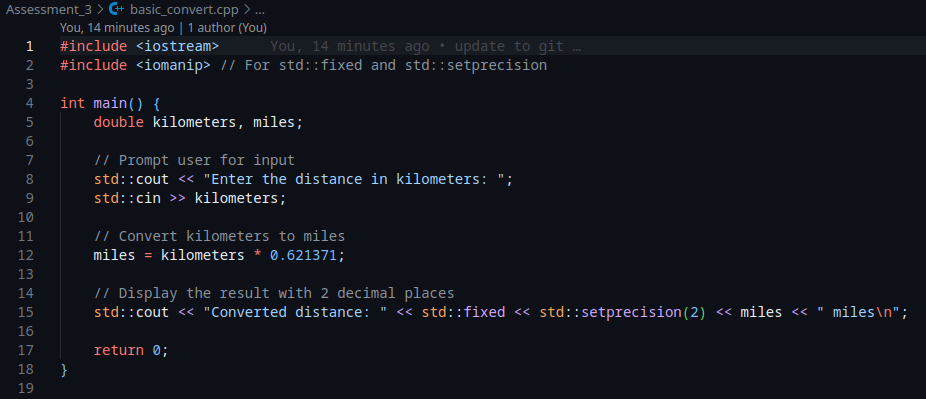
Assessment Task 3: Unit Conversion Tool

This system was created in a C++ program that performs various unit conversions. I gradually implemented more functionalities and added features to the program. The program uses **input/output**, **loops**, **conditional statements** and includes **user interaction**. Below, I document the progress through each stage, showing code, outputs and explanations as i go through the implementation stages.

# Basic Conversion Tool:

This is a simple program that demonstrates how the input is accepted from the user for the value of kilometers and the output provided the result after a simple conversion of the user input (in kilometers) to miles.  
  
**Implementation code:**



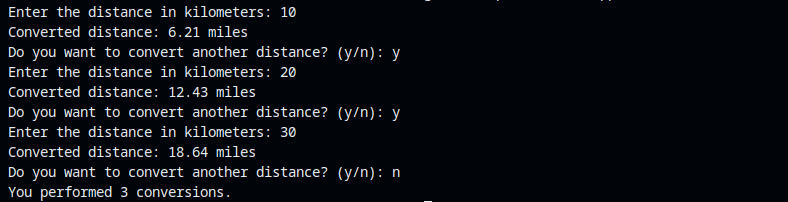
Example Output:  


# Loop for Repeated Conversion

This is the first improvement of the basic conversion code, the new added feature is a loop, that repeatedly enable the user to input multiple values for kilometers and then the output is given. After each input conversion, the user is prompted to choose between making a new conversion or exiting the program. The loop used in this case is the do-while loop.

Implementation Code:

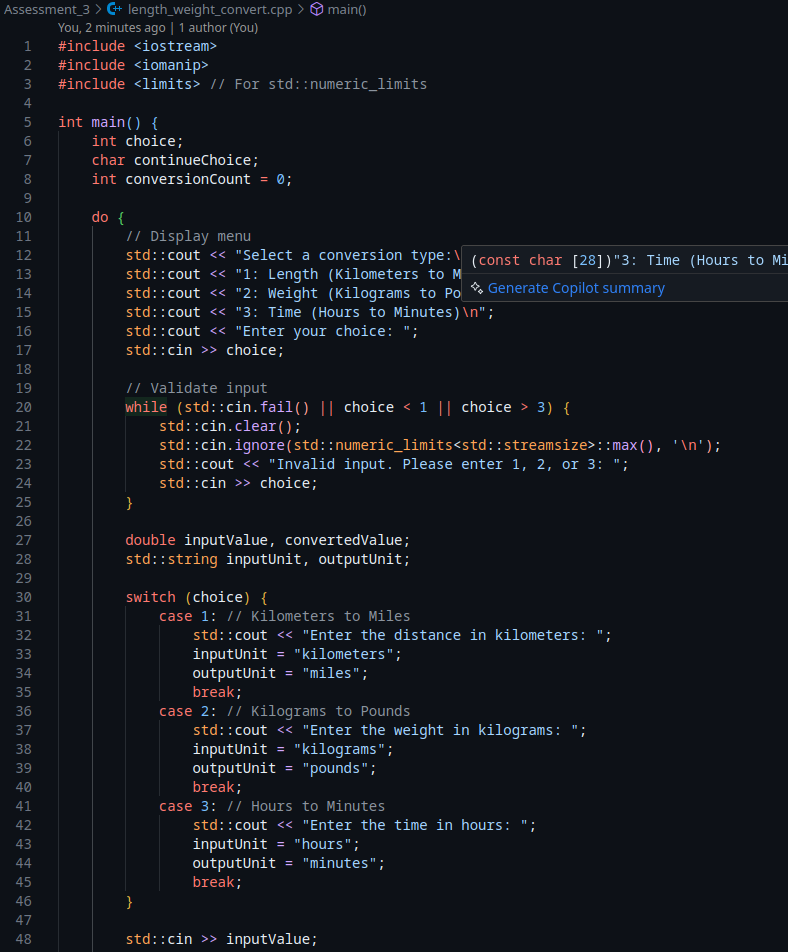


**Example Output:  
**

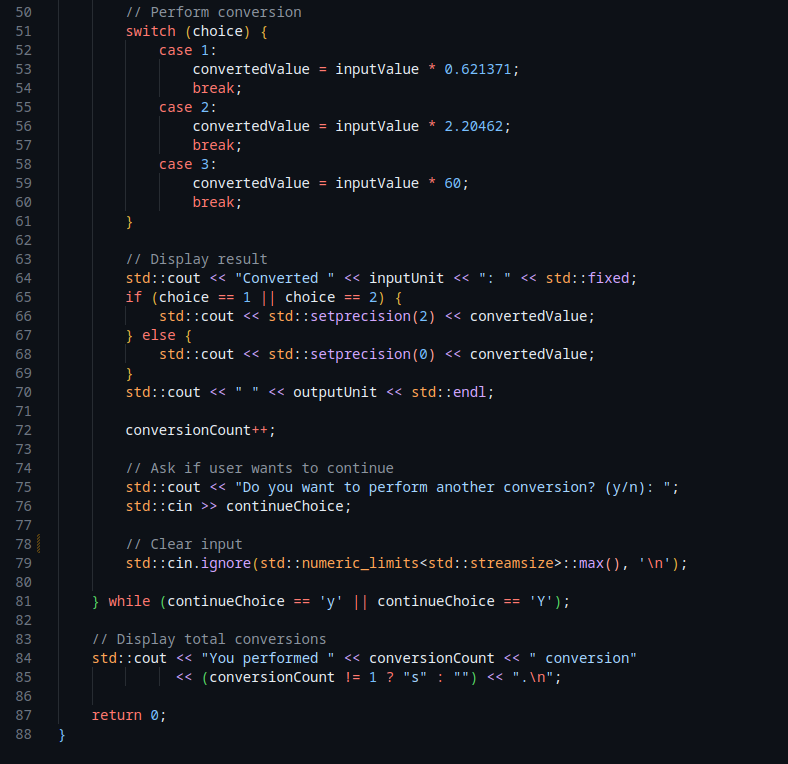
As seen from the output, after each conversion, the user is prompted whether to perform another conversion or to exit the program. On program exit, the number of conversions performed is also generated.

# Multiple Conversion Types

This stage brings further improvements to the program. In this phase, other metrics are added to the program (kilometers to miles, hours to minutes and kilograms to pounds). This improves on the previous programs and adds more complexity and improvement to the program. I implemented the program using do-while loops and switch statements. In the ‘do’ block is the display menu where the user can choose the type of conversion to be made. The conversion types are present in the switch statements which are used to select one of the multiple types available to the user.

**Implementation Code:   
**

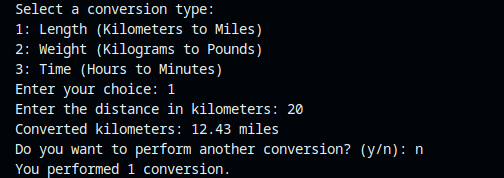
This block displays how the display menu, input validation and conversion type was implemented



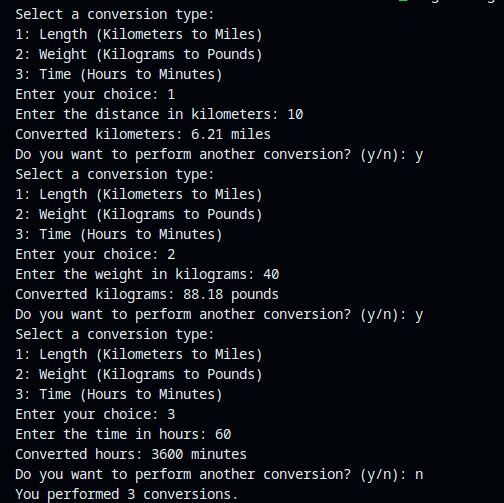
This block shows how the conversion was performed based on the chosen conversion type.

It also shows how the result would be printed.

**Example Single Output:**



**Example Multiple Output:**

****

This output demonstrated the program’s full functionality whereby the user can perform multiple operations of conversions in a single run

# 

# 

# 

# Alternative approach

Instead of using a do-while loop, this program uses a **while loop with a priming read strategy**:

**What is a priming read strategy?**

A priming read is a programming technique where you take an input before entering a loop. It’s used to set up the initial condition that the loop will test.

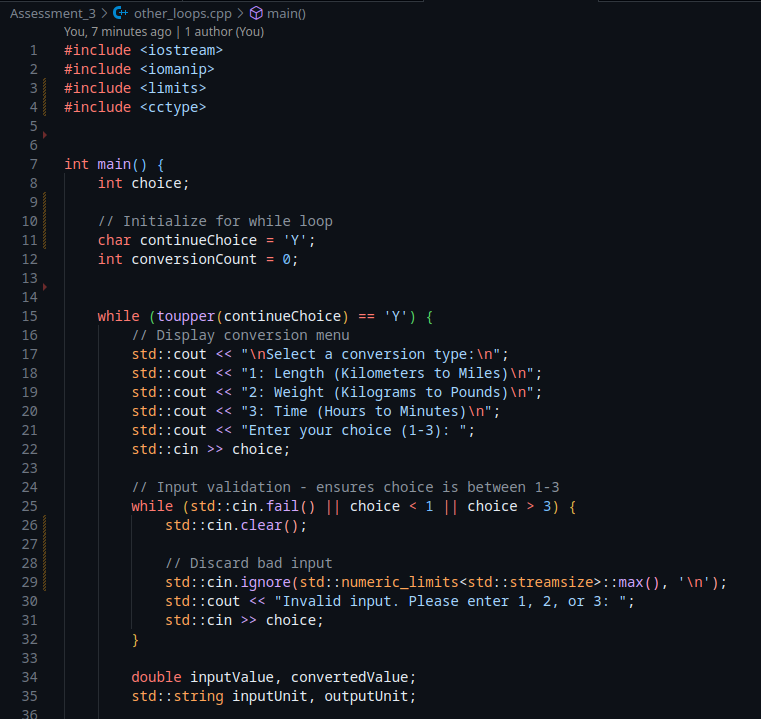
This is especially common with while loops, where the condition is checked before each loop iteration. You must give the loop a valid reason to start, which is done through the priming input (or “read”).

#### **Why Use a Priming Read?**

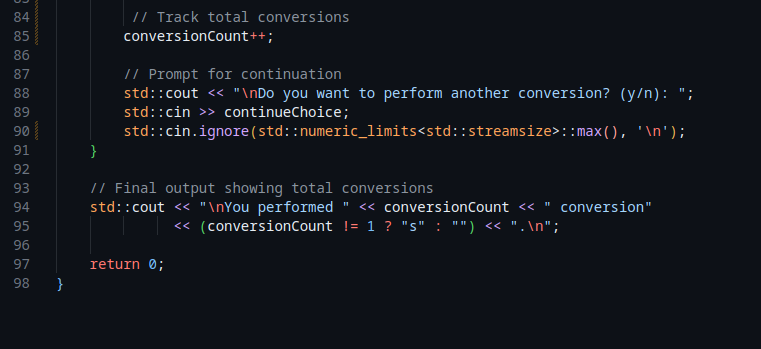
Because a while loop doesn’t execute unless its condition is true from the start, you need to collect the user's first input before the loop begins — that's the priming read.

Without it, the loop condition might be uninitialized or false, and the loop would never run.

Example code







In our program, we ask the user:

"Do you want to perform another conversion? (y/n):"

We store this in a variable (e.g., continueChoice).  
 Before entering the loop, we **set** continueChoice = 'Y' — this is our **priming value**.

Then the while loop checks:

while (toupper(continueChoice) == 'Y')

Because continueChoice is already 'Y', the loop runs for the first time.

Example output:   
