Programación modular en Pascal

- Programación modular
  - Es una técnica para escribir programas grandes y complejos
  - Se basa en subdividir el programa en secciones más simples. Cada sección se llama módulo y realiza una única tarea
- Módulos
  - Cada módulo tiene un nombre (identificador) y consiste en un conjunto de sentencias agrupadas mediante un `begin` y un `end`
  - El módulo (grupo de sentencias) se ejecuta cuando se escribe su nombre en el programa
  - Pascal utiliza dos tipos de módulos: procedimientos y funciones
    - Procedimientos: Agrupación de sentencias que efectúan una tarea específica a partir de una serie opcional de datos de entrada (parámetros).
      - Ejemplo: `writeln (valor_entrada);`
    - Funciones: Agrupación de sentencias que efectúan una tarea que da como resultado un valor específico a partir de una serie de datos de entrada (parámetros)
      - Ejemplo: `valor_salida:=sqrt(valor_entrada);`

Procedimientos (1)

- Declaración por valor

```
procedure nombre_de_procedimiento(lista_de_parámetros);
   otras_declaraciones;
begin
   sentencias
end;
```

  - *lista de parámetros* :
    - Si existe, contiene declaraciones de parámetros separadas por signos de punto y coma con la forma *parámetro:tipo.*
    - Si hay varias declaraciones del mismo tipo se pueden agrupar como:
      parámetro_1,parámetro_2,...,parámetro_n:tipo
    - Si no hay parámetros se omiten los paréntesis.
  - *otras_declaraciones:*
    - hacen referencia a declaraciones de constantes, variables, tipos, procedimientos y funciones que se utilizarán en el procedimiento
    - Excepto el nombre del procedimiento, las declaraciones y parámetros de éste son locales (sólo se pueden utilizar dentro del procedimiento).

- Llamada a un procedimiento:

```
nombre_de_procedimiento(lista_de_parámetros;
```

  - Si el procedimiento no tiene parámetros se omiten los paréntesis.
  - Si hay más de un parámetro se separan mediante comas.

Procedimientos (2)

- Ejemplo de procedimientos

```
program ejemplo_procedimiento;
var tiempo, distancia, velocidad : real;
procedure muestra_titulo;
begin
   writeln('Este programa calcula la distancia recorrida basada');
   writeln('en la velocidad y el tiempo')
end;
procedure muestra_respuesta(dist:real);
begin
   writeln('La distancia recorrida es:',dist)
end;

begin
   muestra_titulo;
   writeln('Introduzca la velocidad en km/h');
   readln(velocidad);
   writeln('Introduzca el tiempo en horas');
   readln(tiempo);
   distancia:=velocidad*tiempo;
   muestra_respuesta(distancia)
end.
```

Procedimientos (3)

- Paso de parámetros a procedimientos
  - Parámetros por valor
    - Mediante el paso de parámetros por valor se realiza una copia de los valores de las variables del programa principal en los parámetros del procedimiento.
    - Aunque los valores de los parámetros del procedimiento se modifiquen dentro de éste, el cambio no se refleja en las variables del programa principal.
  - Parámetros por referencia (o variable).
    - Mediante el paso de parámetros por referencia se pasan las variables del programa principal en los parámetros del procedimiento.
    - Si el procedimiento modifica los valores de los parámetros, estas alteraciones se verán reflejadas en el valor de la variable del programa que lo llamó.

Procedimientos (4)

- Declaración por valor:

```
procedure nombre_procedimiento(var parámetros_por_referencia);
   otras_declaraciones
begin
   sentencias
end;
```

  - La declaración de un parámetro por referencia es idéntica a la de un parámetro por valor, con excepción de la palabra **`var`** que se añade al principio de la declaración.
  - Pueden mezclarse en la misma lista de parámetros parámetros por valor y por referencia.
  - Los parámetros de tipo **`text`** o cualquier tipo de archivo tienen que pasarse por referencia.
  - Si no hay parámetros se omiten los paréntesis.

Procedimientos (5)

- Ejemplo:

```
program ejemplo_procedimiento;
var tiempo, distancia, velocidad : real;
procedure muestra_titulo;
begin
   writeln('Este programa calcula la distancia recorrida basada');
   writeln('en la velocidad y el tiempo')
end;
procedure lee_datos(var vel,tie:real);
begin
   writeln('Introduzca la velocidad en km/h');
   readln(vel);
   writeln('Introduzca el tiempo en horas');
   readln(tie)
end;
procedure muestra_respuesta(dist:real);
begin
   writeln('La distancia recorrida es:',dist)
end;
begin
   muestra_titulo;
   lee_datos(velocidad,tiempo);
   distancia:=velocidad*tiempo;
   muestra_respuesta(distancia);
end.
```

Funciones (1)

- Declaración de funciones:

```
function nombre_función(lista_parámetros):tipo_resultado;
   otras_declaraciones
begin
   sentencias
end
```

  - La lista de parámetros, sigue las mismas reglas de construcción que la lista de los procedimientos
  - Al igual que en los procedimientos, excepto el nombre de la función, las declaraciones y parámetros de la función son locales (sólo se pueden utilizar dentro del procedimiento).
  - *tipo_resultado* es el tipo del valor devuelto por la función.
  - Alguna sentencia debe asignar un valor a *nombre_función*. El último valor asignado a *nombre_función* es el valor devuelto por la función.

- Llamada a una función:

```
resultado:=nombre_de_función(lista de parámetros);
```

  - Si la función no tiene parámetros se omiten los paréntesis.
  - Si hay más de un parámetro se separan mediante comas.

Funciones (2)

- Ejemplo:

```
program ejemplo_procedimiento;
var tiempo, distancia, velocidad : real;
procedure muestra_titulo;
begin
    writeln('Este programa calcula la distancia recorrida basada');
    writeln('en la velocidad y el tiempo')
end;
procedure lee_datos(var vel,tie:real);
begin
    writeln('Introduzca la velocidad en km/h');
    readln(vel);
    writeln('Introduzca el tiempo en horas');
    readln(tie)
end;
function calcula_distancia(vel,tie:real);
begin
    calcula_distancia:=vel*tie
end;
procedure muestra_respuesta(dist:real);
begin
    writeln('La distancia recorrida es:',dist)
end;
begin
    muestra_titulo;
    lee_datos(velocidad,tiempo);
    distancia:=calcula_distancia(velocidad,tiempo);
    muestra_respuesta(distancia);
end.
```

Ámbito local y global de las variables (1)

- Ámbito de los identificadores en Pascal
  - Los identificadores en Pascal sólo pueden utilizarse en el bloque del programa en el que están definidos. Por tanto un identificador en el programa principal tiene como alcance todo el programa y un identificador dentro de un procedimiento es local al procedimiento.
  - En el caso de que hayan variables locales dentro de un procedimiento con el mismo nombre que variables globales las variables locales reemplazan a las globales dentro del procedimiento.
  - No es conveniente la utilización de las variables globales siempre que puedan reemplazarse por locales debido a los efectos no deseados que pueden producir.
- Reglas generales para la utilización de los parámetros
  - Usar los parámetros por valor para los datos que se han de pasar a un procedimiento o función pero que no deben modificarse
  - Utilizar los parámetros por referencia solamente en el caso de que queramos una modificación de éstos.
  - Evitar en lo posible que un procedimiento use variables globales.

Ámbito local y global de las variables (2)

- Ejemplo:

```
program ej_ambito
const
    m=230;
var
    i,j:real;

procedure a(var i:real);
var
    r,s:boolean:

function b(g:real):boolean
var
    m,n:char;
begin
    ...
end;

begin
    r:=b(5.0);
end

    a(i);
end.
```

Bloque	Identificador	Significado de cada identificador
ej_ambito	m	constante global
	i,,j	variables globales
	a	procedimiento declarado en ej_ambito
a	i	parámetros de a
	r,s	variables locales
	b	función local
	j	variable declarada en ej_ambito
	m	constante global
b	g	parámetros de b
	m,n	variables locales
	r,s	variable declarada en a
	i	parámetro de a
	j	variable declarada en ej_ambito
	a	procedimiento declarado en ej_ambito
	m	constante global

Recursión

- Llamadas recursivas:
  - En Pascal, a un procedimiento o función le es permitido no sólo llamar a otro procedimiento o función, sino también invocarse a sí mismo. Una llamada de éste tipo se dice que es recursiva.
- Ejemplo:
  - Calcular el factorial de un número de forma recursiva

```
function factorial(numero:integer):integer;
begin
  if numero = 0 then
        factorial := 1
  else
        factorial := numero * factorial(numero-1)
end;
```