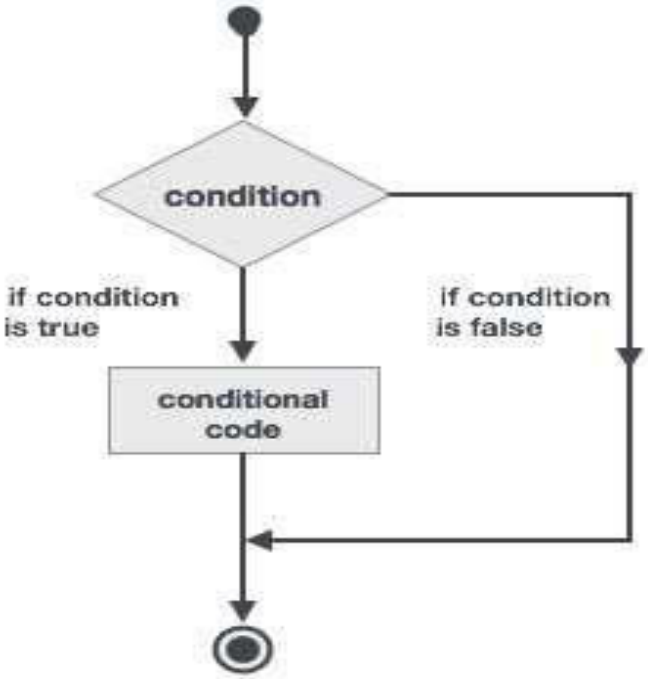
Fortran – Decisions

Decision making structures require that the programmer specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed, if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

Following is the general form of a typical decision making structure found in most of the programming languages:



Fortran provides the following types of decision making constructs.

Statement	Description
If… then construct	An if… then… end if statement consists of a logical expression followed by one or more statements.
If… then...else construct	An if… then statement can be followed by an optional else statement, which executes when the logical expression is false.
nested if construct	You can use one if or else if statement inside another if or else if statement(s).
select case construct	A select case statement allows a variable to be tested for equality against a list of values.
nested select case construct	You can use one select case statement inside another select case statement(s).

If…then Construct

An if… then statement consists of a logical expression followed by one or more statements and terminated by an end if statement.

Syntax : The basic syntax of an if… then statement is

if (logical expression) then

statement

end if

However, you can give a name to the if block, then the syntax of the named if statement would be, like:

[name:] if (logical expression) then

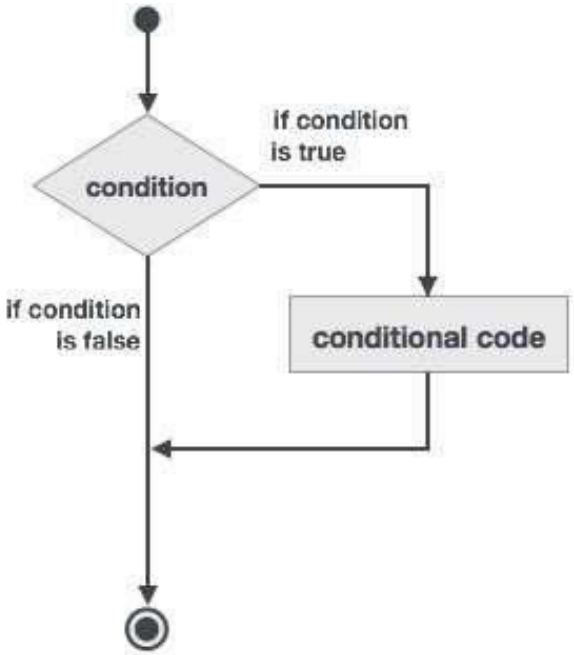
various statements

. . .

end if [name]

If the logical expression evaluates to true, then the block of code inside the if…then statement will be executed. If logical expression evaluates to false, then the first set of code after the end if statement will be executed.

Flow Diagram



Example 1

```
program ifProg
implicit none       ! local variable declaration
integer :: a = 10        ! check the logical condition using if statement
if (a < 20 ) then        ! if condition is true then print the following
print*, "a is less than 20"
end if
print*, "value of a is ", a
end program ifProg
```

When the above code is compiled and executed, it produces the following result:

```
a is less than 20
   value of a is        10
```

Example 2 This example demonstrates a named if block:

```
program markGradeA
implicit none
real :: marks         ! assign marks
marks = 90.4         ! use an if statement to give grade
gr: if (marks > 90.0) then
print *, " Grade A"
end if gr
end program markGradeA
```

When the above code is compiled and executed, it produces the following result:

```
Grade A
```

If… then… else Construct

An if… then statement can be followed by an optional else statement, which executes when the logical expression is false.

Syntax : The basic syntax of an if… then… else statement is:

if (logical expression) then

statement(s)

else

other_statement(s)

end if

However, if you give a name to the if block, then the syntax of the named if-else statement would be, like:

[name:] if (logical expression) then

various statements

. . .

else

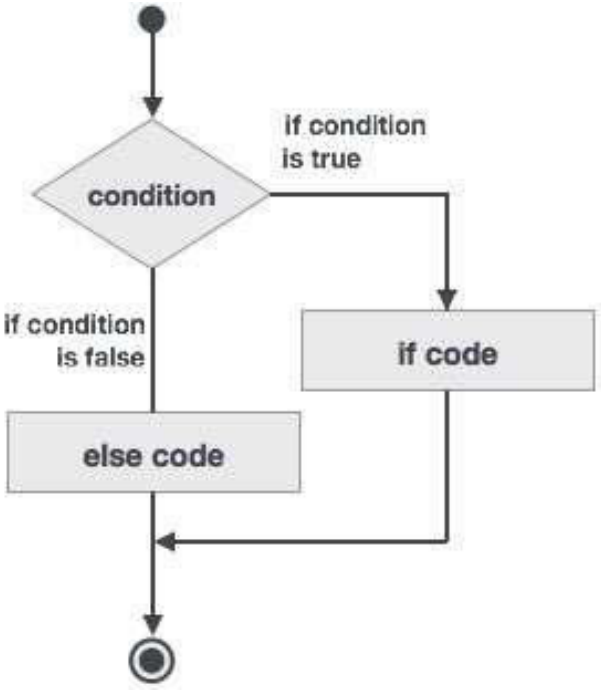
!other statement(s)

. . .

end if [name]

If the logical expression evaluates to true, then the block of code inside the if…then statement will be executed, otherwise the block of code inside the else block will be executed.

Flow Diagram



Example 3

```
program ifElseProg
implicit none          ! local variable declaration
integer :: a = 100          ! check the logical condition using if statement
 if (a < 20 ) then          ! if condition is true then print the following
print*, "a is less than 20"
else
print*, "a is not less than 20"
end if
print*, "value of a is ", a
end program ifElseProg
```

When the above code is compiled and executed, it produces the following result:

```
a is not less than 20
  value of a is     100
```

if...else if...else Statement

An if statement construct can have one or more optional else-if constructs. When the if condition fails, the immediately followed else-if is executed. When the else-if also fails, its successor else-if statement (if any) is executed, and so on. The optional else is placed at the end and it is executed when none of the above conditions hold true.

- All else statements (else-if and else) are optional.
- else-if can be used one or more times
- else must always be placed at the end of construct and should appear only once.

Syntax : The syntax of an if...else if...else statement is:

```
[name:]
if (logical expression 1) then
! block 1
else if (logical expression 2) then
! block 2
else if (logical expression 3) then
block 3
else
block 4
end if [name]
```

Example 4 :

```
program ifElseIfElseProg
implicit none    ! local variable declaration
integer :: a = 100      ! check the logical condition using if statement
 if( a == 10 ) then      ! if condition is true then print the following
print*, "Value of a is 10"
else if( a == 20 ) then       ! if else if condition is true
print*, "Value of a is 20"
else if( a == 30 ) then       ! if else if condition is true
print*, "Value of a is 30"
else         ! if none of the conditions is true
print*, "None of the values is matching"
end if
print*, "exact value of a is ", a
end program ifElseIfElseProg
```

When the above code is compiled and executed, it produces the following result:

```
None of the values is matching
exact value of a is 100
```

Nested If Construct

You can use one if or else if statement inside another if or else if statement(s).

Syntax : The syntax for a nested if statement is as follows:

```
if ( logical_expression 1) then
!Executes when the boolean expression 1 is true
...
if(logical_expression 2)then
! Executes when the boolean expression 2 is true
...
end if
end if
```

Example 5 :

```
program nestedIfProg
implicit none        ! local variable declaration
integer :: a = 100, b= 200        ! check the logical condition using if statement
if( a == 100 ) then        ! if condition is true then check the following
if( b == 200 ) then        ! if inner if condition is true
print*, "Value of a is 100 and b is 200"
end if
end if
print*, "exact value of a is ", a
print*, "exact value of b is ", b
end program nestedIfProg
```

When the above code is compiled and executed, it produces the following result:

```
Value of a is 100 and b is 200
  exact  value  of  a  is        100
  exact  value  of  b  is        200
```

Select Case Construct

A select case statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being selected on is checked for each select case.

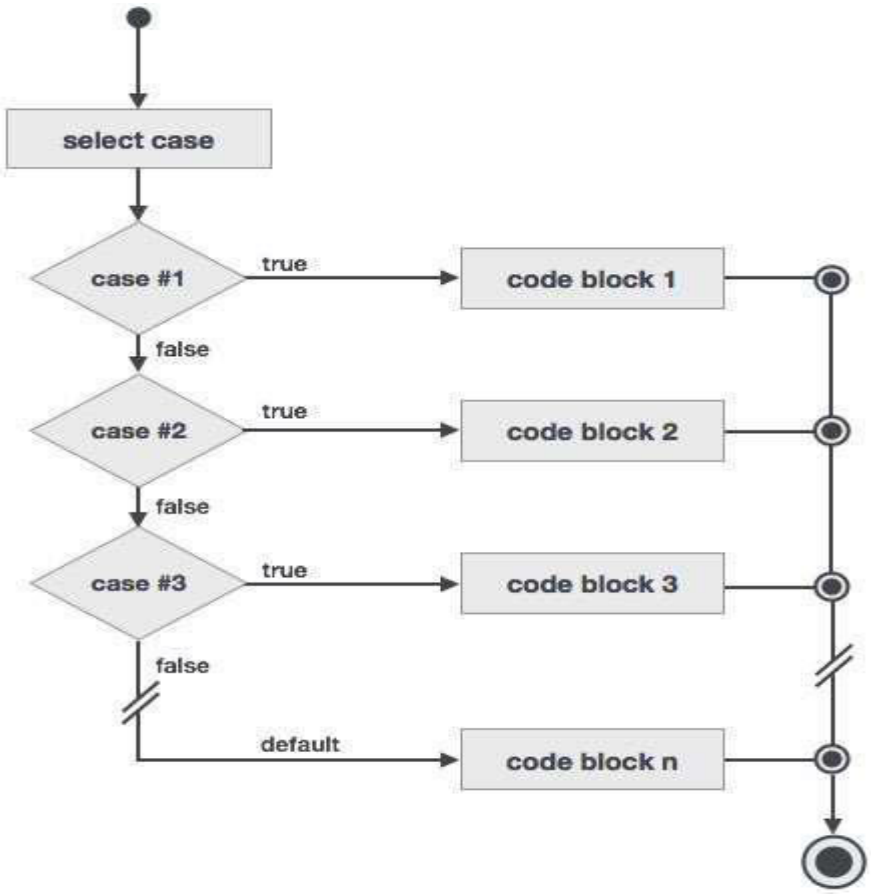
Syntax : The syntax for the select case construct is as follows:

```
[name:] select case (expression)
case (selector1)
! some statements
...        case (selector2)
other statements
...
case default
! more statements
...
end select [name]
```

The following rules apply to a select statement:

- The logical expression used in a select statement could be logical, character, or integer (but not real) expression.
- You can have any number of case statements within a select. Each case is followed by the value to be compared to and could be logical, character, or integer (but not real) expression and determines which statements are executed.
- The constant-expression for a case, must be the same data type as the variable in the select, and it must be a constant or a literal.
- When the variable being selected on, is equal to a case, the statements following that case will execute until the next case statement is reached.
- The case default block is executed if the expression in select case (expression) does not match any of the selectors.

Flow Diagram



Example 6 :

```
program selectCaseProg
implicit none          ! local variable declaration
character :: grade = 'B'
select case (grade)
case ('A')
print*, "Excellent!"
case ('B')
case ('C')
print*, "Well done"
case ('D')
print*, "You passed"
case ('F')
print*, "Better try again"
case default
print*, "Invalid grade"
end select
print*, "Your grade is ", grade
end program selectCaseProg
```

When the above code is compiled and executed, it produces the following result:

```
Your grade is B
```

Specifying a Range for the Selector

You can specify a range for the selector, by specifying a lower and upper limit separated by a colon:

case (low:high)

The following example demonstrates this:

Example 7 :

```
program selectCaseProg
implicit none         ! local variable declaration
integer :: marks = 78
select case (marks)
case (91:100)
print*, "Excellent!"
case (81:90)
print*, "Very good!"
case (71:80)
print*, "Well done!"
case (61:70)
print*, "Not bad!"
case (41:60)
print*, "You passed!"
case (:40)
print*, "Better try again!"
case default
print*, "Invalid marks"
end select
print*, "Your marks is ", marks
end program selectCaseProg
```

When the above code is compiled and executed, it produces the following result:

```
Well done!
 Your marks is          78
```

Nested Select Case Construct

You can use one select case statement inside another select case statement(s).

Syntax :

```
select case(a)
case (100)
print*, "This is part of outer switch", a
select case(b)
case (200)
print*, "This is part of inner switch", a end select
end select
```

Example 8 :

```
program nestedSelectCase
 ! local variable definition
integer :: a = 100
integer :: b = 200
select case(a)
case (100)
print*, "This is part of outer switch", a
select case(b)
case (200)
print*, "This is part of inner switch", a
end select
end select
print*, "Exact value of a is : ", a
print*, "Exact value of b is : ", b
end program nestedSelectCase
```

When the above code is compiled and executed, it produces the following result:

```
This is part of outer switch        100
This is part of inner switch        100
Exact value of a is :               100
Exact value of b is :               200
```