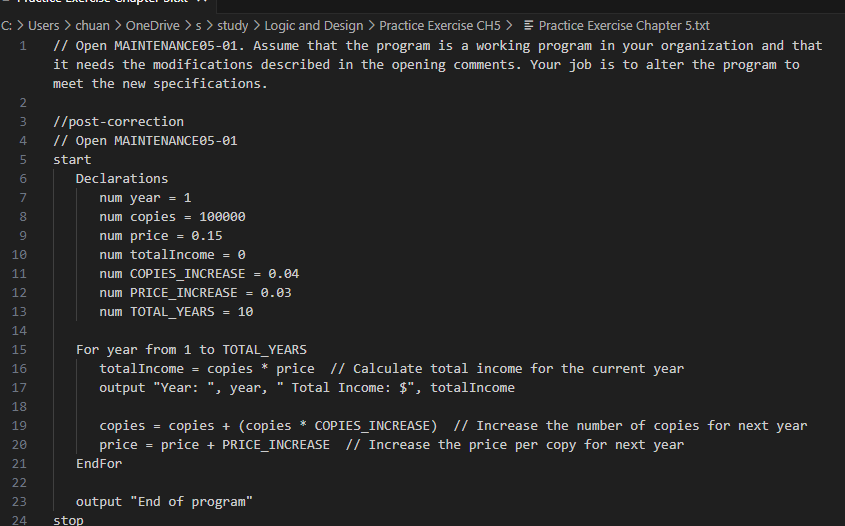
1.Open MAINTENANCE05-01. Assume that the program is a working program in your organization and that it needs the modifications described in the opening comments. Your job is to alter the program to meet the new specifications.

The purpose of this program is to show the expected revenue of QuickCopy, Inc. for each of the next 10 years, taking into account that the number of copies made each year, as well as the price per copy, will grow. However, the structure of this program is not efficient because it calculates and outputs the results for each year by repeating the same blocks of code over and over again without utilizing a loop structure.

To improve efficiency, we can use a for loop or while loop instead of repeating blocks of code, thus only having to code the calculation and output once. This not only reduces the length of the code, but also makes the program easier to maintain and understand.

A For loop is used to repeatedly calculate and output the total income for each of the next 10 years, which makes the program more compact and efficient.At the beginning of each loop, the totalIncome for the current year is first calculated and then that year and its total income are output.Subsequently, the values of copies and price are updated according to the set growth rate in preparation for the calculation of the next loop.At the end of the loop, the program outputs the end message.



DEBUG05-01

1. **End condition error**: **END\_YEAR** was set to 30, which appears to be intended to represent a 30-year time span, not an end year. This resulted in an incorrect end condition for the **while** loop.
2. **The year was not updated in the loop**: within the loop body, the year was not incremented, which would have resulted in an infinite loop.
3. endif **should be** endwhile: the end keyword of the loop should be **endwhile** instead of **endif**.

DEBUG05-02

1. **The** acctNum **variable was misspelled**: **acctNUm** was used instead of **acctNum** in the **while** loop.
2. QUIT **variable not declared**: a special value **QUIT** needs to be declared for the user to exit the program.
3. monthCounter **variable not initialized**: the **monthCounter** variable was used in the **printCoupons() method** but was not initialized before use.
4. **The year counter** yearCounter **is not incremented**: After processing the 12 months of a year, it is necessary to increment the value of the year counter **yearCounter** and reset the **monthCounter** to 1 to start counting the months of the next year.
5. **Logic error in the** printCoupons() method: after the inner loop is finished, **monthCounter** should be reset and the value of **yearCounter** should be incremented, otherwise the inner loop will not be executed again.

DEBUG05-03

The ability to generate all three-digit combinations from 000 to 999 is ensured by appropriately adding the corresponding digit to each level of the loop and resetting the value of the next digit before proceeding to the next level of the loop.

This logic ensures that:

* **digit3** is increased from 0 to 9 in the innermost loop, and then **digit2** is increased by 1.
* Each time **digit2** is increased, **digit3** starts counting again from 0 until 9.
* Similarly, every time **digit2** loops, **digit1** increases by 1, and both **digit2** and **digit3** reset to 0 to start a new round.

1. In Session 2, you learned that many programming languages allow you to generate a random number between one and a limiting value (named LIMIT) by using a statement similar to randomNumber = random(LIMIT). In Session 4, you created the logic for a guessing game in which the application generates a random number and the player tries to guess it. Now, create the guessing game itself. After each guess, display a message indicating whether the player’s guess was correct, too high, or too low. When the player eventually guesses the correct number, display a count of the number of guesses that were required. Save the game file for submission.

We'll implement a loop that continues until the player guesses the correct number, and we'll track the number of attempts. This will provide a good exercise in using loops, conditional statements, and generating random numbers within a specified range.

### Explanation:

1. Initialization: We set up the necessary variables. **LIMIT** determines the maximum possible value for the random number. **randomNumber** is the target number the player needs to guess. **playerGuess** stores each guess from the player. **guessCount** tracks how many guesses the player has made.
2. Random Number Generation: We generate a random number between 1 and **LIMIT**. The **+ 1** is necessary if the **random** function generates numbers starting from 0 to **LIMIT-1**, ensuring the range is 1 to **LIMIT**.
3. Game Loop: The game loop continues until the player's guess matches the **randomNumber**. Inside the loop, we prompt the player for a guess, increment **guessCount**, and provide feedback if the guess is too high or too low.
4. Once the correct number is guessed, we exit the loop and display a message congratulating the player, including the number of guesses it took to find the right number.

This pseudocode represents a simple number-guessing game that introduces fundamental programming concepts such as loops, conditional statements, user input, and generating random numbers.

