Computer Science 1510

Lecture 7

Lecture Outline

- Selection: IF statements
- Selection: SELECT-CASE statements

Control Logic

- So far we have concentrated on programs with statements executed sequentially from top to bottom.

- However, some of the algorithms that we have seen require:
  - control over the order in which statements are executed;
  - control over which of a group of statements is to be executed; or
  - control over whether certain statements are executed at all.

- In the latter two cases we can use IF statements or SELECT-CASE statements.

Logical expressions

- Logical expressions are expressions that evaluate to either true or false.

- Logical expressions can consist of logical constants (`.TRUE.` or `.FALSE.`), logical variables, or relational expressions of the form

```
exp1 relational-operator exp2
```

- Example: `X > 2`.

- Logical expressions can also be formed by combining smaller logical expressions using a logical operator.

- Example: `X >= 5 .AND. X <= 10`.

Relational operators

- Two items can be compared in a logical expression using relational operators.

- Relational operators in Fortran include:

Operator	Meaning
==	Equal to
/=	Not equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to

- Note that the two items being compared would normally (but not always) be of the same type.

Logical operators

- Logical operators can be used to combine conditions in logical expressions.

- Logical operators in Fortran include:

Operator	Meaning
`.NOT.`	Not
`.AND.`	And
`.OR.`	Or
`.EQV.`	Equivalent to
`.NEQV.`	Not equivalent to

- The first operator is a unary operator, meaning that it operates on one item, `.NOT.A`, for example.

- The remaining four operators are binary operators, meaning that they operate on two items. That is `A OP B`, where `OP` is a logical operator.

Logic (truth) tables

A	B	A.AND.B
T	T	T
T	F	F
F	T	F
F	F	F

A	B	A.OR.B
T	T	T
T	F	T
F	T	T
F	F	F

A	B	A.EQV.B
T	T	T
T	F	F
F	T	F
F	F	T

A	B	A.NEQV.B
T	T	F
T	F	T
F	T	T
F	F	F

A	.NOT.A
T	F
F	T

Order of Operations

- Logical operators:

  `.NOT., .AND., .OR., .EQV. (or .NEQV)`

- For expressions containing arithmetic operators, relational operators, and logical operators, the operations are performed in the following order:

  1. Arithmetic
  2. Relational
  3. Logical

- Consider `N**2 + 1 > 10 .AND. .NOT. N < 3`, with `N=4`, we have,

  1. Arithmetic: `17 > 10 .AND. .NOT. 4 < 3`
  2. Relational: `.TRUE. .AND. .NOT. .FALSE.`
  3. Logical: `.TRUE. .AND. .TRUE.` $=$ `.TRUE.`

- Using parentheses for clarity is strongly advised.

Fortran Statements: IF

- The `IF` construct can be used if a given sequence of statements is to be executed or bypassed depending on the result of a logical expression.

- Syntax:

```
IF (logical-expression) THEN
   statement-sequence
END IF
```

  where `logical-expression` is some condition that must be true in order for `statement-sequence` to be executed.

- If the condition is false then `statement-sequence` is not executed and execution continues from the statement following the IF block.

- Example:

```
IF (x>=0) THEN
   y=x*x
   z=SQRT(x)
END IF
```

Fortran Statements: Logical IF

- In the case where only a single statement is to be executed depending on the result of some logical expression, a logical `IF` statement can be used.

- Syntax:

  ```
  IF (logical-expression) statement
  ```

  where `logical-expression` is some condition that must be true in order for `statement` to be executed.

- If the condition is false then `statement` is not executed and execution continues from the statement following the IF statement.

- Note that `statement` is a single command. For conditional execution of more than one command we must use the IF-THEN statement.

Fortran statements: IF-THEN-ELSE

- In the case where we wish to execute a group of statements under one condition, but a different group of statements otherwise then the IF-THEN-ELSE construct can be used.

- Syntax:

```
IF (logical-expression) THEN
   statement-sequence1
ELSE
   statement-sequence2
END IF
```

- Example (evaluation of a piecewise function):

```
IF (X <= 0) THEN
   fval = -x
ELSE
   fval = x**2
END IF
```

Fortran statements: ELSE IF

- The final case occurs when one wishes to execute a group of statements under one condition, but a different group of statements under a different condition. Then the IF-ELSE-IF construct can be used.

- Syntax:

```
IF (logical-expression-1) THEN
   statement-sequence1
ELSE IF (logical-expression-2) THEN
   statement-sequence2
END IF
```

- If neither `logical-expression-1` nor `logical-expression-2` are true, then no portion of the above `IF` block is executed.

- One can also have an ELSE statement at the end of the IF-ELSE-IF construct, in which case, exactly one group of statements within the IF block will be executed.

Example: ELSE IF

```
PROGRAM Rectangles
  IMPLICIT NONE
  REAL::x1,x2,y1,y2
  REAL::x3,x4,y3,y4
  REAL::x,y
  x1=1;y1=1;x2=3;y2=4
  x3=4;y3=5;x4=8;y4=9

  WRITE(*,*) 'Enter coordinates:'
  READ(*,*) x,y

  IF (x>=x1.AND.x<=x2.AND.y>=y1.AND.y<=y2) THEN
    WRITE(*,*) 'Inside rectangle 1'
  ELSE IF (x>=x3.AND.x<=x4.AND.y>=y3.AND.y<=y4) THEN
    WRITE(*,*) 'Inside rectangle 2'
  ELSE
    WRITE(*,*) 'Outside both rectangles'
  END IF
END PROGRAM Rectangles
```

Nested IF statements

- IF statements can also be nested, that is, we are permitted to use IF statements within IF statements.

```
IF (logical-expression-1) THEN
   statement-sequence-1
   IF (logical-expression-2) THEN
      statement-sequence-2
   END IF
   statement-sequence-3
ELSE IF (logical-expression-3) THEN
   statement-sequence-4
ELSE
   statement-sequence-5
END IF
```

- Note the indenting used in this example. Such formatting makes source code much easier to read and debug.

Example: Nested IF

```
PROGRAM Ascending_order
  IMPLICIT NONE
  INTEGER::a,b,c
  WRITE(*,*) 'Please enter 3 integers:'
  READ(*,*) a,b,c
  IF (a<=b.AND.a<=c) THEN ! a is smallest
    IF (b<=c) THEN
      WRITE(*,*) a,b,c
    ELSE
      WRITE(*,*) a,c,b
    END IF
  ELSE IF (b<=a.AND.b<=c) THEN ! b is smallest
    IF (a<=c) THEN
      WRITE(*,*) b,a,c
    ELSE
      WRITE(*,*) b,c,a
    END IF
  ELSE ! c is smallest
    IF (a<=b) THEN
      WRITE(*,*) c,a,b
    ELSE
      WRITE(*,*) c,b,a
    END IF
  END IF
END PROGRAM Ascending_order
```

Logical variables

- Logical data can have one of two values, true or false.
- In Fortran logical data is indicated by .TRUE. or .FALSE. (Note the periods before and after the word).
- Fortran Logical: LOGICAL
- Logical variables can be assigned a value with a statement of the form

```
logical-variable = logical-expression
```

- Example: `test = .TRUE.`
- Printing a logical variable prints `T` or `F`.

Logical variables

- To read a logical value, the input can consist of an optional period followed by `T` or `F` which may be followed by other characters. A value of `.TRUE.` or `.FALSE.` is assigned to the variable according to whether the first letter is `T` or `F`.

- Logical variables can be used as the condition in an IF statement. For example:

```
PROGRAM Admit
   LOGICAL::legal
   WRITE(*,*) 'Are you 19 years of age or older?'
   READ(*,*) legal
   IF (legal) THEN
      WRITE(*,*) 'You can enter'
   ELSE
      WRITE(*,*) 'You may not enter'
   END IF
END PROGRAM Admit
```

Fortran statements: SELECT-CASE

- In the case where we would like to execute a different section of code depending on the value of a single variable or expression, the SELECT-CASE statement may be used instead of an IF statement.

- Syntax:

```
SELECT CASE (selector)
   CASE (label-list-1)
      statement-sequence-1
   CASE (label-list-2)
      statement-sequence-2
   CASE DEFAULT
      default-statement-sequence
END SELECT
```

  where `selector` is an integer, character, or logical expression, and each `label-list` is one or more possible values of the selector.

Fortran statements: SELECT-CASE

- If the value of `selector` is in `label-list-1` then `statement-sequence-1` is executed. If the value of `selector` is in `label-list-2` then `statement-sequence-2` is executed. Otherwise, `default-statement-sequence` is executed.

- Note that there can be any number of cases within a `SELECT` block.

- Like the ELSE portion of an IF statement, the DEFAULT case is optional, and is only executed if no other case matches.

- Each `label-list` can be in one of the following forms:

`value`	a single value
`value1:value2`	range of values from `value1` to `value2` inclusive
`value1:`	all values greater than or equal to `value1`
`:value2`	all values less than or equal to `value2`

Example 1: SELECT-CASE

```
PROGRAM Calculator
! This program reads in a simple arithmetic calculation,
! and computes and displays the result.
   IMPLICIT NONE
   CHARACTER::op
   INTEGER::k,m,result
   WRITE(*,*) 'Enter calculation'
   READ(*,*) k,op,m
   SELECT CASE (op)
      CASE ('+')
         result=k+m
      CASE ('-')
         result=k-m
      CASE ('/')
         result=k/m
      CASE ('*')
         result=k*m
      CASE DEFAULT
         WRITE(*,*) 'Invalid operator',op
   END SELECT
   WRITE(*,2) k,op,m,result
2  FORMAT(' ',I3,' ',A1,I3,' = ',I3)
END PROGRAM Calculator
```

Example 2: SELECT-CASE

```
PROGRAM Display_grade
! This program reads in an integer mark and
! displays the corresponding letter grade.
   IMPLICIT NONE
   INTEGER::mark
   WRITE(*,*) 'Please enter a grade'
   READ(*,*) mark
   SELECT CASE (mark)
      CASE (80:100)
         WRITE(*,*) 'A'
      CASE (65:79)
         WRITE(*,*) 'B'
      CASE (55:64)
         WRITE(*,*) 'C'
      CASE (50:54)
         WRITE(*,*) 'D'
      CASE (0:49)
         WRITE(*,*) 'F'
      CASE DEFAULT
         WRITE(*,*) 'Invalid mark, not within [0,100]'
   END SELECT
END PROGRAM Display_grade
```