# 4.5: The Structure Theorem

IT1406 - Introduction to Programming

Level I - Semester 1

# 4.5. The Structure Theorem

- The Structure Theorem revolutionized program design by establishing a structured framework for representing a solution algorithm.
- The Structure Theorem states that it is possible to write any computer program by using only three basic control structures that are easily represented in pseudocode:

  sequence,

  selection and

  repetition.

# 4.5. The Structure Theorem

## The three basic control structures

### 1. Sequence

- The sequence control structure is the straightforward execution of one processing step after another.
- In pseudocode, this construct is represented as a sequence of pseudocode statements:

```
statement a
statement b
statement c
```

# 1. Sequence

- The sequence control structure can be used to represent the first **four** basic computer operations listed previously:
  - to receive information,
  - put out information,
  - perform arithmetic, and
  - assign values.
- For example, a typical sequence of statements in an algorithm might read:

```
add 1 to pageCount
Print heading line1
Print heading line2
Set lineCount to zero
Read customer record
```

# 1. Sequence (cont.)

- These instructions illustrate the sequence control structure as a straightforward list of steps written one after the other, in a top-to-bottom fashion.
- Each instruction will be executed in the order in which it appears.

# 2. Selection

- The selection control structure is the presentation of a condition and the choice between two actions, the choice depending on whether the condition is true or false.
- This construct represents the decision-making abilities of the computer and is used to illustrate the fifth basic computer operation, namely to compare two variables and select one of two alternative actions.

# 2. Selection (cont.)

- The condition in the IF statement is based on a comparison of two items, and is usually expressed with one of the following relational operators:

< less than

> greater than

= equal to

<= less than or equal to

>= greater than or equal to

<> not equal to

# 2. Selection (cont.)

- In pseudocode, selection is represented by the keywords IF, THEN, ELSE and ENDIF:

```
IF condition p is true THEN
      statement(s) in true case
ELSE
      statement(s) in false case
ENDIF
```

# 2. Selection (cont.)

- If condition p is true, then the statement or statements in the true case will be executed, and the statements in the false case will be skipped.
- Otherwise (the ELSE statement) the statements in the true case will be skipped and statements in the false case will be executed.
- In either case, control then passes to the next processing step after the delimiter ENDIF.

# 2. Selection (cont.)

- typical pseudocode example for selection might read:

```
IF student_attendance_status is part_time THEN
      add 1 to part_time_count
ELSE
      add 1 to full_time_count
ENDIF
```

# 2. Selection (cont.)

There are a number of variations of the selection structure, as follows.

**1. Simple selection (simple IF statement)**

```
IF account_balance < $300 THEN
      service_charge = $5.00
ELSE
      service_charge = $2.00
ENDIF
```

# 2. Selection (cont.)

There are a number of variations of the selection structure, as follows.

**2. Simple selection with null false branch (null ELSE statement)**

```
IF student_attendance = part_time THEN
        add 1 to part_time_count
ENDIF
```

# 2. Selection (cont.)

There are a number of variations of the selection structure, as follows.

**3. Combined selection (combined IF statement)**

In this case, each student record will undergo two tests.

```
IF student_attendance = part_time AND student_gender = female
THEN
        add 1 to female_part_time_count
ENDIF
```

```
IF student_attendance = part_time OR student_gender = female
THEN
        add 1 to female_part_time_count
ENDIF
```

# 2. Selection (cont.)

There are a number of variations of the selection structure, as follows.

**4. Nested selection (nested IF statement)**

Nested IF statements can be classified as linear or non-linear.

# 2. Selection (cont.)

There are a number of variations of the selection structure, as follows.

**4. Nested selection (nested IF statement)**

**Linear nested IF statements**

- The linear nested IF statement is used when a field is being tested for various values and a different action is to be taken for each value.
- This form of nested IF is called linear, because each ELSE immediately
- follows the IF condition to which it corresponds.
- Comparisons are made until a true condition is encountered, and the specified action is executed until the next ELSE statement is reached.
- Linear nested IF statements should be indented for readability, with each IF, ELSE and corresponding ENDIF aligned.

# 2. Selection (cont.)

There are a number of variations of the selection structure, as follows.

**4. Nested selection (nested IF statement)**

**Linear nested IF statements**

```
IF record_code = 'A' THEN
      increment counter_A
ELSE
      IF record_code = 'B' THEN
            increment counter_B
       ELSE
            IF record_code = 'C' THEN
                  increment counter_C
            ELSE
                  increment error_counter
            ENDIF
       ENDIF
ENDIF
```

# 2. Selection (cont.)

There are a number of variations of the selection structure, as follows.

**4. Nested selection (nested IF statement)**

**Linear nested IF statements**

- Note that there are an equal number of IF, ELSE and ENDIF statements, that each ELSE and ENDIF statement is positioned so that it corresponds with its matching IF statement, and that the correct indentation makes it easy to read and understand.
- A block of nested IF statements like this is sometimes referred to as 'cascading IF statements', as they cascade like a waterfall across the page.

# 2. Selection (cont.)

There are a number of variations of the selection structure, as follows.

**4. Nested selection (nested IF statement)**

**Non-linear nested IF statements**

- A non-linear nested IF statement occurs when a number of different conditions need to be satisfied before a particular action can occur.
- It is termed nonlinear because the ELSE statement may be separated from the IF statement with which it is paired.
- Indentation is once again important when expressing this form of selection in pseudocode.
- Each ELSE and ENDIF statement should be aligned with the IF condition to which it corresponds.

# 2. Selection (cont.)

There are a number of variations of the selection structure, as follows.

**4. Nested selection (nested IF statement)**
**Non-linear nested IF statements**

```
IF student_attendance = part_time THEN
        IF student_gender = female THEN
                IF student_age > 21 THEN
                        add 1 to mature_female_pt_students
                ELSE
                        add 1 to young_female_pt_students
                ENDIF
        ELSE
                add 1 to male_pt_students
        ENDIF
ELSE
          add 1 to full_time_students
ENDIF
```

# 2. Selection (cont.)

There are a number of variations of the selection structure, as follows.

**4. Nested selection (nested IF statement)**

**Non-linear nested IF statements**

- Note that the number of IF conditions is equal to the number of ELSE and ENDIF statements.
- Using correct indentation helps to see which set of IF, ELSE and ENDIF statements match.
- However, non-linear nested IF statements may contain logic errors that are difficult to correct, so they should be used sparingly in pseudocode.
- If possible, replace a series of non-linear nested IF statements with a combined IF statement.
- This replacement is possible in pseudocode because two consecutive IF statements act like a combined IF statement that uses the AND operator.

# 2. Selection (cont.)

There are a number of variations of the selection structure, as follows.

**4. Nested selection (nested IF statement)**

**Non-linear nested IF statements**

- Take as an example the following nonlinear nested IF statement:

```
IF student_attendance = part_time

        IF student_age > 21 THEN
            increment mature_pt_student
        ENDIF
ENDIF
```

- This can be written as a combined IF statement:

```
IF student_attendance = part_time AND student_age > 21 THEN
            increment mature_pt_student
ENDIF
```

# 2. Selection (cont.)

There are a number of variations of the selection structure, as follows.

**4. Nested selection (nested IF statement)**

**Non-linear nested IF statements**

- The outcome will be the same for both pseudocode expressions, but the format of the latter is preferred, if the logic allows it, simply because it is easier to understand.

# 2. Selection (cont.)

**Algorithms using selection**

- Let us look at some programming examples that use the selection control structure.
- In each example, the problem will be defined, a solution algorithm will be developed and the algorithm will be manually tested.
- To help define the problem, the processing verbs in each example have been underlined.

# 2. Selection (cont.)

## Algorithms using selection

- Read three characters
  - Design an algorithm that will prompt a terminal operator for three characters, accept those characters as input, sort them into ascending sequence and output them to the screen.

# 2. Selection (cont.)

**Algorithms using selection**

- Read three characters
  - Defining algorithm

| Input | Processing | Output |
|---|---|---|
| char_1<br>char_2<br>char_3 | Prompt for characters<br>Accept three characters<br>Sort three characters<br>Output three characters | char_1<br>char_2<br>char_3 |

# 2. Selection (cont.)

**Algorithms using selection**

- Read three characters
  - ***Solution algorithm***
    - The solution algorithm requires a series of IF statements to sort the three characters into ascending sequence.

```
Read_three_characters
1       Prompt the operator for char_1, char_2, char_3
2       Get char_1, char_2, char_3
3       IF char_1 > char_2 THEN
            temp = char_1
            char_1 = char_2
            char_2 = temp
        ENDIF
```

# 2. Selection (cont.)

**Algorithms using selection**

- Read three characters
  - ***Solution algorithm***
    - The solution algorithm requires a series of IF statements to sort the three characters into ascending sequence.

```
4    IF char_2 > char_3 THEN
         temp = char_2
         char_2 = char_3
         char_3 = temp
     ENDIF
```

# 2. Selection (cont.)

**Algorithms using selection**

- Read three characters
  - ***Solution algorithm***
    - The solution algorithm requires a series of IF statements to sort the three characters into ascending sequence.

```
5       IF char_1 > char_2 THEN
            temp = char_1
            char_1 = char_2
            char_2 = temp
        ENDIF
6       Output to the screen char_1, char_2, char_3
END
```

# 2. Selection (cont.)

**Algorithms using selection**

Process customer record

- A program is required to read a customer's name, a purchase amount and a tax code.
- The tax code has been validated and will be one of the following:

0 tax exempt (0%)

1 state sales tax only (3%)

2 federal and state sales tax (5%)

3 special sales tax (7%)

# 2. Selection (cont.)

## Algorithms using selection

The program must then compute the sales tax and the total amount due, and print the customer's name, purchase amount, sales tax and total amount due.

# 2. Selection (cont.)

- **Algorithms using selection**
- ***Defining diagram***

| Input | Processing | Output |
|---|---|---|
| cust_name | Read customer details | cust_name |
| purch_amt | Calculate sales tax | purch_amt |
| tax_code | Calculate total amount | sales_tax |
| | Print customer details | total_amt |

# 2. Selection (cont.)

**Algorithms using selection**

***Defining diagram***

- ***Solution algorithm***
- The solution algorithm requires a linear nested IF statement to calculate the sales tax.

# 2. Selection (cont.)

**Algorithms using selection**

***Defining diagram***

```
        Process_customer_record
1               Read cust_name, purch_amt, tax_code
```

# 2. Selection (cont.)

**Algorithms using selection**

***Defining diagram***

```
2           IF tax_code = 0 THEN
                sales_tax = 0
            ELSE
                IF tax_code = 1 THEN
                    sales_tax = purch_amt * 0.03
                ELSE
                    IF tax_code = 2 THEN
                        sales_tax = purch_amt * 0.05
                    ELSE
                        sales_tax = purch_amt * 0.07
                    ENDIF
                ENDIF
            ENDIF
```

# 2. Selection (cont.)

**Algorithms using selection**

***Defining diagram***

```
3               total_amt = purch_amt + sales_tax
4               Print cust_name, purch_amt, sales_tax, total_amt
        END
```

# 2. Selection (cont.)

**The case structure**

- The case control structure in pseudocode is another way of expressing a linear nested IF statement. It is used in pseudocode for two reasons: it can be
- translated into many high-level languages, and it makes the pseudocode easier
- to write and understand. Nested IFs often look cumbersome in pseudocode
- and depend on correct structure and indentation for readability. Let us look
- at the example used earlier in this chapter:

# 2. Selection (cont.)

**The case structure**

```
IF record_code = 'A' THEN
      increment counter_A
ELSE
        IF record_code = 'B' THEN
                increment counter_B
        ELSE
                IF record_code = 'C' THEN
                        increment counter_C
                ELSE
                        increment error_counter
                ENDIF
        ENDIF
ENDIF
```

# 2. Selection (cont.)

**The case structure**

- This linear nested IF structure can be replaced with a case control structure.
- Case is not really an additional control structure.
- It simplifies the basic selection control structure and extends it from a choice between two values to a choice from multiple values.
- In one case structure, several alternative logical paths can be represented.
- In pseudocode, the keywords CASE OF and ENDCASE serve to identify the structure, with the multiple values indented, as follows:

# 2. Selection (cont.)

**The case structure**

```
CASE OF single variable
        value_1 : statement block_1
        value_2 : statement block_2
        .
        .
        .
        value_other : statement block_other
ENDCASE
```

# 2. Selection (cont.)

**The case structure**

- The path followed in the case structure depends on the value of the variable specified in the CASE OF clause.
- If the variable contains value_1, statement block_1 is executed; if it contains value_2, statement block_2 is executed, and so on.
- The value_other is included in the event that the variable contains none of the listed values.

# 2. Selection (cont.)

## The case structure

- We can now rewrite the above linear nested IF statement with a case statement, as follows:

```
CASE OF record_code
    'A' : increment counter_A
    'B' : increment counter_B
    'C' : increment counter_C
    other : increment error_counter
ENDCASE
```

# 2. Selection (cont.)

**Algorithms using selection**

- Process customer record
- A program is required to read a customer's name, a purchase amount and a tax code. The tax code has been validated and will be one of the following:

0 tax exempt (0%)

1 state sales tax only (3%)

2 federal and state sales tax (5%)

3 special sales tax (7%)

- The program must then compute the sales tax and the total amount due, and print the customer's name, purchase amount, sales tax and total amount due.

# 2. Selection (cont.)

## Algorithms using selection

- ***Defining diagram***

| Input | Processing | Output |
|---|---|---|
| cust_name | Read customer details | cust_name |
| purch_amt | Calculate sales tax | purch_amt |
| tax_code | Calculate total amount | sales_tax |
| | Print customer details | total_amt |

# 2. Selection (cont.)

**Algorithms using selection**

- ***Solution algorithm***
- The solution algorithm will be expressed using a CASE statement.

```
        Process_customer_record
1           Read cust_name, purch_amt, tax_code
2               CASE OF tax_code
                    0 : sales_tax = 0
                    1 : sales_tax = purch_amt * 0.03
                    2 : sales_tax = purch_amt * 0.05
                    3 : sales_tax = purch_amt * 0.07
                ENDCASE
3               total_amt = purch_amt + sales_tax
4               Print cust_name, purch_amt, sales_tax, total_amt
        END
```

# 3. Repetition

- The repetition control structure can be defined as the presentation of a set of instructions to be performed repeatedly, as long as a condition is true.
- The basic idea of repetitive code is that a block of statements is executed again and again, until a terminating condition occurs.
- There are three different ways in which a set of instructions can be repeated, and each way is determined by where the decision to repeat is placed:

# 3. Repetition (cont.)

- at the beginning of the loop (leading decision loop)
- at the end of the loop (trailing decision loop)
- a counted number of times (counted loop).

# 3. Repetition (cont.)

- This construct represents the sixth basic computer operation, namely to repeat a group of actions.
- It is written in pseudocode as:

```
DOWHILE condition p is true
 statement block
ENDDO
```

# 3. Repetition (cont.)

- The DOWHILE loop is a leading decision loop; that is, the condition is tested before any statements are executed.
- If the condition in the DOWHILE statement is found to be true, the block of statements following that statement is executed once.
- The delimiter ENDDO then triggers a return of control to the retesting of the condition.
- If the condition is still true, the statements are repeated, and so the repetition process continues until the condition is found to be false.
- Control then passes to the statement that follows the ENDDO statement.
- It is imperative that at least one statement within the statement block alters the condition and eventually renders it false, because otherwise the logic may result in an endless loop.

# 3. Repetition (cont.)

- Here is a pseudocode example that represents the repetition control structure:

```
Set student_total to zero
DOWHILE student_total < 50
    Read student record
    Print student name, address to report
    add 1 to student_total
ENDDO
```

# 3. Repetition (cont.)

This example illustrates a number of points:

**1.** The variable student_total is initialised before the DOWHILE condition is executed.

**2.** As long as student_total is less than 50 (that is, the DOWHILE condition is true), the statement block will be repeated.

**3.** Each time the statement block is executed, one instruction within that block will cause the variable student_total to be incremented.

**4.** After 50 iterations, student_total will equal 50, which causes the DOWHILE condition to become false and the repetition to cease.

- It is important to realize that the initializing and subsequent incrementing of the variable tested in the condition is an essential feature of the DOWHILE construct.

# 3. Repetition (cont.)

**Using DOWHILE to repeat a set of instructions a known number of times**

- When a set of instructions is to be repeated a specific number of times, a counter can be used in pseudocode, which is initialized before the DOWHILE statement and incremented just before the ENDDO statement.
- Let's look at an example.

# 3. Repetition (cont.)

**Using DOWHILE to repeat a set of instructions a known number of times**

- Fahrenheit–Celsius conversion Every day, a weather station receives 15 temperatures expressed in degrees Fahrenheit. A program is to be written that will accept each Fahrenheit temperature, convert it to Celsius and display the converted temperature to the screen. After 15 temperatures have been processed, the words 'All temperatures processed' are to be displayed on the screen.

# 3. Repetition (cont.)

**Repetition**

- **Using DOWHILE to repeat a set of instructions a known number of times**

***Defining diagram***

| Input | Processing | Output |
|---|---|---|
| f_temp<br>(15 temperatures) | Get Fahrenheit temperatures<br>Convert temperatures<br>Display Celsius temperatures<br>Display screen message | c_temp<br>(15 temperatures) |

# 3. Repetition (cont.)

**Using DOWHILE to repeat a set of instructions a known number of times**

- Having defined the input, output and processing, you are ready to outline a solution to the problem.
- This can be done by writing down the control structures needed and any extra variables that are to be used in the solution algorithm. In this example, you need:

→a DOWHILE structure to repeat the necessary processing

→a counter, called temperature_count, initialised to zero, that will control the 15 repetitions.

# 3. Repetition (cont.)

- **Using DOWHILE to repeat a set of instructions a known number of times**

***Solution algorithm***

```
        Fahrenheit_Celsius_conversion
1           Set temperature_count to zero
2           DOWHILE temperature_count < 15
3               Prompt operator for f_temp
4               Get f_temp
5               compute c_temp = (f_temp – 32) * 5/9
6               Display c_temp
7               add 1 to temperature_count
            ENDDO
8           Display 'All temperatures processed' to the screen
        END
```

# 3. Repetition (cont.)

**Using DOWHILE to repeat a set of instructions a known number of times**

This solution algorithm illustrates a number of points:

**1** The temperature_count variable is initialised before the DOWHILE condition is executed.

**2** As long as temperature_count is less than 15 (that is, the DOWHILE condition is true), the statements between DOWHILE and ENDDO will be executed.

**3** The variable temperature_count is incremented once within the loop, just before the ENDDO delimiter (that is, just before it is tested again in the DOWHILE condition).

**4** After 15 iterations, temperature_count will equal 15, which causes the DOWHILE condition to become false and control to be passed to the statement after ENDDO.

# 3. Repetition (cont.)

- **Repetition using the REPEAT…UNTIL structure**

**Trailing decision loop**

- The REPEAT…UNTIL structure is similar to the DOWHILE structure, in that a group of statements is repeated in accordance with a specified condition.
- However, where the DOWHILE structure tests the condition at the beginning of the loop, a REPEAT…UNTIL structure tests the condition at the end of the loop.
- This means that the statements within the loop will be executed once before the condition is tested. If the condition is false, the statements will be repeated UNTIL the condition becomes true.

The format of the REPEAT…UNTIL structure is:

```
REPEAT
statement
statement
.
.
.
UNTIL condition is true
```

# 3. Repetition (cont.)

**Repetition using the REPEAT…UNTIL structure**

**Trailing decision loop**

- REPEAT…UNTIL is a trailing decision loop; the statements are executed once before the condition is tested.
- There are two considerations of which you need to be aware before using REPEAT…UNTIL.
- First, REPEAT…UNTIL loops are executed when the condition is false; it is only when the condition becomes true that repetition ceases.
- Thus, the logic of the condition clause of the REPEAT…UNTIL structure is the opposite of DOWHILE. For instance, 'DOWHILE more records' is equivalent to 'REPEAT… UNTIL no more records', and 'DOWHILE number NOT = 99' is equivalent to 'REPEAT…UNTIL number = 99'.

# 3. Repetition (cont.)

## Repetition using the REPEAT…UNTIL structure

## Trailing decision loop

- Second, the statements within a REPEAT…UNTIL structure will always be executed at least once.
- As a result, there is no need for a priming Read when using REPEAT…UNTIL. One Read statement at the beginning of the loop is sufficient;
- however, an extra IF statement immediately after the Read statement must be included, to prevent the processing of the trailer record.
- Let us now compare an algorithm that uses a DOWHILE structure with the same problem using a REPEAT…UNTIL structure. Consider the following DOWHILE loop:

# 3. Repetition (cont.)

- **Repetition using the REPEAT…UNTIL structure**

**Trailing decision loop**

```
Process_student_records
        Set student_count to zero
        Read student record
        DOWHILE student_number NOT = 999
                Write student record
                increment student_count
                Read student record
        ENDDO
        Print student_count
END
```

# 3. Repetition (cont.)

## Repetition using the REPEAT…UNTIL structure

### Trailing decision loop

- This can be rewritten as a trailing decision loop, using the REPEAT…UNTIL structure as follows:

```
Process_student_records
        Set student_count to zero
        REPEAT
                Read student record
                IF student number NOT = 999 THEN
                        Write student record
                        increment student_count
                ENDIF
        UNTIL student number = 999
        Print student_count
END
```

# 3. Repetition (cont.)

**Repetition using the REPEAT…UNTIL structure**

**Trailing decision loop**

- Process inventory items
- A program is required to read a series of inventory records that contain an item number, an item description and a stock figure. The last record in the file has an item number of zero. The program is to produce a low stock items report, by printing only those records that have a stock figure of less than 20 items. A heading is to be printed at the top of the report and a total low stock item count printed at the end.

# 3. Repetition (cont.)

- **Repetition using the REPEAT…UNTIL structure**

| Input | Processing | Output |
|---|---|---|
| inventory record<br>• item_number<br>• item_description<br>• stock_figure | Read inventory records<br>Select low stock items<br>Print low stock records<br>Print total low stock items | heading<br>selected records<br>• item_number<br>• item_description<br>• stock_figure<br>total_low_stock_items |

# 3. Repetition (cont.)

**Repetition using the REPEAT…UNTIL structure**

- You will need to consider the following requirements when establishing a
- solution algorithm:

→ a REPEAT…UNTIL to perform the repetition

→ an IF statement to select stock figures of less than 20

→ an accumulator for total_low_stock_items

→ an extra IF, within the REPEAT loop, to ensure the trailer record is not processed.

# 3. Repetition (cont.)

**Repetition using the REPEAT…UNTIL structure**

***Solution algorithm using REPEAT…UNTIL***

```
        Process_inventory_records
1           Set total_low_stock_items to zero
2           Print 'Low Stock Items' heading
            REPEAT
3               Read inventory record
```

# 3. Repetition (cont.)

## Repetition using the REPEAT…UNTIL structure

- The solution algorithm has a simple structure, with a single Read statement at the beginning of the REPEAT…UNTIL loop and an extra IF statement within the loop to ensure the trailer record is not incorrectly incremented into the total_low_stock_items accumulator.

# 3. Repetition (cont.)

## Counted repetition

- **Counted loop**
- Counted repetition occurs when the exact number of loop iterations is known in advance.
- The execution of the loop is controlled by a loop index, and instead of using DOWHILE, or REPEAT…UNTIL, the simple keyword DO is used as follows:

```
DO loop_index = initial_value to final_value
 statement block
ENDDO
```

# 3. Repetition (cont.)

## Counted repetition

The DO loop does more than just repeat the statement block. It will:

**1** initialize the loop_index to the required initial_value

**2** increment the loop_index by 1 for each pass through the loop

**3** test the value of loop_index at the beginning of each loop to ensure that it is within the stated range of values

**4** terminate the loop when the loop_index has exceeded the specified final_ value.

- In other words, a counted repetition construct will perform the initialising, incrementing and testing of the loop counter automatically.
- It will also terminate the loop once the required number of repetitions has been executed.

# 3. Repetition (cont.)

**Counted repetition**

Fahrenheit–Celsius conversion

Every day, a weather station receives 15 temperatures expressed in degrees Fahrenheit. A program is to be written that will accept each Fahrenheit temperature, convert it to Celsius and display the converted temperature to the screen. After 15 temperatures have been processed, the words 'All temperatures processed' are to be displayed on the screen.

# 3. Repetition (cont.)

**Counted repetition**

Fahrenheit–Celsius conversion

| Input | Processing | Output |
|---|---|---|
| f_temp<br>(15 temperatures) | Get Fahrenheit temperatures<br>Convert temperatures<br>Display Celsius temperatures<br>Display screen message | c_temp<br>(15 temperatures) |

# 3. Repetition (cont.)

- **Counted repetition**

Fahrenheit–Celsius conversion

- ***Solution algorithm***
- The solution will require a DO loop and a loop counter (temperature_count) to process the repetition.

# 3. Repetition (cont.)

**Counted repetition**

```
        Fahrenheit–Celsius conversion
1           DO temperature_count = 1 to 15
2               Prompt operator for f_temp
3               Get f_temp
4               compute c_temp = (f_temp – 32) * 5/9
5               Display c_temp
            ENDDO
6           Display 'All temperatures processed' to the screen
        END
```

# 3. Repetition (cont.)

**Counted repetition**

- Note that the DO loop controls all the repetition:

→ It initialises temperature_count to 1.

→ It increments temperature_count by 1 for each pass through the loop.

→It tests temperature_count at the beginning of each pass to ensure that it is within the range 1 to 15.

→ It automatically terminates the loop once temperature_count has exceeded 15.