COMP 208
Computers in Engineering

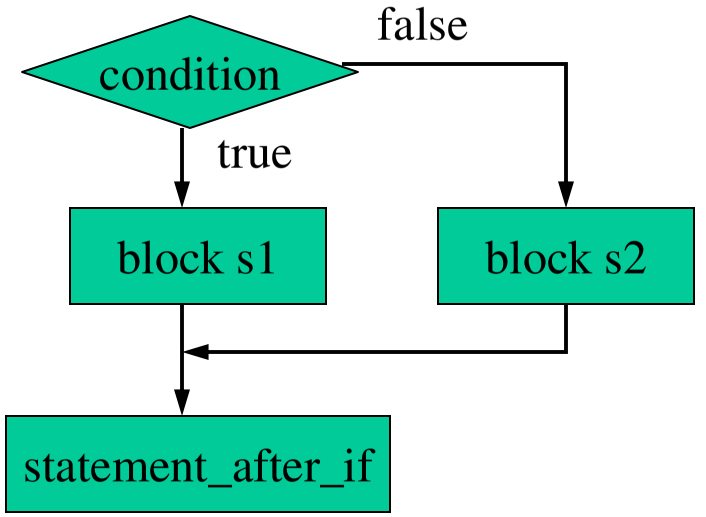
Lecture 07

Jun Wang
School of Computer Science
McGill University
Fall 2007

Review

- IF-THEN-ELSE-END IF

```
IF (logical-expression) THEN
     first statement block s1
ELSE
     second statement block s2
END IF
```



Review

- `IF-ELSE IF` construct

```
IF (logical-exp, e1) THEN
    statement block, s1
ELSE IF (logical-exp, e2) THEN
    statement block, s2
ELSE IF (logical-exp, e3) THEN
    statement block, s3
  . . . . . .
ELSE
    statement block, se
END IF
```

Review

- `STOP` statement: terminates program
- Logical operators

```
.NOT.
.AND.
.OR.
.EQV., .NEQV.
```

  - precedence of ALL logical operators lower than arithmetic operators

- The CHARACTER type:

```
CHARACTER (LEN=n) :: s
```

Nested-IF vs. IF-ELSE IF

```
IF (n == 1) THEN
    WRITE(*,*) "One"
ELSE
    IF (n == 2) THEN
        WRITE(*,*) "Two"
    ELSE
        IF (n == 3) THEN
            WRITE(*,*) "Three"
        ELSE
            WRITE (*,*) "Unknown"
        END IF
    END IF
END IF
```

```
IF (n == 1) THEN
    WRITE(*,*) "One"
ELSE IF (n == 2) THEN
    WRITE(*,*) "Two"
ELSE IF (n == 3) THEN
    WRITE(*,*) "Three"
ELSE
    WRITE (*,*) "Unknown"
END IF
```

- Semantically equivalent
- Syntactically, IF-ELSE IF slightly easier to read and write.

IF statement: conditional execution

```
IF (month == 1 .OR. month == 3 .OR. month == 5 .or.
       month == 7 .OR. month == 8 .OR. month == 10 .or.
       month == 12) THEN
  lastday =31
ELSE IF (month == 4 .OR. month == 6 .OR. month == 9 .OR.
            month == 11) THEN
  lastday =30
ELSE IF ((mod(year,4) == 0 .AND. mod(year,100) /= 0) .OR.
             mod(year,400) == 0) THEN
  lastday = 29  ! February of leap year
ELSE
  lastday = 28
END IF
```

The value of `lastday` is dependent on the value of `month`.

Exercises on IF construct

Which of the following are legal logical IF statement?

```
IF (x > y) WRITE (*,*) 2A
```
2*A

```
IF x < y  x = x + 1
```
(x < y)

```
IF (x = y)  READ (*,*) z
```
x == y

```
IF (0 < x < 10) x = x + 1
```
0<x .AND. x < 10

```
IF (a >= 0) THEN
   IF (b >= 0) THEN
      WRITE (*,*) a+b
   ELSE
      WRITE (*,*) a*b
   END IF
ELSE
   WRITE (*,*) a-b
END IF
```

What's the output if:

a is 3, b is 2 — 5

a is 3, b is -2 — -6

a is -3, b is 2 — -5

Integer division

1. Get 2 numbers: numerator, denominator
2. Compute and output numerator/denominator

```
WRITE (*,*) "Enter denominator:"
READ (*,*) denominator

IF (denominator == 0) THEN
  WRITE (*,*) "The denominator may not be 0”
ELSE  !perform calculation
  output = nominator / denominator
  WRITE (*,*) "The result is: ”, output
END IF
! End of program
```

What if we want to give user another chance after 0 is entered as denominator?

Nested if-statement?

```
WRITE (*,*) "Enter denominator:"
READ (*,*) denominator
IF (denominator == 0) THEN
  WRITE (*,*) "The denominator may not be 0"
  WRITE (*,*) "Enter denominator: "
  READ (*,*) denominator
  IF (denominator == 0) THEN
      WRITE (*,*) "The denominator may not be 0"
  ELSE
    output = nominator / denominator;
    WRITE (*,*) "The result is: ", output
  END IF
ELSE
  output = nominator / denominator;
  WRITE (*,*) "The result is: ", output
END IF
```

How about we give user one more chance?

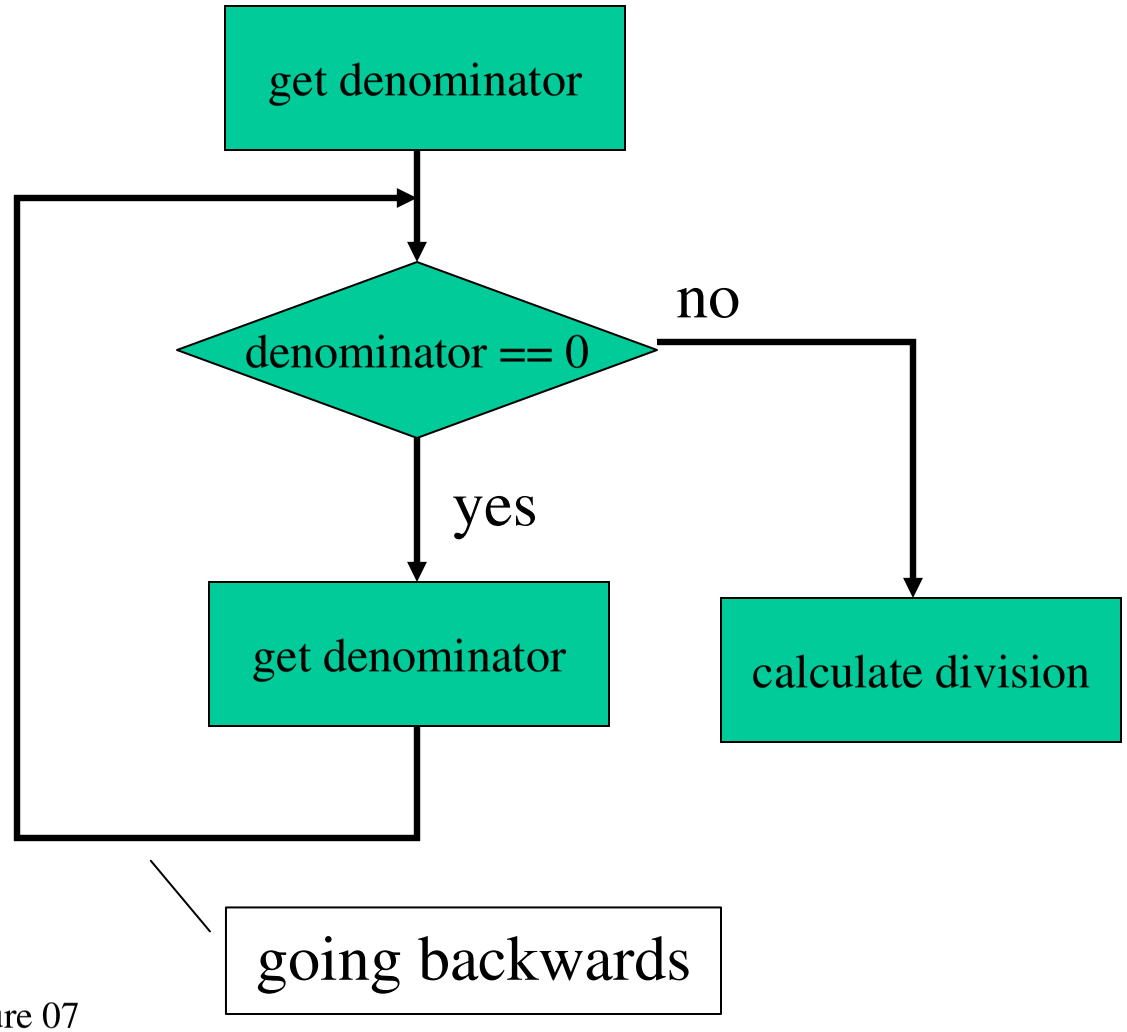
Nested if-statement? (cont'd)

```
WRITE (*,*) "Enter denominator:"
READ (*,*) denominator
IF (denominator == 0) THEN
  WRITE (*,*) "The denominator may not be 0"
  WRITE (*,*) "Enter denominator:"
  READ (*,*) denominator
  IF (denominator == 0) THEN
      WRITE (*,*) "The denominator may not be 0"
      WRITE (*,*) "Enter denominator:"
      READ (*,*) denominator
      IF (denominator == 0) THEN
        WRITE (*,*) "The denominator may not be 0"
      ELSE
        ! calculate and print
      END IF
  ELSE
    ! calculate and print
  END IF
ELSE
  ! calculate and print
END IF
```

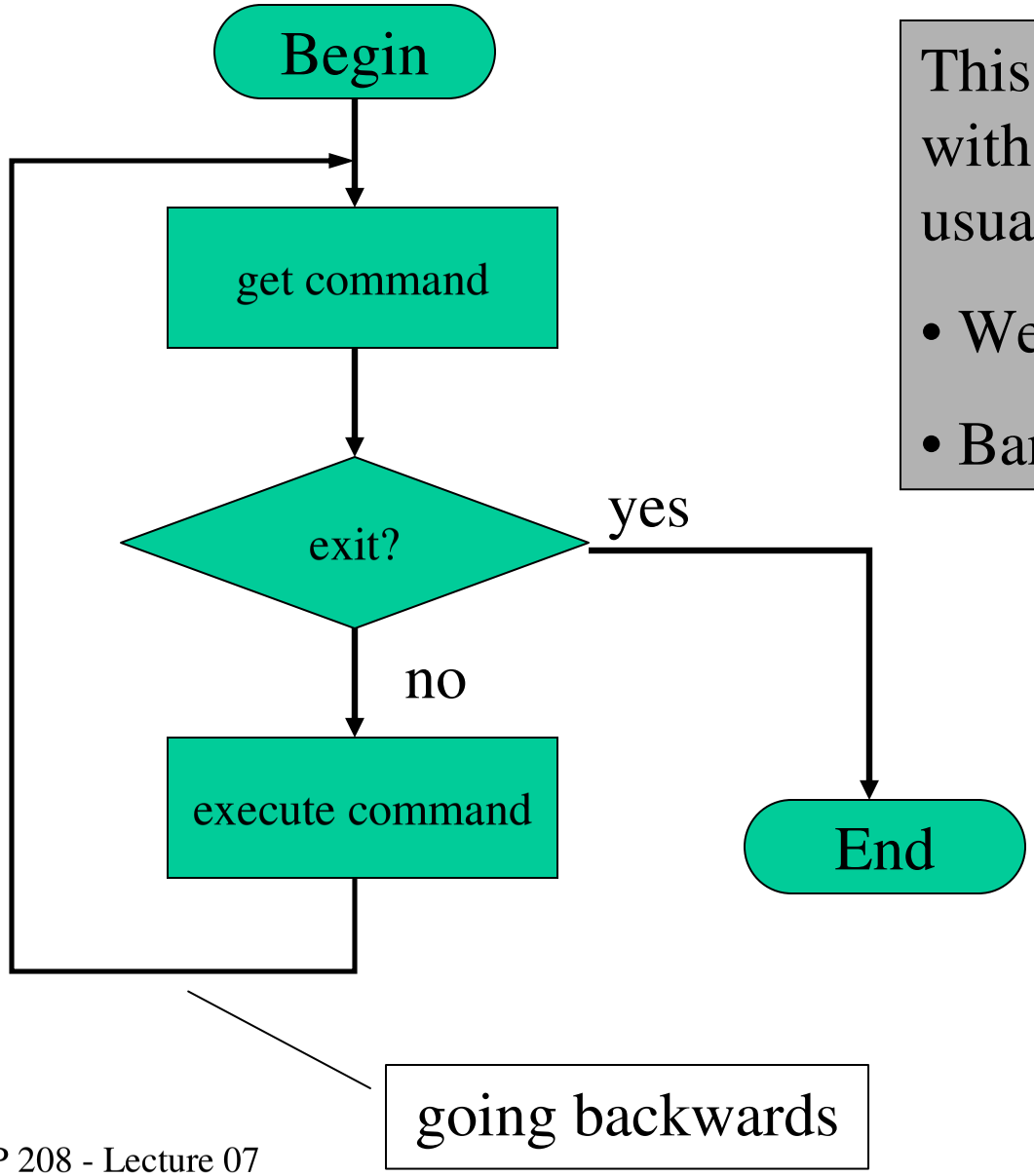
What if we only proceed when denominator is not 0?

nested if won't help!
need something new!

Flow-chart for getting non-0 denominator



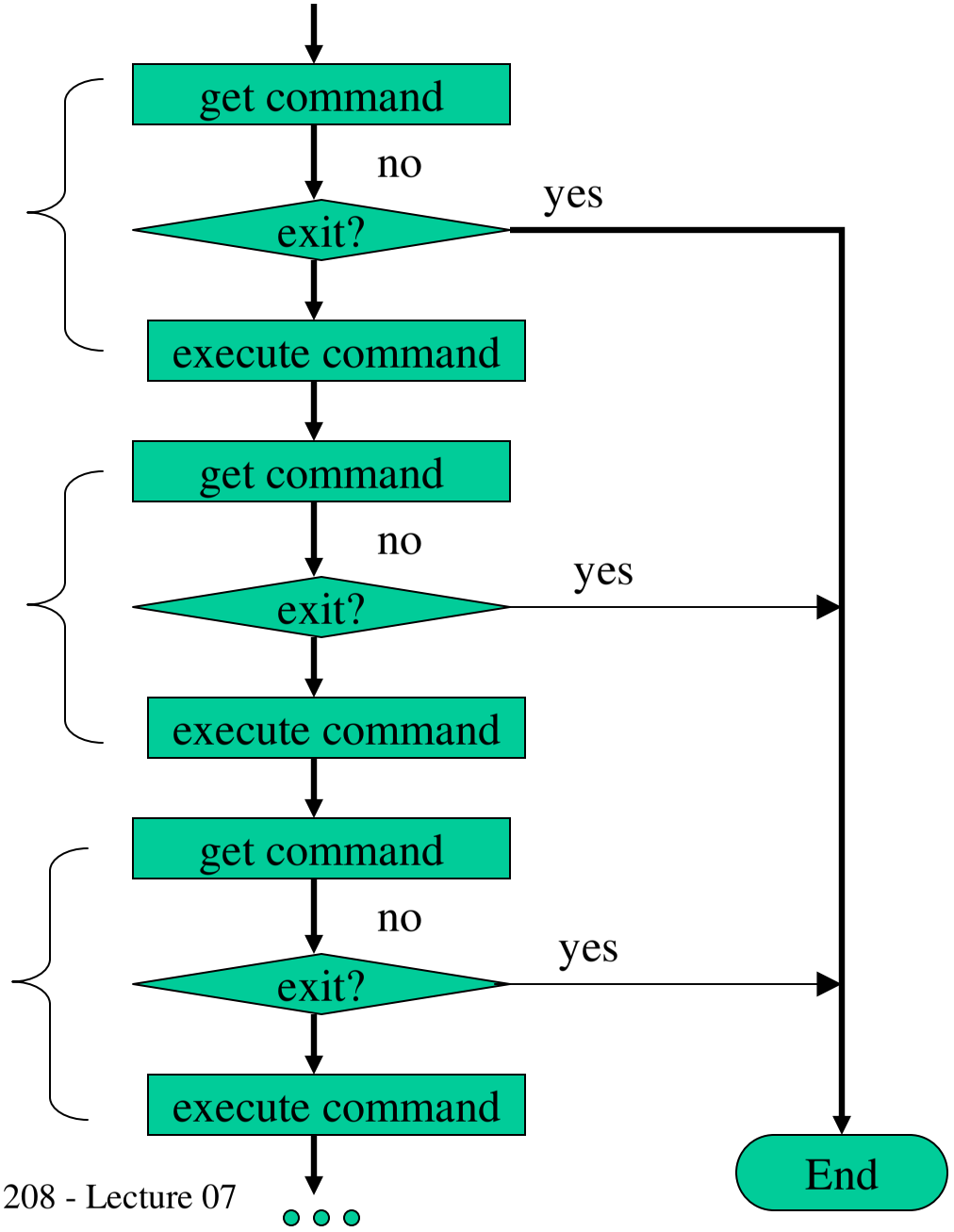
Windows "Command Prompt"



This is how a program with a user interface usually works:

- Web browser
- Banking machine

Windows "Command Prompt"



How many times do we repeat the code?

A Table of Values

- Problem: Output a table of numbers from 1 to 100 with their squares and cubes

```
     1       1       1
     2       4       8
     3       9      27
     4      16      64
          .   .   .
```

```
...
WRITE (*,*) 1, 1*1, 1*1*1
WRITE (*,*) 2, 2*2, 2*2*2
WRITE (*,*) 3, 3*3, 3*3*3
...
```

- We have to be able to repeat a computation over and over for the different numbers without writing 100 `WRITE` statements

A Table of Values

```
INTEGER :: Num

DO Num = 1, 100
  WRITE(*,*) Num, Num*Num, Num*Num*Num
END DO
```

Semantics:

- Repeatedly executing the “WRITE” statement, for Num from 1 to 100.

Counter-controlled loops

The syntax of a **definite iterator** (often called a counted DO loop) is:

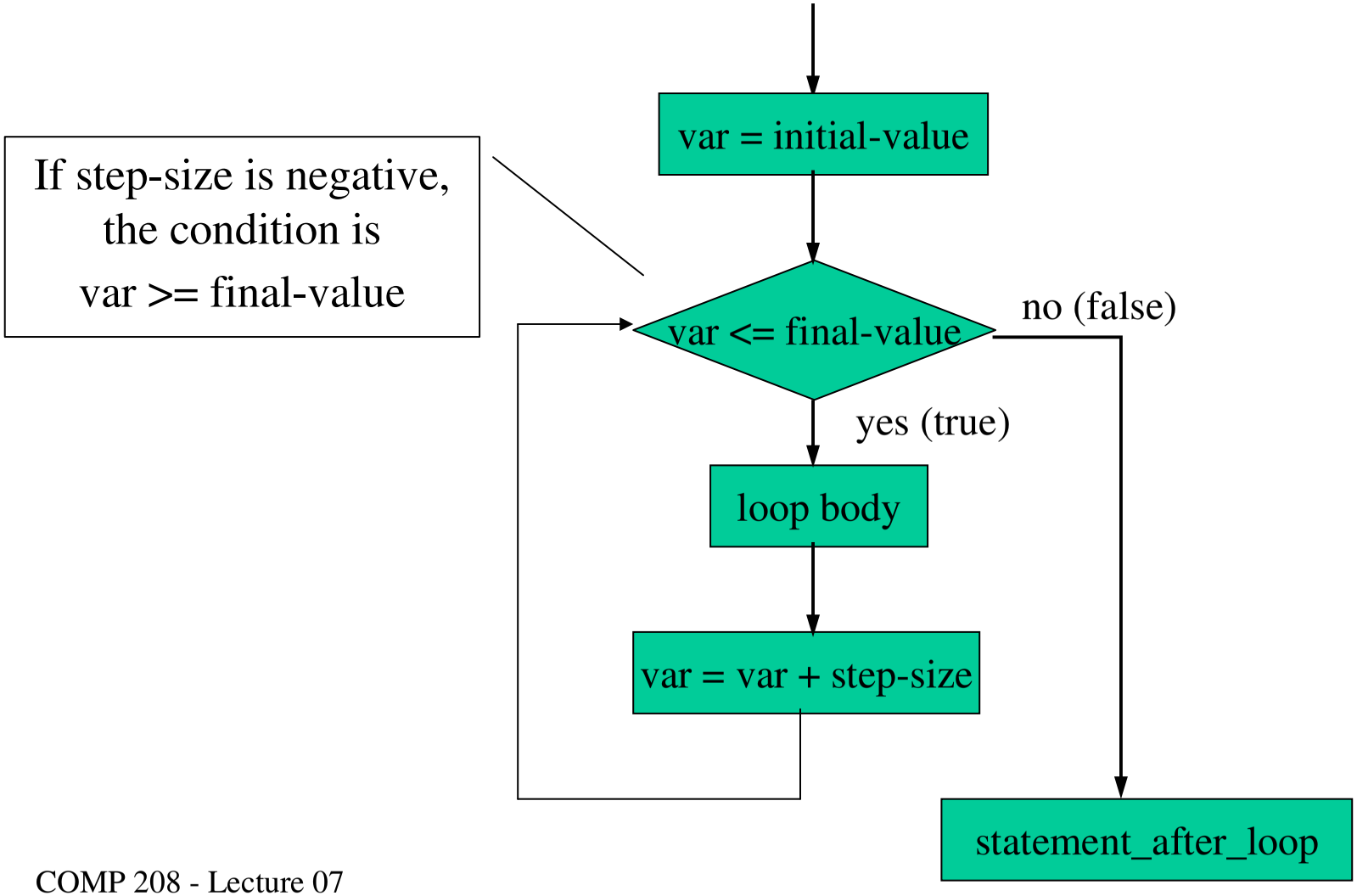
```
DO var = initial-value, final-value, step-size
  statement block, s
END DO
```

- `var` is an INTEGER variable called the control variable
- `initial-value` and `final-value` are INTEGER expressions
- `step-size` is an **optional** INTEGER expression. If omitted the default value is 1

Semantics of Counted DO Initialisation

1. Evaluate the initial, final and step-size expressions.
   - These can be any expressions that give an integer value
   - They are evaluated only once before the loop is entered
2. The step-size should not be 0.
   - The default value if the step size is omitted is 1

Flow-chart of counter-controlled loop



Semantics of Counted DO (Counting Up)

1. If step-size > 0, the loop counts up
   1. var = initial value
   2. If (var <= final value) then
      Execute the statement block, s
      var = var + step-size
      Repeat step 2
   3. When var > final value, the loop ends and the statement after the END DO is executed

Print “hello” 3 times: counting up

```
PROGRAM hello3
  IMPLICIT NONE
  INTEGER :: I

  DO i=1, 3, 1
    WRITE (*,*) “hello!”
  END DO

END PROGRAM hello3
```

When step size is 1, it can be omitted

Semantics of Counted DO (Counting Down)

1. If step-size < 0, the loop counts down
   1. var = initial value
   2. If (var >= final value) then
      Execute the statement block, s
      var = var + step-size (negative)
      Repeat step 2
   3. When var < final value, the loop ends and the statement after the END DO is executed

only difference from counting up is this condition

Print “hello” 3 times: counting down

```
PROGRAM hello3
  IMPLICIT NONE
  INTEGER :: I

  DO i=3, 1, -1
    WRITE (*,*) “hello!”
  END DO

END PROGRAM hello3
```

Table of Odd Numbers

Output the **odd numbers** between 1 and 100, their squares and cubes.

```
INTEGER :: Num

DO Num = 1, 100, 2
  WRITE(*,*) Num, Num*Num, Num*Num*Num
END DO
```

Temperature Conversions

Print a table of Celsius to Fahrenheit conversions:

```
INTEGER :: Celsius
REAL :: Fahrenheit

DO Celsius = -40, 40
  Fahrenheit = 1.8 * Celsius + 32.0
  WRITE(*,*) Celsius, " degrees Celsius = ", &
             Fahrenheit, " degrees Fahrenheit"
END DO
```

Note the negative initial value. The step size is 1.

Table in Descending Order

```
INTEGER :: Celsius
REAL :: Fahrenheit

DO Celsius = 40, -40, -1
 Fahrenheit = 1.8 * Celsius + 32.0
 WRITE(*,*) Celsius, " degrees Celsius = ", &
     Fahrenheit, " degrees Fahrenheit"
END DO
```

Note the negative step size.

Average Value

Input 1000 real numbers and compute the average value:

```
INTEGER :: Count, Number=1000
REAL :: Sum, Input
REAL :: Average

Sum = 0.0
DO Count = 1, Number
  READ(*,*) Input
  Sum = Sum + Input
END DO
Average = Sum / Number
```

Definite Iterator

- The DO loop we have looked at is called a **definite iterator**
- The body of the loop is executed a fixed number of times
- The control variable, `i`, takes on the values `x, x+s, x+2s, …, x+ks` where
  - `x` is the initial value,
  - `s` is the step size and
  - `x+ks` $\leqslant$ final value < `x+(k+1)s`

Arrays

Processing Lists

- Counted do loops are used extensively in processing lists of data
- In the next application, we will see how to represent a list of data in a way that allows us to go through each value in the list using a do loop

ISBN Numbers

- ISBN numbers assign a unique identification number to every book published
- As with many such identification numbers, such as UPC codes, Postal Money Order serial numbers, Credit card numbers, there is a self checking code that allows us to reduce scanning and transmission errors

10 Digit ISBN Codes

An ISBN consists of 10 digits (newer standards will have 13 digits)

For example: `0-7872-9390-3`

1. The first digit is a country or language code
2. The next group of digits is the publisher
3. The next group is the item number
4. The final digit is a check digit

(The lengths of groups 2 and 3 may vary)

The Check Digit

To calculate the check digit the International ISBN Agency specifies that

1. For each of the first nine digits, we multiply the digit by a weight depending on the position of the digit that goes from 10 down to 1
2. We then sum these products
3. The check digit is the number that, if added, will make this sum a multiple of 11

Verifying ISBN Numbers

```
PROGRAM isbn
   IMPLICIT NONE
   INTEGER :: digits(10)
   INTEGER :: pos, sum
   INTEGER :: check

   READ (*,*) digits
   sum = 0

   DO pos = 1,10
        sum = sum + (11-pos)*digits(pos)
   END DO

   check = mod(sum,11)
   IF (check == 0) THEN
        write(*,*) "ISBN is valid"
   ELSE
        write(*,*) "ISBN is invalid"
   END IF
END PROGRAM isbn
```

an array of 10 integers

Verifying ISBN Numbers (with logical variables)

```
PROGRAM isbn
   IMPLICIT NONE
   INTEGER :: digits(10)
   INTEGER :: pos, sum
   LOGICAL :: valid

   READ (*,*) digits
   sum = 0
   DO pos = 1,10
       sum = sum + (11-pos)*digits(pos)
   END DO

   valid = mod(sum,11) == 0
   IF (valid) THEN
       write(*,*) "ISBN is valid"
   ELSE
       write(*,*) "ISBN is invalid"
   END IF
END PROGRAM isbn
```

relational operators have higher precedence than assignment