- Simple IF-THEN-ENDIF

Syntax

```
IF (condition) THEN
    { Block of code to execute
    } if condition is true
ENDIF
```

- Really simple IF-THEN
  - if the body of the IF block consists of only 1 line, you can do

```
IF (condition)  statement
```

Example

```
IF (j > 5)  i = -2
```

This is the same as

```
IF (j > 5) THEN
    i = -2
ENDIF
```

- I don't use this syntax

Nesting of IF - THEN structures

- The bodies of an IF - THEN structure can contain other IF - THEN structures (nesting)
- You can nest any IF - THEN structure within any other structure.

Example

```
IF (cond 1) THEN
    { Block to execute if
      cond 1 is true
ELSEIF (cond 2) THEN
    IF (cond 3) THEN
        { Block to execute if
          cond 1 is false, cond 2 true, cond 3 true
    ELSE
        { Block to execute if
          cond 1 false, cond 2 true, cond 3 false
    ENDIF
ENDIF
```

- indenting is important because it allows you to easily determine what level you are in.

* The "inner IF - THEN" structure must begin and end completely within the block it starts in

* You can nest up to 7 levels deep

```
IF (cond 1) THEN
    IF (cond 2) THEN
        IF (cond 3) THEN
            .
            .
            .
        ENDIF
    ENDIF
ENDIF
```

Example

Given x, y, compute

$$f(x,y) = \begin{cases} \cos(x)\sin(y) & x \geq 0,\ y \geq 0 \\ \sin(x)\cos(y) & x \geq 0,\ y < 0 \\ \cos(x)\cos(y) & x < 0,\ y \geq 0 \\ \sin(x)\sin(y) & x < 0,\ y < 0 \end{cases}$$

Compound Testing

- The conjunction operators allow you to do more complex testing in a single statement

cond 1 .AND. cond 2	→ True if cond 1 and cond 2 are both true
cond 1 .OR. cond 2	→ True if cond 1 and/or cond 2 is true
.NOT. Cond 1	→ False if cond 1 is true and vice versa
cond 1 .EQV. cond 2	→ True if cond 1, 2 are both True or both false
cond 1 .NEQV. cond 2	→ True if cond 1, 2 are opposites

Modifications to order of operations

- ( ) first, (left to right)
- ** (left to right)
- *, / (left to right)
- +, − (left to right)
- Relational (<, <=, ==, /=, >, >=) (left to right)
- .NOT. (right to left)
- .AND. (left to right)
- .OR. (left to right)
- .EQV., .NEQV. (left to right)

<u>Examples</u>

i = 1
j = 2
k = −1

(i < 0) .AND. (j < 7) = False .AND. True
= False

parentheses are optional, but greatly improve readability

(i > 0) .OR. (~~j > 7~~) = True .OR. ~~True~~ False
False = True

.NOT.(i > 2) .OR. ((j < 3) .OR. (K > 0))

.NOT. False True .OR. False

True .OR TRue =True

Example

Previous program using compound testing

Named IF statements

If you wish, you can name an IF statement

- syntax

```
if1:  IF (cond) THEN
        {
      ENDIF if1
```

This statement has been named if1

- name of the block follows the rules as any other program variable
- must not be the same as another variable in your program
- can do this for any IF structure.

Equality testing

- Due to the nature of floating point computing, it is (almost) never safe to test for equality of 2 floating point variables.

  * There are some specific instances where this can be done, but generally it's not safe to do.

- IF (a == b) THEN
  {
  ENDIF

  → IF (err < tol) THEN
  {
  ENDIF

  can and will fail most of the time.

- Instead, you should use a tolerance based test.

- instead of `a == b`

  test "`|a - b| <= small`"

  → This is the idea to use, but need to use a relative difference

- $err = \frac{|a-b|}{|a|+|b|}$ = `ABS(a-b)/(ABS(a)+ABS(b))`

- a and b are close enough to be considered the same if `err <= tol`

  where tol is some prescribed tolerance.

- The size of tol depends on the nature of a, b and what they represent physically.
- use `tol = 1.0d-14` ($= 10^{-14}$) unless otherwise instructed.