Functions and Subroutines

- The objective herein is to use a top-down approach to solve complex problems by dividing them into a structured modular computational form.

- The following topics are covered:
  1. standard functions,
  2. statement functions,
  3. function subprograms, and
  4. subroutine subprograms
Functions and Subroutines

- **Standard (Library) Functions**
  - SIN(\(x\)) Sine of \(x\) (in radian)
  - COS(\(x\)) Cosine of \(x\) (in radian)
  - TAN(\(x\)) Tangent of \(x\) (in radian)
  - LOG(\(x\)) Natural logarithm of \(x\)
  - LOG10(\(x\)) Common (base 10) logarithm

Refer to attached Table for more functions.

- **Statement Functions**
  - The statement should be used in the same unit of program where function is used and immediately after the specification statements (before any executable statements).

  
  \[
  \text{name (argument-list) = expression}
  \]


Functions and Subroutines

Statement Functions (cont’d)

– The function name follows the same rules of variables for type (real or integer). The argument list can be empty

– Example:

```fortran
REAL A, B, Z, T
Z(A,B) = A+B
: 
X = 3 
Y = 4 
T = Z(X,Y) + 2 
PRINT *, T
```

The result is $T = 9$

Function Subprograms

– The structure for a function subprogram is the same as a FORTRAN programs as follows:

```fortran
FUNCTION name(argument-list)
Declaration part
Subprogram statements
RETURN
END
```
Functions and Subroutines

- The statement return is not needed all the time
- It is used to return to the main program
- The computed value goes to the location were the function was called
- The result from the function is returned by the function name

Example:

```fortran
FUNCTION F(X,Y,N)
   REAL X, Y
   INTEGER N
   F = X**N + Y**N
   RETURN
END
```
The above function of the example can be called in the main program as follows:

\[ W = F(A,B+3.0,2) \]

or

\[ Z = F(R(I),\sin(A),K) \]
Subroutines

- Subroutine Subprograms
  - Similar to Function subprograms with the following differences:
    1. Functions return one value; whereas Subroutines return no value, one or more values.
    2. Returned values by Functions are their names; whereas Subroutine names do not return values, Subroutine output is part of the arguments.
    3. A Function is referenced using its name in an expression; whereas subroutines are referenced by a CALL statement.

Structure of Subroutines

```
SUBROUTINE name(argument-list)
  Declaration Part
  Subprogram Statements
  RETURN
END
```
Subroutines

- The input and output list is the argument list which can be empty
- The arrays in a subroutine can be of variable sizes that are declared using the DIMENSION or REAL statement (among others) with a size that is specified by a variable in the argument list
- In the main program, use CALL subroutine-name (argument-list) to execute the subroutine

Example:

```fortran
SUBROUTINE MEAN(X,N,Z)
    SUM = 0
    INTEGER N
    REAL X(N), Z, SUM
    DO 10 I = 1, N
        SUM = SUM + X(I)
    10 CONTINUE
    Z = SUM/N
    RETURN
END
```
**Subroutines**

- In the main program, the following is permissible:

  ```fortran
  REAL Y(100) , AVERAG
  M = 10
  CALL MEAN(Y,M,AVERAG)
  ```

**Subroutines**

- COMMON Statement
  - This statement is used to share information between main Program and Subprograms and among subprograms. It takes the following form:

    ```fortran
    COMMON list
    ```

    - The list is the list of variables or arrays separated by commas
Subroutines

Example
- In the main program, use
  COMMON A, B, C
- In the subprograms, use
  COMMON X, Y, Z
- The results of these two statements is that
  A takes on the same value as X, B takes
  on same values as Y, and C takes on the
  same value as Z.

In general, the statement can be as follows:
COMMON /name1/list1/ name2/list2 ...

Example:
- In the main programs, use
  COMMON/N1/ X, Y, Z/N2/A, B, C/N3/D, E, F
- In subprogram 1, use
  COMMON/N1/ XX, YY, ZZ/N3/DD, EE, FF
- In subprogram 2, use
  COMMON/N3/DD, EE, FF
**Double Precision**

- Real data are commonly treated using single precision, i.e., single memory location for each number which corresponds to 32-bit word. The result is an accuracy of about 7 significant digits. Double precision allows for doubling the above quantities.

---

**Double Precision**

- Use a type statement to specify double precision as follows:
  
  ```fortran
  DOUBLE PRECISION Z, X(2, 3)
  FUNCTION F(X, Y)
  DOUBLE PRECISION F, X, Y
  END
  ```
  
- In writing or printing numbers 1.E-3 becomes 1.D-3
**Double Precision**

In general, the read and write control is

\[ \text{rDw.d} \]

where
- \( r \) = repetitions,
- \( w \) = width, and
- \( d \) = number of digits

---

**Fortran Application Examples**

- **Newton’s Method (Numerical Analysis)**
  - This program illustrates the Newton’s method on the function:
    \[ f(x) = x^3 - 3x^2 - x + 9 = 0 \]
  - Calls: NEWT
  - Output (form Function F)
    - X=Value of X at current iteration
    - F=Functional value at x
Approximation Given by a Newton Step

\[ p(x) = f(x_0) + f'(x_0)(x - x_0) \]

\[ x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \]

**Fortran Application Examples**

```fortran
PROGRAM NEWTON
X0=-2.0
EPS=0.00001

C
C SUBROUTINE NEWT IS USED TO FIND A
C ROOT STARTING WITH THE INITIAL GUESS
C X0

CALL NEWT(X0,X,EPS)
STOP
END
```
SUBROUTINE NEWT(X0,X,EPS)

C ***************************************************************
C * FUNCTION: THE SUBROUTINE APPROXIMATES THE ROOT
C * OF F(X)=0 GIVEN THE INITIAL POINT X0 AND
C * THE DERIVATIVE FUNCTION DF(X) USING THE
C * NEWTON METHOD
C * USAGE:
C * CALL SEQUENC: CALL NEWT(X0,X,EPS)
C* EXTERNAL FUNCTIONS/SUROUTINES:
C* FUNCTION F(X)
C* FUNCTION DF(X)

C **** INITIALIZATION ****
X=X0-(F(X0)/DF(X0))
C *** COMPUTE APPROXIMATE ROOT ITERATIVELY ***
**Fortran Application Examples**

```fortran
DO WHILE(ABS(X-X0) .GT. EPS)
   X0=X
   X=X0-(F(X0)/DF(X0))
END DO
RETURN
END
```

**Fortran Application Examples**

```fortran
C FUNCTION F IS CALLED BY NEWT TO CALCULATE
C FUNCTIONAL VALUES FOR THE PASSED POINT X
FUNCTION F(X)
   F=X**3-3.0*X**2-X+9.0
   WRITE(6,10)X,F
10 FORMAT(22X,F16.7,4X,F16.7)
RETURN
END
```
Fortran Application Examples

Function DF(X) is called by NEWT to calculate the derivative at X.

```fortran
FUNCTION DF(X)
    DF = 3.0 * X**2 - 6.0 * X - 1.0
    RETURN
END
```

Area of a Circle

-- Demonstrating the use of Input and Output files

```fortran
PROGRAM AREA_CIRCLE
    DIMENSION DIA(10), AREA(10)
    OPEN(5, FILE='TEST.DAT', STATUS='UNKNOWN')
    OPEN(6, FILE='TEST.OUT', STATUS='UNKNOWN')
    PI = 22. / 7
    DO I = 1, 10
        READ(5, *) DIA(I)
        AREA(I) = PI * DIA(I)**2 / 4
    END DO
    WRITE(6, *) DIA(I), AREA(I)
END
```
Fortran Application Examples

• A. J. Clark School of Engineering • Department of Civil and Environmental Engineering

**INPUT File (area.dat)**

1
2
3
6
8
9
5
7
9
11

**OUTPUT File (area.out)**

1.00000 0.785714
2.00000 3.14286
3.00000 7.07143
6.00000 28.2857
8.00000 50.2857
9.00000 63.6429
5.00000 19.6429
7.00000 38.5000
9.00000 63.6429
11.0000 95.0714
Numerical Analysis
– The Trapezoidal Rule of Integration

\[
\int_{x_1}^{x_n} f(x) \, dx = h \left[ \frac{1}{2} f(x_0) + f(x_1) + \ldots + f(x_{n-1}) + \frac{1}{2} f(x_n) \right]
\]

\[h = x_{i+1} - x_i\]
Numerical Analysis

- Need for Trapezoidal Formula
- Complex Integral that cannot be evaluated analytically

\[ \int e^x \sin(x^2) \, dx \]

- Set of discrete values obtained from experiments

Fortran Application Examples

**Numerical Analysis**

- The Trapezoidal Rule of Integration

```fortran
PROGRAM TRAPEZOID
A=0.0
B=10.0
N=10

CALL TRAP(A,B,N,EST)
WRITE(5,*),EST
END
```
**Fortran Application Examples**

- **The Trapezoidal Rule of Integration (cont’d)**

  ```fortran
  SUBROUTINE TRAP(A,B,N,E)
  H = (B-A)/N
  E = (F(A)+F(B))/2.0
  IF (N.GT.1) THEN
    X = A
    DO 1 I=1, N-1
      X = X + H
      E = E + F(X)
  1 CONTINUE
  END IF
  E = E * H
  RETURN
  END
  ```

- **The Trapezoidal Rule of Integration (cont’d)**

  ```fortran
  FUNCTION F(X)
  F = 1 + (X**2*exp(x))/(1+x)
  RETURN
  END
  ```