Program Flow Rafting - Dealing with Branching Construct

1. Tracing Code Blocks
   - Hand tracing: Simulating run-time behavior given the value of the variables and the source code.
   - Machine tracing:
     - Inserting debugging WRITE statement inside each code block
     - Use debugger program

   **Practice:** Trace the execution of the claued IF statement in the following partial code for a salary value of $23,500.0 (assume variables \textit{tax} and \textit{salary} are both declared in REAL type, and represent values in US$). Circle the assignment statement that would be executed.

   ```
   IF ( salary < 0. ) THEN !invalid salary
     tax = -1.
   ELSE IF ( salary < 15000. ) THEN !first range
     tax = .15 * salary;
   ELSE IF ( salary < 30000. ) THEN !second range
     tax = (salary - 15000.) * .18 + 2250.
   ELSE IF ( salary < 50000. ) THEN !third range
     tax = (salary - 30000. ) * .22 + 5400.
   ELSE IF ( salary < 80000. ) THEN !fourth range
     tax = (salary - 50000. ) * .27 + 11000.
   ELSE IF ( salary < 150000. ) THEN !fifth range
     tax = (salary - 80000. ) * .33 + 21600.
   ELSE
     tax = -1. !Cannot handle the remaining range
   END IF
   ```

2. Order of Conditions
   When more than one condition in a multiple-alternative decision is true, only the code block following the first true condition executes. Therefore, the order of the conditions can affect the outcome.

   **Practice:** In the previous practice, what would be the effect of reversing the order of the conditions given the same salary value of $23,500.0? Circle the assignment statement that would be executed.

   ```
   IF ( salary < 0. ) THEN !invalid salary
     tax = -1.
   ELSE IF ( salary < 150000. ) THEN !fifth range
     tax = (salary - 80000. ) * .33 + 21600.
   ELSE IF ( salary < 80000. ) THEN !fourth range
     tax = (salary - 50000. ) * .27 + 11000.
   ELSE IF ( salary < 50000. ) THEN !third range
     tax = (salary - 30000. ) * .22 + 5400.
   ELSE IF ( salary < 30000. ) THEN !second range
     tax = (salary - 15000. ) * .18 + 2250.
   ELSE IF ( salary < 15000. ) THEN !first range
     tax = .15 * salary;
   ELSE
     tax = -1. !Cannot handle the remaining range
   END IF
   ```
3. **Nested Branch** An IF statement with another IF statement under its branched block.

*Practice:* A year is a leap year if it is divisible by 4, except that any year divisible by 100 is a leap year only if it is divisible by 400. This leap year verification logic can be presented in the following flowchart:

![Flowchart showing leap year logic](image)

Implement the algorithm into a main program that takes user input of a 4-digit year number, and return user the information whether the year is a leap year or not. Use a nested IF statement in your code. You can verify your implementation using the following testing cases:

<table>
<thead>
<tr>
<th>Year</th>
<th>isLeapYear</th>
</tr>
</thead>
<tbody>
<tr>
<td>1995</td>
<td>false.</td>
</tr>
<tr>
<td>1996</td>
<td>true.</td>
</tr>
<tr>
<td>2000</td>
<td>true.</td>
</tr>
<tr>
<td>2100</td>
<td>false.</td>
</tr>
</tbody>
</table>

4. **Common Syntax Errors**

*Practice:* Observe and fix the errors in the following demo code:

```fortran
PROGRAM main
  IMPLICIT NONE
  INTEGER::x

  WRITE(*,*)"Please enter an integer number:"
  READ(*,*) x

  IF ( x /= 0 )
    WRITE(*,*) "logiExpr1: x is not equal to 0 - .true. ", x
  END IF

  IF ( 0 < x < 4 ) THEN
    WRITE(*,*) "logiExpr2: x is between 0 and 4 - .true. ", x
  END IF

  IF ( x = 10 ) THEN
    WRITE(*,*) "logiExpr3: x is equal to 10 - .true. ", x
  END IF

  IF ( x ) THEN
    WRITE(*,*) "logiExpr4: x is true ", x
  END IF
END PROGRAM
```
5. **Select-Case statement:**

- **Syntax:**
  
  ```
  SELECT CASE ( caseExpression )
  CASE ( caseSelector1 )
    Statement(s) of Block 1
  CASE ( caseSelector2 )
    Statement(s) of Block 2
  ...
  CASE DEFAULT
    Statement(s) of Block n
  END SELECT
  ```

- **Rules:**
  - The *case expression* must hold integer, character, or logical data type, real type is not permitted.
  - The *case selector* can take one of the following forms:
    - `caseValue` Execute the block if `(caseValue == caseExpression)`
    - `lowValue : highValue` Execute the block if `(lowValue <= caseExpression <= highValue)`
  - Or it can be a list of any combinations of these forms separated by commas
  - The *case selectors* must be exclusive from one another, in other words, there is no overlapping between different case selectors.

- **Recommended practice:**
  - Always use a default block to guarantee that the switch construct is complete to cover all possible scenarios.

**Practice(Optional):** Interpret, compile and run the following demo code:

```fortran
PROGRAM main
  IMPLICIT NONE
  CHARACTER(3)::day=''
  DO WHILE ( day /= 'Q' )
    WRITE(*,*) 'Enter the name of the day(type 'Q' to quit):'
    READ(*,*) day
    SELECT CASE (day)
      CASE ( 'Mon', 'Tue', 'Wed', 'Thu', 'Fri' )
        WRITE(*,*) 'It is a weekday'
      CASE ( 'Sat', 'Sun' )
        WRITE(*,*) 'It is a weekend'
      CASE ( 'Q' )
        WRITE(*,*) 'Quit this demo'
      CASE default
        WRITE(*,*) 'Sorry, I cannot understand the name of the day.'
    END SELECT
  END DO
END PROGRAM main
```

Last update: 2009-09-30