Capítulo 5: Subprogramas: Funciones y Procedimientos

1.-Funciones
2.-Procedimientos
3.-Llamados anidados y ámbito de alcance

SUBPROGRAMAS

La solución de un problema puede abarcar varios aspectos o etapas bien diferenciados, cada una de ellas, puede ser pensada e implementada como un proceso individual, la unión de todos ellos lleva a la solución del problema completo. Estos procesos son llamados subprogramas.
Un *subrograma* es un módulo o sección autónoma del programa, que realiza una tarea específica. Puede ser llamado (invocado, activado) por el programa o por otros subprogramas siempre que se necesite ejecutar la tarea para la cual fue hecho.
Al "partir" la solución, cada subprograma, esta acotado y atiende una tarea específica, siendo más fácil de interpretar y corregir, para garantizar el correcto funcionamiento. La "suma" de los mismos conforma la solución integral al problema original.

La utilización de subprogramas aporta las características de:

- Modularidad: los problemas son divididos en subproblemas (subprogramas), este enfoque aporta un nivel de abstracción para controlar la complejidad de los procesos. Para la comprensión de esta abstracción es necesario especificar el intercambio de información que se produce entre ellos a través de *parámetros.*
- Mantenimiento: los subprogramas al ser módulos independientes, pueden ser corregidos y modificados independientemente de los procesos con los que interactúan. Para controlar esta interacción, la comunicación debe darse sólo a través de sus *parámetros.*
- Ahorro de código: las tareas que se repiten pueden ser codificadas solo una vez en un subprograma, luego se lo "llama" (utiliza) tantas veces como sea necesario.

Tipos de subrogramas → FUNCION
→ PROCEDIMIENTO

1.-FUNCIÓN es un subprograma que realiza una tarea específica y devuelve un único valor (de tipo simple) en el nombre.
Consta de un encabezamiento y un cuerpo (este último tiene una parte declarativa y otra ejecutable).
El encabezamiento contiene la palabra reservada *function*, el nombre de la misma (sigue las reglas de un identificador), los parámetros y sus respectivos tipos encerrados entre paréntesis y por último el tipo del resultado que devuelve.
La función recibe información a través de parámetros y devuelve un resultado en el nombre. No lee, ni escribe.

1.1.-Declaración

Lista parámetros formales

Function NombFun (par_1:$tipo_1$; par_2: $tipo_2$;...par_n : $tipo_n$) : $tipo_f$; } ENCABEZAMIENTO
{parte declarativa : Const, Type, Var, declaración de otras funciones}
.....
Begin {parte ejecutable}
.....
NombFun := resultado; {debe figurar el nombre de la función recibiendo el resultado de $tipo_f$ } } CUERPO
End;

1.2.-Invocación o llamado

Cuando se requiere la solución que un determinado subprograma brinda, el **programa invocante** (puede ser un programa principal o un subprograma) especifica el nombre del mismo (lo llama) sustituyendo la lista de parámetros **formales** por los **actuales**. Esta llamada desencadena su ejecución, devolviendo un único valor del tipo de la función (resultado). La llamada puede ser invocada:

a) A la derecha de una asignación
b) En la condición de una estructura de control
c) En una lista de salida

Ejemplo de un programa

```
Program funciones;
Var
   X : real;
Begin
        Readln ( X );
        if  Fracc(X) =0 Then                    b)
            X  := sqrt (X) * 3.4;               a)
        Writeln ( Trunc(X));                    c)
End.
```

1.3.-Parámetros

a) Los parámetros permiten la comunicación del programa invocante con el subprograma.
b) En la declaración del subprograma, se especifican entre paréntesis los parámetros ***formales*** (identificadores y sus respectivo tipos). A través de ellos se **describen** (formalizan) las operaciones y relaciones que forman parte del subprograma.
c) La invocación a un subprograma requiere la especificación entre paréntesis de los parámetros ***actuales*** (no se especifica el tipo). Estos son los datos con los que se **realizan** las operaciones y relaciones especificadas en la declaración.
d) Los parámetros actuales deben coincidir con los formales en cantidad, orden y tipo. La posición en su respectiva lista determina la correspondencia entre ambos.
e) No es necesario que el nombre de los parámetros formales coincida con el de los actuales
f) La comunicación es unidireccional, ya que a través de los parámetros el subprograma recibe los valores, generando una copia de los mismos en su zona de memoria. Cualquier modificación que realice sobre la copia no modifica el valor del parámetro actual. Por lo tanto el programa invocante al retomar el control, encuentra los parámetros actuales valor sin cambios.
g) El parámetro formal VALOR es siempre un identificador, el parámetro actual VALOR puede ser una variable del programa invocante, una constante o una expresión (esta última se evalúa antes de copiar el valor).

Ejemplo1-Desarrollar una función booleana que indique si un carácter es vocal

```
Program Vocales;

Function EsVocal (Letra:Char) : boolean;
   Begin
   Letra:= upcase(Letra);
   EsVocal:= (Letra =´A´) or  (Letra =´E´) or  (Letra =´I´) or  (Letra =´O´) or  (Letra =´U´);
   End;
```

```
Var
Letra :char;

Begin
Write(´ingrese una letra´); Readln(Letra);
If  EsVocal(Letra) Then
   Writeln(Letra, ´ es una vocal´)
Else
   Writeln(Letra, ´ no es una vocal´)
End.
```

{*al ser Letra parámetro valor, la modificación que puede haber hecho la función, no se registra en el parámetro actual}*

Ejemplo2-Desarrollar una función (entera) que a partir de un entero N devuelva la suma de 1 a N.

```
Program SumaN;

Function Suma (N: word) : word;
Var
 Aux, i : word;
Begin
    Aux:= 0;
    For i:= 1 to N do
        Aux:= Aux + i ;
    Suma := Aux
End;
Var
  M:word;
Begin
  Write(´ingrese un número´); Readln(M);
  Writeln(´La suma es ´ , Suma (M))
End.
```

variables locales

1.4.-Variables locales, son las que define un subprograma (función o procedimiento) para tareas específicas dentro de su ámbito. Dichas variables son propias del subprograma, no tienen comunicación o alcance fuera del mismo y su permanencia en memoria se limita al tiempo en el que el subprograma está en ejecución.

Ejemplo3- Se desarrolla para el ejemplo 11 del capítulo 4 una solución que utiliza una función string, que a partir de las tres notas (Matemáticas, Física y Química) devuelve una cadena que indica si el alumno ingreso (‘C1’, ‘C2’, ‘C3’) o no (‘No’).

```
Program Cap3Ej11;    {utilizando una función}
Type
        St2= string[2];     {defino un tipo apropiado}

Function Ingresa(M, F, Q:byte) :St2;
Begin
If (M >= 40) AND (F >= 40) AND (Q >= 40) then
            Ingresa:= 'C1'
else
           if  (M >= 60) AND (Q >= 80) then
                    Ingresa:= 'C2'
           else
                    if  F + Q >= 180   then
                             Ingresa:= 'C3'
                    Else
                             Ingresa:= 'No'
End;

Var
          I, N, NM, NF, NQ, ContIng : byte;
          Nomb: string[25];
          Respuesta: St2;
begin
Readln(N);
ContIng:= 0;
For I := 1 to N do
      begin
      Readln(Nomb);
      Readln(NM, NF, NQ);
      Respuesta:= Ingresa(NM, NF, NQ);
      If   Respuesta = 'No'then
            writeln (Nomb, 'No ingresa')
      else
           Begin
           ContIng:= ContIng + 1;
            writeln (Nomb, 'ingresa por condición', Respuesta);
           End
     End;
writeln ('El % de ingresantes es ', ContIng * 100/ N: 8:2);
end.
```

Desarrollar soluciones, que utilicen funciones para resolver las problemáticas planteadas.

1.- Leer pares de números enteros desde un archivo de texto y para cada uno de ellos verificar mediante función booleana si el mayor de ellos es múltiplo del menor, además calcular e imprimir el porcentaje de los pares que cumplen.

2.- Leer M números y para cada uno calcular e informar el factorial mediante una función entera. Recordar:

$$n! = \begin{cases} 1 \text{ si } n=0 \\ n \cdot (n-1)! \text{ si } n>0 \end{cases}$$

3.- Reescribir el Ejercicio 12 del capítulo 3 para un conjunto de puntos, mediante una función string .

4.- Reescribir el Ejercicio 17 del capítulo 3 para un conjunto de pagos, calculando el importe a abonar mediante una función real.

2.-PROCEDIMIENTO es un subprograma que realiza una tarea específica, puede incluir lectura y/o escritura, devolver cero o más resultados de tipo simple o estructurado (registros, tablas).
Existen procedimientos incorporados al lenguaje Pascal (clrscr, readln), pero el programador puede definir otros "a medida" del problema que resuelve.
Consta de un encabezamiento y un cuerpo (este último tiene una parte declarativa y otra ejecutable).
El encabezamiento contiene la palabra reservada *procedure* el nombre del mismo (sigue las reglas de un identificador), los parámetros y sus respectivos tipos encerrados entre paréntesis
El procedimiento recibe y devuelve información a través de parámetros.

2.1.-Declaración

Sintaxis para la definición de procedimientos. Consta de un encabezamiento y un cuerpo.

```
Procedure  identificador (lista de parámetros);        } ENCABEZAMIENTO
 Var                    (Sección declarativa}
  .....
 Begin                  {Sección ejecutable}
  .....                                                  CUERPO
  .....
  end;
```

2.2.-Parámetros

Los conceptos especificados en el punto 1.3 para los parámetros de las funciones, son válidas para los procedimientos, que además aceptan otra forma de comunicación.

TIPOS de COMUNICACIÓN → por VALOR (unidireccional como en las funciones)

TIPOS de COMUNICACIÓN → por REFERENCIA o VARIABLE (bidireccional)

a) La comunicación bidireccional produce intercambio de información, lo que permite obtener a través de los parámetros resultados de cualquier tipo.
b) El parámetro formal no recibe el valor del parámetro actual (no copia), sino la dirección de éste, por lo tanto cuando el subprograma opera sobre el mismo no lo hace sobre una copia sino sobre la misma variable, la referencia a través de la dirección Cada vez que se modifica el parámetro formal, se modifica el parámetro actual.
c) El parámetro formal VARIABLE es siempre un identificador y el actual debe ser variable (ya que representa una dirección donde se almacenará un resultado).
d) En la definición del subprograma a este tipo de parámetro, se le antepone la palabra VAR.

2.3.-Invocación o llamado

A diferencia de la función, que retorna un valor en el lugar de la invocación, la invocación del procedimiento constituye una sentencia ejecutable en si misma. Ejemplo:

```
Clrscr;
Readln(……..);
```

Ejemplo4.- -Leer dos valores reales, calcular y mostrar su suma y su producto, utilizar un procedimiento que a partir de dos valores reales devuelva su suma y su producto.

```
Program Ej4 ;
Var
  X,Y , Sum, Prod: real;

Procedure Calcula    ( W, Z : real; Var Suma, Producto: real); {define procedimiento,  W y Z
                                  parámetros de entrada, Suma y Producto son parámetros de salida}
Begin
   Suma: =  W + Z;
   Producto : = W * Z;
End;

Begin
   Write('Ingese dos numeros)'; Readln (X, Y);
   Calcula (X, Y, Sum, Prod); {invoca el procedimiento}
   Writeln ('La suma es = ', Sum);
   Writeln ('El producto es = ', Prod);
End.
```

Ejemplo5.- Procedimiento que a partir de dos variables reales intercambia el contenido

```
.....
Procedure Intercambia (Var A,   B : real);  { A y B son parámetros de entrada salida}
Var
  Aux : real;
Begin
  Aux:= A;
  A:= B;
  B:= Aux;
End;
```

¿Cuándo se debe utilizar procedimiento y cuando una función?

Se utiliza una función cuando el proceso arroja un único resultado (de tipo simple), cuando no lee ni escribe. Un procedimiento en otro caso.
Siempre es posible escribir una función como procedimiento, pero no siempre es posible escribir un procedimiento como función. ¿por que?

Desarrollar soluciones, que utilicen procedimientos para resolver las problemáticas planteadas

5.- A partir de dos fracciones devolver la fracción suma. ¿Podría utilizarse una función que realice la misma operación?

6.- Leer de un archivo de texto los coeficientes de un conjunto de polinomios con raíces reales, calcular y mostrar las raíces de cada uno

$P(x) = ax^2 + bx + c$

7- Se tiene un conjunto de datos de N clientes de una compañía de electricidad, estos son:

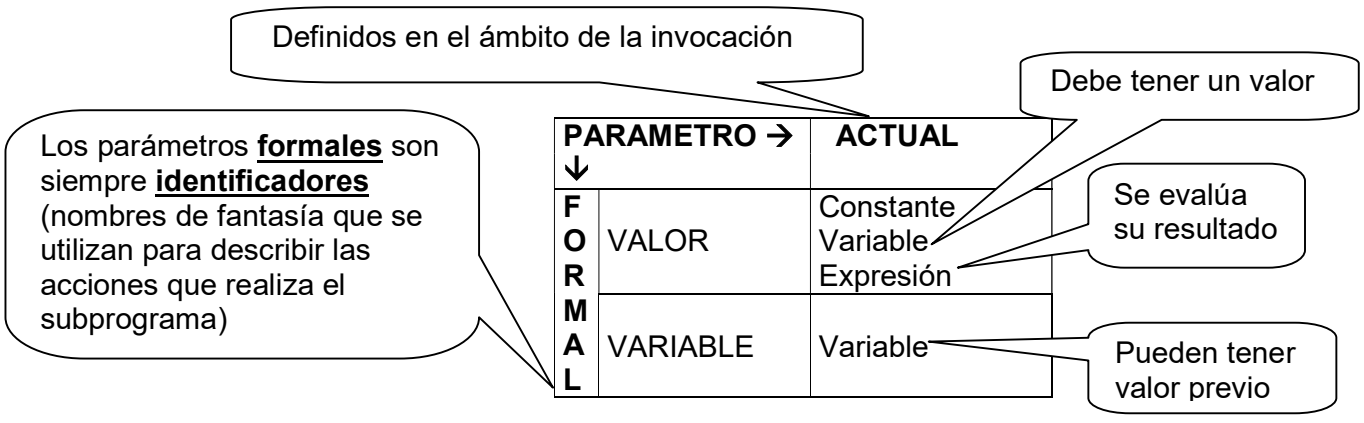
- ✓ Número de cliente
- ✓ Estado actual del medidor (en kw)
- ✓ Estado anterior del medidor

Leer de un archivo los datos de los clientes, calcular e informar el consumo, el importe junto al número de cliente.
El importe se calcula con un básico fijo de $50, más un monto variable que depende del consumo, se establece una escala de valores para el precio por kw según rangos de consumo:

$5 consumo <=100
$3.7 100< consumo <=250
$2.5 250 < consumo

8 – ¿Cual es la diferencia entre variables locales y parámetros formales?

El siguiente cuadro resume la compatibilidad entre parámetros **formales** y **actuales**, de acuerdo a su comunicación (valor : unidireccional; variable: bidireccional)



3.1.-Llamados o invocaciones anidadas

Cuando un subprograma invoca a otro subprograma, se produce un llamado “anidado”.

Notar que el parámetro actual pertenece al ámbito del subprograma que realiza la invocación, puede ser variable local o parámetro formal.

IMPORTANTE, para mantener acotado lo que una función o un procedimiento pueden operar, de acuerdo al paradigma de programación estructurada

- Una función puede invocar otra función (no un procedimiento)
- Un procedimiento puede invocar otro procedimiento o una función
- Estas invocaciones se pueden realizar en forma sucesiva. Ejemplo:

```
Program Llamados;
  Procedure A(….);
    Begin
     ……..
      End;
  Procedure B(….);
    Begin
     A (....);
     ……..
      End;
 Procedure C(….);
    Begin
     B (....);
     ……..
     End;
Begin  {programa principal}
.....
C(.....);
End.
```

3.2.-Alcance local y global

- ✓ Se define el *alcance* de un objeto (variable, tipo, función o procedimiento.) al ámbito de acción de dicho objeto. Puede ser *global* o *local*.
- ✓ En la declaración de un programa (programa principal) se especifican constantes, tipos, variables, procedimientos y funciones que son reconocidos en todo el programa incluyendo los subprogramas definidos dentro del mismo, por lo tanto estos objetos son ***globales***
- ✓ En la declaración de un procedimiento los identificadores de parámetros (formales), las definiciones de constantes, tipos, la declaración de variables, procedimientos y funciones, propios, son *locales* al procedimiento declarado, es decir, no se les conoce fuera de este ámbito, estos objetos son llamados ***locales***.

Los valores compartidos entre procesos deben implementarse mediante parámetros y ***no utilizar variables globales*** para garantizar:

* **Seguridad en el manejo de la información**
Los objetos globales son reconocidos dentro del subprograma, éste puede modificar las variables globales y producir **alteraciones no deseadas.**

*** Reutilizabilidad**
Es conveniente que los subprogramas sea módulos de código independientes, al utilizar una variable global se establece una **dependencia con el identificador** (a la inversa de la independencia de los identificadores entre los parámetros formales y actuales).

*** Claridad**
La cabecera de un subrograma permite a través de su nombre asociar la **tarea que realiza** (Suma, Escribe) y los parámetros informan que **información recibe y que resultados devuelve**.

Identificar ámbito de alcance de variables/parámetros en los siguientes programas.

```
Program Globales;
Var
X,Y, W: real;

Procedure Muestra(Z: real);
  Begin
  Writeln(X, Y, Z)
  End;

Begin {programa principal}
Readln(X,Y, W);
Muestra (W);
End.
```

```
Program Locales;
Var
W: real;

Procedure Muestra(Z: real);
 Var X,Y: real;
   Begin
   Readln(X, Y);
   Writeln(X, Y, Z)
   End;

Begin {programa principal}
Readln(W);
Muestra (W);
End.
```