3. Program Design and Branching Structures
Two broad categories of control statement:
- Branches
- Loops

These will make the program more complex.
To avoid programming errors, a formal program design procedure is needed, which is based on the technique known as top-down design.
Top-down design technique

- Design problem (equations) to be solved
  - Design an algorithm (step-by-step procedure) for finding the solution
    - Subtask
      - Pseudo-code
        - Program unit (statements)
          - Program unit test & validation
    - Subtask
      - Pseudo-code
        - Program unit (statements)
          - Program unit test & validation
    - Pseudo-code
      - Program unit (statements)
        - Program unit test & validation
    - Subtask
      - Pseudo-code
        - Program unit (statements)
          - Program unit test & validation
    - Subtask
      - Pseudo-code
        - Program unit (statements)
          - Program unit test & validation
    - Subtask
      - Pseudo-code
        - Program unit (statements)
          - Program unit test & validation
    - Subtask
      - Pseudo-code
        - Program unit (statements)
          - Program unit test & validation
    - Subtask
      - Pseudo-code
        - Program unit (statements)
          - Program unit test & validation
- Final program
Form to describe the algorithm

- **Pseudo-code**: hybrid mixture of English & program statements
- **Flowchart**: graphical description
- Any other formats you prefer

- Use pencil and paper as much as you can before writing the computer program.
- Your note $\cong$ the computer program.
Assignment Statements and Logical Calculation

logical_variable_name = logical expression

A logical expression is a combination of logical constants, logical variables and logical operators.

**logical constants:**  
.true.  
.false.

**logical variables:**  
logical :: done  
logical :: done1 = .true.  
logical :: done2 = .false.

**logical operators:**  
relational logical operator  
combinational logical operator
Relational logical operators:

<table>
<thead>
<tr>
<th>new style</th>
<th>old style</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>.eq.</td>
</tr>
<tr>
<td>/=</td>
<td>.ne.</td>
</tr>
<tr>
<td>&gt;</td>
<td>.gt.</td>
</tr>
<tr>
<td>&gt;=</td>
<td>.ge.</td>
</tr>
<tr>
<td>&lt;</td>
<td>.lt.</td>
</tr>
<tr>
<td>&lt;=</td>
<td>.le.</td>
</tr>
</tbody>
</table>

Example:

<table>
<thead>
<tr>
<th>logical assignment statement</th>
<th>logical value</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 &lt; 4</td>
<td>.true.</td>
</tr>
<tr>
<td>3 &lt;= 4</td>
<td>.true.</td>
</tr>
<tr>
<td>3 &gt; 4</td>
<td>.false.</td>
</tr>
<tr>
<td>4 &lt;= 4</td>
<td>.true.</td>
</tr>
<tr>
<td>7+3 &lt; 2+11</td>
<td>.true.</td>
</tr>
<tr>
<td>(7+3) &lt; (2+11)</td>
<td>.true.</td>
</tr>
<tr>
<td>4 == 4.0</td>
<td>.true.</td>
</tr>
</tbody>
</table>
Combinational logical operators:

<table>
<thead>
<tr>
<th>Logical statement</th>
<th>The result is .true. if...</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 .and. L2</td>
<td>both L1 and L2 are true</td>
</tr>
<tr>
<td>L1 .or. L2</td>
<td>either L1 or L2 is true</td>
</tr>
<tr>
<td>L1 .eqv. L2</td>
<td>L1 is the same as L2</td>
</tr>
<tr>
<td>L1 .neqv. L2</td>
<td>L1 is not the same as L2</td>
</tr>
<tr>
<td>.not. L1</td>
<td>L1 is false</td>
</tr>
</tbody>
</table>

Example:

L1 = .true.
L2 = .true.
L3 = .false.

<table>
<thead>
<tr>
<th>Logical statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>.not. L1</td>
<td>.false.</td>
</tr>
<tr>
<td>L1 .or. L3</td>
<td>.true.</td>
</tr>
<tr>
<td>L2 .neqv. L3</td>
<td>.true.</td>
</tr>
<tr>
<td>L1 .and. L2 .or. L3</td>
<td>.true.</td>
</tr>
<tr>
<td>L1 .or. L2 .and. L3</td>
<td>.false.</td>
</tr>
<tr>
<td>.not. (L1 .eqv. L2)</td>
<td>.false.</td>
</tr>
</tbody>
</table>
** Copy this program to the directory ./examples, compile and execute it.
Control Constructs: Branch

• Logical IF statement
• Block IF construct
• ELSE and ELSE IF clauses
• Nested Block IF
• Name Block IF
• **Logical IF statement**
  
  if (logical expression) statement

• **Block IF construct**
  
  if (logical expression) then
    statement block
  end if

• **Block IF construct with ELSE IF clause**
  
  if (logical expression 1) then
    statement block 1
  else if (logical expression 2) then
    statement block 2
  else if (logical expression 3) then
    ...
  else !option
    statement block !option
  end if

(* At most, one statement block is executed. ***)
Example of using BLOCK IF:

Solve $ax^2 + bx + c = 0$ for $x$ given $a$, $b$ and $c$.

Design the algorithm (procedure):

1. Read the input data $a$, $b$, and $c$

2. Calculate the roots

   if $b^2 - 4ac > 0$, then $x = -b/(2.*a) + (b**2 - 4.*a*c)/(2.*a)$
   \hspace{5cm} -b/(2.*a) - (b**2 - 4.*a*c)/(2.*a)$

   if $b^2 - 4ac = 0$, then $x = -b/(2.*a)$

   if $b^2 - 4ac < 0$, then $x = -b/(2.*a) + i(4.*a*c - b**2)/(2.*a)$
   \hspace{5cm} -b/(2.*a) - i(4.*a*c - b**2)/(2.*a)$

3. Write out the roots
Flowchart:

start

read a, b, c

echo a, b, c

b**2 - 4.*a*c > 0

false.

write "The equation has two distinct real roots."

calculate x1, x2

write x1, x2

false.

false.

true.

write "The equation has two identical real roots."

calculate x1

write x1

false.

false.

true.

write "The equation has two complex roots."

calculate real_part, imag_part

write real_part \, + \, i \, imag_part

write real_part \, - \, i \, imag_part

stop
Purpose:
This program solves for the roots of a quadratic equation of the form
\[ a \cdot x^2 + b \cdot x + c = 0. \]
It calculates the answers regardless of the type of roots that the equation possesses.

Declare the variables used in this program:
- Coefficient of the form \( a \cdot x^2 \) term of equation
- Coefficient of the form \( b \cdot x \) term of equation
- Constant term of the equation
- Discriminant of the equation
- Imaginary part of equation (for complex roots)
- Real part of equation (for real roots)
- First solution of equation (for real roots)
- Second solution of equation (for real roots)

Prompt the user for the coefficients of the equation:
- Enter the coefficients \( a \), \( b \), and \( c \): 

```
write(*) 'This program solves for the roots of a quadratic equation of the form a \cdot x^2 + b \cdot x + c = 0. Enter the coefficients a, b, and c: ', a, b, c
```

Echo back coefficients:
- The coefficients \( a \), \( b \), and \( c \) are: 

```
write(*,*) a, b, c
```
! Calculate discriminant
discriminant = b**2 - 4.0*a*c

! Solve for the roots, depending upon the value of the discriminant
IF (discriminant > 0.0) THEN
  x1 = (-b + sqrt(discriminant)) / (2.0*a)
  x2 = (-b - sqrt(discriminant)) / (2.0*a)
  write(*,*) 'This equation has two real roots:'
  write(*,*) 'x1 = ', x1
  write(*,*) 'x2 = ', x2
ELSE IF (discriminant == 0.0) THEN
  x1 = (-b) / (2.0*a)
  write(*,*) 'This equation has two identical real roots:'
  write(*,*) 'x1 = x2 = ', x1
ELSE ! there are complex roots, so...
  real_part = (-b) / (2.0*a)
  imag_part = sqrt(abs(discriminant)) / (2.0*a)
  write(*,*) 'This equation has complex roots:'
  write(*,*) 'x1 = ', real_part, ' +i ', imag_part
  write(*,*) 'x2 = ', real_part, ' -i ', imag_part
END IF

end program

** Copy this program to the directory ./examples, compile and execute it.
** Change ELSE to ELSE IF (discriminant < 0.0) THEN, and test it again.
• Using equality test in IF construct

```plaintext
if(x==10.) then
...
```

• This may cause problem, i.e., the logical test is never be true.
• Because of small round-off errors during floating-point arithmetic operations, two numbers that theoretically are equal will be different by a tiny amount.
• Instead of testing equality, it is better to test near equality.

```plaintext
if(abs(x-10.) <= 0.000001) then
...
```

• In the previous example, a better way is to write:

```plaintext
if(abs(discriminant) <= 1.0E-7) THEN
...
```
** Block IF constructs**

\[
\text{if (test 1) then} \\
\quad \text{statement block 1} \\
\text{else if (test 2) then} \\
\quad \text{statement block 2} \\
\text{else if (test 3) then} \\
\quad \text{statement block 3} \\
\text{end if}
\]

** Nested Block IF constructs**

\[
\text{if (test 1) then} \\
\quad \text{statement block 1} \\
\text{else} \\
\quad \text{if (test 2) then} \\
\quad \quad \text{statement block 2} \\
\text{else} \\
\quad \quad \text{if (test 3) then} \\
\quad \quad \quad \text{statement block 3} \\
\text{end if} \\
\text{end if}
\]

** The two are the same. Choose whichever style you prefer.**
**Nested Block IF Constructs with ELSE**

```plaintext
if (test 1) then
    statement block 1
else
    if (test 2) then
        statement block 2
    else
        if (test 3) then
            statement block 3
        end if
    end if
end if
```

**Nested Block IF Constructs without ELSE**

```plaintext
if (test 1) then
    statement block 1
else
    if (test 2) then
        statement block 2
    else
        if (test 3) then
            statement block 3
        end if
    end if
end if
```

**The two are totally different!**

(At most, 1 statement block is executed.)

(At most, 3 statement blocks are executed.)
• Example comparing Block IF and Nested Block IF

<table>
<thead>
<tr>
<th>score</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>95 &lt; x</td>
<td>A</td>
</tr>
<tr>
<td>86 &lt; x ≤ 95</td>
<td>B</td>
</tr>
<tr>
<td>76 &lt; x ≤ 86</td>
<td>C</td>
</tr>
<tr>
<td>66 &lt; x ≤ 76</td>
<td>D</td>
</tr>
<tr>
<td>0 &lt; x ≤ 66</td>
<td>E</td>
</tr>
</tbody>
</table>

if x > 95 ?

YES: write(*,*)'Grade is A'

NO: if x > 86 ?

YES: write(*,*)'Grade is B'

NO: if x > 76 ?

YES: write(*,*)'Grade is C'

NO: if x > 66 ?

YES: write(*,*)'Grade is D'

NO: write(*,*)'Grade is E'

end
• Example comparing Block IF and Nested Block IF

<table>
<thead>
<tr>
<th>score</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>95 &lt; x</td>
<td>A</td>
</tr>
<tr>
<td>86 &lt; x ≤ 95</td>
<td>B</td>
</tr>
<tr>
<td>76 &lt; x ≤ 86</td>
<td>C</td>
</tr>
<tr>
<td>66 &lt; x ≤ 76</td>
<td>D</td>
</tr>
<tr>
<td>0 &lt; x ≤ 66</td>
<td>E</td>
</tr>
</tbody>
</table>

```
write(*,*)'Grade is A'
write(*,*)'Grade is B'
write(*,*)'Grade is C'
write(*,*)'Grade is D'
write(*,*)'Grade is E'
```
if (x > 95) then
  write(*,*) 'Grade is A'
else if (x > 86) then
  write(*,*) 'Grade is B'
else if (x > 76) then
  write(*,*) 'Grade is C'
else if (x > 66) then
  write(*,*) 'Grade is D'
else
  write(*,*) 'Grade is E'
end if

<table>
<thead>
<tr>
<th>score</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>95 &lt; x</td>
<td>A</td>
</tr>
<tr>
<td>86 &lt; x ≤ 95</td>
<td>B</td>
</tr>
<tr>
<td>76 &lt; x ≤ 86</td>
<td>C</td>
</tr>
<tr>
<td>66 &lt; x ≤ 76</td>
<td>D</td>
</tr>
<tr>
<td>0 &lt; x ≤ 66</td>
<td>E</td>
</tr>
</tbody>
</table>

 Nested Block IF

if (x > 95) then
  write(*,*) 'Grade is A'
else
  if (x > 86) then
    write(*,*) 'Grade is B'
else
    if (x > 76) then
      write(*,*) 'Grade is C'
else
      if (x > 66) then
        write(*,*) 'Grade is D'
else
        write(*,*) 'Grade is E'
      end if
    end if
  end if
end if
**Name Block IF constructs**

- to help identify a complex BLOCK IF construct that spans long listing
- `[name] after else if and else are option

```
[name1:] if (logical expression 1) then
    statement block 1
else [name1]
    [name2:] if (logical expression 2) then
        statement block 2
    else [name2]
        [name3:] if (logical expression 3) then
            ...
        else [name3]
            statement block n
        end if [name3]
    end if [name2]
end if [name1]
```

```
[name:] if (logical expression 1) then
    statement block 1
else if (logical expression 2) then [name]
    statement block 2
else if (logical expression 3) then [name]
    ...
else [name]
    statement block n
end if [name]
```
**Nested Block IF constructs**

<table>
<thead>
<tr>
<th>Block IF Constructs</th>
<th>Name Block IF Constructs</th>
<th>Name Nested Block IF Constructs</th>
</tr>
</thead>
<tbody>
<tr>
<td>if (test 1) then</td>
<td>test: if (test 1) then</td>
<td>outer: if (test 1) then</td>
</tr>
<tr>
<td>statement block 1</td>
<td>statement block 1</td>
<td>statement block 1</td>
</tr>
<tr>
<td>else if (test 2) then</td>
<td>else if (test 2) then</td>
<td>else outer</td>
</tr>
<tr>
<td>statement block 2</td>
<td>statement block 2</td>
<td>middle: if (test 2) then</td>
</tr>
<tr>
<td>else if (test 3) then</td>
<td>else if (test 3) then</td>
<td>inner: if (test 3) then</td>
</tr>
<tr>
<td>statement block 3</td>
<td>statement block 3</td>
<td>end if inner</td>
</tr>
<tr>
<td>end if</td>
<td>end if</td>
<td>end if middle</td>
</tr>
<tr>
<td></td>
<td></td>
<td>end if outer</td>
</tr>
</tbody>
</table>

- Nested Block IF constructs
- Name Block IF Constructs
- Name Nested Block IF Constructs
Block IF

if (x > 95) then
   write(*,*) 'Grade is A'
else if (x > 86) then
   write(*,*) 'Grade is B'
else if (x > 76) then
   write(*,*) 'Grade is C'
else if (x > 66) then
   write(*,*) 'Grade is D'
else
   write(*,*) 'Grade is E'
end if

<table>
<thead>
<tr>
<th>score</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>95 &lt; x</td>
<td>A</td>
</tr>
<tr>
<td>86 &lt; x ≤ 95</td>
<td>B</td>
</tr>
<tr>
<td>76 &lt; x ≤ 86</td>
<td>C</td>
</tr>
<tr>
<td>66 &lt; x ≤ 76</td>
<td>D</td>
</tr>
<tr>
<td>0 &lt; x ≤ 66</td>
<td>E</td>
</tr>
</tbody>
</table>

Nested Block IF

aif: if (x > 95) then
   write(*,*) 'Grade is A'
else aif
   bif: if (x > 86) then
      write(*,*) 'Grade is B'
   else bif
      cif: if (x > 76) then
         write(*,*) 'Grade is C'
      else cif
         dif: if (x > 66) then
            write(*,*) 'Grade is D'
         else dif
            write(*,*) 'Grade is E'
         end if dif
      end if cif
   end if bif
end if aif
Exercise 1

Write two Fortran programs using both block if and nested block if respectively to evaluate the function with different user-specified real $x$ and $y$:

$$f(x, y) = \begin{cases} 
  x + y & x \geq 0 \text{ and } y \geq 0 \\
  x + y^2 & x \geq 0 \text{ and } y < 0 \\
  x^2 + y & x < 0 \text{ and } y \geq 0 \\
  x^2 + y^2 & x < 0 \text{ and } y < 0
\end{cases}$$
Consider a cubic equation with real coefficients $a_1$, $a_2$, $a_3$:

$$x^3 + a_1 x^2 + a_2 x + a_3 = 0.$$ 

Let

$$Q = \frac{1}{9}(3a_2 - a_1^2), \quad R = \frac{1}{5}(9a_1 a_2 - 27a_3 - 2a_1^3), \quad D = Q^3 + R^2,$$

$$S = \text{sgn}(R + D^{1/2})|R + D^{1/2}|^{1/3}, \quad T = \text{sgn}(R - D^{1/2})|R - D^{1/2}|^{1/3}.$$

When $D > 0$, the equation has a real root and 2 non-real complex conjugate roots:

$$\begin{align*}
    x_1 &= S + T - \frac{1}{3}a_1 \\
    x_2 &= -\frac{1}{2}(S + T) - \frac{1}{3}a_1 + i\frac{1}{2}3^{1/2}(S - T) \\
    x_3 &= -\frac{1}{2}(S + T) - \frac{1}{3}a_1 - i\frac{1}{2}3^{1/2}(S - T).
\end{align*}$$

When $D = 0$, all roots are real and at least two are equal:

$$\begin{align*}
    x_1 &= 2R^{1/3} - \frac{1}{3}a_1 \\
    x_2 &= -R^{1/3} - \frac{1}{3}a_1 \\
    x_3 &= -R^{1/3} - \frac{1}{3}a_1.
\end{align*}$$

When $D < 0$, the equation has 3 distinct real roots:

$$\begin{align*}
    x_1 &= 2(-Q)^{1/2} \cos\left(\frac{1}{3}\theta\right) - \frac{1}{3}a_1 \\
    x_2 &= 2(-Q)^{1/2} \cos\left(\frac{1}{3}\theta + \frac{2}{3}\pi\right) - \frac{1}{3}a_1 \\
    x_3 &= 2(-Q)^{1/2} \cos\left(\frac{1}{3}\theta + \frac{4}{3}\pi\right) - \frac{1}{3}a_1,
\end{align*}$$

where, $\cos\theta = R/(-Q)^{3/2}$.

Implement these into a Fortran program given the three coefficients as input data.
Exercise 3

The implementation of the solution of a cubic equation can be tested by substituting the three solved roots into the original equation and see if it is nearly equal to zero.

It can also be tested whether the following identities are satisfied to the appropriate precision:

\[ x_1 + x_2 + x_3 = -a_1, \quad x_1 x_2 + x_2 x_3 + x_3 x_1 = a_2, \quad x_1 x_2 x_3 = -a_3, \]

where \( x_1, x_2, x_3 \) are the three roots.

Incorporate these two tests into the program.

Examples of cubic equations having different types of roots are:

\[ x^3 - 6x^2 + 11x - 6 = 0 \]
\[ x^3 - 5x^2 + 8x - 4 = 0 \]
\[ x^3 - 3x^2 + 3x^2 - 1 = 0 \]
\[ x^3 + x^2 + x - 3 = 0 \]