

Control Structures and Program Design

Program Design Process

1. Clearly state the problem that you are trying to solve.
2. Define the inputs required by the program and the outputs to be produced by the program.
3. Design the algorithm that you intend to implement in the program.
4. Turn the algorithm into FORTRAN statements.
5. Test the FORTRAN program.

Follow the steps of the program-design process to produce reliable, understandable FORTRAN programs.

Flowcharts

Flowcharts are a way to describe algorithms graphically. In a flowchart, different graphical symbols represent the different operations in the algorithm and our standard constructs (standard forms used to describe algorithms) are made up of collections of one or more of these symbols.

Control Constructs: Branches

Branches are FORTRAN statements that permit us to select and execute specific sections of code (called blocks) while skipping other sections of code. They are variations of the IF statement, plus the SELECT CASE.

The Block IF construct

This construct specifies that a block of code will be executed if and only if a certain logical expression is true. Its General form is

```
IF (logical_expr)then  
  Statement 1  
  Statement 2  
  ....  
END IF
```

} *Block 1*

If the logical expression is true, the program executes the statements in the block between the IF and END IF statements. If the logical expression is false, then the program skips all the statements in the block between the IF and END IF statements and executes the next statement after the END IF.

The Else and ELSE IF clauses

The Block IF construct with an ELSE clause and an ELSE IF clause has this form:

```

    IF (logical_expr_1) then
        Statement 1
        Statement 2
        ....
    ELSE IF (logical_expr_2) then
        Statement 1
        Statement 2
        ....
    ELSE
        Statement 1
        Statement 2
        ....
    END IF

```

} Block 1
 } Block 2
 } Block 3

If logical_expr_1 is true, then the program executes the statements in block 1 and skips to the first executable statement following the END IF. Otherwise, the program checks for the status of the logical_expr_2. If logical_expr_2 is true, then the program executes the statements in block 2 and skips to the first executable statement following the END IF. If both logical expressions are false, then the program executes the statements in Block 3. The line containing the ELSE and ELSE IF statement should not have a statement number.

Named Block IF constructs

General form is

```

[name:] IF (logical_expr_1) then
    Statement 1
    Statement 2
    ....
ELSE IF (logical_expr_2) then [name]
    Statement 1
    Statement 2
    ....
ELSE
    Statement 1
    Statement 2
    ....
END IF [name]

```

} Block 1
 } Block 2
 } Block 3

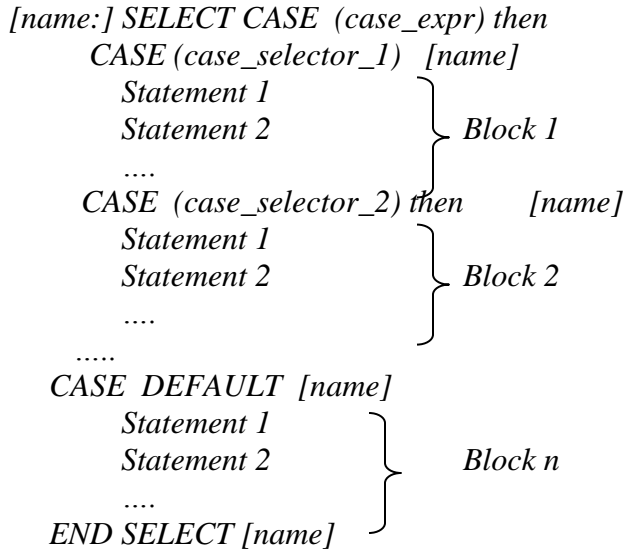
Where name may be up to 31 alphanumeric characters long, beginning with a letter. It should be unique within each program unit and must not be the same as any constant or variable name within the program unit. If the name is assigned to an IF, then the same name must appear on the associated END IF. Names are optional on the ELSE and ELSE IF statements of the construct, but if they are used, they must be same as the name on the IF.

Assign a name to any large and complicated IF constructs in your program to help you keep track of the parts of the construct.

The Case Construct

It permits a programmer to select a particular code block to execute based on the value of the single integer, character, or logical expression. The general form is

```
[name:] SELECT CASE (case_expr) then
  CASE (case_selector_1) [name]
    Statement 1
    Statement 2
    ....
  CASE (case_selector_2) then [name]
    Statement 1
    Statement 2
    ....
  ....
  CASE DEFAULT [name]
    Statement 1
    Statement 2
    ....
  END SELECT [name]
```



If the value of the `case_expr` is in the range of values included in `case_selector_1`, then the first code block will be executed. Similarly, if the value of `case_expr` is in the range of values included in `case_selector_2`, then the second code block will be executed. The same idea applies for any other cases in the construct. The default code block is optional. If it is present, the default code block will be executed whenever the value of `case_expr` is outside the range of all of the case selectors. If it is not present and the value of `case_expr` is outside the range of all of the case selectors, then none of the code blocks will be executed.

If the name assigned to a `SELECT CASE` statement, then the same name must appear on the associated `END SELECT`. Names are optional on the `CASE` statements of the construct, but if they are used, they must be the same as the name on the `SELECT CASE` statement.

The `case_expr` may be any integer, character, or logical expression. Each case selector must be an integer, character, or logical value or range of values. All case selectors must be mutually exclusive: no single value can appear in more than one case selector.

Control Constructs: Loops

Loops permit us to execute a sequence of statements more than once. Two basic forms of loop constructs are While loops and iterative loops (or counting loops).

While loop

It is a block of statements that are repeated indefinitely as long as some condition is satisfied. General form is

```
Do
  ...
  IF (logical_expr) Exit
  ...
END Do
```

} Code Block

The block of statements between the DO and END DO is repeated indefinitely until the logical_expr becomes true and the EXIT statement is executed. After the EXIT statement is executed, control transfers to the first statement after the END DO.

The IF statement may be located anywhere within the body of the loop, and it is executed once each time that the loop is repeated. If the logical_expr in the IF is false, when the statement is executed, the loop continues to execute. If the logical_expr in the IF is true when the statement is executed, control transfers immediately to the first statement after the END DO.

If the logical expression is false the first time we reach the while loop, the statements in the loop below the IF will never be executed!

The Iterative or Counting Loop

General form is

```
Do index = istart, iend, incr
  Statement 1
  Statement 2
  ...
  Statement n
END Do
```

} Body

Index is an integer variable used as the loop counter (also known as the loop index). The integer quantities istart, iend, and incr are the parameters of the counting loop. They control the values of the variable index during the execution. The parameter incr is optional. If it is missing, it is assumed to be 1.

The counting loop construct functions as follows:

1. The Do loop parameters may be a constant, a variable, or an expression.
2. At the beginning of the execution of the DO loop, the program assigns the value istart to control variable index. If $\text{index} * \text{incr} \leq \text{iend} * \text{incr}$, the program executes the statements within the body of the loop.
3. After the statements in the body of the loop have been executed, the control variable is recalculated as $\text{index} = \text{index} + \text{incr}$. If $\text{index} * \text{incr}$ is still less than or equal to $\text{iend} * \text{incr}$, the program executes the statements within the body again.
4. Step 2 is repeated over and over as long as $\text{index} * \text{incr} \leq \text{iend} * \text{incr}$. When this condition is no longer true, execution skips to the first statement following the end of the Do loop.

Always indent the body of a Do loop by two or more spaces to improve the readability of the code.

Never modify the value of a Do loop index variable while inside the loop.

Never modify the control values of a Do loop (istart, iend, incr) while inside the loop

Never depend on an index variable to retain a specific value after a Do loop completes normally.

The CYCLE and EXIT statements

If the CYCLE statement is executed in the body of a loop, the execution of the body will stop and control will be returned to the top of the loop. The loop index will be incremented, and execution will resume again if the index has not reached its limit.

If the EXIT statement is executed in the body of the loop, the execution of the body will stop and control will be transferred to the first executable statement after the loop.

Examples:

```
(1)  PROGRAM test_cycle
      INTEGER :: i
      Do i = 1,5
        IF(i == 3) CYCLE
        Write(*,*) i
      END DO
      Write(*,*) 'End of Loop!'
END PROGRAM
```

```
(2)  PROGRAM test_exit
      INTEGER :: i
      Do i = 1,5
        IF(i == 3) EXIT
        Write(*,*) i
      END DO
      Write(*,*) 'End of Loop!'
END PROGRAM
```

Named Loops

While loop

```
[name:]      Do
    Statement 1
    Statement 2
    ...
    IF (logical_expr) CYCLE [name]
    IF (logical_expr) Exit  [name]
    ....
END Do  [name]
```

Iterative or counting loop

```
[name: ] Do index = istart, iend, incr
    Statement 1
    Statement 2
    ...
    IF (logical_expr) CYCLE [name]
    ....
END DO [name]
```

The accidental deletion of an END DO statement in a large set of nested DO loops can produce a hard- to- find error. Use names on nested Do loops to avoid this problem. Assign names to all nested loops so that they will be easier to understand and debug. Use independent index variables for each loop in a set of nested Do loops. If two loops are nested, always ensure that one of them lies completely within the other. Use loop names with CYCLE or EXIT statements in nested loops to make sure that the statements affect the proper loop.