Practical 2- Programming paradigms.

IT23632264

HITHUSH M

1. Why was efficiency so important for early-generation programming languages?

   Early computers had limited memory and processing power, so languages like FORTRAN were designed to maximize performance. Efficient code meant faster execution and better use of scarce resources.

2. What were the advantages and disadvantages of pseudocode interpreters?

   Advantages:

- Easier to understand and teach.
- Platform-independent logic representation.

  Disadvantages:

- Not executable directly.
- May lack precision in syntax and semantics.

3. Does FORTRAN comply with the syntactic consistency principle? Consider: (-B + SQRT(B**2-4*A*C)) / (2 * A)

   FORTRAN has inconsistent syntax, especially in expressions like: (−B+B2−4AC)/(2A)(−B+B2−4AC)/(2A)
   FORTRAN requires verbose and sometimes unintuitive syntax (e.g., `SQRT(B**2 - 4*A*C)`), which can hinder readability.

4. Compare FORTRAN's syntax to C or Python. Which is more consistent?

   FORTRAN is verbose, has older syntax and less intuitive. Python and C are more structured and readable.

5. The GOTO statement is unavoidable in early FORTRAN. How does it affect readability, writability, and reliability?

   The GOTO statements made the language less readable as it was harder follow flow. It was easy to implement but error prone and less reliable due to the mazed code.

6. Does FORTRAN allow for information hiding (a form of abstraction)?

   Early FORTRAN versions do not support information hiding. Later versions (like F95) introduced modules for abstraction, but it's still limited compared to modern languages.

7. What are the problems with the COMMON block statement?

   Shared memory across procedures, hard to debug and maintain, risk of name conflicts.

8. What is the computed GOTO statement? How does it work?

   The computed GOTO statement is a control flow mechanism that allows the program to jump to one of several labels based on the value of an index variable. For example, GOTO (10, 20, 30), I will jump to label 10 if I = 1, label 20 if I = 2, and so on. It functions like a switch-case statement but is less readable and more error-prone, especially if the index value is out of range.

9. What were FORTRAN's major contributions to programming?

   First high-level language. Introduced structured programming (later versions). Optimized for scientific computing. Influenced many languages

10. Why is FORTRAN still in use today despite its criticisms?

    ORTRAN is still used today because of its reliability, performance, and the vast amount of legacy code in scientific and engineering fields. Many high-performance computing applications, such as climate modeling and physics simulations, were written in FORTRAN and continue to be maintained. Its compilers are highly optimized for numerical tasks, and rewriting these systems in newer languages would be costly and risky.

11. Name two major problems associated with FORTRAN. Provide examples.

    One major problem with FORTRAN is its reliance on GOTO statements, which leads to unreadable and error-prone code. For example, using multiple GOTO statements to simulate loops can result in tangled logic. Another problem is the use of COMMON blocks, which allow global variable sharing and make it difficult to manage data safely. These features reduce modularity and increase the risk of bugs.

12. How does the INTEGER type affect syntax design in FORTRAN?

    The INTEGER type in FORTRAN requires explicit declaration, which can lead to type mismatch errors if not handled carefully. Unlike modern languages that support type inference, FORTRAN demands that programmers specify types for each variable. This adds verbosity to the code and increases the chance of mistakes, especially in large programs.

13. Name three features that could be removed to make FORTRAN more secure.

To make FORTRAN more secure, the COMMON block feature should be removed to prevent uncontrolled data sharing. The GOTO statement should also be eliminated in favor of structured control flow like DO loops and IF blocks. Finally, implicit typing should be disabled by enforcing the use of IMPLICIT NONE, which requires all variables to be explicitly declared, reducing errors from undeclared or misspelled variables.

14. New: How were loops implemented in early FORTRAN before DO loops?

    Before DO loops were introduced, loops in early FORTRAN were implemented using a combination of conditional IF statements and GOTO labels. A counter variable would be initialized, and the program would jump back to a label as long as a condition was met. This method worked but lacked structure and clarity, making the code harder to read and maintain.

Comprehensive exercise

```
PROGRAM GUESS_GAME
    INTEGER CHOICE, TARGET, GUESS, ATTEMPTS, TOTAL_GAMES
    REAL R
    COMMON /GAME_STATS/ TOTAL_GAMES
    TOTAL_GAMES = 0
1    PRINT *, '============================'
    PRINT *, '    GUESSING GAME MENU    '
    PRINT *, '============================'
    PRINT *, '1. Start New Game'
    PRINT *, '2. Show Total Games Played'
    PRINT *, '3. Exit'
    PRINT *, 'Enter your choice (1-3):'
    READ *, CHOICE
    IF (CHOICE .EQ. 1) GOTO 10
    IF (CHOICE .EQ. 2) GOTO 20
    IF (CHOICE .EQ. 3) GOTO 30
    PRINT *, 'Invalid choice. Try again.'
```

```
    GOTO 1
10  CALL RANDOM_NUMBER(R)
    TARGET = INT(R * 100) + 1
    ATTEMPTS = 0
    PRINT *, 'Guess a number between 1 and 100:'
11  READ *, GUESS
    ATTEMPTS = ATTEMPTS + 1
    IF (GUESS .GT. TARGET) THEN
      PRINT *, 'Too High. Try again:'
      GOTO 11
    ELSE IF (GUESS .LT. TARGET) THEN
      PRINT *, 'Too Low. Try again:'
      GOTO 11
    ELSE
      PRINT *, 'Correct! You guessed it in', ATTEMPTS, 'attempts.'
    END IF
    TOTAL_GAMES = TOTAL_GAMES + 1
    GOTO 1
20  PRINT *, 'Total games played:', TOTAL_GAMES
    GOTO 1
30  PRINT *, 'Thank you for playing!'
    STOP
    END
```