Goals for This Lecture:

- Examine named form for block IF constructs
- Understand nesting of block if IF constructs
- Alternative ways to structure conditional execution
- Introduce the IF statement
- Introduce the CASE construct
- Describe a few obsolete programming constructs:
  - CONTINUE statements
  - Arithmetic IF statements
  - GOTO statements
Naming block IF constructs

- The block IF construct can accommodate a optional name which can help identify the clauses of the construct.

Form:

```
name: if(logical_expr1) then
  statement block 1
...
elseif(logical_expr2) then name
  statement block 2
...
elseif(logical_expr3) then name
  statement block 3
...
else name
  alternative statement block
...
endif name
```

Example:

```
first1: if(x > 3.0) then
  f = 2.0e0*(x**2)+3.0e0
elseif(x > 2.0) then first1
  f = x
elseif(x > 1.0) then first1
  f = -1.0*x
else first1
  f = -1.0*(x**2)
endif
```
Nesting block IF constructs

- Block IF constructs can be nested within one another

Form:

```plaintext
if(logical_expr1) then
    statement block 1
    ...
else
    if(logical_expr2) then
        statement block 2
        ...
    else
        statement block 3
        ...
    endif
endif
```

Example:

```plaintext
if(x > 3.0) then
    f = 2.0e0*(x**2)+3.0e0
else
    if(x > 2.0) then
        f = x
    else
        if(x > 1.0) then
            f = -1.0*x
        else
            f = -1.0*(x**2)
        endif
    endif
endif
endif
```
Nesting block IF constructs

- Named block IF constructs can help to clarify code structure

Example w/o names:

```plaintext
if(x > 3.0) then
  f = 2.0*(x**2)+3.0
else
  if(x > 2.0) then
    f = x
  else
    if(x > 1.0) then
      f = -1.0*x
    else
      f = -1.0*(x**2)
    endif
  endif
endif
```

Example:

```plaintext
outer: if(x > 3.0) then
  f = 2.0*(x**2)+3.0
else outer
  middle: if(x > 2.0) then
    f = x
  else middle
    inner: if(x > 1.0) then
      f = -1.0*x
    else inner
      f = -1.0*(x**2)
    endif inner
  endif middle
endif outer
```
Alternative ways to structure conditional execution

Example using elseif clauses:

```plaintext
first1: if(x > 3.0) then
    f = 2.0e0*(x**2)+3.0e0
elseif(x > 2.0) then
    f = x
elseif(x > 1.0) then
    f = -1.0*x
else
    f = -1.0*(x**2)
endif
```

Example using nested ifs:

```plaintext
outer: if(x > 3.0) then
    f = 2.0*(x**2)+3.0
else
middle: if(x > 2.0) then
    f = x
else
inner: if(x > 1.0) then
    f = -1.0*x
else
    f = -1.0*(x**2)
endif
endif
endif
```
IF statement

• FORTRAN provides a simple way to execute a single line of code conditionally using the IF statement

  Form:
  \[
  \text{if}(\text{logical\_expr1}) \text{ statement}
  \]

  Example:
  \[
  \text{if}( x < 0.0 ) \ x = -x
  \]

• This for is very useful for small modifications to the flow of the program
CASE constructs

- The CASE construct is a useful way of branching based on more complex sets of expressions.

Form:
```plaintext
select case(case_expression)
  case(case1_selector_list)
    statement block 1
  ...
  case(case2_selector_list)
    statement block 2
  ...
  case(case3_selector_list)
    statement block 3
  ...
  case default
    alternative statement block
  ...
end select
```

Example:
```plaintext
select case(i)
  case(10:)
    f = 2.0
  case(1,3,5,7,9)
    f = 0.75
  case(2,4,6,8)
    f = 0.25
  case(0)
    write(*,*) ' i = 0 not allowed'
  case default
    f = -1.0
end select
```
CASE selector lists

- Expressions and selectors in case statements must have integer, logical or character values

- The `case_selector_list` can consist of one of a list of case selector expressions separated by commas

- The case selector expressions take on one of four forms:
  - `case_value` executes block if `case_expression` is equal to this value
    - Example: 3
  - `low_value: executes block if case_expression >= low_value`
    - Example: 3:
  - `:high__value executes block if case_expression <= this high_value`
    - Example: :3
  - `low_value:high_value executes block if low_value <= case_expression <= high_value`
    - Example: 3:7
Obsolete Programming Constructs That You Will Encounter in FORTRAN code

• In your programming endeavors you will have occasion to make use of code that was written some time ago (legacy or dusty-deck code).

• For this reason we will briefly describe several programming constructs that are sometimes encountered in legacy code:
  – GOTO statements
  – CONTINUE statements
  – Arithmetic IF statements

• Avoid using these statements at all cost! They are obsolete and dangerous
GOTO & CONTINUE statements

• The GOTO statement has been highly abused and is now considered a deprecated feature of the FORTRAN language (meaning it may be removed in future versions of FORTRAN)

• The GOTO transfers execution to another portion of the program described by a statement number
• Often combined with IF statements to transfer execution conditionally

• Form:
  
goto statement_number

• Example:
  
if(x < 0.0) goto 10
  y = log(x)
goto 20
  10 continue
  y = 0.0
  20 continue
write(*,*) ' y = ',y

• See how confusing this is!
• CONTINUE is a meaningless statement that has no effect
  – It is often used as a line to transfer execution to
Arithmetic IF statements

• Another deprecated language feature that you may encounter if the arithmetic IF statement

Form:

\[ \text{if(expression) label1, label2, label3} \]

Example:

\[ \text{if(x) 10, 20, 30} \]
\[ 30 \text{ continue} \]
\[ f = 2.0e0*(x**2)+3.0e0 \]
\[ \text{goto 40} \]
\[ 20 \text{ continue} \]
\[ f = 0.0 \]
\[ \text{goto 40} \]
\[ 10 \text{ continue} \]
\[ f = x**3 \]
\[ 40 \text{ continue} \]
Reading Assignment

– Read Sections 3.4-3.6