Branching (IF–THEN) Part 2 Outline

1. Branching (IF–THEN) Part 2 Outline
2. A Sequence of Statements to Execute When the IF Condition Fails
3. Flowchart for When the IF Condition Fails
4. The ELSE Clause
5. The Meaning of ELSE
6. Flowchart for IF – THEN – ELSE
7. IF - THEN - ELSE Example
8. IF - THEN - ELSE Example’s Flowchart
9. Indenting Inside IF Blocks
10. Multiple, Related Conditions
11. The ELSE IF Clause
12. ELSE IF Example
13. ELSE IF Example’s Flowchart
14. Mixing Branching Clauses
15. ELSE IF – ELSE Example
16. ELSE IF – ELSE Example’s Flowchart
17. Multiple ELSE IF Clauses
18. Multiple ELSE IF Clauses (continued)
19. Multiple ELSE IF Example
20. Multiple ELSE IF Example’s Flowchart
21. IF, Plus Multiple ELSE IF, Plus ELSE
22. IF, Plus Multiple ELSE IF, Plus ELSE Example
23. Nested IF Blocks
24. Nested IF Block Example
25. How Nested IF Blocks Work
26. Nested IF Indentation
27. Nested IF Block Example
28. Nested IF Block Example (continued)

See Programming in Fortran 90/95, 1st or 2nd ed, Chapter 12.

A Sequence of Statements to Execute When the IF Condition Fails

What if we have something that we want executed only when the LOGICAL expression in the IF–THEN condition fails? (That is, when it evaluates to .FALSE.)

Well, we could simply use another IF block:

```fortran
IF ((users_number < minimum_number) .OR. &
 & (users_number > maximum_number)) THEN
   PRINT *, "Hey! That’s not between ", &
   & minimum_number, " and ", &
   & maximum_number, "!
END IF

IF (.NOT. &
 & ((users_number < minimum_number) .OR. &
 & (users_number > maximum_number))) THEN
   PRINT *, "Woohoo! That’s between ", &
   & minimum_number, " and ", &
   & maximum_number, "!
END IF
```

But that’s kind of cumbersome. Plus, it increases the chances of a mistake, since we might mistype the second condition, or we might decide to change the first but forget to change the second.
The ELSE Clause

Often, we want to have not only a sequence of statements to execute in the event that the IF condition evaluates to .TRUE., but also a sequence of statements to execute in the event that the IF condition evaluates to .FALSE.

So, Fortran 90 (like most programming languages) allows us to set up a special group of statements within the IF block, called an ELSE clause:

```fortran
IF ((users_number < minimum_number) .OR. 
   & (users_number > maximum_number)) THEN 
   PRINT *, "Hey! That’s not between ", 
   & minimum_number, " and ", 
   & maximum_number, "!
ELSE 
   PRINT *, "Woohoo! That’s between ", 
   & minimum_number, " and ", 
   & maximum_number, "!
END IF
```

In such a case, the sequence of statements that execute when the IF condition evaluates to .TRUE. is called the IF clause, and the sequence of statements that execute when the IF condition evaluates to .FALSE. is called the ELSE clause.

Notice that the ELSE statement does not have a condition of its own: it’s simply the keyword ELSE, with its condition implied by the IF statement. That is, the ELSE clause’s condition is the opposite of the IF clause’s condition.

Also notice that the presence of the ELSE clause guarantees that at exactly one of the clauses of this IF block will be executed.
The Meaning of ELSE

IF ((users_number < minimum_number) .OR. (users_number > maximum_number)) THEN
  PRINT *, "Hey! That's not between ", minimum_number, " and ", maximum_number, "!"
ELSE
  PRINT *, "Woohoo! That's between ", minimum_number, " and ", maximum_number, "!"
END IF

The statements inside the IF clause are executed if and only if the condition in the IF-THEN statement evaluates to .TRUE.

By contrast, the statements inside the ELSE clause are executed if and only if the IF condition evaluates to .FALSE.

So, in programming, the word ELSE is used to mean “otherwise.”

Thus, exactly one of these two clauses will be executed.

Notice that there’s only one END IF statement for the whole set of clauses. Thus, the IF block includes both the IF clause and the ELSE clause.

Again, regardless of the value of the LOGICAL expression in the IF-THEN statement’s condition, the statements after the END IF statement are executed.
**IF – THEN – ELSE Example**

```fortran
PROGRAM is_less_else
IMPLICIT NONE
INTEGER :: computers_number = 5
INTEGER :: users_number
PRINT *, "Pick an integer."
READ *, users_number
IF (users_number < computers_number) THEN
  PRINT *, "That’s unbelievable! Your number is"
  PRINT *, "less than mine!"
  PRINT *, "Well, okay, maybe it’s believable."
ELSE
  PRINT *, "Wow, you picked a number that isn’t"
  PRINT *, "less than mine. Good work!"
END IF
PRINT *, "And now I’m sick of you."
PRINT *, "Bye!"
END PROGRAM is_less_else
```

**IF–THEN–ELSE Example’s Flowchart**

1. **Start**
2. Prompt for user’s number.
3. Input user’s number.
4. User’s number < computer’s?
5. True
   - Output less.
6. False
   - Output not less.
7. Output goodbye.
8. **End**
**Indenting Inside IF Blocks**

The IF statement, the ELSE statement and the END IF statement are indented the same amount as other statements — e.g., declarations, PRINT statements — that are between the PROGRAM statement and the END PROGRAM statement.

However, the statements inside an IF clause or inside an ELSE clause are indented some extra space.

Specifically, however much the IF statement is indented past the PROGRAM statement, the statements inside the IF block — that is, not including the IF statement, the ELSE statement and the END IF statement — are indented that much again.

So, in CS1313 programming projects, the IF statement, the ELSE statement and the END IF statement should be indented four spaces farther than the PROGRAM statement and the END PROGRAM statement, but the statements inside the IF clause and the ELSE clause should be indented eight spaces farther than the PROGRAM statement and the END PROGRAM statement (i.e., 4 + 4).

```fortran
IF ((users_number < minimum_number) .OR. users_number > maximum_number) THEN
  PRINT *, "Hey! That's not between ", minimum_number, " and ", maximum_number, "!
END IF

ELSE
  PRINT *, "Woohoo! That's between ", minimum_number, " and ", maximum_number, "!
END IF
```

**Multiple, Related Conditions**

What if we have multiple, related conditions and we want to be able to handle each?

Well, we could simply use multiple IF blocks:

```fortran
IF ((users_number < minimum_number) .OR. &
   users_number > maximum_number)) THEN
  PRINT *, "Hey! That’s not between ", minimum_number, " and ", maximum_number, "!
END IF

IF (ABS(users_number - computers_number) <= &
   close_distance) THEN
  PRINT *, "Close, but no cigar."
END IF
```

That’s not too cumbersome.

But notice that there’s a case where both PRINT statements might be executed: in the event that both

- `users_number` is less than `minimum_number`, and
- `users_number` is within `close_distance` of `computers_number`.

In that case, both outputs will be printed, which is not what we want; we want either to be told that we’re outside the range, or to be told that we’re close. We definitely don’t want both.
The ELSE IF Clause

Fortran 90 allows us to set up another special clause of statements attached to the first IF clause, called an ELSE IF clause:

```
IF ((users_number < minimum_number) .OR. &
   & (users_number > maximum_number)) THEN
   PRINT *, "Hey! That’s not between ", &
   & minimum_number, " and ", &
   & maximum_number, "!
ELSE IF (ABS(users_number - computers_number) <= &
   & close_distance) THEN
   PRINT *, "Close, but no cigar."
END IF
```

As usual, the statements inside the IF clause are executed if and only if the condition in the IF-THEN statement evaluates to .TRUE.

By contrast, the statements inside the ELSE IF clause are executed if and only if both of the following occur:

1. The IF condition evaluates to .FALSE. and
2. the ELSE IF condition evaluates to .TRUE.

Note: in the case that the IF condition evaluates to .TRUE., it is also the case that the ELSE IF condition isn’t evaluated at all. Why? Because in that case the statements inside the ELSE IF clause will be skipped regardless of the value of the ELSE IF condition, so the evaluation of the ELSE IF condition would be irrelevant. Why do work that isn’t going to help?

By the way, notice that it could be the case that no clause of this IF block gets executed, if both conditions evaluate to .FALSE.

ELSE IF Example

```
% cat islesselseif.f90
PROGRAM is_less_else_if
IMPLICIT NONE
INTEGER :: computers_number = 5
INTEGER :: users_number
PRINT *, "Pick an integer:
READ *, users_number
IF (users_number < computers_number) THEN
   PRINT *, "That’s unbelievable! Your number is" &
   PRINT *, " less than mine!"
ELSE IF (users_number > computers_number) THEN
   PRINT *, "Well, okay, maybe it’s believable."
END IF
PRINT *, "Surprise, surprise! Your number is" &
PRINT *, " greater than mine!"
END IF
PRINT *, "And now I’m sick of you."
PRINT *, "Bye!"
END PROGRAM is_less_else_if
% f95 --o islesselseif islesselseif.f90
% islesselseif
Pick an integer:
4
That’s unbelievable! Your number is less than mine!
Well, okay, maybe it’s believable.
And now I’m sick of you.
Bye!
% islesselseif
Pick an integer:
5
And now I’m sick of you.
Bye!
% islesselseif
Pick an integer:
6
Surprise, surprise! Your number is greater than mine!
And now I’m sick of you.
Bye!
```
**ELSE IF Example’s Flowchart**

```plaintext
PRINT *, "Pick an integer:"
READ *, users_number
IF (users_number < computers_number) THEN
  PRINT *, "That’s unbelievable! Your number is"
  PRINT *, "  less than mine!"
ELSE IF (users_number > computers_number) THEN
  PRINT *, "Well, okay, maybe it’s believable."
ELSE IF (users_number == computers_number) THEN
  PRINT *, "That’s amazing!"
ELSE
  PRINT *, "Bzzzt! Not even close."
END IF
END IF
PRINT *, "And now I’m sick of you."
PRINT *, "Bye!"
```

**Mixing Branching Clauses**

Not only can we have an ELSE IF clause, we can also have an ELSE clause as well, as the final clause of the entire IF block.

```plaintext
IF ((users_number < minimum_number) .OR. &
    users_number > maximum_number)) THEN
  PRINT *, "Hey! That’s not between ", &
  minimum_number, " and ", &
  maximum_number, "!
ELSE IF (users_number == computers_number) THEN
  PRINT *, "That’s amazing!"
ELSE
  PRINT *, "Bzzzt! Not even close."
END IF
```

The statements inside the ELSE clause are executed if and only if the IF condition and the ELSE IF condition both evaluate to .FALSE.

Notice that the presence of the ELSE clause guarantees that at exactly one of the clauses of this IF block will be executed. If the ELSE clause were absent, then it might be that no clause is executed, if both of the conditions evaluated to .FALSE.

Again, notice that there’s only one END IF statement for this whole set of clauses.

Also, notice that the indenting rules that apply to IF clauses and ELSE clauses also apply to ELSE IF clauses.
ELSE IF - ELSE Example

% cat islesselseifelse.f90
PROGRAM is_less_else_if_else
  IMPLICIT NONE
  INTEGER :: computers_number = 5
  INTEGER :: users_number
  PRINT *, "Pick an integer:
  READ *, users_number
  IF (users_number < computers_number) THEN
    PRINT *, "That's unbelievable! Your number is"
    PRINT *, "less than mine!"
    PRINT *, "Well, okay, maybe it's believable."
    ELSE IF (users_number > computers_number) THEN
      PRINT *, "Surprise, surprise! Your number is"
      PRINT *, "greater than mine!"
    ELSE
      PRINT *, "Yowza! Your number is equal to mine!"
    END IF
    PRINT *, "And now I'm sick of you."
    PRINT *, "Bye!"
  END IF
END PROGRAM is_less_else_if_else

% f95 -o islesselseifelse islesselseifelse.f90
% islesselseifelse
Pick an integer:
4
That’s unbelievable! Your number is less than mine!
Well, okay, maybe it’s believable.
And now I’m sick of you.
Bye!

% islesselseifelse
Pick an integer:
5
Yowza! Your number is equal to mine!
And now I’m sick of you.
Bye!

% islesselseifelse
Pick an integer:
6
Surprise, surprise! Your number is greater than mine!
And now I’m sick of you.
Bye!

ELSE IF - ELSE Example’s Flowchart

Prompt for user’s number.

Input user’s number.

user’s < computer’s?
  Output less.
  True

user’s > computer’s?
  Output greater.
  False

user’s = computer’s?
  Output equal.
  False

Output goodbye.
Multiple ELSE IF Clauses

We don’t have to stop at just one ELSE IF clause; we can have as many as we like:

```fortran
IF ((users_number < minimum_number) .OR. 
    & (users_number > maximum_number)) THEN 
    PRINT *, "Hey! That’s not between ", 
    & minimum_number, " and ", 
    & maximum_number, "!"
ELSE IF (users_number == computers_number) THEN 
    PRINT *, "That’s amazing!"
ELSE IF (ABS(users_number - computers_number) <= 
          & close_distance) THEN 
    PRINT *, "Close, but no cigar."
END IF
```

As usual, the statements inside the IF clause are executed if and only if the condition (a LOGICAL expression completely enclosed in parentheses) in the IF-THEN statement evaluates to .TRUE.

Also as usual, the statements inside the first ELSE IF clause are executed if and only if both of the following occur:

1. The IF condition evaluates to .FALSE.
2. The first ELSE IF condition evaluates to .TRUE.

As for the second ELSE IF clause, its statements are executed if and only if all of the following occur:

1. The IF condition evaluates to .FALSE.
2. All prior ELSE IF conditions within the entire IF block (if there are any) evaluate to .FALSE.
3. The given ELSE IF condition evaluates to .TRUE.

Again, the conditions (LOGICAL expressions completely enclosed in parentheses) in the IF-THEN and ELSE-IF-THEN statements are evaluated until one of them results in .TRUE.; the conditions in subsequent ELSE-IF-THEN statements within the IF block are skipped.

Multiple ELSE IF Clauses (continued)

The statements inside the second ELSE IF clause are executed if and only if all of the following occur:

1. The IF condition evaluates to .FALSE., and
2. The first ELSE IF condition evaluates to .FALSE., and
3. The second ELSE IF condition evaluates to .TRUE.

We can generalize this principle for an arbitrary number of ELSE IF clauses.

General Rule for Multiple ELSE IF Clauses

For a given ELSE IF clause, the statements inside it are executed if and only if all of the following occur:

1. The IF condition evaluates to .FALSE., and
2. All prior ELSE IF conditions within the entire IF block (if there are any) evaluate to .FALSE., and
3. The given ELSE IF condition evaluates to .TRUE.

As for the second ELSE IF clause, its statements are executed if and only if all of the following occur:
Multiple ELSE IF Example

% cat islesselseifs.f90
program is_less_else_ifs
implicit none
integer :: computers_number = 5
integer :: users_number
print *, "Pick an integer:"ead *, users_number
if (users_number < computers_number) then
print *, "That's unbelievable!  Your number is" print *, "less than mine!"
else if (users_number > computers_number) then
print *, "Surprise, surprise!  Your number is"
else if (users_number == computers_number) then
end if
end program is_less_else_ifs

% f95 -o islesselseifs islesselseifs.f90
% islesselseifs

Pick an integer:
4
That's unbelievable!  Your number is
less than mine!
Well, okay, maybe it's believable.
And now I'm sick of you.
Bye!
% islesselseifs
Pick an integer:
5
Yowza!  Your number is equal to mine!
And now I'm sick of you.
Bye!
% islesselseifs
Pick an integer:
6
Surprise, surprise!  Your number is
greater than mine!
And now I'm sick of you.
Bye!

Multiple ELSE IF Example’s Flowchart

Prompt for user's number.
Input user's number.

user's < computer's? True
False

user's > computer's? True
False

user's == computer's? False
True

Output less.
Output greater.
Output equal.

Output goodbye.
IF, Plus Multiple ELSE IF, Plus ELSE

Not surprisingly, we can not only have as many ELSE IF clauses as we like, we can also have an ELSE clause as well, as the final clause of the entire IF block.

```
IF ((users_number < minimum_number) .OR. &
& (users_number > maximum_number)) THEN
  PRINT *, "Hey! That's not between ", &
  minimum_number, " and ", &
  maximum_number, "!
ELSE IF (users_number == computers_number) THEN
  PRINT *, "That's amazing!"
ELSE IF (ABS(users_number - computers_number) <= &
  close_distance) THEN
  PRINT *, "Close, but no cigar."
ELSE
  PRINT *, "Bzzzt! Not even close."
END IF
```

The statements inside the ELSE clause are executed if and only if the IF condition and all of the ELSE IF conditions in the block evaluate to .FALSE.

Again, the ELSE clause guarantees that exactly one of the clauses of this IF block will be executed. If the ELSE clause were absent, and all of the IF block’s conditions evaluated to .FALSE., then no clause would be executed.

As usual, notice that there’s only one END IF statement for this whole set of clauses.

---

Example

```
% cat islesselseifselse.f90
PROGRAM is_less_else_ifs_else
IMPLICIT NONE
INTEGER :: computers_number = 5
INTEGER :: users_number
PRINT *, "Pick an integer:"
READ *, users_number
IF (users_number < computers_number) THEN
  PRINT *, "That's unbelievable! Your number is" PRINT *, " less than mine!"
  PRINT *, "Well, okay, maybe it's believable."
ELSE IF (users_number > computers_number) THEN
  PRINT *, "Surprise, surprise! Your number is" PRINT *, " greater than mine!"
ELSE IF (users_number == computers_number) THEN
  PRINT *, "Yowza! Your number is equal to mine!"
ELSE
  PRINT *, "This statement will never be executed."
  PRINT *, "Why?"
END IF
PRINT *, "And now I'm sick of you."
PRINT *, "Bye!"
END PROGRAM is_less_else_ifs_else
```

```
f95 --islesselseifselse islesselseifselse.f90
% f95 --islesselseifselse islesselseifselse.f90
% islesselseifselse
Pick an integer:
4
That's unbelievable! Your number is less than mine!
Well, okay, maybe it's believable. And now I'm sick of you.
Bye!
% islesselseifselse
Pick an integer:
5
Yowza! Your number is equal to mine!
And now I'm sick of you.
Bye!
% islesselseifselse
Pick an integer:
6
Surprise, surprise! Your number is greater than mine!
And now I'm sick of you.
Bye!
```
Nested IF Blocks

Inside each clause of an IF block, we can nest more IF blocks:

```
IF ( condition ) THEN
    statement
    IF ( condition ) THEN
        statement
    ELSE IF ( condition ) THEN
        statement
    ELSE
        statement
    END IF
ENDIF
```

23

Nested IF Block Example

```
PROGRAM nested_if
IMPLICIT NONE
INTEGER,PARAMETER :: minimum_number = 1
INTEGER,PARAMETER :: maximum_number = 10
INTEGER,PARAMETER :: computers_number = 5
INTEGER,PARAMETER :: close_distance = 1
INTEGER :: users_number
PRINT *, "I'm thinking of a number between ", &
   minimum_number, " and ", maximum_number, "."
PRINT *, "What number am I thinking of?"
READ *, users_number
IF ((users_number < minimum_number) .OR. &
   (users_number > maximum_number)) THEN
    PRINT *, "Hey! That's not between ", &
   minimum_number, " and ", maximum_number, ":"
ELSE IF (users_number == computers_number) THEN
    PRINT *, "That's amazing!"
ELSE
    PRINT *, "Well, at least you were ", &
   "within the range"
    IF (ABS(users_number - computers_number) <= &
       close_distance) THEN
        PRINT *, "and you were close!"
    ELSE IF (users_number < computers_number) THEN
        PRINT *, "but you were way too low."
    ELSE
        PRINT *, "but you were way too high."
    END IF
    PRINT *, "My number was ", computers_number, "."
END IF
END PROGRAM nested_if
```

24
How Nested IF Blocks Work

Suppose that an IF block is nested inside another IF block. What will happen?

Well, a statement inside a clause of an IF block is executed if and only if the clause’s condition evaluates to .TRUE. and all prior conditions within the IF block evaluate to .FALSE. (or, in the case that the clause is an ELSE clause, it is executed if and only if all of the IF block’s conditions evaluate to .FALSE.).

On the other hand, an IF-THEN statement is a normal executable statement (more or less).

So, the inner IF-THEN statement will be reached, and therefore executed, if and only if the outer clause that contains it has a condition that evaluates to .TRUE. and if all of the outer IF block’s prior clauses have conditions that evaluate to .FALSE. — or, in the case of an outer ELSE clause, if all of the conditions of the outer IF block’s prior clauses evaluate to .FALSE.

Once the inner IF block is reached, it will be executed exactly like any other IF block.

Nested IF Indentation

Notice that the statements inside the nested IF blocks are indented several extra spaces, so that it’s obvious which statements go with which blocks.

In CS1313 programming projects, statements should be indented an extra four spaces for each block that they are inside.

We’ll see later that this rule applies not only to IF blocks but to other kinds of blocks as well (e.g., DO loops).
Nested IF Block Example

PROGRAM nested_if
IMPLICIT NONE
INTEGER,PARAMETER :: minimum_number = 1
INTEGER,PARAMETER :: maximum_number = 10
INTEGER,PARAMETER :: computers_number = 5
INTEGER,PARAMETER :: close_distance = 1
INTEGER :: users_number
PRINT *, "I’m thinking of a number between ", &
 minimum_number, " and ", maximum_number, "."
PRINT *, "What number am I thinking of?"
READ *, users_number
IF ((users_number < minimum_number) .OR. &
 (users_number > maximum_number)) THEN
 PRINT *, "Hey! That’s not between ", &
 minimum_number, " and ", maximum_number, "."
ELSE IF (users_number == computers_number) THEN
 PRINT *, "That’s amazing!"
ELSE
 PRINT *, "Well, at least you were within the range"
 "but you were way too low."
IF (ABS(users_number - computers_number) <= &
 close_distance) THEN
 PRINT *, "and you were close!"
ELSE IF (users_number < computers_number) THEN
 PRINT *, "but you were way too high."
ELSE
 PRINT *, "but you were way too high."
END IF
PRINT *, "My number was ", computers_number, "."
END IF
END PROGRAM nested_if

Nested IF Block Example (continued)

% f95 -o nestedif nestedif.f90
% nestedif
I’m thinking of a number between 1 and 10 .
What number am I thinking of?
0
Hey! That’s not between 1 and 10 !
% nestedif
I’m thinking of a number between 1 and 10 .
What number am I thinking of?
11
Hey! That’s not between 1 and 10 !
% nestedif
I’m thinking of a number between 1 and 10 .
What number am I thinking of?
1
Well, at least you were within the range
 but you were way too low.
My number was 5 .
% nestedif
I’m thinking of a number between 1 and 10 .
What number am I thinking of?
9
Well, at least you were within the range
 but you were way too high.
My number was 5 .
% nestedif
I’m thinking of a number between 1 and 10 .
What number am I thinking of?
4
Well, at least you were within the range
 and you were close!
My number was 5 .
% nestedif
I’m thinking of a number between 1 and 10 .
What number am I thinking of?
5
That’s amazing!