

## Forward References I:

In Pascal, all identifiers can only be used after they have been well defined.  
For example, you can't assign a value to a variable that hasn't been declared, call a procedure without defining it.

These will result in "Compile Error: Unknown Identifier".

For procedures or functions, if we need two procedures calling each other.  
Say p1 call p2 and p2 call p1, we can't simply declare a procedure as follow.

```
procedure p1;  
begin  
p2; ←  
end;  
procedure p2;  
begin  
p1;  
end;  
begin  
p1;  
end.
```

Unknown identifier p2

Compile Error occurs,  
simply because p2 is not defined  
before the line "p2;" is being called.

If you remove the line "p2;",  
there is no more syntax error.

Since p1 has been well defined before  
procedure p2 calls p1, there is no error.

So, it seems that we can't declare two procedures/functions calling each other.

But, Pascal provides us a way to do so. It is called **forward references**.

It allows us to declare the header of a procedure/function first, and leaves the body to be specified later on.

To do so, we need to add the reserved word, more correctly called directive, **forward** after the procedure/function header.

**Syntax:** **Procedure** *identifier* [(*parameter list*)]; **forward**;

```
...  
Procedure identifier [(parameter list)];  
Begin  
Statement;  
End;
```

**Function** *identifier* [(*parameter list*)] : *r\_type*; **forward**;

```
...  
Function identifier [(parameter list)] : r_type;  
Begin  
Statement;  
End;
```

## Forward References II:

### Example:

```
procedure p1(i:integer);forward; {This line just give a header of p1}
procedure p2(i:integer);
{The word forward states the real body is defined later on, so procedure p2 is not a procedure of p1.}
{p2 is a procedure of main body.}
begin
p1(1);
end;
procedure p1(i:integer); {p1 is defined at here}
begin
p2(1);
end; {Obviously, this program cause infinite calling between 2 procedures}
begin
p1(1); {Therefore, a runtime error 202: Stack Flow Error appears}
end.
```

### Example 2:

```
const n:integer=10;

procedure k;forward;
procedure g;
begin
dec(n);
writeln(n);
k;
end;
procedure k;
begin
dec(n);
if n>0 then g;
end;

begin
k;
end.
```

This is an example showing **forward reference**.

The output of this program:

```
8
6
4
2
0
```

---

If you find difficulties when dry run this example, study the notes **Recursion** again.

In TPW, the parameter list for the body is not necessary, but in FP the list must not be omitted.

You can find a question from HKOI past paper using **forward reference** without parameter list for the body declaration.

In FP, if omitted, it becomes **overriding** instead.

Please download and install FreePascal at <http://www.freepascal.org/download.html>  
intel / i386 → Win32.

## Override (重載):

*This part is only available in FreePascal only.*

In TPW, it is impossible to declare a procedure/function that can accept different data types and for each data types gives different result.

In FP, it is possible to do so. We can define many procedures/functions with same name but different parameter list.

When that procedures/functions is called, we find the procedure with corresponding data type and parameter list declared, and execute that procedure.

Example:

```
var r1,r2:real;
    i1,i2:integer;
procedure p;
begin
writeln('Called without passing any parameter. ');
end;

procedure p(i:integer);
begin
writeln('Called with one integer ');
end;

procedure p(r:real);
begin
writeln('Called with one real ');
end;

procedure p(i1,i2:integer);
begin
writeln('Called with two integers ');
end;

procedure p(i1:integer;r1:real);
begin
writeln('Called with one integer followed by one real');
end;

procedure p(r1:real;i1:integer);
begin
writeln('Called with one real followed by one integer');
end;

begin
p(r1,i1);
p(i1,r1);
p(i1,i1);
p(i1);
p(r1);
p;
end.
```

Output:

---

Called with one real followed by one integer  
Called with one integer followed by one real  
Called with two integers  
Called with one integer  
Called with one real  
Called without passing any parameter.

---

Override is not any difficult thing.

Just need to find the procedure  
with corresponding declaration and run it.