

Recursion (遞歸) I:

Last lesson, you have learnt that a procedure can call another procedures.
This time, we try to write a procedure that calls itself.

Example 9.1:

<pre>Procedure p1; Begin p1; End; Begin p1; End.</pre>	<p>Obviously when you run it, a run time error 202: Stack flow error will produce. It's because p1 calls p1, and calls p1... and so on. (infinitely) Such an infinite calling will result in Stack flow error.</p> <p>This error occurs very often when you handle recursion improperly. i.e. We must stop the procedure from calling itself if some condition does not fulfill.</p>
---	--

So, there must be a **stop case** for recursion.
Besides, a recursion usually comes with a recursive formula.

Example 9.2:

the power function can form a recursive formula.
 $2^n = 2 \times 2^{n-1}$. Pascal: power2(n) = 2* power2(n-1)
 Stop case is $2^0 = 1$. Pascal: power2(0) = 1

<pre>function power2(n:integer):integer; begin if n=0 then power2=1 else power2:=2*power2(n-1); end; begin writeln(power2(4)); end.</pre>	<p>For each different call to procedure or function, local variables are created. So, each function or procedure has its own set of local variables. Output 16.</p>
--	---

There are total 5 local variables n created in this case.
The computer use the corresponding set of local variables for each function.

<p>Dry Run Flow :</p>			
<p>Writeln(power2(4)); ← find the value of power2(4). Call function power2(4).</p>			
<table border="1" style="width: 100%; text-align: center;"> <tr> <td style="width: 50%;">n</td> <td style="width: 50%;">4</td> </tr> </table>	n	4	<p>Since n > 0, it return 2*power2(3) ← find the value of power2(3)</p>
n	4		
<p>Call function power2(3).</p>			
<table border="1" style="width: 100%; text-align: center;"> <tr> <td style="width: 50%;">n</td> <td style="width: 50%;">3</td> </tr> </table>	n	3	<p><i>Notes: there actually two different local variables n.</i> Since n > 0, it return 2*power2(2) ← find the value of power2(2)</p>
n	3		
<p>Call function power2(2).</p>			
<table border="1" style="width: 100%; text-align: center;"> <tr> <td style="width: 50%;">n</td> <td style="width: 50%;">2</td> </tr> </table>	n	2	<p><i>Now there are total 3 local variables n, one for each function.</i> Since n > 0, it return 2*power2(1) ← find the value of power2(1)</p>
n	2		
<p>Call function power2(1).</p>			
<table border="1" style="width: 100%; text-align: center;"> <tr> <td style="width: 50%;">n</td> <td style="width: 50%;">1</td> </tr> </table>	n	1	<p>Since n > 0, it return 2*power2(0) ← find the value of power2(0)</p>
n	1		
<p>Call function power2(0).</p>			
<table border="1" style="width: 100%; text-align: center;"> <tr> <td style="width: 50%;">n</td> <td style="width: 50%;">0</td> </tr> </table>	n	0	<p>Since n=0, it return 1, the value of power2(0) is found, then power2(1) returns 2*1 = 2 , power2(2) returns 2*2 = 4 and so on.</p>
n	0		

Recursion (遞歸) II:

When dry-running recursion, you have to mark down each local variables of different function/procedure, as they do only have same name. Like a global variable having the same name as a local variable.

But, all global variables occur inside function/procedure, are the same.

Example 9.3:

<pre> procedure k(n:integer); begin if n>1 then k(n-1); writeln(n); end; begin k(5); end. </pre>
<p>Output:</p> <pre> 1 2 3 4 5 </pre>

Dry Run Flow :		
Main body call k(5):		
k(5)	n	5
k(5) call k(4):		
k(4)	n	4
k(4) call k(3):		
k(3)	n	3
k(3) call k(2):		
k(2)	n	2
k(2) call k(1):		
k(1)	n	1
<p>Since n=1, k(1) will not call k(0). And proceed to writeln(n). So 1 is outputted at the first line. Then, we see the reserved word end, knowing that we have finished calling k(1). Go back to k(2), writeln(n), 2 is outputted. The go back to k(3), do the statement writeln(n), 3 is outputted. Go back to k(4) and outputted 4. Finally, k(4) was done, then go back to k(5) and writeln(n). Therefore, the last line is 5.</p>		

Example 9.4:

<pre> procedure k(n:integer); begin writeln(n); if n>1 then k(n-1); end; begin k(5); end. </pre>
<p>Output:</p> <pre> 5 4 3 2 1 </pre>

If two lines are swapped, the order of output becomes totally different.

It is very important when you writing codes with recursion.

Just few differences of the codes can result in another output etc.

Recursion (遞歸) III:

Example 9.5:

<pre> procedure k; var c:char; begin if not eoln then begin read(c); k; write(c); end; end; begin k(5); end. </pre>
Input:abcde Output:edcba

Dry Run Flow :		
Main body call k:		
k ₁	c	'e'
Not eoln, call k again;		
k ₂	c	'd'
Not eoln, call k again;		
k ₃	c	'c'
Not eoln, call k again;		
k ₄	c	'b'
Not eoln, call k again;		
k ₅	c	'a'
<p>After reading 'e', the cursor is located at end-of-line, so eoln is true now. This time, k will do nothing. Then come back to the previous k, and write(c); so 'e' is outputted first, then come back to previous k again, and output 'd' and so on.</p>		

Example 9.6:

<pre> procedure k; var c:char; begin if not eoln then begin read(c); write(c); k; end; end; begin k(5); end. </pre>
Input:abcde Output:abcde

Try to dry run 9.6 to make you understand clearly.

Recursion (遞歸) IV:

Fibonacci Sequence: $F_1=F_2=1$, $F_n = F_{n-1} + F_{n-2}$ (for $n > 2$)

Example 9.7: Using Recursion to find F_n .

```
const count:integer=0;
function fib(n:integer):integer;
begin
inc(count);
if n<=2 then fib=1
else fib:=fib(n-1)+fib(n-2);
end;

begin
writeln(fib(5));
writeln(count);
end.
```

Output:
5
9

Always Remember, each time you call a function, you must re-calculate all the part of the function. As computer does not memorize the result.

Dry Run Flow :

Main body call fib(5):

k(5)	n	5
------	---	---

Inc(count)

Fib(5)=fib(4)+fib(3) ← We have to find fib(4) and fib(3)
First, find fib(4)
Fib(4)=fib(3)+fib(2)
Find fib(3) first
Fib(3)=fib(2)+fib(1)
Find fib(2) first,
Fin(3) become 1+fib(1)
Find fib(1) , = 1
Fin(3) = 2
Fib(4) become 2 + fib(2)
Find fib(2) , = 1
Fib(4) = 3
Fib(5) becomes 3 + fib(3)
Then find fib(3) , fib(3) = fib(2) + fib(1)
Find fib(2) , = 1 , then find fib(1) , = 1 , fib(3) = 2
Fib(5) = 3+2 = 5

For each call to fib, count increases 1, so count is 9.

Example 9.8: n is value parameter.

```
const i:integer=5;
procedure k(n:integer);
begin
if n>1 then
begin
dec(n);
k(n);
writeln(n);
end;
end;
begin
k(i);
writeln("Variable i is ",i);
end.
```

Output:
1
2
3
4
Variable i is 5

Example 9.9: n is variable parameter.

```
const i:integer=5;
procedure k(var n:integer);
begin
if n>1 then
begin
dec(n);
k(n);
writeln(n);
end;
end;
begin
k(i);
writeln("Variable i is ",i);
end.
```

Output:
1
1
1
1
Variable i is 1

