

## Input I:

∇ stands for a space. ↓ stands for end of line.  
End of line is usually a <enter> key.

How do computers read data in Pascal, when we use the **read** or **readln** statements?  
For each different type of variables, the computer use different ways to get the input.

**Here are the rules:**

```
Var N,N2,N3:integer; R,R2,R3:real;
    C,C2,C3:char;
    S,S2,S3:string;
```

### Integer type variable:

The blinking straight line is called cursor.

When the computer read an integer type variable,

It will first ignore all the spaces between the cursor and the first non-space character.  
Then it read starting from that non-space character until first space or end of line.

The whole thing is stored to n.

```
Read(n);
```

*\*Except the first one may be a negative sign, this thing can only consist of 0-9 only.  
Otherwise, a runtime error will occur due to type mismatch of assignment.*

Example:

User input 1

```
∇∇∇8889↓
```

```
Read(n);
```

N will store 8889 after the read statement.

User input 2

```
∇∇∇-88.89↓
```

A runtime error is produced as we try to store -88.89 to an integer variable.

### Character type variable:

It read exactly the first character, no matter what it is, and store to c.

User input 1

```
∇ASD8889↓
```

```
Read(c);
```

c will store ∇.

User input 2

```
ASD8889↓
```

```
Read(c);
```

c will store 'A'.

## Input II:

### Real type variable:

The same reading steps as integer.

But, the input can be in scientific notation or usual notation.

Example:

User input 1

```
▽▽▽888.9↓
```

```
Read(R);
```

R will store 888.9 after the read statement.

User input 2

```
▽▽▽8.889E+02↓
```

```
Read(R);
```

R will store 888.9 after the read statement..

### String type variable:

```
Read(s);
```

It read the whole line and store to S.

Example:

User input 1

```
AHJFSJ7476537^%$@&%$^@&$#@▽▽888.9↓
```

S will store ‘AHJFSJ7476537^%\$@&%\$^@&\$#@▽▽888.9’ after the statement

```
Read(s);
```

### Readln

If you use

```
Read(n);Read(n2);Readln(n3);
```

Then n will store 1 , n2 store 8 , n3 store 15.

But if you use

```
Readln(n);Readln(n2);Readln(n3)
```

Then n will store 1 , n2 store 2 , n3 store 3.

### Sample Input

```
1▽8▽15↓
2▽9▽16↓
3▽10▽17↓
4▽11▽18↓
5▽12▽19↓
6▽13▽20↓
7▽14▽21↓
```

Readln means after reading go to prepare to read the next line, so after “Readln(n);” the remaining “▽8▽15↓” is ignored , that’s why n2 store 2 instead of 8.

*Only this 4 types can perform read or readln.*

## Conditional Looping IV:

Revision Exercise: Reverse the digit of integer n and output it.

*This example shows you another way of using **while..do** , **repeat...until**.*

Step1: Ask user to input a integer n.

Step2: Use another variable named “ans” to store 0. (Why?)

Step3: While the integer n still has significant digit, do the following steps:

Step A: copy the unit digit out from n, and put it to ans

Step B: delete the unit digit of n.

Step4: Output ans.

```

Program div_and_mod;
Var ans,n:integer;
Begin
  Readln(n);
  Ans:=0;
  While n<>0 do      {What is n if n has no sig. digit?}
  begin
    ans := ans*10 + n mod 10 ;
    n := n div 10 ;
  end;
  writeln( ans );
End.

```

Actually, this is the most common way to use **while..do** , **repeat..until**.

Example: Find the greatest power of 2 which is just smaller than 10000.

```

Program power_2;
Var n:integer;
Begin
  N:=1;
  While 2*N<10000 do
  N:=n*2;
  writeln(n);
End.

```

Example: Find the greatest power of 2 which is just greater than 10000.

```

Program power_2;
Var n:integer;
Begin
  N:=1;
  repeat
  N:=n*2;
  until n>10000;
  writeln(n);
End.

```

## Conditional Looping V:

Example: Find HCF of two numbers m and n. ( $m > n$ ) (輾轉相除法)

```

Program Euclidean_Algorithm_for_finding_HCF;
Var m,n,t:integer;
Begin
  Readln(m,n);
  while n<>0 do
  begin
    t:=m;
    m:=n;
    n:=t mod n;
  end;
  writeln(m);
end.

```

Example: Find LCM of two numbers m and n. ( $m > n$ )

```

Program finding_LCM;
Var m2,n2,m,n,t:integer;
Begin
  Readln(m,n);
  m2:=m;
  n2:=n;
  while n<>0 do
  begin
    t:=m;
    m:=n;
    n:=t mod n;
  end;
  writeln(m2*n2 div m); { LCM of M,N = M x N / HCF of M , N }
end.

```

Example:  $F_1=1$ ,  $F_2=1$ ,  $F_n=F_{n-1} + F_{n-2}$  ( $n>2$ ), find  $F_n$ .

```

Program Fibonacci_Number;
Var a,b,c,n,i:integer;
Begin
  Readln(n);
  A:=1;
  B:=1;
  i:=2;
  While I<n do
  Begin
    C:=b;
    B:=a;
    A:=a+c;
    I:=I+1;
  End;
  Writeln(a);
End.

```

The above number  $\{1,1,2,3,5,8,13\dots\}$  is called **Fibonacci Number**.

*\* Very Challenging Question: \**

*Try to modify the program using only a,b,n,i 4 variables to find out  $F_n$ .*

## Conditional Looping VI:

Example: Ask User to enter a password

```
Program Password;
const password=4321;
var entry,count:integer;
begin
count :=0;
while entry<>password do
begin
count:=count+1;
write('Trial ',Count,'! Enter a 4-digit password: ');
readln(Entry);
end;
writeln('Valid password! You succeed in ',count,' trails.');
```

Example: Validate the marks.

```
Program Validation;
var mark:integer;
begin
repeat
write('Enter a mark (0-100): ');
readln(mark);
if (mark<0) or (mark>100) then write('Invalid mark!');
until (mark>=0) and (mark<=100);
writeln('The entered mark is ',mark);
end.
```

Example: Sum up n numbers, find their sum and average.

```
Program sum_and_average;
var n,total,count,mark:integer;
begin
write('Enter how many numbers you want to input: ');
readln(n);
count:=0;
total:=0;
while count<n do
begin
write('Enter a number :');
readln(mark);
total:=total+mark;
count:=count+1;
end;
writeln('The sum is ',total);
writeln('The average is ',total/n:0:4);
end.
```

## Definite looping (有限的迴圈):

Unlike conditional looping, which we can't predict numbers of times that the loops are executed, so it is in sense of indefinite looping (無限迴圈).

Definite looping is in sense definite.

This is call the **for-loop**.

**Syntax :**

```
For control_variable := initial_value to Final_value do statement1;
```

```
For control_variable := initial_value downto Final_value do statement1;
```

p.s. The rules for using **being...end;** is the same.

*control\_variable* is a **ordinary-type** (基本型) variable .

*initial\_value* and *Final\_value* must be the same type as the *control\_variable*.

**Ordinary** means that the type of the variable, is very very clear and well-organised.

For example, **integer** , **char** and **Boolean**.

You can find the next one or the previous of a given value.

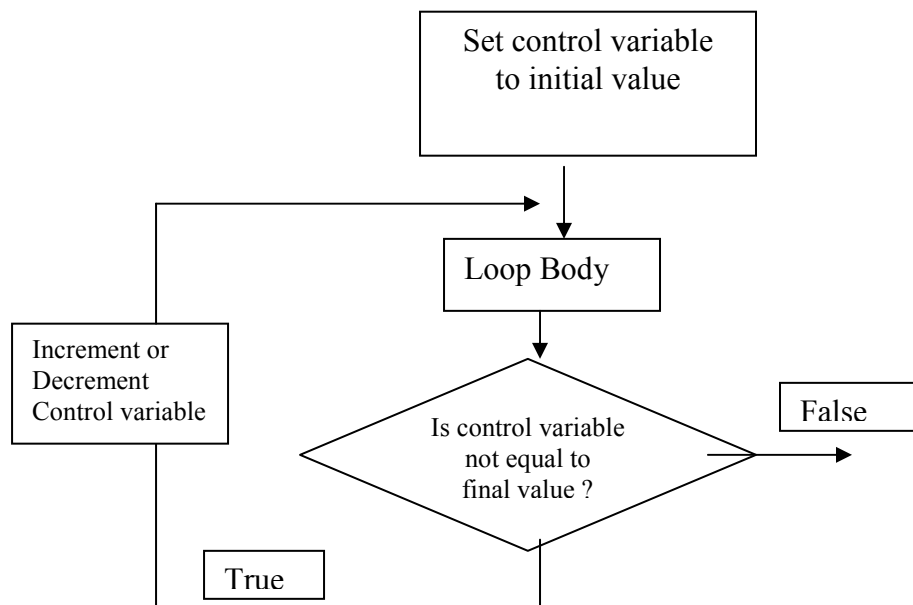
The next one of 1 is 2 , the next one of 'A' is 'B' , the next one of FALSE is TRUE.

However, in a real or string type ,

Do you know the next number of 1.1 ? or , Do you know the next string of 'AAA' ?

Is it 1.2 ? 1.100001? or 1.100000000001? , 'AAB' ? 'AAAA'? No one knows.

## Flow Chart of for-loop :



Example : output 1 to 10 in asc order.

```
Program for_loop1;
Var I:integer;
Begin
For I:=1 to 10 do
Writeln(i);
End.
```

Example : output 1 to 10 in desc order.

```
Program for_loop2;
Var I:integer;
Begin
For I:=10 downto 1 do
Writeln(i);
End.
```

Example : Check if n is prime.

```
Program prime;
Var I,n:integer; z:boolean;
Begin
Readln(n);
Z:=true;
For I:=2 to n-1 do
Z:= z and (n mod I<>0);
If z then writeln(n,' is prime. ');
else writeln(n,' isn"t prime. ');
End.
```

Example : Calculate  $a^n$ .

```
Program power;
Var I,a,n,answer:integer;
Begin
Readln(a,n);
Answer:=1;
For I:=1 to n do
Answer:=answer*a;
Writeln(answer);
End.
```

Example : Output n odd numbers.

```
Program odd_no;
Var I,n:integer;
Begin
Readln(n);
For I:=1 to n do
Writeln(2*I-1);
End.
```

Example : Writeln 10 times "Hello".

```
Program Hello;
Var I:integer;
Begin
For I:=1 to 10 do
write('Hello');
writeln;
End.
```

*Try to use for-downto-do to rewrite the above example.*

*Use for-loop to rewrite the example sum\_and\_average, Fibonacci\_number.*

## Nested-Loop I:

A nested-loop is a loop in another loop.

```
{This is a nested-loop}
program square;
var I,j:integer;
begin
for I:=1 to 5 do
begin
for j:=1 to 5 do write('*');
writeln;
end;
end.
```

```
{But this is not a nested-loop}
program square;
var I,j:integer;
begin
for I:=1 to 5 do writeln(i);
for j:=1 to 5 do writeln(j);
end;
end.
```



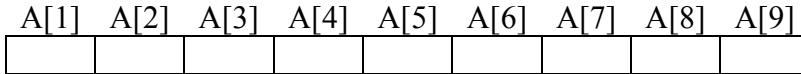
## Data Structure – Array (陣列):

Array is like a list of variables.

If we need many variables, it is very unwise to use name like Var1,var2,var3 , ... ,varN to open N variables.

We use a new data type called Array instead.

The structure of Array.



A[K] means the variable having index K of array A.

**Var** identifier: **array** [start\_value ... final\_value ] **of** type ;

*Start\_value ≤ final\_value , and they must be the same ordinary type.*

So to declare an integer array like the above example.

**Var** a: **array** [1..9 ] **of** integer;

Each variable can be called by the statement like

A[1];

### Why or When and How do we use array?

When writing a program, you have to think about what is the use of each variable, say the Boolean variable z in previous example represents whether a number is prime.

When using array, you must also think about what each variable in an array represents.

Example : Ask user to input subject marks , Chinese , English , Maths.  
And check if all of them pass.

```

Program mark_example;
Var mark:array[1..3] of integer; {Since there is 3 subjects}
Begin
write('Enter Chinese mark :');
readln(mark[1]);
write('Enter English mark :');
readln(mark[2]);
write('Enter Maths mark :');
readln(mark[3]);
if (mark[1]<50) or (mark[2]<50) or (mark[3]<50) then writeln('Oh..some of them fail!');
end.

```



**Exercise I :**

Rewrite all example in **while..do** , **repeat..until** ,  
and **for-to-do** , **for-downto-do** (if possible).

Finish the mastermind so that it contains no bug.

**Exercise II:**

Complete **Exer1.doc** and **G1&2Quiz.doc** .

**Exercise III\*:**

*This time, you may do with your computer.*

*But, always try to do it once without the help of computer.*

2002hje.doc	12, 20-24 , 36 , 37
2003hje.doc	A1 , A2 , A12** , A13 , A20

**Exercise IV\*\*\*:**

Write program to complete the following task :

1999hfe.doc	Question 1 Pythagoras Numbers
-------------	-------------------------------

\*\* : Very difficult and tricky, please spend time to think about it.