

## Typed-Constant variable:

In Pascal, there is a weird statement call **typed-constant**.  
Usually, we define a constant by using statements like `pi=3.14`

A typed-constant is adding a type to such constant, making it be a variable.

**Syntax:** `Const Identifier : anytype = corresponding_value ;`

**Example:** Actually, both are the same.

```
const i:integer=6;
begin
writeln(i); { 6 }
i:=i+1;
writeln(i); { 7 }
end.
```

```
var i:integer;
begin
i:=6;
writeln(i); { 6 }
i:=i+1;
writeln(i); { 7 }
end.
```

The statement `const I:integer=6;` means declare a integer-typed variable `i`  
And assign 6 to it at the start of the program.

**Array-typed constant:** {We use `'()`' to enclose the array element}

```
const i:array[1..3] of integer = (7,8,9);
begin
writeln(I[1]); {7}
writeln(I[2]); {8}
writeln(I[3]); {9}
end.
```

```
const i:array[1..4,1..3] of integer = ((7,8,9),(5,6,7),(-1,-2,-3),(0,1,2));
begin
writeln(I[1,1]); {7}
writeln(I[2,2]); {6}
writeln(I[3,2]); {-2}
writeln(I[4,2]); {1}
end.
```

```
{String typed Constant}
const
Heading: string[7] = 'Section';
NewLine: string[2] = #13#10;
TrueStr: string[5] = 'Yes';
FalseStr: string[5] = 'No';
```

```
{Char-array typed Constant , both are the same}
const
Digits: array[0..9] of Char =
('0', '1', '2', '3', '4', '5', '6', '7', '8', '9');
const
Digits: array[0..9] of Char = '0123456789';
```

*You must type all the values in a typed-constant statement.*  
So `const I:array [1..3] of integer =(1,2) ;` is wrong.

## Set:

In mathematics, there are objects called set, which can be thought as a collection of some special things.

For example, set of natural numbers means the collection of all natural numbers i.e. positive integers.

We can declare a set in Pascal by using the syntax **set of** char; etc.

```
Var identifier : set of ordinary_datatype;
```

The cardinality of a set must not exceed 256, meaning a set can contain maximum 256 elements.

### The in operator:

The statement “**x in y**” checks if x is in the set y.

For example, ‘a’ is in the set of char since char include all characters.

**A + B** means the union of set A and set B.

**A \* B** means the intersection of set A and set B.

**A - B** means the difference of set A and set B. {set contains elements in A and not in B}

**A < B** checks if set A is proper subset of set B.

**A <= B** checks if set A is subset of set B.

**A > B** checks if set A is proper superset of set B.

**A >= B** checks if set A is superset of set B.

We use the middle-bracket [ ] to enclose an element of a set.

### Example:

```
Var c:char; setc:set of char; s:string;  
    I:integer;  
Begin  
  Readln(s);  
  Setc:=[]; {An empty set }  
  For I:=1 to length(s) do  
    Setc:=setc+ [s[I]];  
  For c:=#0 to #255 do  
    If c in setc then write(c);  
  Writeln('These are the characters in the string');  
End.
```

```
Var c:char;  
Begin  
  Readln(c);  
  If c in ['a' .. 'z'] then writeln('Small letter.')
```

```
  Else if c in ['A'.. 'Z'] then writeln('Capital')
```

```
  Else if c in ['0' .. '9'] then writeln('Digit')
```

```
  Else writeln('Symbol');
```

```
End.
```

In fact, you can consider a set as a range.

A set-typed variable is like a variable range.

## Self-defined data type I:

We can create a new data type by using the reserved word **type**.

### Enumerated type:

We can create a new type with at most 255 self-defined elements.

However, this is useless.

### Syntax: (Self-define type)

```
Type identifier = ( identifier1, identifier2, identifier3, ..., identifierN);
```

### Example:

```
type day=(Mon,tue,wed,thr,fri,sat,sun);  
var today : day;  
begin  
today:=thr; {thr is a new value we defined}  
if today=thr then writeln('Today is thursday.').  
end.  
{You can't directly use writeln(today) to output today}
```

### Range type:

```
Type identifier = start_value..end_value;
```

### Example:

```
type digit=0..9; {This defined digit as a type storing 0 to 9 only}  
var unit_digit : digit; {in fact, longint is defined by the similar way}  
begin {i.e. type longint=-2147483648.. 2147483647;}  
unit_digit:=1;  
writeln(unit_digit);  
end.
```

*Except the enumerated type, you can use the self-defined type as follow.*

```
var unit_digit : 0..9;  
begin  
unit_digit:=1;  
writeln(unit_digit);  
end.
```

## Self-defined data type II:

Self-defined data type is important when you deal with two arrays.  
Consider the following two examples

```
var a:array[1..3] of integer;
    b:array[1..3] of integer;
begin
a:=b
end.
{This results in Type Mismatch}
```

```
Type array3= array[1..3] of integer;
var a: array3;
    b:array3;
begin
a:=b {This works normally}
end.
```

In the left one, the computer thinks their type is different while in the right one, it treats them as the same data type.

Such array in the right example is called parallel array.

You can do assignment between parallel arrays, which means for I:=1 to 3 do a[I]:=b[I]; .

Another importance is that, we can create complex data structure with the reserved word **record...end;** or **record...case...of...end;** together

### Record:

This pack small types into a larger type.

**Syntax :** ( 3 different versions)

```
Type identifier = record
    Field_identifier1 : datatype1 ;
    Field_identifier2 : datatype2 ;
    ....
    identifierN : datatypeN ;
end;
```

```
Type identifier = record
    identifier1 : datatype1 ;
    identifier2 : datatype2 ;
    ....
    identifierN : datatypeN ;
case datatypeX of
value1 : ( identifierY1 : datatypeY1);
value2 : ( identifierY2 : datatypeY2);
v3,v4 : ( identifierY3 : datatypeY3);
    ...
valueN : ( identifierN : datatypeN);
end;
```

```
Type identifier = record
    identifier1 : datatype1 ;
    identifier2 : datatype2 ;
    ....
    identifierN : datatypeN ;
case identifierX : datatypeX of
value1 : ( identifierY1 : datatypeY1);
value2 : ( identifierY2 : datatypeY2);
v3,v4 : ( identifierY3 : datatypeY3);
    ...
valueN : ( identifierN : datatypeN);
end;
```

## Self-defined data type III:

The **record..case..of..end**; statement is rarely used for your level , or even OI level. It is related to how a computer stores data in memory and how such data is represented. It also requires knowledge of superposition of data in computer. It was originally designed for low-level programming like generating a midi, control a light of a keyboard, or some file structure like old-style midi-sound-file uses it.

So, I will only discuss the **record..end**; statement here. The others may be taught later on in the Memory-sized, Stack and Heap section when free.

**Example:** define a xy-coordinate pair data type

The integer x, y is called a field of the record coor.

```
Type coor=record
  x,y:integer;
End;
```

**Example:** define a personal information with name , phone , address and age data type.

```
Type info=record
  Name , phone_no , Address : string;
  Age: integer;
End;
```

The following code demonstrates the use of record.

```
Type coor=record
  x,y:integer;
End;
Var pt1 , pt2 :coor;
  Distance : integer;
Begin
Pt1.x:=1;
Pt1.y:=1;
Pt2.x:=3;
Pt2.y:=3;
Distance:=round(sqrt(sqrt(pt1.x-pt2.x)+sqrt(pt1.y-pt2.y)));
Writeln('The distance between two points is ',distance);
End.
```

```
Type coor=record
  x,y:integer;
End;
Var pt1 , pt2 :coor;
  Distance : integer;
Begin
With pt1 do
  begin
x:=1; {Means pt1.x}
y:=1; {Means pt1.y}
  End;
Pt2.x:=3;
Pt2.y:=3;
Distance:=round(sqrt(sqrt(pt1.x-pt2.x)+
```

*We use a full-stop to access the variable inside a record.*  
So pt1.x means the integer x of record-typed variable pt1

*You can use **with..do..begin..end** to access the element easily like the right example.*

## Self-defined data type IV:

More complex example: This shows you a record of record, array of record and record of array

```
Type basic_info=record
  Name , phone_no , Address : string;
  Age: integer;
End;

Complex_info = record
  Lastname , middlename , firstname : string;
  nickname : array[1..10] of string;
  Basic : basic_info;    { You can use the datatype defined previously}
End;

School = record
  Schname : string;
  No_of_std : integer;
  Std_info : array [ 1..1000] of complex_info;  {Array of record is also allowed}
End;

Var  MC:school;    {Declare MC as school record-typed }
Begin
mc.schname:= 'methodist college';
mc.no_of_std:=997;
{Store information of the first student in MC}
mc.std_info[1].lastname:= 'Chan' ;
mc.std_info[1].middlename:= " ;  { Usually empty for a Chinese}
mc.std_info[1].firstname:= 'Tai Man';
mc.std_info[1].nickname:= 'Ar Man' ;
mc.std_info[1].basic.name := 'Chan Tai Man' ;
mc.std_info[1].basic.Age := 16 ;
mc.std_info[1].basic.phone_no := '23843543' ;
mc.std_info[1].basic.Address := '50 Gxxxxxxx Rod , Yau Ma Tei' ;

End.
```

If there are more than 1 school , we can also use

```
Var sch:array[1..100] of school;
```

And replace all mc with sch[1] in the previous example.

No exercise for this notes though.

Please try to use **type**, **record**, **set..of** and the operator **in** to write program.