Week 5: Revision of FORTRAN and BASIC Programming

1. Introduction to FORTRAN and BASIC

Importance of Revisiting FORTRAN and BASIC:

FORTRAN and BASIC are two of the earliest high-level programming languages that have significantly influenced the development of modern programming paradigms. Revisiting these languages provides engineers with a deeper understanding of programming fundamentals, historical context, and the evolution of computational methods used in engineering applications.

Applications in Engineering:

- **FORTRAN:** Predominantly used in numerical and scientific computing, including computational fluid dynamics, finite element analysis, and weather prediction models.
- **BASIC:** Utilized in educational settings, early personal computing, and simple control systems in machinery and robotics.

---

2. FORTRAN Programming Language

History and Significance:

Developed in the 1950s by IBM, FORTRAN (Formula Translation) was designed to simplify the programming process for mathematical and scientific computations. It introduced many programming concepts such as loops, conditionals, and subroutines that are fundamental in modern programming.

Syntax and Structure:

- **Program Structure:**
    - **Program Statement:** Begins with PROGRAM and ends with END PROGRAM.
    - **Variable Declarations:** Explicitly declare variables with types like INTEGER, REAL, DOUBLE PRECISION, etc.
    - **Control Structures:** Utilize IF, DO loops, and GOTO statements for flow control.
    - **Input/Output:** Handled using READ and PRINT statements.
- **Example: Simple FORTRAN Program**

fortran

Copy code

```
PROGRAM HelloWorld
  PRINT *, 'Hello, World!'
END PROGRAM HelloWorld
```

Explanation:

1. **PROGRAM HelloWorld**
   - Defines the start of the program named HelloWorld.
2. **PRINT *, 'Hello, World!'**
   - Outputs the string "Hello, World!" to the console.
3. **END PROGRAM HelloWorld**
   - Marks the end of the program.

---

3. BASIC Programming Language

History and Significance:

BASIC, an acronym for "Beginner's All-purpose Symbolic Instruction Code," was developed in the mid-1960s at Dartmouth College. Its primary goal was to make programming accessible to students and beginners, promoting widespread use of computers in education and personal computing.

Syntax and Structure:

- **Basic Structure:**
  - **Line Numbers:** Traditional BASIC uses line numbers to indicate the order of execution.
  - **Variable Naming:** Variables are dynamically typed based on their names (e.g., variables ending with $ are strings).
  - **Control Structures:** Utilize IF...THEN, FOR...NEXT loops, and GOTO statements.
  - **Input/Output:** Managed through INPUT and PRINT statements.
- **Example: Simple BASIC Program**

basic

Copy code

```
10 PRINT "Hello, World!"

20 END
```

Explanation:

1. **10 PRINT "Hello, World!"**
   - Displays the string "Hello, World!" on the screen.
2. **20 END**
   - Terminates the program.

---

4. Comparative Analysis of FORTRAN and BASIC

Feature	FORTRAN	BASIC
Purpose	Scientific and engineering computations	Beginner-friendly programming
Syntax	Strict, requires explicit declarations	Flexible, often uses line numbers
Control Structures	IF, DO, GOTO	IF...THEN, FOR...NEXT, GOTO
Variable Typing	Explicit (e.g., REAL, INTEGER)	Implicit (based on variable names)
Use Cases	Numerical simulations, computational physics	Educational purposes, early personal computing
Convergence	Designed for high-performance computing	Designed for ease of use and learning

---

5. Applications of FORTRAN and BASIC in Engineering

FORTRAN:

- **Computational Fluid Dynamics (CFD):** Simulating fluid flows in various engineering systems.
- **Finite Element Analysis (FEA):** Analyzing structural integrity and stress distribution.
- **Weather Prediction Models:** Numerical weather forecasting and climate modeling.
- **Aerospace Engineering:** Trajectory calculations and simulations for spacecraft and aircraft.

BASIC:

- **Educational Tools:** Teaching programming fundamentals to engineering students.
- **Early Control Systems:** Implementing simple control algorithms in machinery.
- **Rapid Prototyping:** Developing quick simulations and calculations for engineering problems.
- **Legacy Systems:** Maintaining older engineering software that was originally developed in BASIC.

---

6. Writing and Executing Simple Programs

FORTRAN Programming:

1. **Choose an IDE or Compiler:**
    - Examples: GNU Fortran (gfortran), Intel Fortran Compiler, Visual Studio with FORTRAN extensions.
2. **Write the Program:**

- Use a text editor or IDE to create a .f90 file containing FORTRAN code.

3. **Compile the Program:**
   - Example Command: gfortran CircleArea.f90 -o CircleArea
4. **Execute the Program:**
   - Run the compiled executable.
   - Example Command: ./CircleArea

BASIC Programming:

1. **Choose a BASIC Interpreter or Compiler:**
   - Examples: QB64, FreeBASIC, Microsoft Small Basic.
2. **Write the Program:**
   - Use a text editor or IDE to create a .bas file containing BASIC code.
3. **Run the Program:**
   - Use the interpreter to execute the code directly or compile it.
   - Example: Open QB64, load the program, and run.

---

7. Hands-On Exercises

Exercise 1: FORTRAN Programming

Task: Write a FORTRAN program to calculate the factorial of a number.

Instructions:

1. **Define the Program Structure:**
   - Begin with PROGRAM Factorial.
   - End with END PROGRAM Factorial.
2. **Declare Variables:**
   - Use INTEGER for the number and factorial.
3. **Input the Number:**
   - Use PRINT and READ statements.
4. **Calculate the Factorial:**
   - Use a DO loop.
5. **Output the Result:**
   - Use PRINT statement.

Sample Code:

fortran

Copy code

```
PROGRAM Factorial
  IMPLICIT NONE
  INTEGER :: N, i, fact

  PRINT *, 'Enter a positive integer:'
  READ *, N

  IF (N < 0) THEN
    PRINT *, 'Factorial is not defined for negative numbers.'
    STOP
  END IF

  fact = 1
  DO i = 1, N
    fact = fact * i
  END DO

  PRINT *, 'The factorial of ', N, ' is ', fact
END PROGRAM Factorial
```

Exercise 2: BASIC Programming

Task: Write a BASIC program to solve the quadratic equation ax2+bx+c=0ax^2 + bx + c = 0ax2+bx+c=0.

Instructions:

1. **Prompt User for Coefficients:**
   - Use PRINT and INPUT statements.
2. **Calculate Discriminant:**
   - D=b2−4acD = b^2 - 4acD=b2−4ac
3. **Determine Nature of Roots:**
   - Based on the discriminant.

4. **Calculate and Display Roots:**
    - Use IF...THEN statements.

Sample Code:

basic

Copy code

```
10 PRINT "Quadratic Equation Solver"
20 PRINT "Enter coefficient a:"
30 INPUT A
40 PRINT "Enter coefficient b:"
50 INPUT B
60 PRINT "Enter coefficient c:"
70 INPUT C
80 D = B^2 - 4*A*C
90 IF D > 0 THEN GOTO 110
100 IF D = 0 THEN GOTO 130
110 PRINT "Roots are real and distinct."
120 GOTO 160
130 PRINT "Roots are real and equal."
140 GOTO 160
160 IF D >= 0 THEN
170 X1 = (-B + SQR(D)) / (2*A)
180 X2 = (-B - SQR(D)) / (2*A)
190 PRINT "Root 1: "; X1
200 PRINT "Root 2: "; X2
210 ELSE
220 REAL_PART = -B / (2*A)
230 IMAG_PART = SQR(-D) / (2*A)
240 PRINT "Roots are complex."
250 PRINT "Root 1: "; REAL_PART; " + "; IMAG_PART; "i"
260 PRINT "Root 2: "; REAL_PART; " - "; IMAG_PART; "i"
270 END IF
```

Exercise 3: Programming Practice

- **FORTRAN:** Implement a program to perform matrix multiplication.
- **BASIC:** Create a program to simulate simple harmonic motion.

Exercise 4: Visualization Project

- **FORTRAN:** Modify the factorial program to record and display each multiplication step.
- **BASIC:** Extend the quadratic solver to plot the roots as coefficients vary.

---

8. Historical Significance and Modern Applications

FORTRAN:

- **Foundation for Scientific Computing:**
  - Influenced the development of languages like MATLAB and Julia.
- **Legacy Code:**
  - Extensive use in legacy systems within aerospace, automotive, and energy sectors.
- **High-Performance Computing:**
  - Optimized for numerical computations and parallel processing.

BASIC:

- **Democratizing Programming:**
  - Made programming accessible to non-experts and beginners.
- **Personal Computing Revolution:**
  - Integral in the development of early personal computers (e.g., Apple II, Commodore 64).
- **Educational Tool:**
  - Continues to be used for teaching fundamental programming concepts.

---

9. Comparative Code Explanation

FORTRAN vs. BASIC: Factorial Calculation

FORTRAN:

fortran

Copy code

```
PROGRAM Factorial
  IMPLICIT NONE
```

```
   INTEGER :: N, i, fact

   PRINT *, 'Enter a positive integer:'
   READ *, N

   IF (N < 0) THEN
      PRINT *, 'Factorial is not defined for negative numbers.'
      STOP
   END IF

   fact = 1
   DO i = 1, N
      fact = fact * i
   END DO

   PRINT *, 'The factorial of ', N, ' is ', fact
END PROGRAM Factorial
```

Explanation:

- **Variable Declarations:** FORTRAN requires explicit declaration of variable types.
- **Control Structures:** Uses DO...END DO for looping.
- **Input/Output:** Utilizes PRINT and READ for user interaction.
- **Error Handling:** Checks for negative input and terminates the program if found.

BASIC:

basic

Copy code

```
10 PRINT "Factorial Calculator"
20 PRINT "Enter a positive integer:"
30 INPUT N
40 IF N < 0 THEN
50 PRINT "Factorial is not defined for negative numbers."
60 END
```

```
70 fact = 1

80 FOR i = 1 TO N

90 fact = fact * i

100 NEXT i

110 PRINT "The factorial of "; N; " is "; fact

120 END
```

Explanation:

- **Line Numbers:** BASIC uses line numbers to determine the order of execution.
- **Control Structures:** Uses FOR...NEXT loops for iteration.
- **Variable Typing:** BASIC infers variable types based on naming conventions (no explicit declarations).
- **Input/Output:** Utilizes PRINT and INPUT for user interaction.
- **Error Handling:** Uses conditional IF...THEN statements to handle invalid input and ends the program accordingly.

---

10. Practical Exercise - Implementing Gaussian Elimination in FORTRAN

Task: Extend the Gaussian Elimination program to handle any size of the system.

Instructions:

1. **Modify the Program to Accept Variable Sizes:**
   - Remove fixed N = 3 and allow user input for the number of equations.
2. **Dynamic Memory Allocation:**
   - Use allocatable arrays or dynamic memory structures if supported.
3. **Enhance Error Handling:**
   - Include checks for singular matrices and infinite solutions.
4. **Optimize for Efficiency:**
   - Implement partial pivoting to improve numerical stability.

Sample Code Adjustments:

fortran

Copy code

```
PROGRAM GaussianElimination

  IMPLICIT NONE

  INTEGER :: N, i, j, k
```

```
REAL, ALLOCATABLE :: A(:,:)
REAL, ALLOCATABLE :: x(:)
REAL :: factor, sum

PRINT *, 'Enter the number of equations:'
READ *, N

ALLOCATE(A(N, N+1))
ALLOCATE(x(N))

PRINT *, 'Enter the augmented matrix coefficients:'

DO i = 1, N
   PRINT *, 'Row ', i, ':'
   DO j = 1, N+1
      READ *, A(i,j)
   END DO
END DO

! Forward Elimination with Partial Pivoting
DO k = 1, N-1
   ! Partial Pivoting
   IF (A(k,k) == 0.0) THEN
      PRINT *, 'Zero pivot encountered. Attempting to swap rows.'
      DO i = k+1, N
         IF (A(i,k) /= 0.0) THEN
            A([k,i], :) = A([i,k], :)
            PRINT *, 'Swapped row ', k, ' with row ', i
            EXIT
         END IF
      END DO
```

```
        IF (A(k,k) == 0.0) THEN
          PRINT *, 'Cannot perform partial pivoting. Singular matrix.'
          STOP
        END IF
      END IF
      DO i = k+1, N
        factor = A(i,k) / A(k,k)
        A(i, k:N+1) = A(i, k:N+1) - factor * A(k, k:N+1)
      END DO
    END DO

    ! Back Substitution
    DO i = N, 1, -1
      sum = 0.0
      DO j = i+1, N
        sum = sum + A(i,j) * x(j)
      END DO
      x(i) = (A(i,N+1) - sum) / A(i,i)
    END DO

    PRINT *, 'Solution:'
    DO i = 1, N
      PRINT *, 'x(', i, ') = ', x(i)
    END DO

    DEALLOCATE(A)
    DEALLOCATE(x)
  END PROGRAM GaussianElimination
```

Explanation:

- **Dynamic Sizing:** The program now dynamically allocates memory based on user input for the number of equations.

- **Partial Pivoting:** Enhances numerical stability by swapping rows when a zero pivot is encountered.
- **Error Handling:** Checks for singular matrices and terminates the program gracefully if detected.
- **Flexibility:** Can handle systems of any size up to memory constraints.

Appendix: C# Code Examples

1. FORTRAN Factorial Program Explained

Code Snippet: Calculating Factorial in FORTRAN

fortran

Copy code

```
PROGRAM Factorial
  IMPLICIT NONE
  INTEGER :: N, i, fact

  PRINT *, 'Enter a positive integer:'
  READ *, N

  IF (N < 0) THEN
    PRINT *, 'Factorial is not defined for negative numbers.'
    STOP
  END IF

  fact = 1
  DO i = 1, N
    fact = fact * i
  END DO

  PRINT *, 'The factorial of ', N, ' is ', fact
END PROGRAM Factorial
```

Explanation:

1. **PROGRAM Factorial**
    - Starts the program named Factorial.
2. **IMPLICIT NONE**
    - Disables implicit variable typing, requiring all variables to be explicitly declared.
3. **INTEGER :: N, i, fact**
    - Declares three integer variables: N (the number to calculate factorial for), i (loop counter), and fact (to store the factorial result).
4. **PRINT *, 'Enter a positive integer:'**
    - Prompts the user to input a positive integer.
5. **READ *, N**
    - Reads the user input and assigns it to the variable N.
6. **IF (N < 0) THEN ... END IF**
    - Checks if the input is negative and handles it by displaying a message and terminating the program.
7. **fact = 1**
    - Initializes the factorial result to 1.
8. **DO i = 1, N**
    - Begins a loop from 1 to N.
9. **fact = fact * i**
    - Multiplies the current value of fact by i in each iteration.
10. **END DO**
    - Ends the loop.
11. **PRINT *, 'The factorial of ', N, ' is ', fact**
    - Outputs the calculated factorial.
12. **END PROGRAM Factorial**
    - Marks the end of the program.

---

2. BASIC Quadratic Solver Explained

Code Snippet: Solving Quadratic Equations in BASIC

basic

Copy code

```
10 PRINT "Quadratic Equation Solver"
```

```
20 PRINT "Enter coefficient a:"

30 INPUT A

40 PRINT "Enter coefficient b:"

50 INPUT B

60 PRINT "Enter coefficient c:"

70 INPUT C

80 D = B^2 - 4*A*C

90 IF D > 0 THEN GOTO 110

100 IF D = 0 THEN GOTO 130

110 PRINT "Roots are real and distinct."

120 GOTO 160

130 PRINT "Roots are real and equal."

140 GOTO 160

160 IF D >= 0 THEN

170 X1 = (-B + SQR(D)) / (2*A)

180 X2 = (-B - SQR(D)) / (2*A)

190 PRINT "Root 1: "; X1

200 PRINT "Root 2: "; X2

210 ELSE

220 REAL_PART = -B / (2*A)

230 IMAG_PART = SQR(-D) / (2*A)

240 PRINT "Roots are complex."

250 PRINT "Root 1: "; REAL_PART; " + "; IMAG_PART; "i"

260 PRINT "Root 2: "; REAL_PART; " - "; IMAG_PART; "i"

270 END IF
```

Explanation:

1. **10 PRINT "Quadratic Equation Solver"**
   - Displays the program title.
2. **20 PRINT "Enter coefficient a:"**
   - Prompts the user to enter the coefficient a.
3. **30 INPUT A**

   - Reads the user input for a.
4. **40 PRINT "Enter coefficient b:"**
   - Prompts the user to enter the coefficient b.
5. **50 INPUT B**
   - Reads the user input for b.
6. **60 PRINT "Enter coefficient c:"**
   - Prompts the user to enter the coefficient c.
7. **70 INPUT C**
   - Reads the user input for c.
8. **80 D = B^2 - 4*A*C**
   - Calculates the discriminant DDD.
9. **90 IF D > 0 THEN GOTO 110**
   - Checks if discriminant is positive for real and distinct roots.
10. **100 IF D = 0 THEN GOTO 130**
    - Checks if discriminant is zero for real and equal roots.
11. **110 PRINT "Roots are real and distinct."**
    - Informs the user about the nature of roots.
12. **120 GOTO 160**
    - Jumps to root calculation.
13. **130 PRINT "Roots are real and equal."**
    - Informs the user about the nature of roots.
14. **140 GOTO 160**
    - Jumps to root calculation.
15. **160 IF D >= 0 THEN**
    - Checks if roots are real.
16. **170 X1 = (-B + SQR(D)) / (2*A)**
    - Calculates the first root.
17. **180 X2 = (-B - SQR(D)) / (2*A)**
    - Calculates the second root.
18. **190 PRINT "Root 1: "; X1**
    - Displays the first root.

19. **200 PRINT "Root 2: "; X2**
    - Displays the second root.
20. **210 ELSE**
    - Handles complex roots.
21. **220 REAL_PART = -B / (2*A)**
    - Calculates the real part of the roots.
22. **230 IMAG_PART = SQR(-D) / (2*A)**
    - Calculates the imaginary part of the roots.
23. **240 PRINT "Roots are complex."**
    - Informs the user about complex roots.
24. **250 PRINT "Root 1: "; REAL_PART; " + "; IMAG_PART; "i"**
    - Displays the first complex root.
25. **260 PRINT "Root 2: "; REAL_PART; " - "; IMAG_PART; "i"**
    - Displays the second complex root.
26. **270 END IF**
    - Ends the conditional statement.

---

3. Additional FORTRAN and BASIC Code Examples

FORTRAN: Matrix Multiplication

fortran

Copy code

```
PROGRAM MatrixMultiplication
  IMPLICIT NONE
  INTEGER, PARAMETER :: N = 3
  INTEGER :: i, j, k
  REAL :: A(N, N), B(N, N), C(N, N)

  PRINT *, 'Enter matrix A (', N*N, ' elements):'
  DO i = 1, N
    DO j = 1, N
      READ *, A(i, j)
```

```
        END DO
    END DO

    PRINT *, 'Enter matrix B (', N*N, ' elements):'
    DO i = 1, N
        DO j = 1, N
            READ *, B(i, j)
        END DO
    END DO

    ! Initialize matrix C
    C = 0.0

    ! Perform multiplication
    DO i = 1, N
        DO j = 1, N
            DO k = 1, N
                C(i, j) = C(i, j) + A(i, k) * B(k, j)
            END DO
        END DO
    END DO

    PRINT *, 'Resultant matrix C = A * B:'
    DO i = 1, N
        PRINT *, (C(i, j), j = 1, N)
    END DO
END PROGRAM MatrixMultiplication
```

BASIC: Simple Harmonic Motion Simulation

basic

Copy code

```
10 PRINT "Simple Harmonic Motion Simulator"
```

```
20 PRINT "Enter amplitude (A):"

30 INPUT A

40 PRINT "Enter angular frequency (omega):"

50 INPUT OMEGA

60 PRINT "Enter phase (phi in degrees):"

70 INPUT PHI

80 PRINT "Enter number of time steps (n):"

90 INPUT N

100 PRINT "Time (t) | Position (x)"

110 FOR I = 1 TO N

120     T = I * 0.1

130     X = A * COS(OMEGA * T + PHI * 3.14159 / 180)

140     PRINT T; " | "; X

150 NEXT I

160 END
```

Explanation:

- **Lines 10-70:** Prompt the user for amplitude, angular frequency, phase, and number of time steps.
- **Lines 100-150:** Loop through each time step, calculate the position using the simple harmonic motion formula x(t)=Acos(ωt+ϕ)x(t) = A \cos(\omega t + \phi)x(t)=Acos(ωt+ϕ), and print the results.
- **Line 160:** Ends the program.

---

4. Enhancing Understanding Through Visualization

FORTRAN:

- **Example:** Modify the factorial program to log each multiplication step and plot the growth of the factorial value.
- **Visualization Tool:** Export data to Excel or use plotting libraries to create a graph of factorial growth.

BASIC:

- **Example:** Extend the quadratic solver to plot the discriminant and roots as coefficients vary.
- **Visualization Tool:** Use BASIC's graphics capabilities (if available) or export data for external plotting.

---

5. Historical Reflection

Essay Topic: The Influence of FORTRAN and BASIC on Modern Programming Languages

Guidelines:

- **Introduction:**
  - Brief overview of FORTRAN and BASIC.
  - Their roles in the history of programming.
- **Body:**
  - **FORTRAN's Influence:**
    - Introduction of high-level programming concepts.
    - Impact on scientific computing and numerical methods.
    - Legacy in modern languages (e.g., MATLAB, Julia).
  - **BASIC's Influence:**
    - Democratizing programming and education.
    - Influence on early personal computing and interactive programming environments.
    - Legacy in modern educational programming tools (e.g., Scratch, Python for beginners).
- **Conclusion:**
  - Summary of the enduring legacy of FORTRAN and BASIC.
  - The importance of understanding these languages for appreciating modern programming advancements.

Length: 300-500 words.