

## Comments / Remarks (註解/註釋):

When we write a large program, say a few thousand lines of code, it is very difficult for one to memorize the whole body, meaning of each variable and what each part of code does actually do etc.

So we need some remarks to help us.

In Pascal, the **syntax for remarks** are enclosing the comments by a pair of { } or ( \* \* ).

Example BII1.pas:

Computers will ignore all remarks, they are for human beings only.

```
Program First;
Var i:integer;
Begin
i:=1; i:=i+1; i:=i+2; i:=i-1;
Writeln(i); {This will output 3}
End.
```

## Increment and Decrement:

You may notice there are 2 'i' in the statement "i:=i+2;".

This kind is called **self-increment** (自增) or **self-decrement** (自減).

That means *first, add 2 to the original value of i and then store it to i.*

Here is a more complex example of increment and decrement:

Read it carefully before you move to the next section.

Example BII2.pas:

**Try to find the output:** BIIe1.pas

```
Program Exer1;
Var i,j:integer;
Begin
i:=-6; j:=2; i:=i-1; j:=j+1;
j:=3-i*j;
i:=j div -i;
if i=4 then
case i of
1..4:writeln(i*i);
1..1000:writeln(i*i*i)
else
if i=3 then writeln(i);
end
Else if i=3 then writeln(i*i*i*i)
Else writeln(i*i+1);
End.
```

```
program Second;
var a,b,c,i,j,k,aa,bb,cc,dd:integer;
begin
a:=1;
b:=2;
a:=a+b;      { a is 3}
b:=a-b;      { b is 3-2 , so b is 1}
a:=a-b;      { a is 3-1 , so a is 2}
writeln(a,b); { This will output 21}
c:=(a+b)*b;
aa:=c*a;
bb:=aa+c;
cc:=bb-aa;
dd:=aa*bb*cc;
writeln(dd); {Output 162}
c:=c*c;
c:=c*c;
writeln(c); {Output 81}
end.
```

## Errors:

### Compile Error (編譯時出現的錯誤):

Errors occurred during compile time, which stop the compiler immediately.

e.g.

Syntax error (語法錯誤) : Missing a semi-colon, missing **end** in a **case...of** statement etc.

Type mismatch: Storing a real number to an integer type variable etc.

Data structure too large: Defining a structure larger than 64KB etc.

Too many variables: Declaring more than 64K of variables etc.

### Logical Error (邏輯錯誤):

Commonly known as “**bug**” in a program.

For example, mistyping a variable name.

### Runtime Error (程式運行時出現的致命錯誤):

**Fatal Error** occurred when a program is running, which usually **cause a program to die**.

For example, division by zero, input a string instead of an integer for a integer type.

### Precision Error (精密度錯誤):

Like your calculators, it can't properly handle numbers that are too large or too small.

If you force it to do so, it just stores those numbers by **approximation** etc.

Underflow: Storing a very very small real number to a variable. Like  $10^{-1000}$  etc.

Overflow: Storing a number larger than its maximum allowed value.

For example, store 2147483649 to a integer.

Examples:

SyntaxError.pas:

```
program syntax;
var i:integer;
begin
readln(i);
case i of
1:writeln('It is one. ');
1000..10000:writeln('It is large. ');
end.
{Missing end; }
```

RunTimeError.pas:

```
program run_time_error;
var a,b:integer;
begin
a:=100;
b:=0;
writeln(a/b);
end.
{Division by zero.}
```

## Predefined Function (函數) and Procedure (過程):

For F1 to F3 students, you may think a **function** as something that accepts some values and returns another value. For example, sine is one that *returns the ratio of two sides of a triangle with a given angle*.

You may consider **procedure** as some special command that will do some specific action. In Pascal **readln** is a procedure that *asks user to input N values and then store them to those N variables respectively*.

Occasionally, we need to perform complex calculation like taking natural logarithm, exponential, sine, cosine, square root, get a random number or get the length of a given string; or do action such as outputting things, deleting files.

Do we need to code those functions every time we use it?

Say, in order to find the value of  $\sin(x)$ , if we needed to do so, all would become very difficult. Every time we find  $\sin(x)$ , we have to find the limit of the infinite series

$$x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots + \frac{(-x)^n}{n!} \quad \text{or the equivalent form} \quad \sum_{i=0}^{\infty} \frac{(-1)^i x^{(2i+1)}}{(2i+1)!}$$

So, in Pascal, many functions and procedures are already defined.

Below is some of them. Notes that all procedures have no return value.

Name	Parameter (參數)	Return type	Description
Abs(x)	x : numeric type	Same as x	x
Arctan(x)	x : real	Real	$\tan^{-1}x$
Cos(x)	x : real	Real	cosx , x is in radian
Odd(x)	x : numeric type	Boolean	Return whether x is odd.
Random(x)	x : word	word	Return a integer y, $0 \leq y \leq x-1$
Random	No parameter	real	Return a real y, $0 < y < 1$
Round(x)	x : real	integer	Round up(四捨五入) x
Sin(x)	x : real	real	sinx , x is in radian
Sqr(x)	x : numeric	Same as x	$x^2$
Sqrt(x)	x : real	Real	$\sqrt{x}$
Trunc(x)	x : real	integer	Return value before decimal place
Writeln(x)	x : any type	No value	Output x
Readln(x)	x : variable	No value	Ask user to input and store it to x

## Formatting (格式):

In this section, I will use  $\nabla$  to stand for one space.

As people always like to see well-formatted output,  
In Pascal, there is an operator ( : ) to print formatted words etc.

1	4	5
1	32	22

<pre> Var I,J:integer; Begin I:=81; J:=3; Writeln( I : 5 , J:5 ); End.</pre>
--

This code will output  $\nabla\nabla\nabla 81\nabla\nabla\nabla 3$ .  
The colon in “I:5” means that *prepare 5 spaces first, then put the value of I align right* (向右對齊).

Explain: First, a computer will prepare  $\nabla\nabla\nabla\nabla\nabla$  and then put 81 to them, which become  $\nabla\nabla\nabla 81$ . Second, it will prepare another  $\nabla\nabla\nabla\nabla\nabla$  and then put 3 to them, getting  $\nabla\nabla\nabla\nabla 3$ . Finally, output them together, i.e.  $\nabla\nabla\nabla 81\nabla\nabla\nabla\nabla 3$ .

*If the length is greater than number of spaces prepared, just output them.*

*For example, `writeln(123435:1);` is the same as `writeln(123435);`*

You may notice that when you output a real number, the result is not what you expected like 1.58, 22.4447, it is outputted in the format  $\nabla 2.00000E+02$  etc.

When a computer output real number, it uses scientific notation instead of common one.

Remember **the first one must be the sign symbol**, *either a negative sign or a space.*

So, -122.5 is displayed as  $\nabla 1.2250000E+02$ , 122.5 is displayed as  $\nabla 1.2250000E+02$ .

But, how can we exactly output 122.5 to the screen?

Use two colons to do so.

In the statement “`writeln(i:0:1)`”, the first number means preparing 0 spaces for the output, the second one stands for correct the real number to 1 decimal place and then output.

<pre> var i:real; begin i:=122.5; writeln(i); writeln(i:0:1); end.</pre>
--

Consequently, if the statement is changed to “`writeln(i:0:0)`” 123 will be outputted.

Notes: In this case, unlike outputting in scientific notation where a space is printed even it's a positive number, the sign symbol is not important. Only when the number is negative, a negative sign is printed.

Also, Only Real Number can have a second colon.

## Conditional Looping I (條件性迴圈):

In many cases, we need to repeatedly do the same process or some similar process for many many times.

Here is one example:  
Output 10 times “Hello”.

Obviously, we can copy and paste 10 times.  
But how if I need 1000 times, or even a billion times?

I introduce a new concept here.

### Looping :

It can tell the computer to do some similar or same process for as many times as you want.

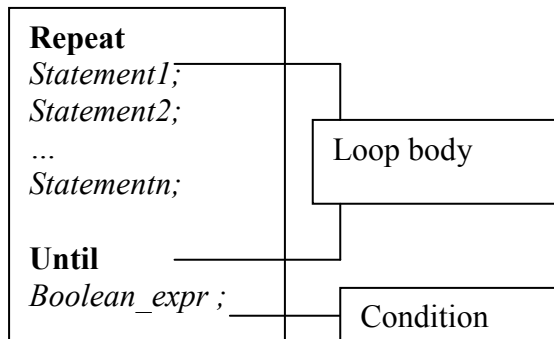
Stupid example.pas

```
Begin
writeln('Hello');
writeln('Hello');
writeln('Hello');
writeln('Hello');
writeln('Hello');
writeln('Hello');
writeln('Hello');
writeln('Hello');
writeln('Hello');
writeln('Hello');
End.
```

There are 2 conditional looping :

- 1) **repeat... until;**
- 2) **while ... do;**

### Syntax :



Flow:

- 1) Do the loop body first
- 2) Check whether the expr is true
- 3) If so, proceed to the next statement
- 4) Otherwise, go back step 1).

The **repeat ... until** statement means repeat to do those statements until the expr is true.

To write 10 times of “Hello”, we can count how many times “Hello” was outputted.  
So, repeat to write “Hello” until 10 lines have been written.

```
program hello;
var count:integer;
begin
count:=0; { there is no "Hello" printed at this moment }
repeat
writeln("Hello");
count:=count+1; {Self-increment count by 1}
until count=10;
end.
```



## Conditional Looping III (條件性迴圈):

Example:

Output m distinct integers from 1 to n. ( $0 < m \leq n$ )  
 ( n must be divided by m)

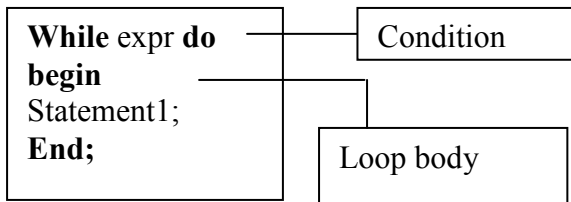
- 1) Ask user to input m and n.
- 2) Make use of the random function
- 3) Divide 1 to n into m intervals
- 3) Starting from 1 to m, generate a random number

```

program hello;
var m,n,i:integer;
begin
  readln(m,n);
  i:=0;
  repeat
    writeln(random(n div m)+i*n div m +1);
    i:=i+1;
  until i=m;
end.
    
```

**While...do:**

**Syntax :**



Flow:

- 1) Check whether the expr is true
- 2) If so, Do the loop body and go back to step 1).
- 3) Otherwise, proceed to the next statement

The **while ... do** statement means while the expr is true do the loop body.

Notes : *If you want to execute more than 1 statement for a **while...do** loop, use **begin...end;** to enclose the loop body.*

Example: Output all odd numbers between 1 and n.

<pre> program hello; var n,i:integer; begin   i:=1;   readln(n);   while i&lt;=n do   begin     writeln(i);     i:=i+2;   end; end.         </pre>	<p>{Try to write in <b>repeat...until</b>}</p>
--	--

**Exercise 1:**

Rewrite the above example with **while...do**:

Finish **ioexercise.doc** and **conditional.doc**.

**Exercise II:**

Without using computers, complete the following questions:

English	Chinese	Question No.
1999hje.doc	1999hjc.doc	4,5,10,16-18,20,23,29,30,32,35,45*
2000hje.doc	2000hjc.doc	4,5,9,10,14,15
2001hje.doc	2001hjc.doc	3,14,17,19,26,38
2003hje.doc	2003hjc.doc	A15,A30
2004hje.doc	2004hjc.doc	A10,A13,B5

**Exercise III \* :**

Without using computers, complete the following questions:

English	Chinese	Question No.
1999hje.doc	1999hjc.doc	28
2000hje.doc	2000hjc.doc	27
2003hje.doc	2003hjc.doc	A8,A16,B7,B11
2004hje.doc	2004hjc.doc	A2, A14

\* : Optional, sometimes very difficult and tricky, please spend time to think about it.