Branching (IF–THEN) Part 1 Outline

1. Branching (IF–THEN) Part 1 Outline
2. Branching with IF–THEN
3. IF–THEN Flowchart
4. The Meaning of IF–THEN
5. The Meaning of IF–THEN (continued)
6. IF–THEN Example
7. IF–THEN Example’s Flowchart
8. The Condition Is a LOGICAL Expression Completely Enclosed in Parentheses
9. The Condition Is a LOGICAL Expression Completely Enclosed in Parentheses (continued)
10. The Condition That Is NOT Completely Enclosed in Parentheses
    Is WRONG WRONG WRONG
11. The Condition That Is NOT Completely Enclosed in Parentheses
    Is WRONG WRONG WRONG: Example
12. The Condition That IS Completely Enclosed in Parentheses
    Is Correct: Example
13. What Kind of Statements Can Go Inside an IF Block?
14. IF Block Example
15. IF Block Example (continued)

See Programming in Fortran 90/95, 1st or 2nd ed, Chapter 12.
Branching with **IF–THEN**

*Branching* is a way to *select* between possible sets of statements.

In Fortran 90, the most common kind of branching is the *IF block*:

```
IF ( condition ) THEN
   statement1
   statement2
   ...
END IF
```

Notes:

1. The condition is a **LOGICAL** expression, so it evaluates either to **.TRUE.** or to **.FALSE.**
2. The **LOGICAL** expression that constitutes the condition **MUST** be completely enclosed inside parentheses.
**IF–THEN Flowchart**

```
statement_before
IF ( condition ) THEN
  statement_inside1
  statement_inside2
  ...
END IF
statement_after
```

A **diamond** denotes a branch.
The Meaning of IF–THEN

In my_number.f90, we saw something like this:

```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
```

What does this mean?

First, the condition

```fortran
    ((users_number < minimum_number) .OR. &
    & (users_number > maximum_number))
```

(which is a LOGICAL expression completely enclosed in parentheses) is evaluated, resulting in either .TRUE. or .FALSE.

Next, if and only if the condition evaluates to .TRUE., then the sequence of statements inside the IF block — that is, between the IF–THEN statement and the END IF statement — are executed in order; otherwise, these statements are skipped.

Finally, regardless of whether the condition evaluates to .TRUE. or .FALSE., execution picks up at the next statement immediately after the END IF statement, and continues along from there.
Consider this code fragment:

```
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 the condition

```
((users_number < minimum_number) .OR. &
& (users_number > maximum_number))
```

evaluates to .TRUE. — that is, if it’s the case that `users_number` is less than `minimum_number` or it’s the case that `users_number` is greater than `maximum_number` — then the statement

```
PRINT *, "Hey! That’s not between ", &
&    minimum_number, " and ", &
&    maximum_number, "!
```

is executed, in which case

```
Hey! That’s not between 1 and 10!
```

is output.

On the other hand, if `users_number` lies between `minimum_number` and `maximum_number` — that is, if the condition evaluates to .FALSE. — then the PRINT statement is not executed, and so no such output is produced.
IF–THEN  Example

% cat isless.f90
PROGRAM is_less
    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."
    END IF
    PRINT *, "And now I’m sick of you."
    PRINT *, "Bye!"
END PROGRAM is_less

% f95 -o isless isless.f90
% isless
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!
% isless
Pick an integer:
5
And now I’m sick of you.
Bye!
% isless
Pick an integer:
6
And now I’m sick of you.
Bye!
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."
END IF

PRINT *, "And now I’m sick of you."
PRINT *, "Bye!"

```
IF-THEN Example's Flowchart

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."
END IF
PRINT *, "And now I’m sick of you."
PRINT *, "Bye!"
```
The Condition Is a LOGICAL Expression Completely Enclosed in Parentheses

Consider this 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, "!
END IF
```

Notice that the condition

```plaintext
((users_number < minimum_number) .OR. &
& (users_number > maximum_number))
```

is a LOGICAL expression completely enclosed in parentheses. How do we know this?

We know that the expressions

```plaintext
users_number < minimum_number
```

and

```plaintext
users_number > maximum_number
```

are relational expressions, because each of them has two numeric operands and a relational operator (the “less than” operator and the “greater than” operator, respectively). Relational expressions are one of the categories of LOGICAL expressions, so each of these relational expressions is, by definition, a LOGICAL expression.
The Condition Is a LOGICAL Expression Completely Enclosed in Parentheses
(continued)

We also know that the expression

\[(\text{users\_number} < \text{minimum\_number}) \ \text{.OR.} \ \& \ \& \ (\text{users\_number} > \text{maximum\_number})\]

is a LOGICAL expression, because it is an expression with two LOGICAL operands (the two relational expressions, above) and a LOGICAL operator (the .OR. operator).

Given that the above expression is a LOGICAL expression, then if we enclose it in parentheses, we have a LOGICAL expression completely enclosed in parentheses:

\[((\text{users\_number} < \text{minimum\_number}) \ \text{.OR.} \ \& \ \& \ (\text{users\_number} > \text{maximum\_number}))\]

Therefore, the expression immediately above, which is a LOGICAL expression completely enclosed in parentheses, can be used as the condition of an IF–THEN statement:

IF \[((\text{users\_number} < \text{minimum\_number}) \ \text{.OR.} \ \& \ \& \ (\text{users\_number} > \text{maximum\_number}))\] THEN
  PRINT *, "Hey! That’s not between ", &
  & minimum\_number, " and ", &
  & maximum\_number, "!"
END IF
A Condition That Is NOT Completely Enclosed in Parentheses Is WRONG WRONG WRONG

If we wanted to use a condition that was a perfectly valid LOGICAL expression, but the LOGICAL expression was NOT completely enclosed in parentheses, then we could not use that expression as the condition of an IF–THEN statement.

Therefore, the IF–THEN statement below is invalid, and the compiler will flag it as an error:

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

The above IF–THEN statement is WRONG WRONG WRONG, as we’ll see in an example on the next page.
A Condition That Is NOT Completely Enclosed in Parentheses Is WRONG WRONG WRONG:

Example

In the example below, the condition is a valid LOGICAL expression, but it is NOT completely enclosed in parentheses, which is WRONG WRONG WRONG:

```
% cat condnotenclosed.f90
PROGRAM condition_not_enclosed
  IMPLICIT NONE
  INTEGER,PARAMETER :: minimum_number = 1
  INTEGER,PARAMETER :: maximum_number = 10
  INTEGER :: users_number = 0
  IF (users_number < minimum_number) .OR. &
    (users_number > maximum_number) THEN
    PRINT *, "Hey! That's not between ", &
    minimum_number, " and ", &
    maximum_number, "!"
  END IF !! (users_number < minimum_number) ...
END PROGRAM condition_not_enclosed
% f95 -o condnotenclosed condnotenclosed.f90
Error: condnotenclosed.f90, line 6:
  syntax error detected at )@.OR.
***Malformed statement
Error: condnotenclosed.f90, line 11:
  Unexpected ENDIF statement
detected at END IF@<end-of-statement>
[f95 terminated - errors found by pass 1]
```

Notice that the compiler flags the IF–THEN statement as an error.

The compiler also flags the END IF statement as an error, because it there’s no VALID IF–THEN statement associated with it.
A Condition That IS Completely Enclosed in Parentheses Is Correct: Example

In the example below, the condition is a valid LOGICAL expression, and it IS completely enclosed in parentheses, which is correct:

```fortran
% cat condenclosed.f90
PROGRAM condition_enclosed
  IMPLICIT NONE
  INTEGER,PARAMETER :: minimum_number = 1
  INTEGER,PARAMETER :: maximum_number = 10
  INTEGER :: users_number = 0
  IF ((users_number < minimum_number) .OR. &
      (users_number > maximum_number)) THEN
    PRINT *, "Hey! That’s not between ", &
    minimum_number, " and ", &
    maximum_number, "!
  END IF !! (users_number < minimum_number) ...
END PROGRAM condition_enclosed
% f95 -o condenclosed condenclosed.f90
% condenclosed
Hey! That’s not between 1 and 10 !
```

Notice that the ONLY difference between the above example and the example on the previous page is whether the condition’s LOGICAL expression is completely enclosed in parentheses. If it is, then the program compiles; if it isn’t, then the program doesn’t compile.
What Kind of Statements Can Go Inside an IF Block?

Between the IF–THEN statement and the END IF statement, there can be any kind of executable statements, and any number of them. For example:

- PRINT statements;
- READ statements;
- assignment statements;
- IF–THEN statements.

There are other kinds of executable statements that can occur inside an IF block, some of which we’ll learn later in the semester.

In the event that the IF condition evaluates to .TRUE., these statements will be executed one by one, in the order in which they appear in the IF block.

Notice that an IF block CANNOT contain declaration statements, because the IF–THEN statement is an executable statement, and ALL declarations MUST come before ANY executable statements.
IF Block Example

PROGRAM absolute_value_by_type
 IMPLICIT NONE
 INTEGER,PARAMETER :: integer_code = 1, real_code = 2
 REAL :: real_input_value, real_output_value
 INTEGER :: integer_input_value, integer_output_value
 INTEGER :: data_type_code
 PRINT *, "I’m going to calculate the absolute ", &
 "value of a number"
 PRINT *, "that you input."
 PRINT *, "Would you like to input an INTEGER ", &
 "or a REAL?"
 PRINT *, (Enter ", integer_code, "for an ", &
 "INTEGER or ", real_code, " for a REAL.)"
 READ *, data_type_code
 IF ((data_type_code /= integer_code) .AND. &
 (data_type_code /= real_code)) THEN
 PRINT *, "ERROR: I don’t recognize the ", &
 "data type code ", data_type_code, "."
 STOP
 END IF !!(data_type_code /= integer_code) ...
 IF (data_type_code == integer_code) THEN
 PRINT *, "Please input the INTEGER."
 READ *, integer_input_value
 IF (integer_input_value < 0) THEN
 integer_output_value = &
 -integer_input_value
 ELSE !! (integer_input_value < 0)
 integer_output_value = &
 +integer_input_value
 END IF !!(integer_input_value < 0)ELSE
 PRINT *, "The absolute value of ", &
 integer_input_value, " is ", &
 integer_output_value, "."
 END IF !!(integer_input_value < 0)
 ELSE
 IF (data_type_code == real_code) THEN
 PRINT *, "Please input the REAL."
 READ *, real_input_value
 IF (real_input_value < 0) THEN
 real_output_value = &
 -real_input_value
 ELSE !! (real_input_value < 0)
 real_output_value = &
 +real_input_value
 END IF !!(real_input_value < 0)ELSE
 PRINT *, "The absolute value of ", &
 real_input_value, " is ", &
 real_output_value, "."
 END IF !!(real_input_value < 0)
 END IF !!(data_type_code == real_code)
 END PROGRAM absolute_value_by_type
IF Block Example (continued)

```bash
% f95 -o absvalbytype absvalbytype.f90
% absvalbytype
I’m going to calculate the absolute value of a number that you input.
Would you like to input an INTEGER or a REAL?
   (Enter 1 for an INTEGER or 2 for a REAL.)
0
ERROR: I don’t recognize the data type code 0.
% absvalbytype
I’m going to calculate the absolute value of a number that you input.
Would you like to input an INTEGER or a REAL?
   (Enter 1 for an INTEGER or 2 for a REAL.)
1
Please input the INTEGER.
-5
The absolute value of -5 is 5.
% absvalbytype
I’m going to calculate the absolute value of a number that you input.
Would you like to input an INTEGER or a REAL?
   (Enter 1 for an INTEGER or 2 for a REAL.)
1
Please input the INTEGER.
5
The absolute value of 5 is 5.
% absvalbytype
I’m going to calculate the absolute value of a number that you input.
Would you like to input an INTEGER or a REAL?
   (Enter 1 for an INTEGER or 2 for a REAL.)
2
Please input the REAL.
-0.75
The absolute value of -0.7500000 is 0.7500000.
% absvalbytype
I’m going to calculate the absolute value of a number that you input.
Would you like to input an INTEGER or a REAL?
   (Enter 1 for an INTEGER or 2 for a REAL.)
2
Please input the REAL.
0.75
The absolute value of 0.7500000 is 0.7500000.
```