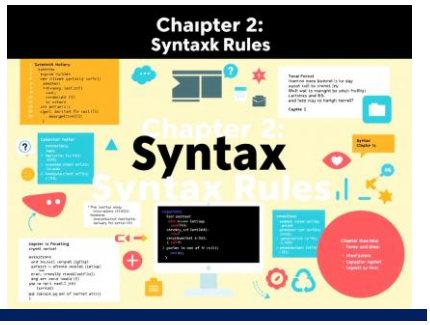
Chapter 2:

Syntax Rules



Chapter 2: Syntax Rules in Fortran

2.1 Basic Syntax Rules

- **Fortran is not case-sensitive**

  PRINT *, 'Hello' and print *, 'Hello' are equivalent.

- **Each instruction must be on a separate line**

  A line usually contains one complete instruction.

- **Statements can be continued on the next line using &**

  PRINT *, 'This is a very long statement & that continues on the next line'

- **Comments begin with !**

  Everything after **!** is ignored by the compiler.

```
! This is a comment
PRINT *, 'Hello, World!'  ! This is also a comment
```

2.2 Variable and Constant Declaration

2.2.1 Declaring Variables

In Fortran, variables must be declared before use.

Table 1: Basic Data Types in Fortran

Type	Description	Example
INTEGER	Whole numbers	INTEGER :: n = 10
REAL	Floating-point numbers (single precision)	REAL :: x = 3.14
DOUBLE PRECISION	Floating-point numbers (double precision)	DOUBLE PRECISION :: y = 3.1415926535
CHARACTER(LEN=n)	Strings of text	CHARACTER(LEN=10) :: name = "Fortran"
LOGICAL	Boolean values (.TRUE. or .FALSE.)	LOGICAL :: flag = .TRUE.

Example of Variable Declaration

```
PROGRAM VariablesExample
   IMPLICIT NONE
   INTEGER :: age
   REAL :: height
   CHARACTER(LEN=20) :: name
   age = 25
   height = 1.75
   name = "John Doe"
   PRINT *, "Name:", name
   PRINT *, "Age:", age
   PRINT *, "Height:", height
END PROGRAM VariablesExample
```

2.2.2 Constants in Fortran

A constant is a variable that does not change during execution. It is defined using the PARAMETER attribute.

Example of Constants

```
PROGRAM ConstantsExample
   IMPLICIT NONE
   REAL, PARAMETER :: PI = 3.14159
   INTEGER, PARAMETER :: MAX_VALUE = 100
   PRINT *, "The value of PI is:", PI
   PRINT *, "The maximum value is:", MAX_VALUE
END PROGRAM ConstantsExample
```

2.3 IMPLICIT NONE and Variable Declaration

2.3.1 Fortran Variable Type Defaults Without IMPLICIT NONE

In **Fortran**, when IMPLICIT NONE is **not** used, the language follows a **default typing rule** based on the **first letter of a variable's name**:

First Letter	Default Data Type
I, J, K, L, M, N	**INTEGER**
A - H, O - Z	**REAL**

This means that:

- Variables **starting with I, J, K, L, M, N** are **automatically considered integers**.
- All other variables (starting with A–H, O–Z) are **automatically considered real numbers**.

Example Without **IMPLICIT NONE**

```
PROGRAM default_types
   INTEGER :: a  ! Explicit declaration
   REAL :: x     ! Explicit declaration
   a = 5
   x = 2.5
   PRINT *, "Value of A (declared INTEGER) = ", a
   PRINT *, "Value of X (declared REAL) = ", x
   ! Implicit types
   i = 10     ! Implicit INTEGER (because it starts with I)
   y = 3.14   ! Implicit REAL (because it starts with Y)
   PRINT *, "Implicit INTEGER I = ", i
   PRINT *, "Implicit REAL Y = ", y
END PROGRAM default_types
```

Output

```
Value of A (declared INTEGER) = 5
Value of X (declared REAL) = 2.5
Implicit INTEGER I = 10
Implicit REAL Y = 3.14
```

- The **explicitly declared** variables (a and x) have known types.
- **Implicitly declared** variables:
  - i is **INTEGER** because it starts with **I**.
  - y is **REAL** because it starts with **Y**.

Example Showing a Logical Error Due to Implicit Typing

```
PROGRAM implicit_error
   r = 5.5   ! Fortran considers R as a REAL (default rule)
   n = 2.5   ! Fortran considers N as an INTEGER (default rule)

   PRINT *, "R = ", r
   PRINT *, "N = ", n
END PROGRAM implicit_error
```

Output

R = 5.500000

N = 2

Issue: N is an **INTEGER**, so 2.5 is **automatically truncated to 2** without warning! This is a classic source of errors in older Fortran codes.

2.3.2 Why Use IMPLICIT NONE?

Using IMPLICIT NONE forces **explicit declarations**, avoiding unintended type assignments.

- **Corrected Example with IMPLICIT NONE**

```
PROGRAM corrected_code
  IMPLICIT NONE
  REAL :: r
  INTEGER :: n
  r = 5.5
  n = 2.5  ! Compiler will give an error if IMPLICIT NONE is used
  PRINT *, "R = ", r
  PRINT *, "N = ", n
END PROGRAM corrected_code
```

- **Without IMPLICIT NONE**, n = 2.5 would silently convert to 2, which could lead to logical errors.
- **With IMPLICIT NONE**, the compiler forces a correction.

3. How to Explicitly Define Precision? (**DOUBLE PRECISION**, **KIND**)

By default, Fortran assigns:

- **INTEGER** → 4 bytes (standard integer)
- **REAL** → 4 bytes (single precision)
- **DOUBLE PRECISION** → 8 bytes

3.1 Using DOUBLE PRECISION

```
PROGRAM double_precision_example
  IMPLICIT NONE
  REAL :: x_single
  DOUBLE PRECISION :: x_double
  x_single = 3.141592653589793  ! Single precision (truncated)
  x_double = 3.141592653589793  ! Double precision (more accurate)
  PRINT *, "Single precision value:", x_single
  PRINT *, "Double precision value:", x_double
END PROGRAM double_precision_example
```

Output

Single precision value: 3.141593

Double precision value: 3.14159265358979

- REAL uses **single precision** (limited decimal places).
- DOUBLE PRECISION provides **higher accuracy**.

3.2 Using KIND for More Control

Instead of DOUBLE PRECISION, you can define precision using KIND:

```
PROGRAM kind_example
  IMPLICIT NONE
  INTEGER, PARAMETER :: sp = SELECTED_REAL_KIND(6)   ! Single precision
  INTEGER, PARAMETER :: dp = SELECTED_REAL_KIND(15)  ! Double precision
  REAL(KIND=sp) :: x
  REAL(KIND=dp) :: y
  x = 3.141592653589793
  y = 3.141592653589793
  PRINT *, "Single precision:", x
  PRINT *, "Double precision:", y
END PROGRAM kind_example
```

- SELECTED_REAL_KIND(6) → Ensures **at least** 6 significant digits.
- SELECTED_REAL_KIND(15) → Ensures **at least** 15 significant digits.

4. Understanding READ, WRITE, and * in Fortran

4.1 **WRITE(*,*)** – Standard Output (Screen)

- The first * → Uses the default output unit (usually screen).
- The second * → Uses the default format.

```
PROGRAM write_example
  IMPLICIT NONE
  INTEGER :: a
  REAL :: b
  a = 10
  b = 3.14
  WRITE(*,*) "Integer A:", a
  WRITE(*,*) "Real B:", b
END PROGRAM write_example
```

```
Output
Integer A: 10
 Real B: 3.14
```

4.2 **READ(*,*)** – Standard Input (Keyboard)

```
PROGRAM read_example
   IMPLICIT NONE
   INTEGER :: age
   REAL :: height
   PRINT *, "Enter your age:"
   READ(*,*) age
   PRINT *, "Enter your height (m):"
   READ(*,*) height
   PRINT *, "You are", age, "years old and", height, "meters tall."
END PROGRAM read_example
```

Example User Input

```
Enter your age:
25
Enter your height (m):
1.75
You are 25 years old and 1.75 meters tall.
```

- READ(*,*) **waits for user input**.
- The entered values are stored in variables.

5. Summary

Concept	Explanation
Default Typing	Variables starting with **I, J, K, L, M, N** are **INTEGER**, all others are **REAL**.
Without **IMPLICIT NONE**	Variables get **implicitly** typed based on first letter.
With **IMPLICIT NONE**	Forces explicit declaration, avoiding errors.
Precision (**DOUBLE PRECISION**, **KIND**)	Controls floating-point accuracy.
WRITE(*,*)	Displays output on screen.
READ(*,*)	Reads input from keyboard.

7 Increasing Precision: KIND and DOUBLE PRECISION

Fortran allows defining variable precision using KIND.

7.1 Defining Precision using KIND

```
INTEGER, PARAMETER :: dp = KIND(1.0D0)  ! Double precision
REAL(KIND=dp) :: x
x = 3.14159265358979_dp
```

1.0D0 ensures that the number is stored in double precision.

7.2 Input and Output: READ and WRITE Statements

7.2.1 The PRINT and WRITE Statements

- PRINT *, displays output on the screen.
- WRITE (*,*) is another way to print output.
- The * symbol indicates **default format**.

Example of Output

```
PRINT *, "Hello, World!"
WRITE (*,*) "This is Fortran programming."
```

7.2.2 The READ Statement (User Input)

- READ (*,*) reads input from the keyboard.
- The * indicates default input format.

Example of Input and Output

```
PROGRAM ReadExample
   IMPLICIT NONE
   CHARACTER(LEN=20) :: name
   INTEGER :: age
   PRINT *, "Enter your name:"
   READ (*,*) name
   PRINT *, "Enter your age:"
   READ (*,*) age
   PRINT *, "Hello", name, "you are", age, "years old."
END PROGRAM ReadExample
```