

## Preface:

### High Precision Addition:

Adding two numbers M and N, ( $0 \leq M, N \leq 10^{254}$ ), and output the result without approximation.

Obviously, you can't use integer, longint, real, extended etc. to store the numbers.

Longint can store only about  $10^{10}$ , which is far less than  $10^{254}$

Although extended can store up to  $10^{4932}$ , it uses approximation.

So, we use string to store them, and “add” them, finally, output it.

```
var ans,m,n:string;
    c,m1,n1,i,j:integer;
begin
readln(m);
readln(n);
ans:="";
c:=0;
for i:=1 to abs(length(m)-length(n)) do
if length(m)>length(n) then n:='0'+n
else m:='0'+n;
for i:=length(m) downto 1 do
begin
m1:=ord(m[i])-48;
n1:=ord(n[i])-48;
ans:=chr((m1+n1+c) mod 10+48)+ans;
c:=(m1+n1+c) div 10;
end;
if c<>0 then ans:=chr(48+c)+ans;
writeln(ans);
end.
```

m,n store the numbers, since length of two numbers may be different. In order to simplify the job, we add zero to the beginning of the shorter one, so that they have equal length.

E.g. 999+1 becomes 999+001.

Next, we perform direct addition starting from the end of string that is the unit digit of two numbers.

Use C stores the carry (進位) of the previous addition. And proceed to the next digit.

Finally, we check if the carry is non-zero, if so, the length of the answer is longer than original.

*Try to code a High Precision Multiplication program.*

*Written in Procedure version and Function version.*

*Like the example in P.4.*

How about if we need to sum up 3 numbers in arbitrary order?

i.e. given a,b,c 3 numbers, I would like to add a and c, then add b, or add b and c first then a.

You may answer that store them to m and n first, sum up them then swap the value of ans and m, and store the third number to n, and add them again.

This is not bad, but it increases the difficulty to dry run the program.

Some may even say that, copy the above code and replace all m and n with ans and c respectively, this works only for small numbers.

In fact, there is an efficient way for copying code.

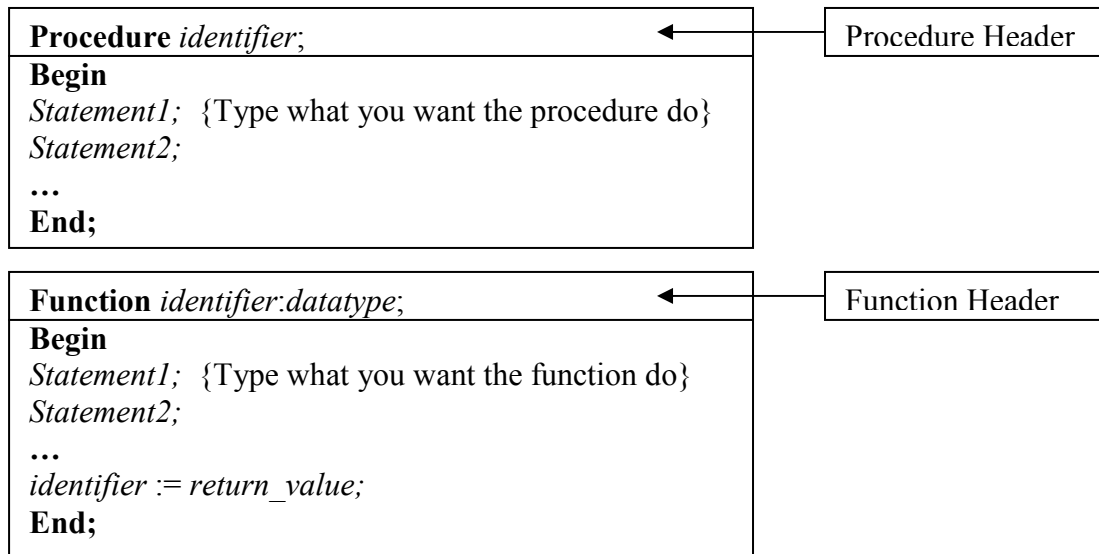
The use of sub-programming- Procedure and Function.

This helps us to simplify the main body.

## Self-Defined Procedure and Function I:

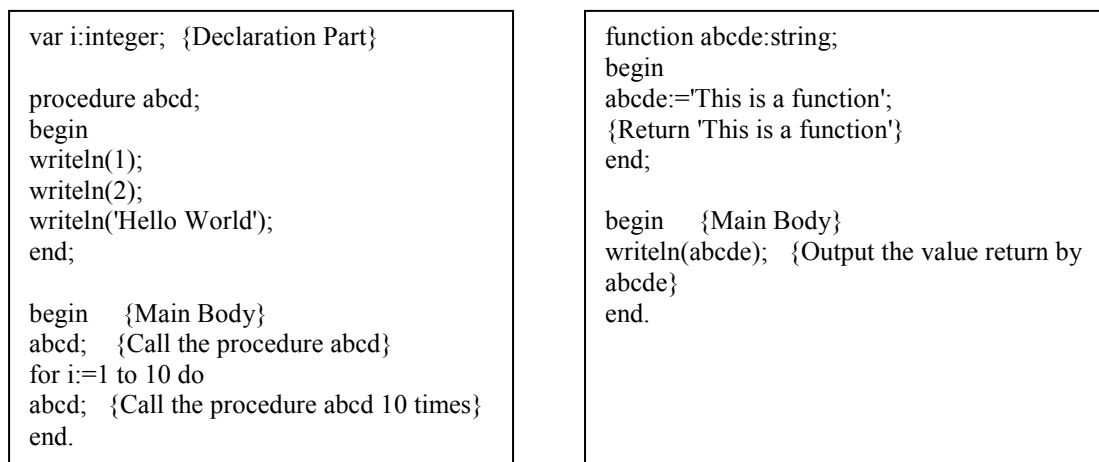
Previously, you have already learnt system-defined procedures and functions. In this lesson, you can create a procedure or a function yourself. Please be careful that there are a few differences between TPW and FP.

### Syntax:



These statements just create a procedure/function, they will not run until you call them. In Pascal, we type the name of a procedure or a function in the main body to call it once.

Here are two examples demonstrating the use of procedure and function.



There is almost no max. limit of numbers of procedures and functions. The only differences between procedure and function is that function return a value when being called, that value is stored in the variable having the same name as the function name.

*Notes: In TPW, you can't use the value of function abcde within itself, e.g. inc(abcde), "Abcde:=abcde+1;" , statements like this have another meaning. → call abcde again.*

## Self-Defined Procedure and Function II:

A program liked this still works.

```
procedure a;  
begin {Begin of a}  
writeln(1);  
end; {End of a}  
procedure b;  
begin  
writeln(2);  
end;  
procedure c;  
begin  
writeln(3);  
end;  
procedure d;  
begin  
writeln(4);  
end;  
begin {Main Body}  
a;b;c;d;  
end.
```

A sub-program is very liked a program, except that sub-program will not run until we call them in the main program.

Also, you can define variables, constants, type etc. within a sub-program.

You can even define function, procedure within a procedure or a function, but those things can only be accessed, used, called within that procedure.

It is liked the concept of scope of looping, we called this block.

Things enclosed by the reserved word procedure...begin... end; are considered as the same block.

The variables inside are called local variables, which are only accessible within the same block. On the other hand, variables that are declared under the block of main body are called Global variables.

They can be accessed everywhere in the program.

Example:

```
Var Glo_i:integer;  
procedure aa;  
var loc_k:integer;  
begin  
loc_k:=random(glo_i);  
writeln(loc_k);  
inc(glo_i);  
end;  
begin  
glo_i:=100;  
aa;  
writeln(glo_i);  
{writeln(loc_k);}  
end.
```

← Global variable

← Local variable

You can see that the statement “inc(glo\_i)” inside the procedure aa, this is valid because glo\_i is a global variable.

But, if you try to uncomment the last statement before “end.”, you may get a compile error, because loc\_k is only accessible within procedure aa.

### \* Important \*

Local variable can have the same name as global one, in such cases, the name refer to the nearest variable within the block having the same name.

Local variables are created every time when the corresponding procedure is called, therefore it is meaningless to think of the previous value of local variables.

```
procedure k;  
var i:integer;  
begin  
writeln(i);  
i:=99;  
end;  
begin  
k; k;  
end.
```

The second line of the output is not as expected 99.

New local variables are always created when being called, so the previous value has not effect.

If you really want to do so, use typed-const variable.

## Parameters(參數) I:

Still remember the addition example?

```
{Procedure}
var ans,m,n:string;
procedure add;
var c,m1,n1,i,j:integer;
begin
ans:= "";
c:=0;
for i:=1 to abs(length(m)-length(n)) do
if length(m)>length(n) then n:='0'+n
else m:='0'+n;
for i:=length(m) downto 1 do
begin
m1:=ord(m[i])-48;
n1:=ord(n[i])-48;
ans:=chr((m1+n1+c) mod 10+48)+ans;
c:=(m1+n1+c) div 10;
end;
if c<>0 then ans:=chr(48+c)+ans;
end;
begin
m:= '12345';
n:= '12345';
m:=ans;
n:= '1111';
writeln(ans);
end.
```

This add Global variables m,n and store to ans.

```
{Function}
var m,n:string;
function add:string;
var c,m1,n1,i,j:integer; ans:string;
begin
ans:= "";
c:=0;
for i:=1 to abs(length(m)-length(n)) do
if length(m)>length(n) then n:='0'+n
else m:='0'+n;
for i:=length(m) downto 1 do
begin
m1:=ord(m[i])-48;
n1:=ord(n[i])-48;
ans:=chr((m1+n1+c) mod 10+48)+ans;
c:=(m1+n1+c) div 10;
end;
if c<>0 then ans:=chr(48+c)+ans;
add:=ans;
end;
begin
m:= '12345';
n:= '12345';
m:=add;
n:= '1111'; m:=add;
writeln(add);
end.
```

This add Global variables m,n and return the result.

*Notes: the function “add” is called 3 times, don’t have a wrong concept that the value are memorized in the computer.*

You may realize that the result of procedure and function varies as m and n change. In this case, m and n are in the sense of parameters.

Parameters are values to which a procedure/function refers and gives different results. For example, if the value of m and n is changed to ‘123’ and ‘555’ then the return value of add will become ‘678’.

But, if you use parameters like the above two examples, you will definitely find difficulties in using them as you have to memorize those global variables which is used as parameters, and therefore you have to pay attention when modifying values of them. Apart from this, you have to create a lot more variables for parameters. It is very terrible if you code liked this.

In Pascal, we use a pair of brackets to enclosing the parameters to be passed to a procedure or a function. Like `sqrt(100)`, 100 is the parameter passed to function `sqrt`.

## Parameters(參數) II:

Use the following headers to replace the previous for parameters in a procedure/function.

### Syntax:

```
Procedure identifier ( p_identifier1, p_identifier2:ptype1; p_identifier3, p_identifier4:ptype2 );
```

### Syntax:

```
Procedure identifier (var p_identifier1, p_identifier2:ptype1; p_identifier3, p_identifier4:ptype2 );
```

### Syntax:

```
Function identifier ( p_identifier1, p_identifier2:ptype1; p_identifier3, p_identifier4:ptype2 ):return_type;
```

### Syntax:

```
Function identifier (var p_identifier1, p_identifier2:ptype1; p_identifier3, p_identifier4:ptype2 ):return_type;
```

Parameters can be divided into two groups, the formal parameters and the actual ones. Formal parameters are referring the parameters at function/procedure headers, while actual parameters refer to the parameters you pass to function/procedure when you call it. So, sqrt(100) , 100 is an actual parameter while p\_identifier1 is a formal one.

Formal parameters can also be divided into two subgroups, value parameters and variable parameters.

Value parameter means we are only interested in the value only.

Namely, variable parameters mean we are interested in the variable.

Value parameters are passed by value only while variable parameters are passed by variables or technically called passed by reference.

You may treat a value parameter as another local variable with initial value the same as the passed value, but the variable parameter is actual the variable passed.

Therefore, if we assign new value to a variable parameter, the actual value of the passed variable will be changed.

So, you must pass a variable for a variable parameter, also they must be the same type or a compile error occurs.

Example:

```
Str('111',integer_A);          Assign(f, 'abc.txt');
```

'111' and 'abc.txt' are passed by value, while integer\_A and f are passed by reference, as the content of it will change after the execution of the procedure Str and Assign.

Variable parameters are declared within the reserved word **var** and semi-colon.

## Parameters(參數) III:

Actual parameters are passed corresponding to their position to formal parameters. When procedures/functions are called with parameters, all parameters will be initialized with the appropriate values, and then run the block of the sub-program.

Example:

Check a input number is prime or not

```
var i:integer; {Global}
function chk(i:integer):boolean;
var k:integer; z:boolean;
begin
z:=i>1;
for k:=2 to trunc(sqrt(i)) do
z:=z and (k mod i <>0);
chk:=z;
{The i within function chk }
{refers to the parameter i}
end;
begin
readln(i); {This is global variable i}
if chk(i) then writeln(i,' is a prime number.');
```

Dry run and read this output carefully.

```
var k1,k2,k3:integer;
procedure abcd(var i,k:integer;j:integer);
{i and k are variable parameter}
{j is value parameter}
var h:integer;
begin
for h:=1 to 3 do
begin
writeln('h:',h,' i:',i,' j:',j);
writeln('k:',k,' k1:',k1,' k2:',k2,' k3:',k3);
i:=i+k;
k:=k+i;
j:=j+k;
end;
end;
begin
k1:=1;
k2:=10;
k3:=100;
abcd(k1,k2,k3);
writeln(k1);
writeln(k2);
writeln(k3);
end.
```

k1 and I,k2 and k have the same value, as they refer to the same variable.  
J and k3 are different.

Try to change the statement  
“abcd(k1,k2,k3)” to “abcd(k1,k1,k1)”, see what happen.

```
procedure Biga;
procedure Biga_1;
begin
writeln(1);
end;
procedure Biga_2;
begin
writeln(2);
end;
procedure Biga_3;
procedure Biga_3_a;
begin {Accessible only within Biga_3}
writeln('a');
end;
begin
Biga_3_a;
writeln(3);
end;
procedure Biga_4;
begin
Biga_1;
Biga_2;
Biga_3;
end;
begin
Biga_4;
end;
begin
Biga;
end.
```

Example showing procedure calling another procedure.  
Only Biga can be called in the main body.

It will do Biga first, then do Biga\_4, and then Biga\_1 to writeln(1), then do Biga\_2, writeln(2), do Biga\_3 then call Biga\_3\_a writeln('a') then go back to do writeln(3).

## Parameters(參數) IV:

Variable parameters cannot be used as the control variable of for-loop.

Value parameters can be passed within compatible value.

e.g. longint , integer, and shortint.

However, variable parameters must have the identical data type.

So, given a procedure header:

```
Procedure aaa(var I:integer;k:longint;r:real);
```

And a list of global variables:

```
Var r1,r2,r3:real; i1,i2,i3:integer; long1:longint
```

Determine if the following calling is valid:

“aaa(i1,i2,i3)”

“aaa(r1,0,0)”

“aaa(i1,0.0,0.0)”

“aaa(long1,long1,long1)” {This is a very special case}

“aaa(i1,i1,i1)”

“aaa(i1,long1,r1)”;

“aaa(i3,r2,i1)”;

A procedure can call itself with its block, such calling is named **recursion (遞歸)**.

Usually, if you do not handle in correct way, a runtime error will produce.

### Special procedure: Exit;

Calling it can quit from the current block and continue to do the outer block.

If this procedure is called in the main body, it stops the program immediately.

Always remember that if you call procedure/function for N times, no matter using exit or reach the end of the procedure, you have to quit for the same number of times.

Except when you use the procedure halt, which immediately stops the program.

### Special procedure: Halt; Halt(number);

This stops the program immediately.

### Practical Exercise:

Use procedures and functions to simplify your program as much as you can.

- 1) Write a program to perform high precision subtraction.
- 2) Write a program to perform high precision multiplication.

You may get hints from [http://www.hkoi.org/training2003/inter\_hpa.ppt]

Till now, you are able to complete all Junior Hear Event Past Paper, except only few questions.

Try to spend only one and half hour to complete a whole paper without referring to any notes or computers. Stop going on when 90 minutes is passed, mark the place where you stop at, and finish the rest with another pen.

I suggest doing 2003 then to 2000 first.

**Exercise I:** (20<sup>th</sup> – Jul)

ex-proc1.doc

program-rev2.doc

**Exercise II:** (20<sup>th</sup> – Jul)

Within 90 minutes do 2003 or 2000 Junior Past Paper.

That is 2003hje.doc or 2000hje.doc

**Exercise III:** (25<sup>th</sup> – Jul )

2003fje.doc #2 Narrow Range Broadband

**Exercise IV:** (25<sup>th</sup> – Jul )

2005fje.doc #4 Competition

If you have time, read the file **Recursion.doc** located in Lesson9 before the next lesson.