SCSJ 2733
Fortran - Subprograms

Mohsin Mohd Sies

Faculty of Chemical and Energy Engineering

Outline

- Motivation
- Function
- Subroutine
- COMMON Statement

Motivation

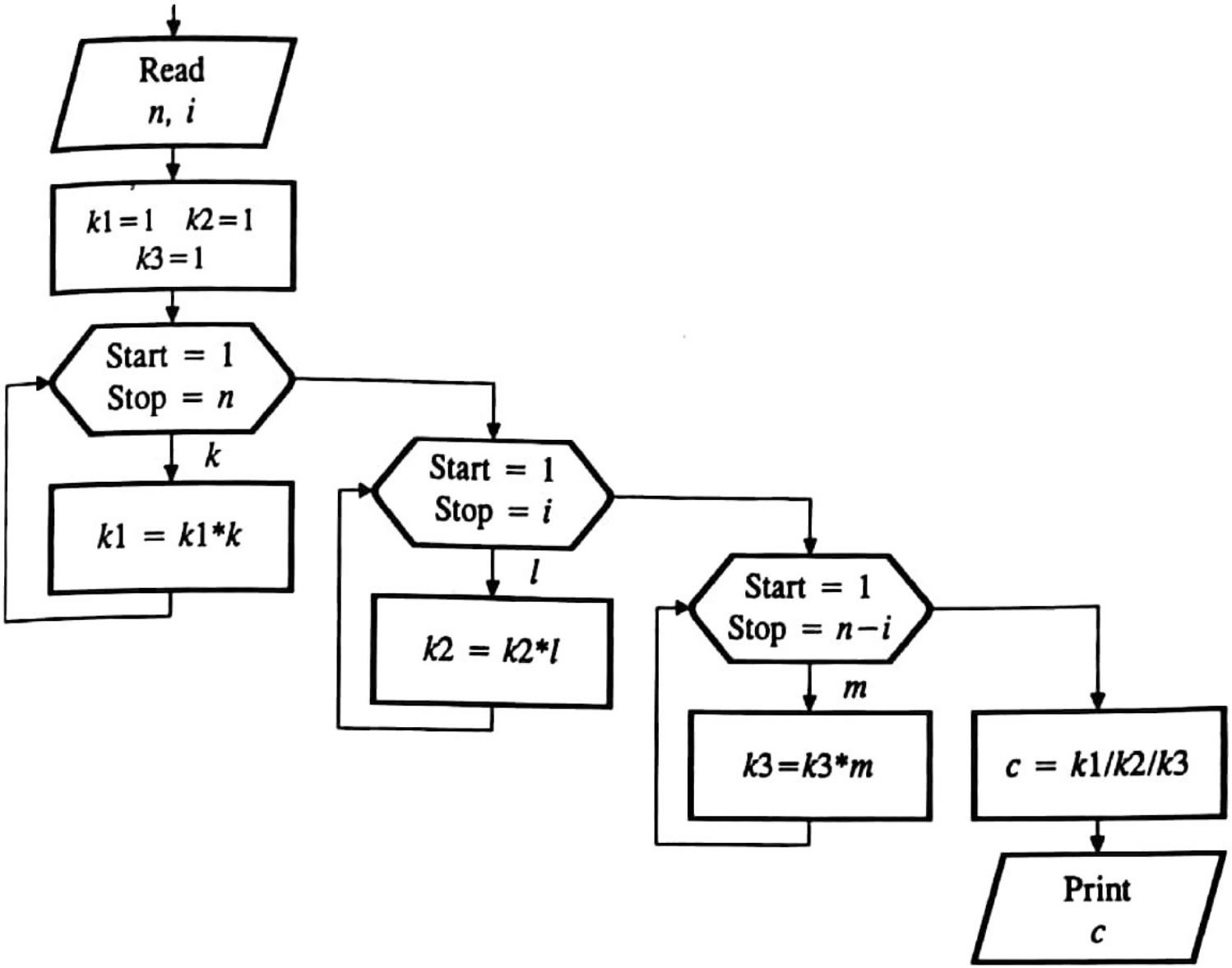
- *Modularity* – Key to programming success
- Break complex problems into simpler *subtasks*
- Recognize that the same subtasks occurs many times in the same complex problem
- The same subtask can also occur in a different problem
  - Reuse the subtask *module* that we have already programmed

Example – Factorial subtask

$$c = \frac{n!}{i!\,(n - i)!}$$

A program to calculate c would require three separate loops to calculate $n!$, $i!$, and $(n-i)!$ as shown in the following program and flowchart:

Flowchart


Read
n, i
k1=1 k2=1
k3=1
Start = 1
Stop = n
k
k1 = k1*k
Start = 1
Stop = i
l
k2 = k2*l
Start = 1
Stop = n−i
m
k3=k3*m
c = k1/k2/k3
Print
c

Program

```
      READ * , N , I
      K1 = 1
      DO 10 K = 1 , N
            K1 = K1 * K
10    CONTINUE
      K2 = 1
      DO 20 L = 1 , I
            K2 = K2 * L
20    CONTINUE
      K3 = 1
      DO 30 M = 1 , N - I
            K3 = K3 * M
30    CONTINUE
      C = K1 / K2 / K3
      PRINT * , C
      END
```

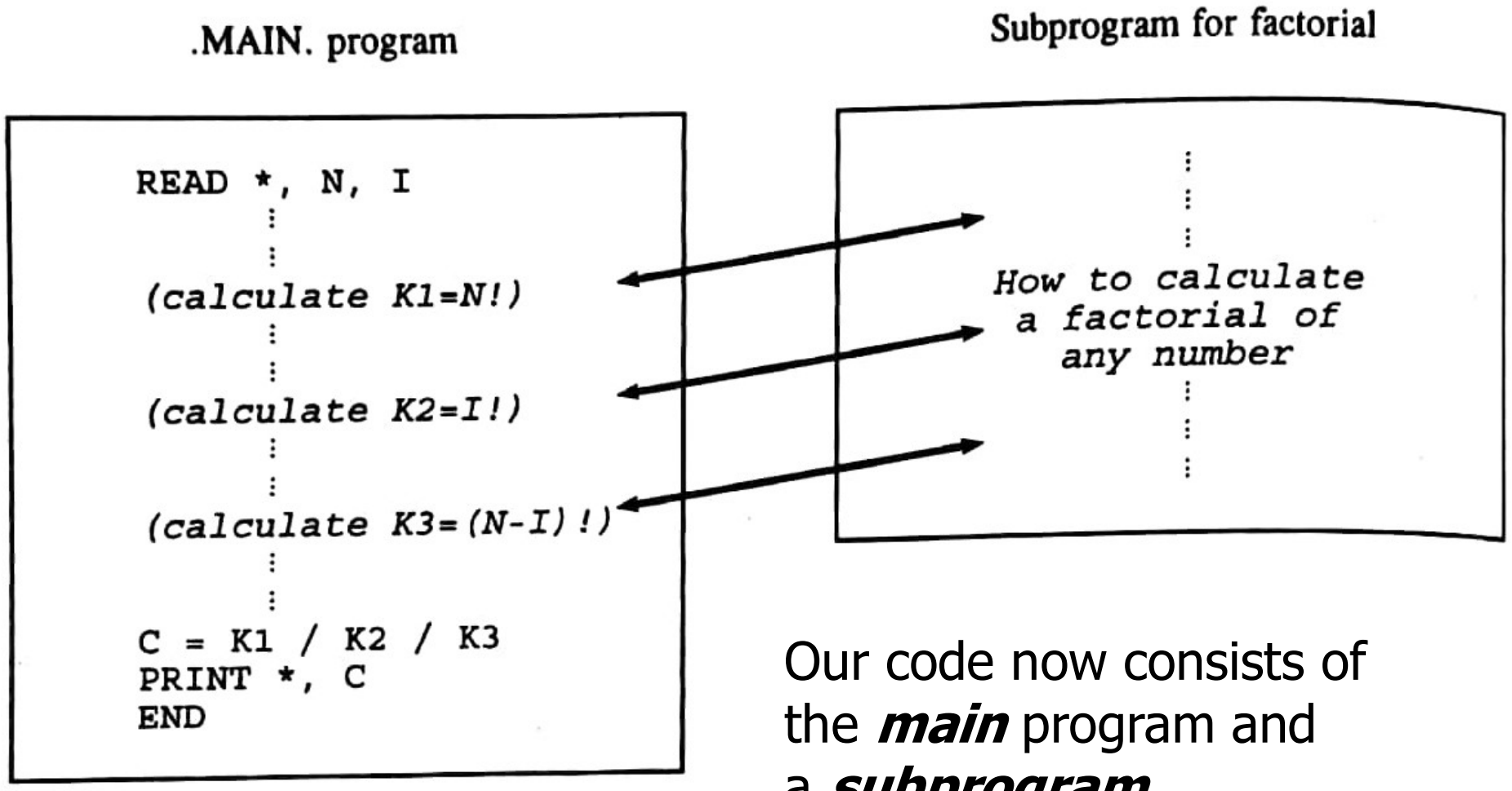
Loop to calculate N!

Loop to calculate I!

Loop to calculate (N−I)!

It works, but in a clumsy way

Better to code the common task *factorial* into a reusable subprogram



Our code now consists of the ***main*** program and a ***subprogram***

Advantages of using Subprograms

- *Simplify* job by focusing on a small task assigned to the subprogram
- *Reuse* the code in the subprogram
- It is *portable* – can be saved in a *library* and used by other programs or other programmers
- Other *libraries* (already optimized and tested) are available for your use (LAPACK, etc.)
- Subprograms make codes smaller – easier to *debug*

Types of Subprograms

- FUNCTION
  - Intrinsic Function (Built-in Function)
  - User-defined Function
  - A Function returns only one value.
- SUBROUTINE
  - Can return many values

Functions

- Returns a single value when *called*
- *Intrinsic* (built-in) or *user defined*
- Intrinsic function example: `SQRT(X)`

  `Y = SQRT(X)` ← *calling* statement

  – Returns the value for SQRT of X and assigns the value to variable Y
- For other tasks not available as already built-in in FORTRAN, we need *user-defined functions*.

User-defined Function

- As a separate subprogram, it can be coded as a separate program file, or separately from the MAIN program block in the same file.
- The structure of a user-defined function is

```
type FUNCTION name (list of variables)
        subprogram instructions
RETURN
END
```

Examples of first line of *function* subprogram

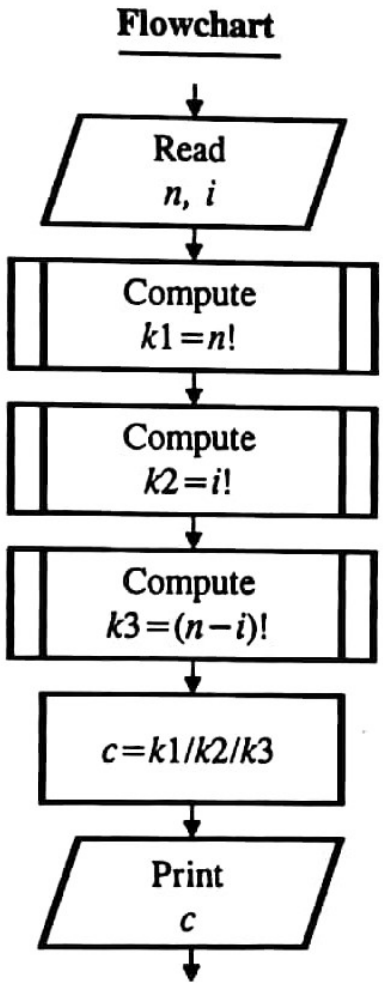
List of variables is also called function *arguments*

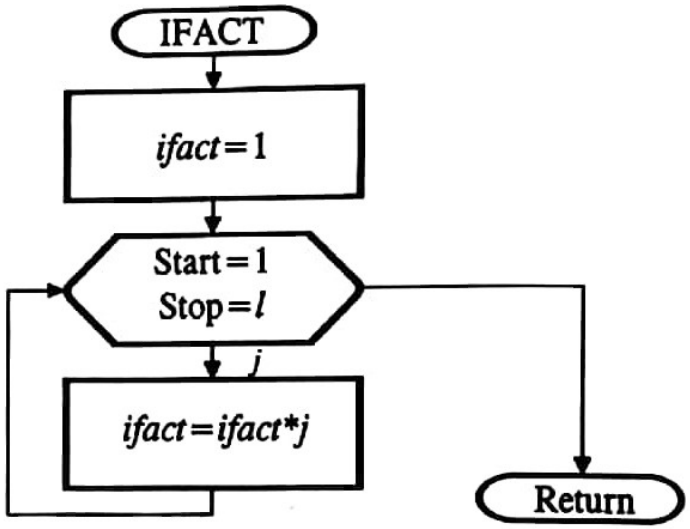
```
INTEGER FUNCTION IFACT(N)   or   FUNCTION IFACT(N)

DOUBLE PRECISION FUNCTION COMPUTE(A, B, C)
```

Can be more than one variable

Back to this example, but using *function*

$$c = \frac{n!}{i!\,(n-i)!}$$

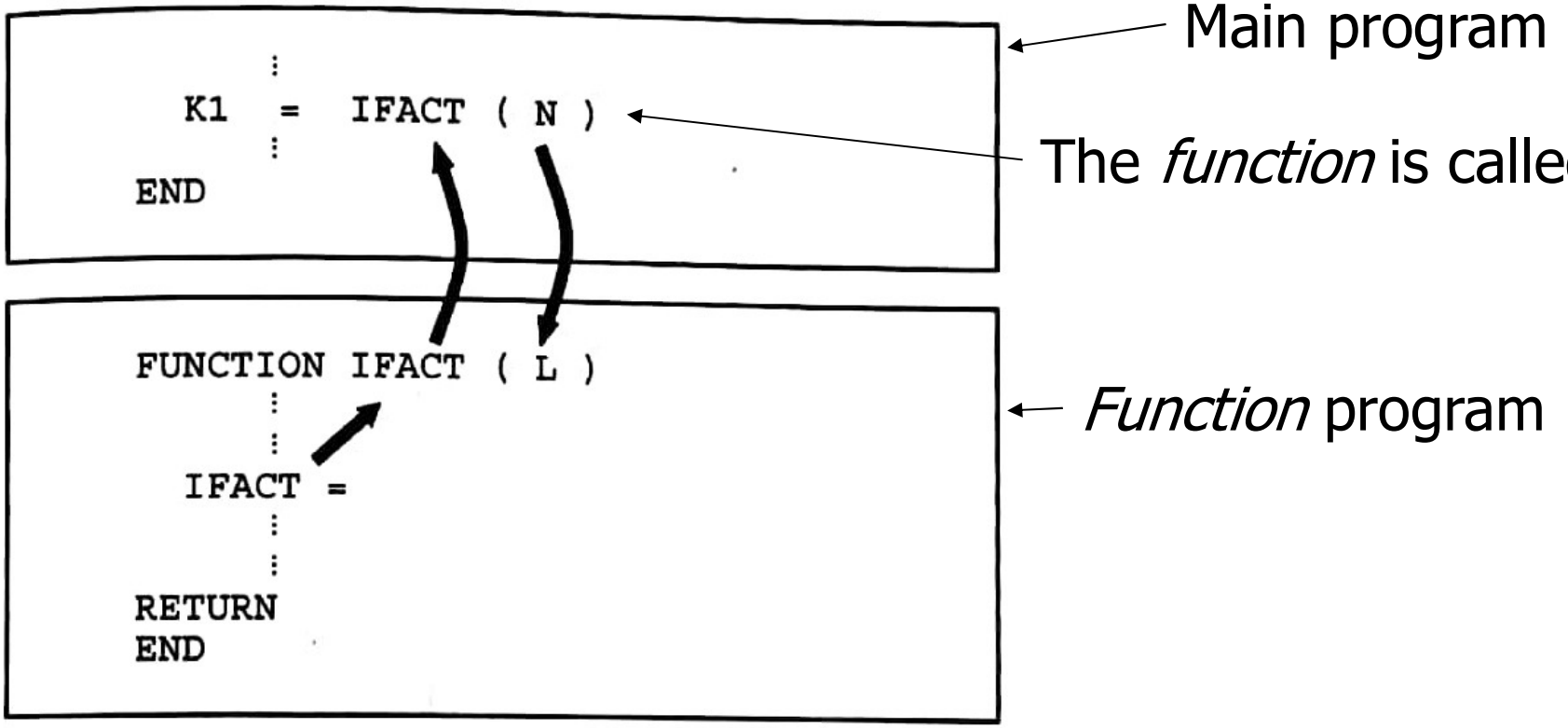




Program

```
C The MAIN program is primarily concerned with control. Details
C of the factorial calculation are handled in the function.
C*************************************************************
      PRINT *, 'ENTER N & I:'
      READ *, N, I

C Three calls to the function to compute N!, I! and (N-I)!
C*************************************************************
      K1 =  IFACT ( N )
      K2 =  IFACT ( I )
      K3 =  IFACT ( N-I )
      C = K1 / K2 / K3
      PRINT *, 'C= ', C
      END
C*************************************************************
C The function is written as a stand alone module.
C*************************************************************
      FUNCTION IFACT ( L )
      IFACT = 1
      DO 10 J = 1 , L
         IFACT = IFACT * J
10    CONTINUE
      RETURN
      END
```

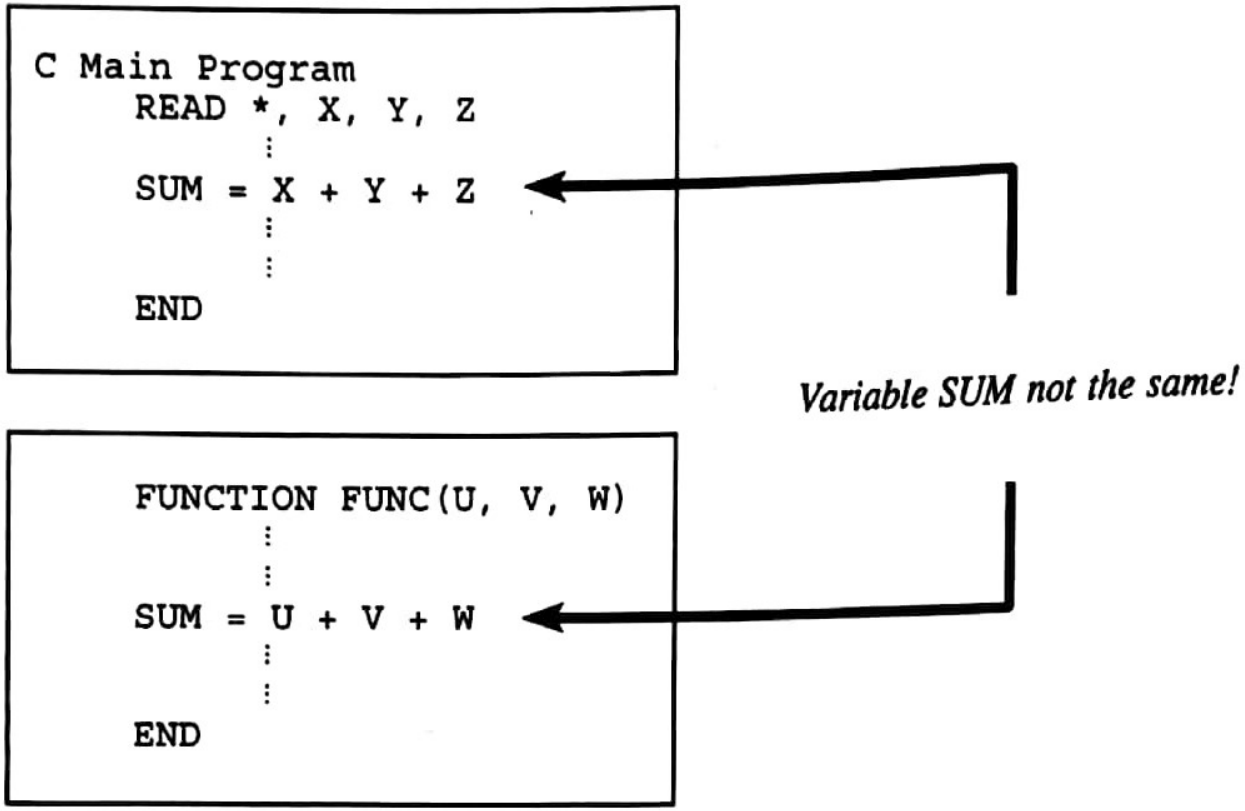
How it behaves



The function name occurs three times:

- In the *calling statement* in the main program
- In the *name of the function* subprogram
- As the *variable* whose value is calculated within the subprogram

Variables are *local*



Variables are *local*

EXAMPLE 7.4

Trace through the following program segment to see what the output of the program will be:

```
        X = 1.2345
        Y = 9.8765
        SUM = X + Y
        PRINT *, SUM
        PRINT *, FUNC(X, Y)
        PRINT *, A, B
        END
C*************************************************************
C The values of X and Y are transferred to the local variables
C A and B. The function computes A+B and returns the result.
C*************************************************************
        FUNCTION FUNC(A, B)
        FUNC = A + B
        RETURN
        END
```

Passing of variables (Data passing) between Main and Subprograms

EXAMPLE 7.5

Assume that we wish to send three variables to a function called ADD. The first two variables, X and Y, are real, but the third, J, is an integer:

```
                    real  real integer
                      ↕     ↕     ↕
Y = ADD  ( X,    Y,    J )
```

The corresponding function statement variables must match in number, order and type:

```
                        real  real  integer
                          ↕     ↕     ↕
FUNCTION ADD  ( A  ,  X  ,  I )
```

Note that X from the main will be assigned to A in the function, not to X in the function, even though they have the same variable name. Position within the list determines the assignment, not similarity of variable names! This confusion often creeps in when you are using subprograms written by someone else.

Subroutines

- Can return more than a single numerical value
  - Calculate many variables
  - Work on data arrays
  - Curve fitting of data
  - Etc.
- Three step process:
  - Call the subroutine from the main program
  - Pass data to the subroutine
  - Set up the subroutine to receive the passed data, process it, and return the results

Calling statement for subroutines

In the main program;

CALL ***subroutinename*** **(** ***variable1*** **,** ***variable2*** **, . . . ,** ***variableN*** **)**

- *subroutinename* is the name of the subroutine being called
- *variable1,…variableN* are two-way variables, they can be the variables passed from the main program to the subroutine, and be the variables returned from the subroutine to the main program

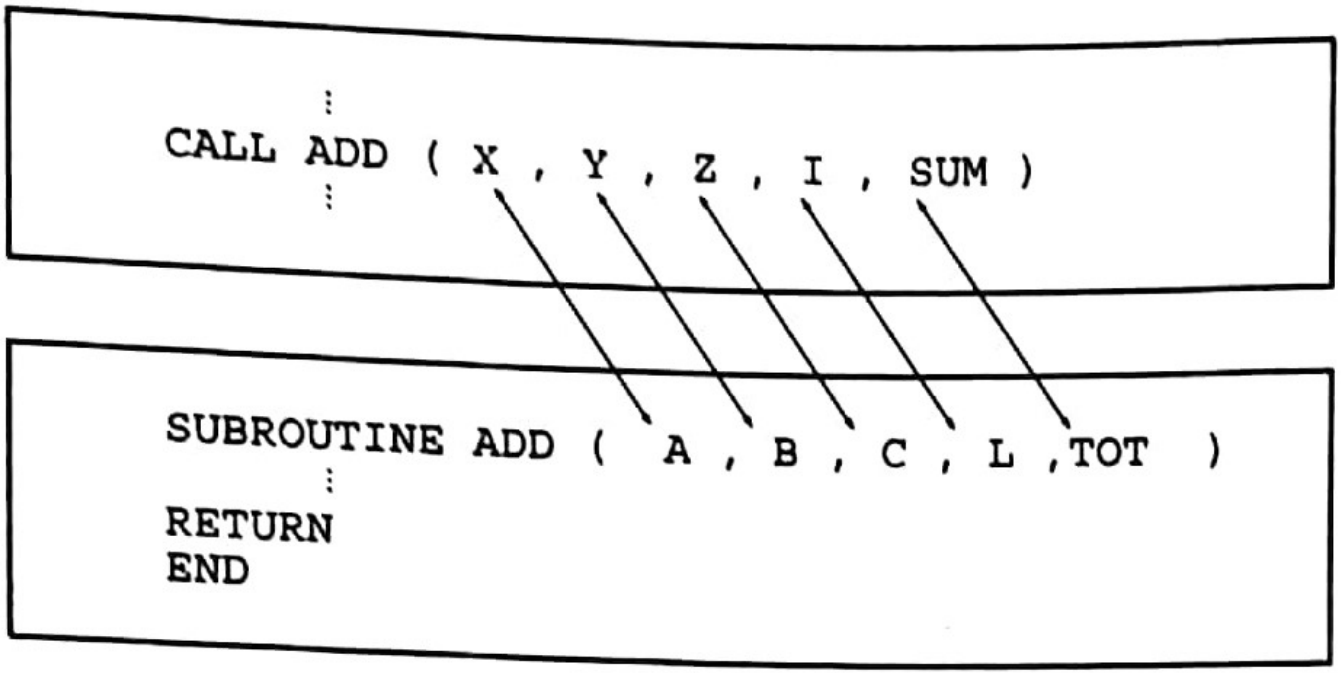
Structure of a subroutine

```
SUBROUTINE subroutinename ( variable1 , variable2 , . . . , variableN )
          :
          :
RETURN
END
```

subroutinename

- Same as the one in the calling statement
- Follow the standard rules for any variable name

Data passing



The passed data

- can have *different names*
- must be the same *number* of variables
- must be the same *types*
- must be the same *order* as their intended meaning

Two-way data passing

- Subroutine arguments are two-way variables, they can be the variables passed from the main program to the subroutine, and be the variables returned from the subroutine to the main program
- If the subroutine changes the value of a variable that it receives from MAIN, the changed value will also be returned and changed in MAIN

```
                ⋮
      DATA X, Y, Z, I/2.0, 3.0, 4.0, 3/
      CALL ADD ( X , Y , Z , I , SUM )
      PRINT *, SUM
                ⋮
```

```
      SUBROUTINE ADD(A , B , C , L ,TOTAL )
      DO 10    K = 1 , L
         TOTAL = TOTAL  + A / B
10    CONTINUE
      A = TOTAL/C
      RETURN
      END
```

Arrays and Subroutines

To pass arrays between MAIN and subroutine;

- We have to *declare the array* twice; in MAIN, and in subroutine
- To pass whole array, just use the array name (without the index) in the subroutine arguments
- We may also pass individual elements of an array
  - This needs the index along with the array name
  - The receiving variable will also be a single value variable

Whole array passing

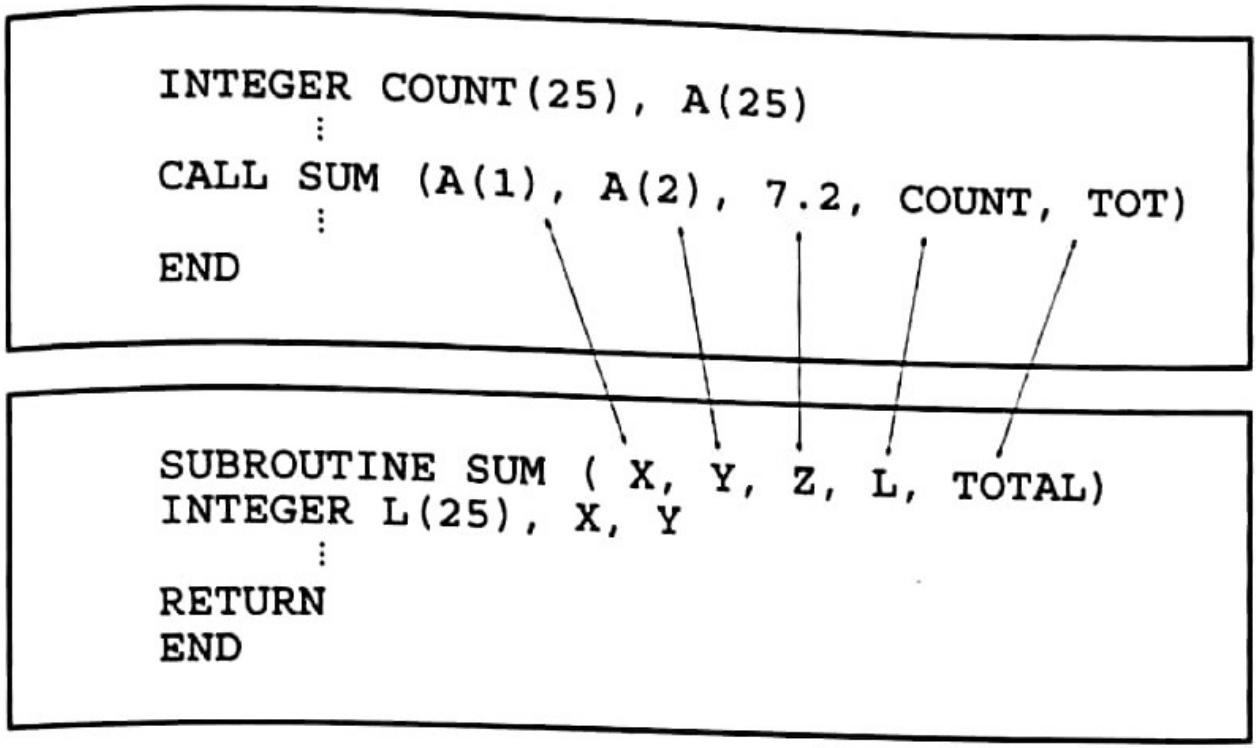
EXAMPLE 7.8

When we pass the entire array A to the subroutine SUM, we have to declare the array twice:

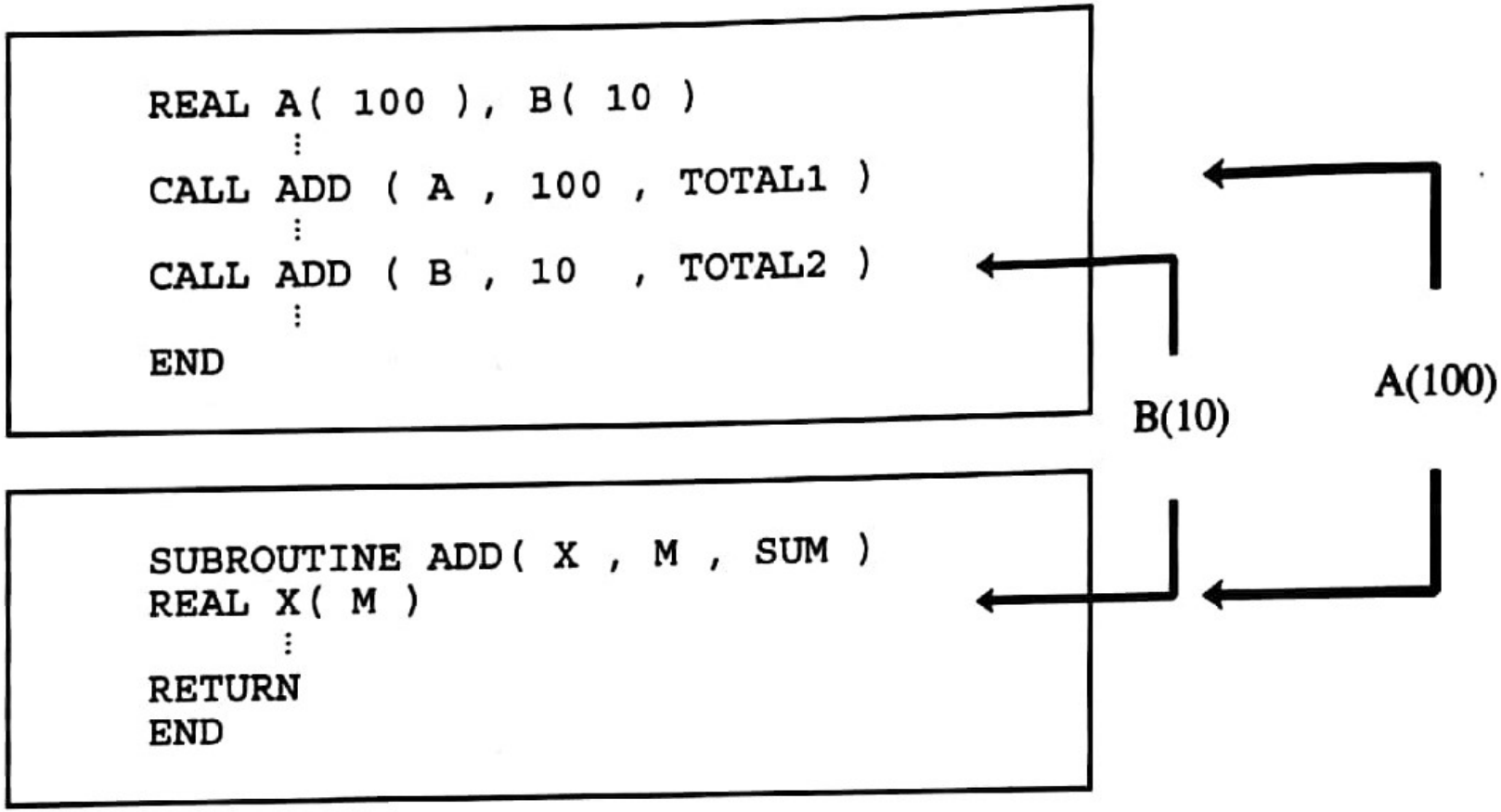
```
      REAL A(100)
           ⋮
      CALL SUM ( A, TOT )
           ⋮
      END
```

```
      SUBROUTINE SUM( X , TOTAL )
      REAL X(100)
           ⋮
      RETURN
      END
```

Passing of individual element of an array



Variable-sized array trick



Variable-sized array is not allowed in MAIN, but allowed in subroutines

The COMMON data block

- Variables are local, but sometimes it is more convenient to have data be globally available
- Example below;

```
CALL DUMMY(A, B, I, J, C, D, E, F, U)
CALL DUMMY(A, B, I, J, C, D, E, F, V)
CALL DUMMY(A, B, I, J, C, D, E, F, W)
```

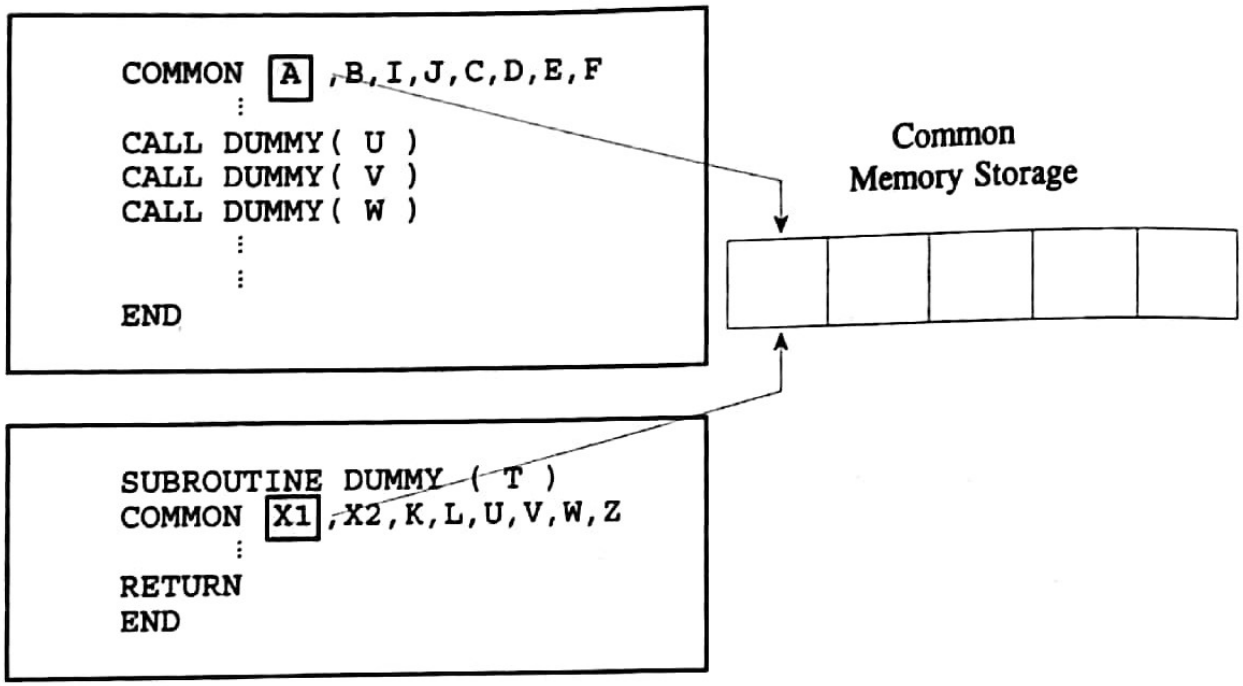
- Only the last variable is different, so it is more convenient to have the other data be the common data (global data)
- The COMMON block allows us to declare a list of variables to be global

The COMMON data block

- The syntax of the statement is

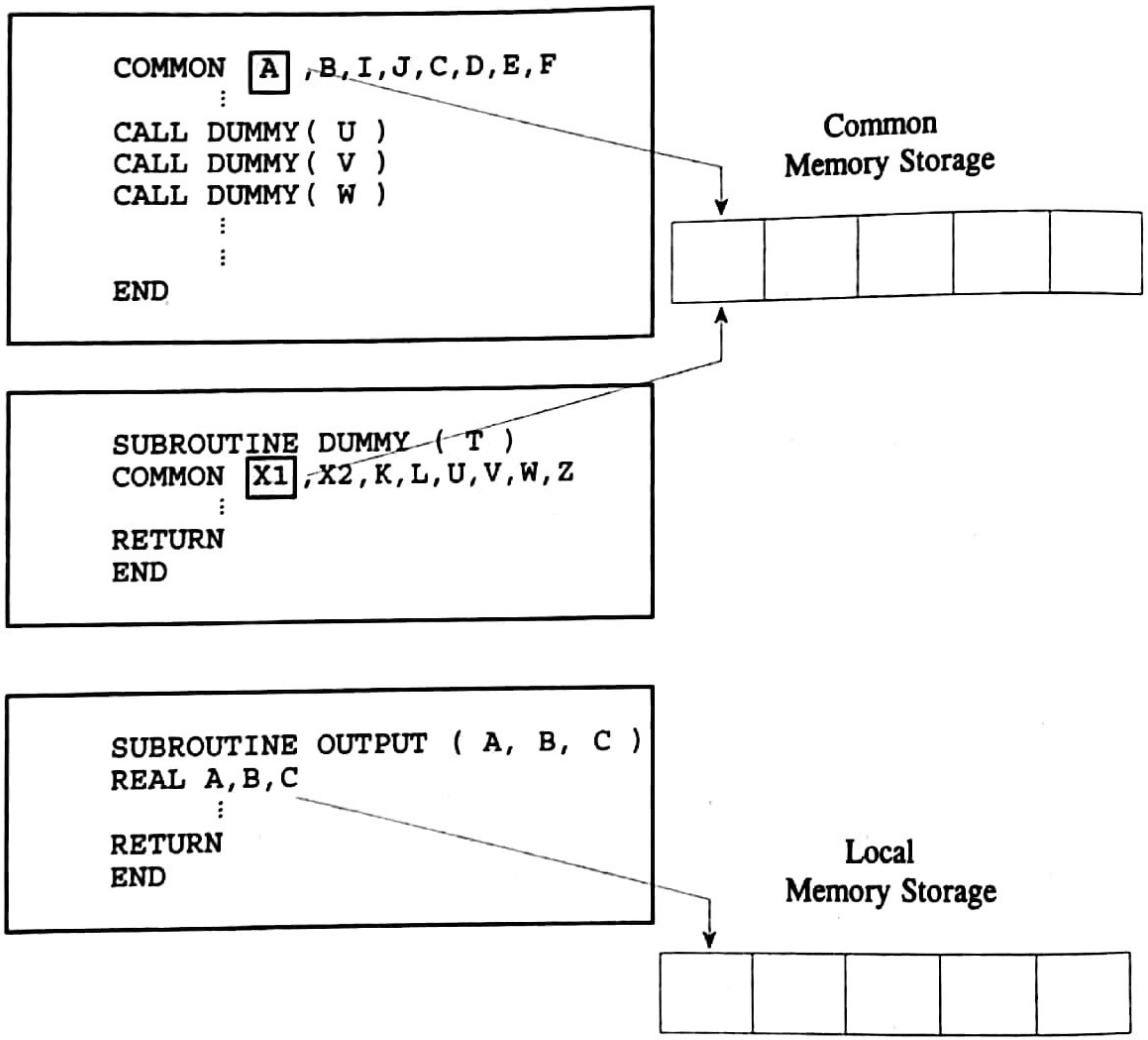
COMMON *variable1, variable2,...,variableN*

- The usage is



The COMMON data block

- Common (global) data vs local variable



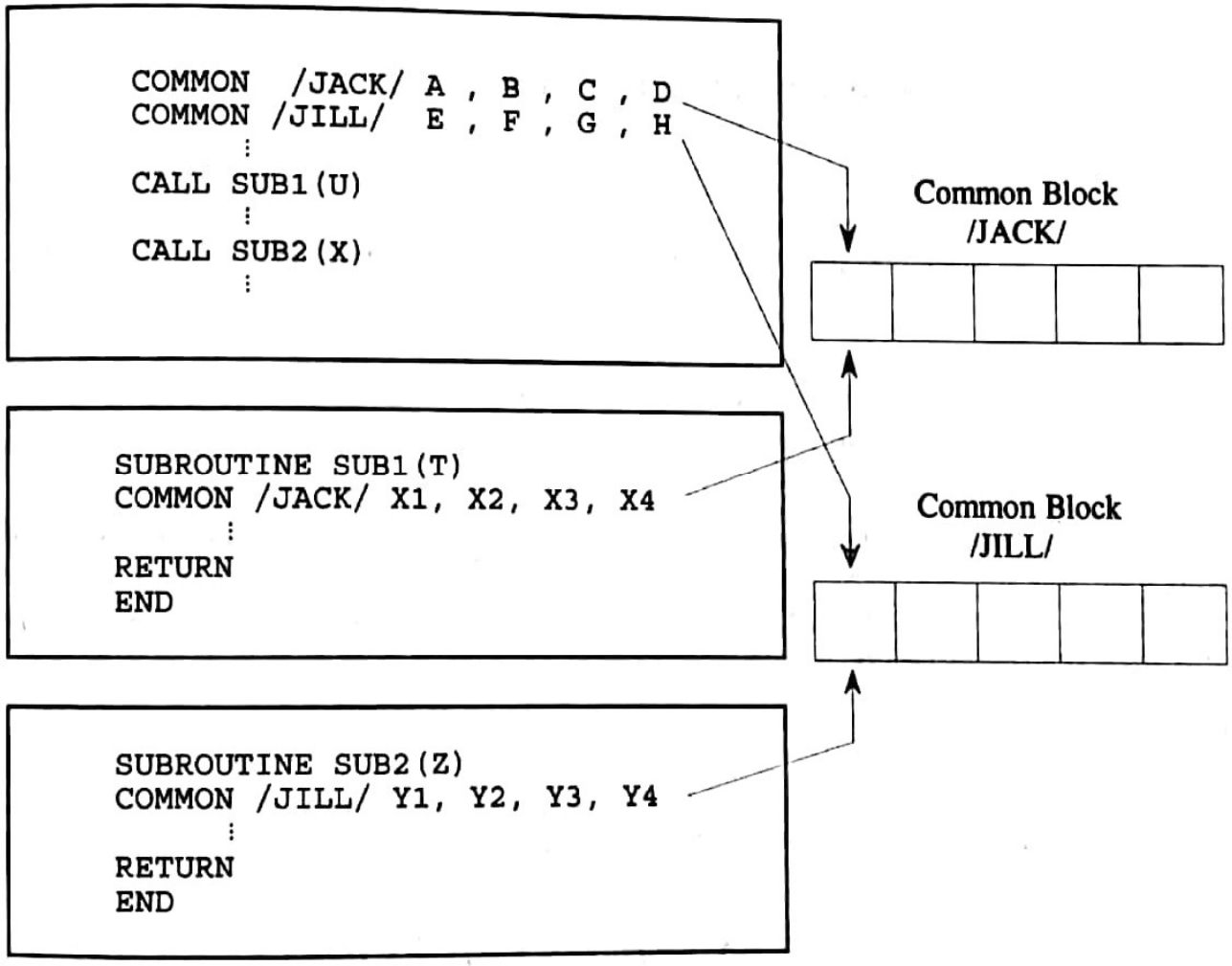
The named COMMON block

- Sometimes convenient to group some data into different blocks
- These blocks are then given different names
- The syntax is

COMMON */name/ variable1, variable2, ... , variableN*

The named COMMON block

- The usage is



The COMMON data block

Final remarks

- Even though COMMON block is convenient, try to avoid from using it 😱
- Subprograms are coded to be independent of each other, but usage of the COMMON block goes against this
- This might produce errors that are difficult to debug

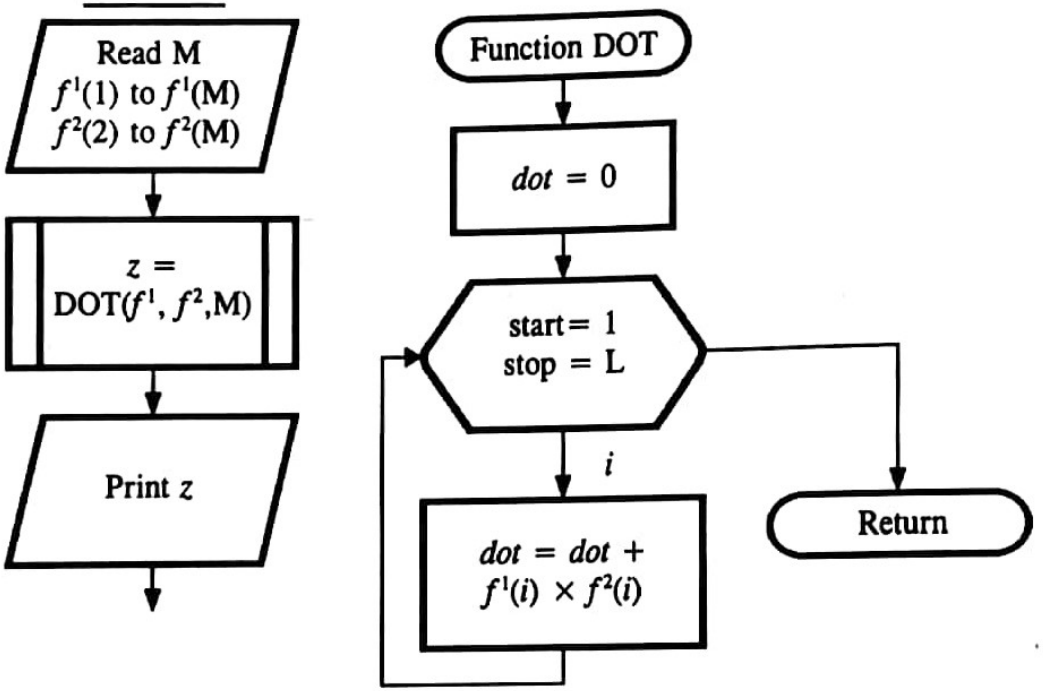
FUNCTION application – The dot product

EXAMPLE 7.12

Assume that we have two vectors, f^1 and f^2. The *dot product* of the two vectors (indicated by the symbol $\odot$) is defined by summing the products of the components of each vector:

$$f^1 \odot f^2 = \sum_{i=1}^{i=3} f_i^1 f_i^2 = f_1^1 f_1^2 + f_2^1 f_2^2 + f_3^1 f_3^2$$

Flowchart



FUNCTION application – The dot product

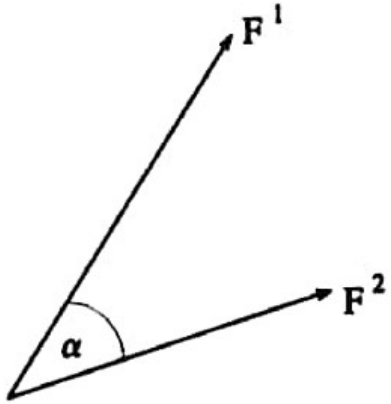
Program

```
C*************************************************************
C Main program reads in A and B vectors and calls function DOT
C to compute the dot product of A and B
C*************************************************************
      REAL A(100), B(100)
      PRINT *, 'Enter number of components, and A & B vectors:'
      READ *, M, (A(I), I = 1, M), (B(I), I = 1, M)
      Z = DOT(A, B, M)
      PRINT *, 'DOT PRODUCT = ', Z
      END

C*************************************************************
C Function subprogram to calculate the dot product of two vectors
C of arbitrary size (L)
C*************************************************************
      FUNCTION DOT(F1, F2, L)
      REAL F1(L), F2(L)
      DOT = 0.0
      DO 10 I = 1, L
         DOT = DOT + F1(I)*F2(I)
10    CONTINUE
      RETURN
      END
```

FUNCTION application – Determining angle between two vectors

EXAMPLE 7.13

Consider two vectors, F^1 and F^2, where we wish to know the angle between them:



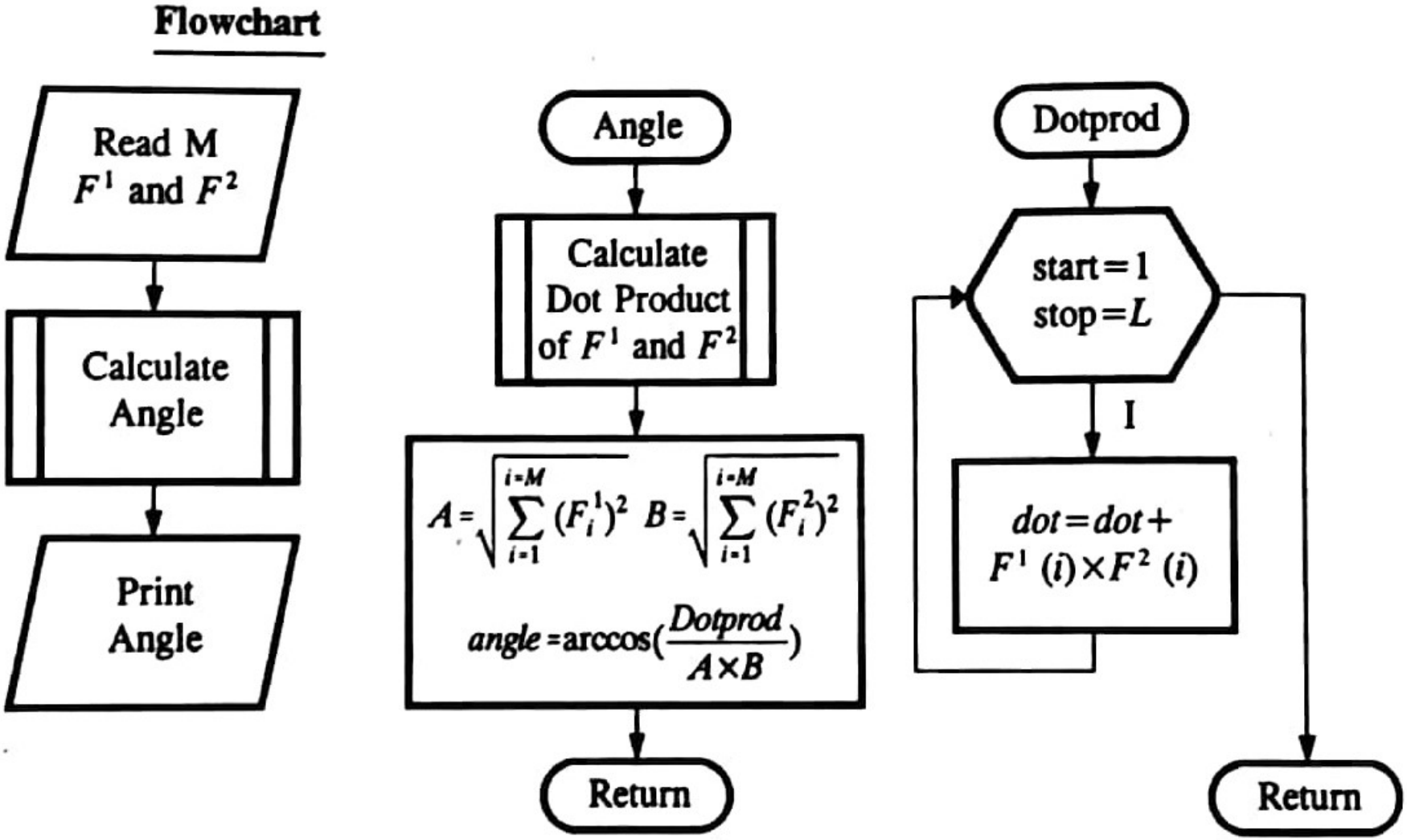
We can find the angle α between the two vectors, F^1 and F^2, with the formula:

$$\cos(\alpha) = \frac{F^1 \odot F^2}{|F^1||F^2|}$$

where $F^1 \odot F^2$ = the dot product of the two vectors, and $|F^1|$ and $|F^2|$ are the lengths of the vectors respectively. The length can be computed by

$$|F| = [(F_1)^2 + (F_2)^2 + (F_3)^2]^{1/2}$$

FUNCTION application – Determining angle between two vectors



FUNCTION application – Determining angle between two vectors

Program

```
C In the main program, we read in the vectors and call the
C appropriate functions. We use the main program to control the
C sequence of operations and leave the details to the functions.
C*************************************************************
      REAL F1(100), F2(100)
      PRINT *, 'Enter number of elements and F1 and F2:'

      READ *, M, (F1(I), I=1,M), (F2(I), I=1,M)
      ANS = ANGLE(F1, F2, M)
      PRINT *, 'Angle = ', ANS * 57.296
      END
```

Functions on the next page

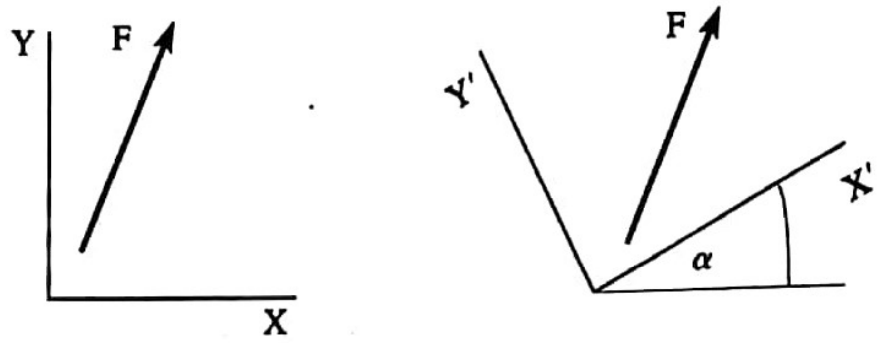
FUNCTION application – Determining angle between two vectors

```
C*************************************************************
C Function Angle to compute the angle between two vectors. This
C function calls another function (DOT) to compute the dot
C product that is needed for the computation of the angle.
C*************************************************************
      FUNCTION ANGLE(F1, F2, M)
      REAL F1(M), F2(M)
      A = 0.0
      B = 0.0
      DO 10 I = 1, M
           A = A + F1(I)**2
           B = B + F2(I)**2
 10   CONTINUE
      DOTPROD = DOT(F1, F2, M)
      ANGLE = ACOS(DOTPROD/SQRT(A*B))
      RETURN
      END
C*************************************************************
C Function to compute the dot product of two vectors
C of arbitrary size (L)
C*************************************************************
      FUNCTION DOT(F1, F2, L)
      REAL F1(L), F2(L)
      DOT = 0.0
      DO 10 I = 1, L
         DOT = DOT + F1(I)*F2(I)
 10   CONTINUE
      RETURN
      END
```

SUBROUTINE application – Vector coordinate transformation

EXAMPLE 7.14

If we have a vector F in the X-Y-Z coordinate system, it is sometimes easier to work in a different coordinate system such as X'-Y'-Z' shown below (Z and Z' axes not shown for clarity). The new, transformed coordinate system might be more desirable due to the fact that the mathematics might be simpler. The simplest transformation is a rotation about one axis:

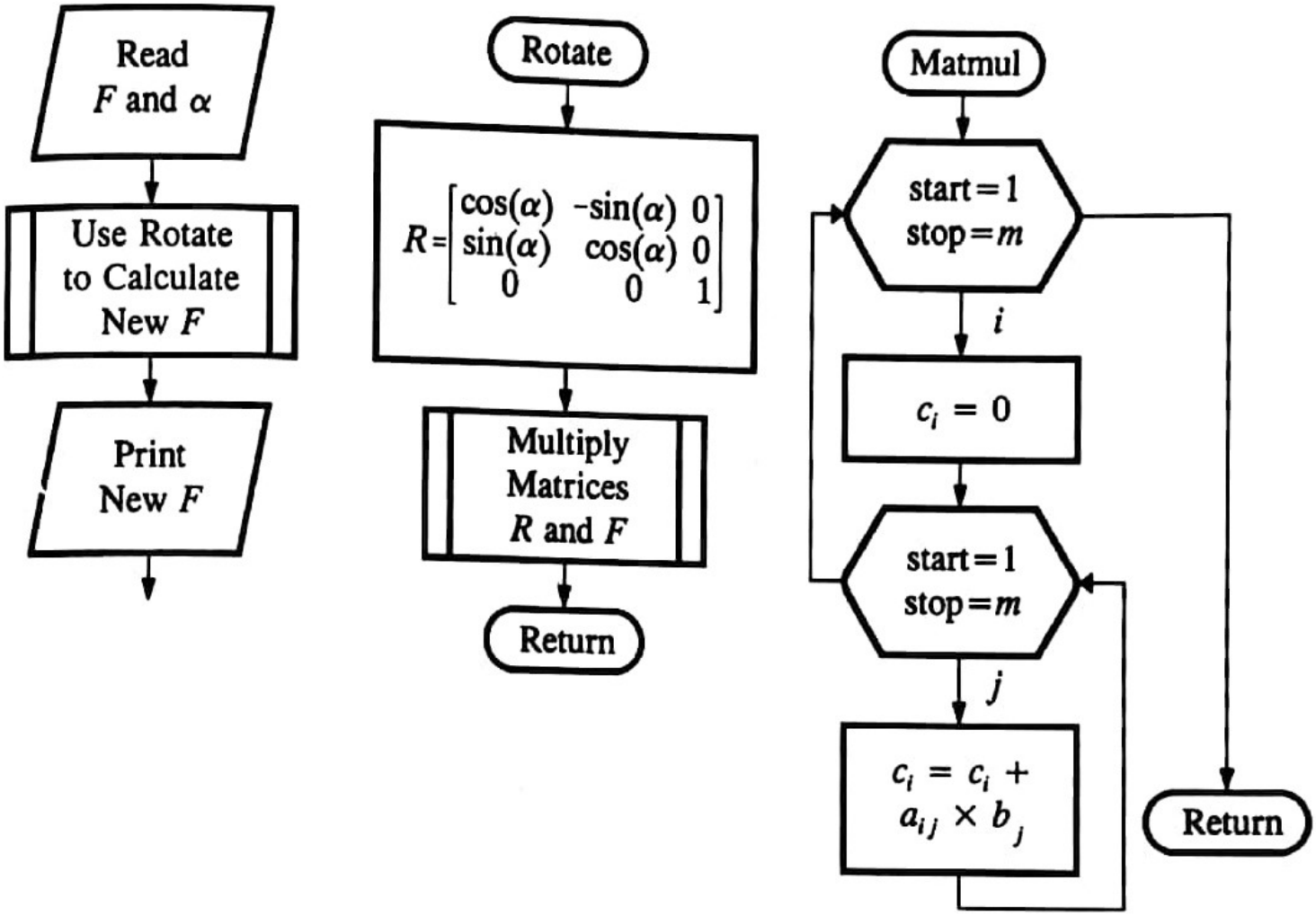


In this transformation, we have taken a rotation through an angle α about the axis perpendicular to the page. Notice that the vector has not changed position, nor has its length changed. But the *components* in the new X'-Y'-Z' coordinate system have changed. If the vector coordinates in the old X-Y-Z system were $F = (F_1, F_2, F_3)$, then we can calculate the components of the new vector as (F'_1, F'_2, F'_3) with the aid of the equation:

$$\begin{bmatrix} F'_1 \\ F'_2 \\ F'_3 \end{bmatrix} = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} F_1 \\ F_2 \\ F_3 \end{bmatrix}$$

SUBROUTINE application – Vector coordinate transformation

Flowchart



SUBROUTINE application – Vector coordinate transformation

Program

```
C Main program reads in F and angle alpha, then calls the
C subroutine Rotate to perform the transformation.
C*************************************************************
      REAL F(3), F2(3)
      PRINT *, 'ENTER THE VECTOR AND THE ROTATION ANGLE:'
      READ *, (F(I), I = 1, 3), ANGLE
      CALL ROTATE (F, ANGLE, F2)
      END
C*************************************************************
C  Subroutine rotate for a simple rotation about the Z axis
C*************************************************************
      SUBROUTINE ROTATE (F, ANGLE, F2)
      REAL F(3), F2(3), R(3,3)
      ANGLE=ANGLE/57.2958
      R(1,1) = COS(ANGLE)
      R(1,2) = -SIN(ANGLE)
      R(2,1) = SIN(ANGLE)
      R(2,2) = COS(ANGLE)
      R(3,3) = 1.0
      CALL MATMUL( R, F, F2, 3, 3)
      RETURN
      END
```

SUBROUTINE application – Vector coordinate transformation

```
C*************************************************************
C Subroutine MATMUL performs multiplication of two matrices
C (A and B) and storing the result in array C.
C*************************************************************
      SUBROUTINE MATMUL( A, B, C, M, N)
      REAL A(M,N), B(N), C(M)
      DO 10 I = 1, M
         C(I) = 0.0
         DO 10 J = 1, N
            C(I) = C(I) + A(I,J)*B(J)
10    CONTINUE
      RETURN
      END
```