

**FORM 4 PASCAL PROGRAMMING****Unit 5: Repetitive Statements**

March, 01

**5.1 WHAT ARE ITERATION & LOOPS?**

- The ability of the computer to handle *repetitive tasks* simply and quickly makes it a powerful tool .
- In programming terms, repetitive tasks are done by \_\_\_\_\_ or \_\_\_\_\_.
- There are three types of looping (iteration) statements provided by PASCAL:
  - I. \_\_\_\_\_ loop
  - II. \_\_\_\_\_ loop
  - III. \_\_\_\_\_ loop

**5.2 FOR ... TO & FOR ... DOWNTO (RESERVED WORDS)**

- The FOR statement causes the <STATEMENT> (or several statements enclosed by BEGIN and END) after DO to be executed once for each value in the range FIRST to LAST.

*Syntax:***FOR..TO STATEMENT**

```
for counter := first to last do
  <statement>
```

- With TO, the value of the control variable is incremented (increased) by 1 for each loop.
- No iteration will be performed if FIRST > LAST

**FOR..DOWNTO STATEMENT**

```
for counter := first downto last do
  <statement>
```

- With DOWNTO, the value of the control variable is decrement (decreased) by 1 for each loop.
- No iteration will be performed if FIRST < LAST

*Remarks:*

- The control variable (e.g. COUNTER) and the initial (e.g. FIRST) and final values (e.g. LAST) must be of an ordinal type (e.g. integers, Boolean, char, etc.) and of the same data type;
- FOR-loop applies to the problems which have KNOWN number of repetitions at the time when the FOR-loop begins to execute.
- The value of the COUNTER (called control variable) can never be changed within the loop statements.

<b>Example 1</b>	<b>Output:</b>
<pre> var   count:integer; begin   for count :=1 to 5 do     writeln('Hello!'); </pre>	
<b>Example 2</b>	<b>Output:</b>
<pre> var   letter:char; begin   for letter := 'a' to 'z' do     write(letter);   writeln;   for letter := 'y' downto 'a' do     write(letter); </pre>	
<b>Example 3</b>	<b>Output:</b>
<pre> var   count:integer; begin   for count := 10 downto 1 do     writeln(count:5);     writeln('blast off!!!!') end. </pre>	
<b>Example 4</b>	<b>Output:</b>
<pre> type   color = (red,orange,yellow,green); var   c:color;   n,j:integer; begin   n:=0;   j:=0;   for c:=green downto red do     {repeat 4 times}     begin       n:=n+1;       j:=j+1     end;     writeln('n = ',n);     writeln('j = ',j); end. </pre>	<p>This program segment will increment the value of N and J both by one 4 times.</p> <p><b>Output:</b></p>

### 5.3 WHILE-DO (RESERVED WORD)

- A WHILE statement contains an expression that controls the repeated execution of a statement (which can be a compound statement).

*Syntax:*

```
WHILE <BOOLEAN-EXPRESSION> DO
  <STATEMENT>
```

*Remarks:*

- The statement after DO is executed repeatedly as long as the Boolean expression is True.
- The expression is evaluated BEFORE the statement is executed, so if the expression is False at the beginning, the statement is not executed at all it will exit the loop.
- It applies to problems that the number of repetitions is not known before. The number of loops is determined within the loop. In the following example, the number of executions depends on the statement `count := count + 1`.

<i>Example:</i>	<i>Output:</i>
<pre>var   count, limit: integer;  begin   count:=0;   limit:=15;   while count &lt;= limit do     begin       count:=count+1;       write(count:3)     end;     writeln;     writeln('end of while-loop')   end.</pre>	

## 5.4 REPEAT...UNTIL (RESERVED WORDS)

- The statements between REPEAT and UNTIL are executed in sequence until, at the end of a sequence, the Boolean expression is TRUE.

*Syntax:*

```
repeat
  <statement>;
  <statement>;
  ...
  <statement>
until <Boolean-expression>;
```

*Remarks:*

- As the Boolean expression is evaluated at the end of the loop, the sequence of statement(s) is executed at least once;
- Contrast to the other statements, the statements within REPEAT-loop need no enclosure by BEGIN and END.

<i>Example:</i>	<i>Output:</i>
<pre>var   count, limit: integer;  begin   count:=0;   limit:=15;   repeat     count:=count+1;     write(' ', count)   until count &lt;= limit;   writeln('end of repeat-loop') end.</pre>	

## 5.5 NESTING OF LOOPS (LOOP WITHIN LOOP)

- Controls structures (conditional or iterative statements) can be nested, one within another;
- e.g. We can place while-do loop, repeat-until loop, or for loop in the body of another loop

<b>Example:</b>	<b>Output:</b>
<pre> var   row,column:integer;  begin   for row := 1 to 3 do     begin       for column := 1 to 4 do         write('*');       writeln;     end;   writeln('end of nested for-loop') end. </pre>	

## 5.6 DECIDE WHICH LOOP TO USE

- The main differences among three types of looping statement are summarized as follows:

	<b>FOR-DO LOOP</b>	<b>WHILE-DO LOOP</b>	<b>REPEAT-UNTIL LOOP</b>
<i>Is the number of repetitions known?</i>	YES	NO	
<i>Where is the Boolean expression evaluated?</i>	---	at the beginning of the loop	at the end of the loop
<i>Minimum number of execution</i>	0	0	1

## 5.7 COMMON APPLICATIONS OF LOOPING STATEMENTS

- Looping statements play an important role in PASCAL programming, below are several examples on how we can use these looping statements to accomplish many simple but useful tasks:

- ◇ *Counting*
- ◇ *Summation / finding cumulative product*
- ◇ *Finding the maximum of a group of numbers*

### 5.7.1 Counting

e.g. To count the number of positive integers read from keyboard:

```
var
  num, count:integer;
begin
  write('enter a positive integer : '); *
  readln(num);
  count := 0; {initialize count to zero}
  while num>0 do
    begin
      count := count + 1; #
      write('enter a positive integer : ');
      readln(num)
    end {while}
  end.
```

\* NUM has to be read before entering the WHILE loop so that the Boolean expression containing it can be evaluated. These shaded input statements(\*) is called *priming read* and they are also inserted just before the end of WHILE loop;

# COUNT is used to count the number of positive integers inputted.

### 5.7.2 Summation / Finding Cumulative Product

e.g. To calculate the sum of the N input numbers

```
var
  i,N,x,sum:integer;
begin
  sum:=0; {initialize sum to zero}
  for i := 1 to N do
    begin
      write('enter an integer : ');
      readln(x);
      sum:=sum+x
    end; {for}
  end.
```

e.g. To calculate the cumulative product of the N input numbers:

```
var
  i,N,x,product:integer;
begin
  product:=1; {initialize product to one}
  for i := 1 to N do
    begin
      write('enter an integer : ');
      readln(x);
      product:=product*x
    end; {for}
  end.
```

### 5.7.3 Finding The Maximum Of A Group Of Numbers

e.g. To determine the maximum of a group of input positive numbers:

```
var
  max, num:integer;
begin
  max:=-1;
  repeat
    write('enter a positive integer ');
    write('<type -999 to end>: ');
    readln(num);
    if max<num then
      max:=num;
  until num=-999; {repeat-until}
end.
```

\* set MAX to a number that is smaller than all numbers in the set;

\* if MAX is smaller to any input number, the value of MAX should be replaced by that of NUM.

end of unit 5