APÉNDICE 2: MÓDULOS EN SUN PASCAL E EN TURBO PASCAL.-

Os módulos están compostos de duas partes diferenciadas:

1. *Interface*. Declaración dos recursos (constantes, tipos, variábeis, funcións/procedimentos) exportados polo módulo.

2. *Implementación.* Implementación concreta das declaracións da interface.

Os *clientes do módulo* só poden acceder aos recursos exportados na interface (proceso de ocultación de recursos).

TURBO PASCAL

Interface e implementación do módulo atopan-se no mesmo arquivo, coñecido como *unit* (unidade).

```
Unit oMeuModulo ;

interface

        (* declaración de constantes, variábeis globais ao módulo e tipos *)

        (* declaración de cabeceiras das funcións/procedimentos exportadas/os *)

        function oMeuFibonacci ( n : integer ) : integer ;

        procedure oMeuPrint ( msx : string ) ;

implementation

        (* declaracións e definicións normais en programas Turbo Pascal *)
        (* deben estar as definicións declaradas na interface *)

        begin
                ...     (* código de inicialización do módulo *)
        end .
```

Para que o cliente poda utilizar os recursos exportados polo módulo, debe incluir ao módulo na cláusula *uses* (usa):

```
program proba ;

uses oMeuModulo ;

begin
        oMeuPrint ( 'olá' ) ;
end .
```

SUN PASCAL

O sistema de módulos en Sun Pascal é semellante ao usado en C. Só é posíbel exportar procedimentos/funcións e variábeis globais ao módulo. Debemos crear dous arquivos diferentes:

1. *Implementación:* extensión .pas p.ex. *modulo.pas*
2. *Interface:* extensión .h p.ex. *modulo.h*

Interface

Deben declarar-se as funcións e as variábeis globais ao módulo. As variábeis deben declarar-se coa palabra reservada *extern*, encanto que os procedimentos tên como corpo *external*.

```
(* modulo.h *)

var
        x : extern integer ;

procedure oMeuPrint ( msx : string ) ; external ;
function oMeuFibonacci ( n : integer ) : integer ; external ;
```

Implementación

A implementación é un arquivo Pascal normal, substituíndo *program* por *module.* O corpo do programa suprime-se.

```
(* modulo.pas *)

module oMeuModulo ;

var
        x : integer ;

procedure oMeuPrint ( msx : string ) ;
begin
        (* aqui vai o corpo de oMeuPrint *)
end ;

function oMeuFibonacci ( n : integer ) : integer ;
begin
        (* aqui vai o corpo de oMeuFibonacci *)
end ;
```

Cliente

Para utilizar o módulo inclui-se a interface con *#include*

```
(* proba.pas *)

program proba ;

#include “modulo.h”

begin
        writeln ( oMeuFibonacci ( 10 ) ) ;
end .
```

Tanto programa como módulos deben compilaren-se coa opción *-c* (compilar só) co que se xeran os arquivos obxecto con extensión *.o* (modulo.o e proba.o, no exemplo). Posteriormente ambos obxectos enlazan-se para xerar un executábel.

```
$ pc -c modulo.pas
$ pc -c proba.pas
$ pc -o proba modulo.o proba.o
```

$ é o prompt de Unix. Obviamente a execución fai-se tecleando

```
$ proba
```

Note-se que lle podemos subministrar ao usuário o módulo xá compilado (é dicer, damos-lle o modulo.o pero non o modulo.pas) e mais a interface (modulo.h). Asi logramos que saiba como usá-lo mediante o modulo.h, pero ocultamos a implementación porque ao ser o modulo.o un arquivo binário o usuário non pode averiguar a implementación interna.

Se se desexan exportar constantes e tipos que son utilizados tanto no módulo como nos seus clientes, o mais conveniente é definir un terceiro arquivo (constantes.h, por exemplo) coas definicións e incluí-las con *#include* en ambos (módulos e clientes).