Practical 2

Procedural Programming Paradigm

Guidance for developing solutions:

- Focus on breaking down each exercise into smaller, well-defined procedures.
- Use clear variable names, comments, and indentation for readability.
- Test your code thoroughly with different inputs to ensure correctness.
- Consider edge cases and error handling for robust programs.

Exercise 1a: Area and Perimeter Calculator

Write separate procedures to:

1. Calculate the area of a rectangle given its length and width.
2. Calculate the perimeter of a rectangle given its length and width.

Then, write a main procedure that prompts the user for the length and width, calls the two previously defined procedures to calculate the area and perimeter, and prints the results.

Exercise 1b: Enhancements

Modify the main procedure to handle different shapes (e.g., square, triangle) by asking the user for the shape type and using appropriate calculations.

- Add error handling to check for invalid input values (e.g., negative dimensions).

Exercise 2: Guessing Game with Limits

Write procedures to:

1. Generate a random number between 1 and 100.
2. Get the user's guess and validate it to ensure it's within the valid range (1-100).
3. Check if the guess is correct.
4. Provide hints (higher or lower) if the guess is incorrect.

Then, write a main procedure that:

- Calls the generate_random_number procedure.
- Starts a loop that:
    - Calls the get_user_guess procedure and checks for valid input.
    - Calls the check_guess procedure and handles the result.
        - If correct, congratulate the user and end the game.
        - If incorrect, call the provide_hint procedure and continue looping.
- Set a limit on the number of guesses allowed (e.g., 10 guesses).
- After the limit is reached, inform the user they ran out of guesses and reveal the secret number.

Exercise 3: Text Analysis

Write procedures to:

1. Count the number of characters in a given string.
2. Count the number of words in a string, considering spaces and punctuation.
3. Count the number of sentences in a string, considering punctuation marks like periods and exclamation points.

Then, write a main procedure that:

- Prompts the user for a text input.
- Calls the character counting, word counting, and sentence counting procedures and stores the results.
- Prints the analysis results, including the number of characters, words, and sentences.

Exercise 4: Text Analyzer with Statistics **(optional)**

- Expand the text analysis exercise to calculate additional statistics:
    - Average word length
    - Most frequent word (excluding stop words like "the", "a", "is")
    - Number of unique words
- Implement procedures for each statistic calculation.
- Modify the main procedure to display the extended analysis results.

Exercise 5: Write a Python program that takes three numerical inputs from the user (a, b, and c) representing the coefficients of a quadratic equation (ax^2 + bx + c = 0). The program should:

1. Use separate procedures for:
    - Calculating the discriminant (b^2 - 4ac).
    - Checking if the discriminant is positive, negative, or zero.
    - Calculating the roots (x1 and x2) based on the discriminant value using the appropriate formulas:
        - For positive discriminant: x1 = (-b ± √discriminant) / (2a) and x2 = (-b ∓ √discriminant) / (2a)
        - For negative discriminant: Complex roots (provide real and imaginary parts separately: x1 = (-b ± √(-discriminant)) / (2a)i and x2 = (-b ∓ √(-discriminant)) / (2a)i
        - For zero discriminant: Single root: x1 = x2 = -b / (2a)
2. Print the calculated roots (x1 and x2) in a user-friendly format.

Formulas:

- Discriminant: b^2 - 4ac
- Roots:
    - Positive discriminant:
        - x1 = (-b + √discriminant) / (2a)
        - x2 = (-b - √discriminant) / (2a)
    - Negative discriminant (complex roots):
        - x1 = (-b + √(-discriminant)) / (2a)i
        - x2 = (-b - √(-discriminant)) / (2a)i
    - Zero discriminant:
        - x1 = x2 = -b / (2a)

Procedure Usage:

- Use one procedure for each calculation (discriminant, checking discriminant value, and calculating roots based on the value).
- The main program prompts for user input, calls the procedures, and formats the output.

Example:

Enter coefficients (a, b, c): 1, 2, 1
Discriminant is 0.
Roots are: x1 = x2 = -1.0