McGill

COMP 208 – Computers in Engineering

COMP 208
Computers in Engineering

Lecture 12

Jun Wang
School of Computer Science
McGill University
Fall 2007

Review

- 2-dimensional arrays

```
INTEGER :: i, j, a(2,3)

DO i=1,2
  DO j=1,3
    a(i,j) = i*3 + j
  END DO
END DO
```

```
4  5  6
7  8  9
```

```
DO i=1,2
  WRITE (*,*) (a(i,j), j=1,3)
END DO
```

```
4  5  6
7  8  9
```

```
WRITE (*,*) ((a(i,j), j=1,3), i=1,2)
```

```
4  5  6  7  8  9
```

```
WRITE (*,*) a
```

```
4  7  5  8  6  9
```

Review

- Functions
  - One type of sub-program (the other type is subroutines)
  - Function may take parameters
  - Function returns a value by assigning the value to the function name
  - Function can define local variables
  - A function invocation can appear anywhere an expression of the same type can appear
  - Caller of a function must provide exact number of parameters with matching types as defined by the function

```
type FUNCTION function-name(arg1, arg2, …, argn)
   IMPLICIT NONE
   [declarations]
   [statements]
END FUNCTION function-name
```

Function example

```
! computes sqrt(a) + sqrt(b) using Newton's method
PROGRAM fun
  REAL :: a, b, result
  REAL :: nsqrt

  WRITE (*,*) "Enter a, b: "
  READ (*,*) a, b  ! assuming a, b are positive

  result = nsqrt(a) + nsqrt(b)
  WRITE (*,*) "result= ", result
END PROGRAM fun

REAL FUNCTION nsqrt(A)
  IMPLICIT NONE
  REAL :: A, R, Tolerance = 0.001

  R = A   ! Initial approximation
  DO
    IF (ABS(R*R - A) < Tolerance) EXIT  !If close enough, exit
    R = 0.5*(R + A/R)     ! compute a new approximation
  END DO

  nsqrt = R
END FUNCTION nsqrt
```

Without Function

```
! computes sqrt(a) + sqrt(b) using Newton's method
PROGRAM fun
  REAL :: a, b, sqa, sqb, R, result, Tolerance=0.001

  WRITE (*,*) "Enter a, b: "
  READ (*,*) a, b

  R = A
  DO
    IF (ABS(R*R - A) < Tolerance) EXIT
    R = 0.5*(R + A/R)
  END DO

  sqa = R

  R = B
  DO
    IF (ABS(R*R - A) < Tolerance) EXIT
    R = 0.5*(R + A/R)
  END DO

  sqb = R
  result = sqa + sqb
  WRITE (*,*) "result= ", result
END PROGRAM fun
```

code repeated

Program is flat, structureless

Function invocation example

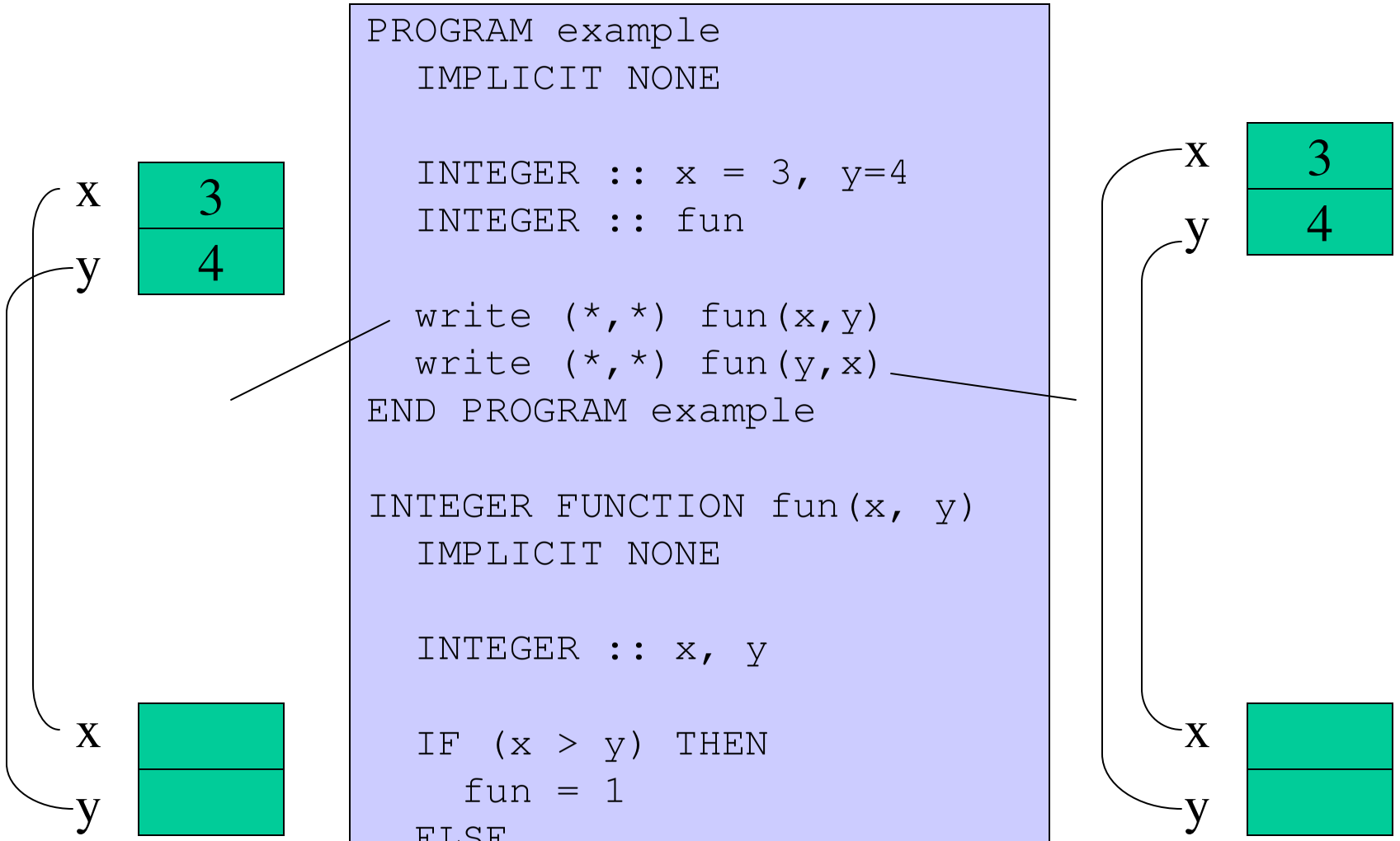
```
PROGRAM fun
  INTEGER :: i, j, k
  INTEGER :: square
  ...
  IF (square(i) > 100) WRITE (*,*) "blah"

  WRITE (*,*) square(i+j)

  k = i + square(j)

END PROGRAM fun

INTEGER FUNCTION square(x)
  IMPLICIT NONE
  INTEGER :: x

  square = x * x
END FUNCTION square
```

in a relational expression

in a WRITE statement

in an arithmetic expression

Actual argument /formal argument association



Argument-Parameter Association

- When a function is used, the argument values are used to initialize the parameters
- The way this is done is not as simple as it might seem
- It varies from language to language and is very different in Fortran than in C
- We present the three formal rules that Fortran uses

Warning!

- The next few slides are very technical in nature
- The material might be a bit tedious but it is important to understand
- It will become even more important later in the course when we study C

Argument-Parameter Association

- The first rule tells us what happens when a constant or expression is used as an argument
  - This is the most intuitive case
- The second rule deals with arguments that are variables
  - The association is a bit more complex here

Example – Function Definition

```
! returns minimum of 3 numbers
INTEGER FUNCTION Minimum (x, y, z)
  IMPLICIT NONE
  INTEGER :: x, y, z

  IF (x <= y .AND. x <= z) THEN
    Minimum = x
  ELSE IF (y <= x .AND. y <= z) THEN
    Minimum = y
  ELSE
    Minimum = z
  END IF

END FUNCTION Mimimum
```

Rule 1 -- Expressions

If an actual argument is an expression or a constant,

it is evaluated and the result is saved into a temporary location. Then, the parameter becomes a reference to this temporary cell

```
INTEGER :: a = 10, b = 3, c = 37
WRITE(*,*) Minimum(18,c-a,a+b)
```

When the function is invoked, new temporary variables named x, y and z are created. The value of x is initialized to 18, y to 27 and z to 13.

The function returns 13.

Pass by value

Rule 2 -- Variables

If an actual argument is a variable, the corresponding formal argument is made to refer to the same memory cell.

```
INTEGER :: a = 10, b = 3, c = 37
WRITE(*,*) Minimum(a,b,c)
```

When the function is invoked, there are no new variables created. The parameter x refers to a, y to b and z to c. We say x is an **alias** for a. There are two names for the same memory cell.

The function returns 3.

Pass by reference

What Does This Function Do?

```
REAL FUNCTION DoSomething (a, b)
  IMPLICIT NONE
  INTEGER :: a, b
  a = a - b
  b = a + b  ! b equals original a
  a = b - a  ! a equals original b
  DoSomething = a
END FUNCTION DoSomething
```

NOTE here the formal arguments are assigned to.

What Happens?

```
INTEGER :: x = 12, y = 5
WRITE (*,*) x, y
WRITE (*,*) DoSomething (x+y, y)
WRITE (*,*) x, y
```

```
REAL FUNCTION DoSomething (a, b)
  IMPLICIT NONE
  INTEGER :: a, b
  a = a - b
  b = a + b
  a = b - a
  DoSomething = a
END FUNCTION DoSomething
```

Output:

```
12  5
5
12  17
```

a is given the value of x+y
b becomes an alias of y

What Happens?

```
INTEGER :: x = 12, y = 5
WRITE (*,*) x, y
WRITE (*,*) DoSomething (12, 5)
WRITE (*,*) x, y
```

```
REAL FUNCTION DoSomething (a, b)
  IMPLICIT NONE
  INTEGER :: a, b
  a = a - b
  b = a + b
  a = b - a
  DoSomething = a
END FUNCTION DoSomething
```

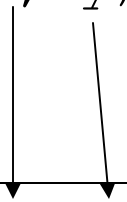
Output:

```
12  5
5
12  5
```

a is given the value of 12
b is given the value of 5

What Happens?

```
INTEGER :: x = 12, y = 5
WRITE (*,*) x, y
WRITE (*,*) DoSomething (x, y)
WRITE (*,*) x, y
```



```
REAL FUNCTION DoSomething (a, b)
  IMPLICIT NONE
  INTEGER :: a, b
  a = a - b
  b = a + b
  a = b - a
  DoSomething = a
END FUNCTION DoSomething
```

Output:

```
12  5
5
5  12
```

a becomes an alias of x
b becomes an alias of y

Parameter passing summary

- When a constant or an expression is passed to a formal parameter, changing the formal parameter (through assignment) does NOT affect program outside the function.
- (with some compiler, a run-time error occurs if a constant is passed to a formal parameter which is changed inside the function)
- When a simple variable is passed to a formal parameter, the formal parameter becomes an alias of the variable, therefore, changing the formal parameter will also change the actual parameter

Example

```
PROGRAM example
  IMPLICIT NONE
  INTEGER :: fun
  INTEGER :: x=1

  WRITE (*,*) x
  WRITE (*,*) fun(x)
  WRITE (*,*) x
  WRITE (*,*) fun(x+1)
  WRITE (*,*) x
END PROGRAM example

INTEGER FUNCTION fun (a)
  IMPLICIT NONE
  INTEGER :: a

  a = 2*a
  fun = a
END FUNCTION fun
```

1

2

2

6

2

Function Execution (summary)

When a function is invoked

1. arguments are evaluated
2. parameter-argument associations are made
3. function body executes until it reaches the END FUNCTION statement
4. value assigned to the function name is returned
5. all temporary storage is released
6. this value is used where the function was invoked

Where Do Function Definitions Go?

The structure of a program is:

```
PROGRAM  program-name
IMPLICIT  NONE
[declarations]
[statements]
END PROGRAM  program-name
[function definitions]
```

Function definitions are placed in the same file as the program

They can be anywhere in the file but usually follow the program

Average of Three Numbers

```
PROGRAM  Avg
  IMPLICIT  NONE
  REAL :: a, b, c, Mean, Average
  READ(*,*)  a, b, c
  Mean = Average (a, b, c)
  WRITE(*,*) a, b, c, Mean
END PROGRAM  Avg

REAL FUNCTION  Average(a, b, c)
  IMPLICIT  NONE
  REAL :: a, b, c
  Average = (a + b + c) / 3.0
END FUNCTION  Average
```

Two Functions (part 1)

```
PROGRAM  TwoFunctions
   IMPLICIT  NONE
   INTEGER :: a, b, BiggerOne
   REAL    :: GeometricMean

   READ(*,*)  a, b
   BiggerOne = Maximum(a,b)
   GeometricMean = GeoMean(a,b)
   WRITE(*,*)  "Input = ", a, b
   WRITE(*,*)  "Larger one = ", BiggerOne
   WRITE(*,*)  "Geometric Mean = ", GeometricMean
END PROGRAM  TwoFunctions
```

Two Functions (part 2)

```
INTEGER FUNCTION  Maximum(a, b)
   IMPLICIT  NONE
   INTEGER :: a, b
   IF (a >= b) THEN
      Maximum = a
   ELSE
      Maximum = b
   END IF
END FUNCTION  Maximum

REAL FUNCTION  GeoMean(a, b)
   IMPLICIT  NONE
   INTEGER :: a, b
   GeoMean = SQRT(REAL(a*b))
END FUNCTION  GeoMean
```

McGill

COMP 208 – Computers in Engineering

Delegating

Subroutines

Nathan Friedman

Fall, 2007

Subprograms

- Functions are one type of subprogram in FORTRAN
- Another type of subprogram FORTRAN allows is called a **subroutine**
- There are many similarities between them and we must be careful not to confuse the two types of subprograms

Subroutines

- Subroutines are used to define new actions
- Unlike functions, **they do not return values**
- They can modify the values of arguments or return values indirectly through the arguments
- For example a Sort subroutine may take an array as an argument and return the array with the values in sorted order

Subroutines vs. functions

- Subroutines don't return values
  - Therefore, subroutines usually modify the parameters
- Subroutines are invoked with the **`CALL`** statement

Computing Statistics

```
! ------------------------------------------------------
! Read an indeterminate number of real values and compute
! their mean, variance and standard deviation.
! ------------------------------------------------------
PROGRAM  MeanVariance
   IMPLICIT   NONE
   INTEGER :: Number, IOstatus
   REAL    :: Data, Sum, Sum2
   REAL    :: Mean, Var, Std
   Number = 0
   Sum    = 0.0
   Sum2   = 0.0
   DO
      READ(*,*,IOSTAT=IOstatus)  Data
      IF (IOstatus < 0)  EXIT
      Number = Number + 1
      CALL  Sums(Data, Sum, Sum2)
   END DO
   CALL  Results(Sum, Sum2, Number, Mean, Var, Std)
   CALL  PrintResult(Number, Mean, Var, Std)
END PROGRAM MeanVariance
```

IOSTAT

We can write:

```
INTEGER :: IOstatus
READ(*,*,IOSTAT=IOstatus) v1, v2, . . ., vn
```

The variable following " IOSTAT = " can be any variable of type INTEGER

This variable is assigned a value after the READ is executed

1. If the value is zero, the read was successful
2. If the value is negative, the end of file was reached
3. If the value is positive, there was an error in the input

An end of file signal is sent from the keyboard when you press Ctrl-Break

Compute Sum and Sum of Squares

```
! ------------------------------------------------------
!    This subroutine takes three REAL values:
!    (1)  x      - the input value
!    (2)  Sum    - x will be added to this sum-of-input
!    (3)  SumSQR - x*x is added to this sum-of-squares
! ------------------------------------------------------
SUBROUTINE  Sums(x, Sum, SumSQR)
   IMPLICIT  NONE
   REAL :: x                    ! Input Parameter
   REAL :: Sum, SumSQR          ! Input and Output Parameters
   Sum    = Sum + x
   SumSQR = SumSQR + x*x
END SUBROUTINE  Sums
```

Computing the Statistics

```
! ------------------------------------------------------------
! Compute the mean, variance and standard deviation
!
!    (1) Sum        - sum of input values
!    (2) SumSQR     - sun-of-squares
!    (3) n          - number of input data items
!    (4) Mean       - computed mean value
!    (5) Variance   - computed variance
!    (6) StdDev     - computed standard deviation
! ------------------------------------------------------------
   SUBROUTINE  Results(Sum, SumSQR, n, Mean, Variance, StdDev)
      IMPLICIT  NONE
      INTEGER :: n
      REAL :: Sum, SumSQR                ! Input Parameters
      REAL :: Mean, Variance, StdDev     ! Output Parameters
      Mean = Sum / n
      Variance = (SumSQR - Sum*Sum/n)/(n-1)
      StdDev   = SQRT(Variance)
   END SUBROUTINE
```

Output the Results

```
! ------------------------------------------------------
! Display the computed results.
! ------------------------------------------------------
   SUBROUTINE  PrintResult(n, Mean, Variance, StdDev)
      IMPLICIT  NONE
      INTEGER :: n
      REAL :: Mean, Variance, StdDev
      WRITE(*,*)
      WRITE(*,*) "No. of data items  = ", n
      WRITE(*,*) "Mean               = ", Mean
      WRITE(*,*) "Variance           = ", Variance
      WRITE(*,*) "Standard Deviation = ", StdDev
   END SUBROUTINE  PrintResult
```

Subroutine Definitions

Syntax

```
SUBROUTINE  subroutine-name (arg1,arg2,...,argn)
   IMPLICIT  NONE
   [declarations]
   [code]
END SUBROUTINE  subroutine-name
```

The definition starts with **SUBROUTINE**, followed by the subroutine's name.

Following the name, the list of parameters is specified

A Factorial Subroutine

- Previously we defined a function to compute factorials
- It acted as a new operator (like `sqrt`) to return a value directly
- We could also define a subroutine to compute factorials
- The result must be returned using an extra parameter

A Factorial Subroutine

```
SUBROUTINE Factorial(n,Fact)
  IMPLICIT NONE
  INTEGER :: n, Fact
  INTEGER :: i

  Fact = 1
  DO i = 1, n
    Fact = Fact * i
  END DO
END SUBROUTINE Factorial
```

- The subroutine does not return a value directly
- The parameter Fact is used to hold the value that is being computed

Function or Subroutine

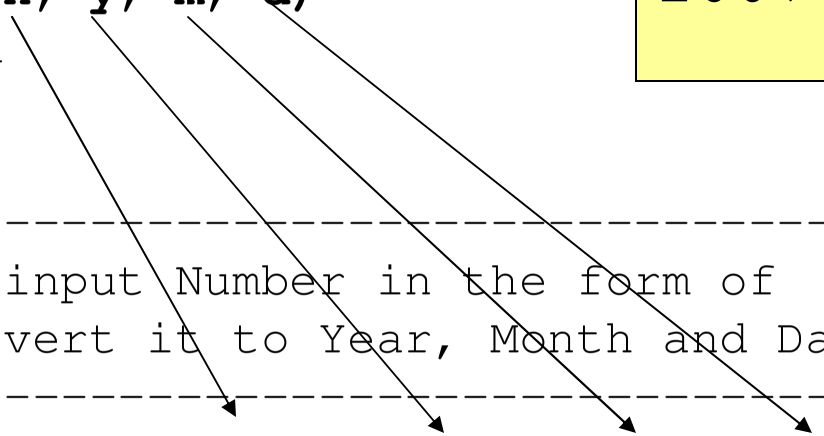
- Factorial takes a single argument and returns a single value
- Defining it as a function seems more natural
- Defining it as a subroutine is more forced
- Sometimes subroutine is more natural

Multiple Results

- Sometimes we want operators that return multiple results
- Since FORTRAN functions can only return one value, it's more natural to use subroutines

Date Conversion

```
PROGRAM prog
  IMPLICIT NONE
  INTEGER :: n=20071016, y, m, d

  CALL Conversion(n, y, m, d)
  WRITE(*,*) y,m,d
END PROGRAM prog

! ------------------------------------------
! Take an integer input Number in the form of
! YYYYMMDD and convert it to Year, Month and Day
! ------------------------------------------
SUBROUTINE  Conversion (Number, Year, Month, Day)
  IMPLICIT  NONE
  INTEGER :: Number
  INTEGER :: Year, Month, Day
  Year  = Number / 10000
  Month = MOD(Number, 10000) / 100
  Day   = MOD(Number, 100)
END SUBROUTINE  Conversion
```



OUTPUT:

```
2007 10 16
```

Factorial
Function vs. Subroutine

```
INTEGER FUNCTION Factorial(n)
  IMPLICIT NONE
  INTEGER :: n
  INTEGER :: i, Fact
  Fact = 1
  DO i = 1, n
    Fact = Fact * i
  END DO
  Factorial = Fact
END FUNCTION Factorial
```

```
SUBROUTINE Factorial(n, Fact)
  IMPLICIT NONE
  INTEGER :: n, Fact
  INTEGER :: i
  Fact = 1
  DO i = 1, n
    Fact = Fact * i
  END DO
END SUBROUTINE Factorial
```

Factorial Function vs. Subroutine

```
INTEGER FUNCTION Factorial(n)
  IMPLICIT NONE
  INTEGER :: n
  INTEGER :: i, Fact
  Fact = 1
  DO i = 1, n
    Fact = Fact * i
  END DO
  Factorial = Fact
END FUNCTION Factorial
```

```
SUBROUTINE Factorial(n, Fact)
  IMPLICIT NONE
  INTEGER :: n, Fact
  INTEGER :: i
  Fact = 1
  DO i = 1, n
    Fact = Fact * i
  END DO
END SUBROUTINE Factorial
```

Functions, Subroutines What's the Difference?

- A function defines a new operation
- It takes some "input" values (arguments) and returns a result
  - For example, sqrt, mod, factorial
- A subroutine defines a new action analogous to a statement.
- It might modify its arguments but doesn't return a result directly
  - For example, sort

Functions, Subroutines What's the Difference?

- A function must assign a value to the dummy variable which is the name of the function
- The name of the subroutine is not a dummy variable and is not assigned a value

Functions, Subroutines What's the Difference?

- A function is invoked implicitly by using it in an expression.
- After executing, it returns a value to be used in evaluating the expression

- A subroutine is called explicitly. It appears in the program where a statement can appear
- After executing it just returns
- The argument values may have changed

Definition vs. Usage

- We have discussed how to define subprograms and the difference between function and subroutine definition
- Definitions are a one time thing
- Once defined, the subprograms can be used throughout the program
- The way functions and subroutines are used differs

Using Subroutines

Subroutines are invoked with a CALL statement.

Syntax of CALL

```
CALL subroutine-name (e1, e2, ..., en)
CALL subroutine-name ()
CALL subroutine-name
```

In the first form, the subroutine has n parameters.

If a subroutine does not have any arguments, it can be called with or without parentheses

Example

To use the Factorial **subroutine,** use the statement

```
CALL Factorial(7,result)
WRITE (*,*) result+9
```

To use the Factorial **function**, reference it directly in an expression

```
WRITE (*,*) Factorial(7)+9
```

The Semantics of Call

1. When a CALL statement is executed,
   - The values of the arguments are passed to the parameters
   - The number of actual arguments in the CALL statement must match the number of formal parameters
   - The type of each argument must match the type of the corresponding formal parameter
2. The body of the called subroutine is executed.

The Semantics of Call

3. When END SUBROUTINE is reached,
   - Execution of the subprogram ends
   - The next statement following the CALL statement is executed.
4. If a variable was passed as an argument, any changes made to it remain

Minimum Function

```
INTEGER FUNCTION MinimumF (x, y, z)
  IMPLICIT NONE
  INTEGER :: x, y, z
  IF (x <= y .AND. x <= z) THEN
    MinimumF = x
  ELSE IF (y <= x .AND. y <= z) THEN
    MinimumF = y
  ELSE
    MinimumF = z
  END IF
END FUNCTION MimimumF
```

Minimum Subroutine

```
SUBROUTINE MinimumS (x, y, z, m)
  IMPLICIT NONE
  INTEGER :: x, y, z, m
  IF (x <= y .AND. x <= z) THEN
    m = x
  ELSE IF (y <= x .AND. y <= z) THEN
    m = y
  ELSE
    m = z
  END IF
END SUBROUTINE MimimumS
```

Examples of Use

```
INTEGER :: a, b, c, result
READ (*,*) a, b, c

CALL MinimumS(a,b,c,result)
WRITE (*,*) "The minimum of ", a, b, c, " is: ",&
            result

WRITE (*,*) "The minimum of ", a, b, c, " is: ",&
            MinimumF(a,b,c)
```

Rules for Argument Association

The rules for associating arguments with formal parameters are the same as the rules we described for functions

"Means" Example

Problem:

Compute the arithmetic, geometric and harmonic means of three real values

Question:

Do we use a function or subroutine?

A function only returns a single result.

To compute three different values, we have to use a subroutine

"Means" Example

```
! ---------------------------------------------
! Subroutine to take three REAL values and compute
! their arithmetic, geometric, and harmonic means.
! ---------------------------------------------
SUBROUTINE  Means(a, b, c, Am, Gm, Hm)
  IMPLICIT  NONE
  REAL :: a, b, c
  REAL :: Am, Gm, Hm
  Am = (a + b + c)/3.0
  Gm = (a * b * c)**(1.0/3.0)
  Hm = 3.0/(1.0/a + 1.0/b + 1.0/c)
END SUBROUTINE  Means
```

Swap

Problem:

Interchange the values stored in two integer variables

Question:

Do we use a function or subroutine?

A function returns a result.

To perform an action that modifies variables, we have to use a subroutine

Swap

```
SUBROUTINE  Swap(a, b)
  IMPLICIT  NONE
  INTEGER :: a, b
  INTEGER :: temp
  temp = a
  a = b
  b = temp
END SUBROUTINE  Swap
```

Example:

```
i = 4
j = 9
CALL Swap(i, j)
WRITE (*,*) i, j
```

Array Parameters

- When we declare an array in a program, we must specify its size
- The compiler allocates storage for the specified number of memory cells
- When we write a subprogram definition to process an array, we want it to be generic
- That is, we want to be able to use it with different arrays, possibly of different sizes

Minimum Value in an Array

```
REAL FUNCTION Min (A, n)
  IMPLICIT NONE
  INTEGER :: n
  INTEGER :: I
  REAL :: A(n)
  Min = A(1)
  DO I = 2, n
    IF (A(I) < Min) Min = A(I)
  END DO
  RETURN Min
END FUNCTION Min
```

The size of the array is not a constant

Array Parameters

- Why can we declare an array when we don't know its size?
- The compiler does not allocate storage when we define a subprogram
- When we invoke a subprogram the compiler just makes the name an alias for an existing block of storage

Sorting Values in an Array

```
SUBROUTINE Sort (A, n)
  IMPLICIT NONE
  INTEGER :: n
  REAL :: A(n)
  INTEGER :: j, k, minptr
  DO j = 1,n-1
    minptr = j
    DO k = j+1,n
      IF (A(k)<A(minptr)) minptr = k
    END DO
    IF ( J /= minptr) THEN
      CALL SwapReals (A(j), A(minptr))
    END IF
  END DO
END SUBROUTINE Sort
```

1	2	3	4	5
9	6	8	3	7

Iteration 1. J = 1, K=2, 5

minptr = 4

1	2	3	4	5
3	6	8	9	7

Iteration 2. J = 2, K=3, 5

minptr = 2

Iteration 3. J = 3, K=4, 5

minptr = 5

1	2	3	4	5
3	6	7	9	8

Iteration 4. J = 4, K=5, 5

minptr = 5

1	2	3	4	5
3	6	7	8	9