Capítulo 4

1. Subprogramas - Ventajas y características de la programación modular
2. Funciones
3. Procedimientos
4. Ámbito de alcance (Local- Global)
5. Definiciones anidadas

1. 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 *subprograma* 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, está acotado y atiende una tarea específica, siendo más fácil de interpretar y corregir, para garantizar el correcto funcionamiento.
La unión de los subprogramas conforma la solución integral al problema original.

1.1. Cohesión y Acoplamiento

En la programación modular, se persigue obtener *alta cohesión* y *bajo acoplamiento*

- Cohesión: Se produce cuando los elementos (instrucciones, declaraciones, etc.) se agrupan juntos en un módulo por una razón lógica, no al azar, teniendo en cuenta que todas ellas contribuyen a realizar un objetivo común que es el funcionamiento del módulo.
  Un módulo tiene **alta cohesión** si todos sus componentes están relacionados fuertemente.
- Acoplamiento: se dice al grado de dependencia que tienen dos módulos o subprogramas entre sí. Cuando dos módulos son independientes (cada uno puede hacer su trabajo sin contar para nada con el otro), encontramos el grado más **bajo** de **acoplamiento**, y decimos que ambas unidades están totalmente desacopladas.

1.2. La utilización de subprogramas aporta las siguientes ventajas:

- 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.

1.3. Parámetros

Los parámetros permiten la comunicación del **programa invocante** (puede ser un programa principal o un subprograma) con el subprograma (procedimiento o función).
Dicha comunicación puede ser unidireccional (solo envía información al subprograma) o bidireccional (envía y recibe información del subprograma).

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.

1.5.Tipos de subprogramas

- FUNCION
- PROCEDIMIENTO

1.6. Declaración de subprogramas

La definición de funciones y procedimientos se realiza en la sección declarativa del proceso (programa o subprograma) que la utiliza, o en un archivo independiente (Unit) que es utilizado como "librería" por diferentes programas (Crt).

2. Funciones

Una función es un módulo de programa 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*, seguida del 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.

2.1. Sintaxis de la 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; *{ el nombre de la función*
End; *recibe el resultado de $tipo_f$ }*

} CUERPO

2.2. Invocación o llamado

Cuando se requiere la solución que un determinado subprograma brinda, el **programa invocante** (puede ser un programa principal o un subprograma) invoca el nombre del mismo (lo llama) sustituyendo la lista de parámetros **formales** por los **actuales**. Esta llamada desencadena su ejecución, devolviendo un valor del tipo de la función. La llamada puede formar parte de una expresión a la derecha de una asignación, en una condición que controla una estructura (if, while,etc.) o en una lista de salida.
La lista de parámetros formales es reemplazada por la lista de parámetros actuales, la función opera con los valores de estos últimos.

Una función devuelve un único valor de tipo simple (resultado) en el lugar donde se realiza la invocación. Ejemplo de un programa que invoca funciones predefinidas en Pascal.

```
Program funciones;
Var
    X : real;
Begin
   Readln ( X );
   X:= Abs( X );                    { a la derecha de una asignación }
   If  Frac( X ) < 0.5 Then         { Como parte de una condición }
       X:= X+1;
   Writeln ( Trunc ( X):8)          { En una lista de salida }
End.
```

2.3.Parámetros

Los parámetros permiten la comunicación entre la función y el programa que la invoca.

- En la declaración del subprograma, se especifican entre paréntesis los parámetros ***formales, de definición o huecos*** (identificadores y sus respectivo tipos). A través de ellos se **describen** (formalizan, definen) las operaciones y relaciones que forman parte del subprograma. Están huecos, no contienen datos.
- La invocación a un subprograma requiere la especificación entre paréntesis de los parámetros ***actuales o de llamada*** (no se especifica el tipo). Estos son los datos con los que se **realizan** las operaciones y relaciones especificadas en la declaración.
- Los parámetros actuales deben coincidir con los formales en cantidad, orden y tipo. La posición en su respectiva lista determina la correspondencia entre ambos.
- No es necesario que el nombre de los parámetros formales coincida con el de los actuales

TIPOS de COMUNICACIÓN → por VALOR (unidireccional)

TIPOS de COMUNICACIÓN → por REFERENCIA o VARIABLE (bidireccional)

2.3.1. Parámetro VALOR (unidireccional)

- ✓ Se establece la comunicación del programa invocante hacia el subprograma invocado, pasando el dato a través del parámetro..
- ✓ Al ejecutarse el subrograma se crea un espacio en memoria para el parámetro formal y se le *copia el valor* del parámetro actual.
- ✓ Cualquier modificación que el subprograma 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 (no devuelven resultados).
- ✓ El parámetro formal VALOR es siempre un identificador, el parámetro actual VALOR puede ser una variable, una constante o una expresión (esta última se evalúa antes de copiar el valor).

Los parámetros de una función son siempre por valor. (el resultado lo devuelve en el nombre)
Si los parámetros valor se reemplazan por variables, estas deben ser propias del programa invocante (y/o parámetros formales si este es a su vez otro subprograma).

2.4.Ejemplos:

Ej1-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.
```

Letra es parámetro valor, por lo tanto la modificación que hace la función, no se registra en el parámetro actual

Ej2-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

Ej3-Desarrollar una función (real) que a partir de dos fracciones devuelva su suma como un real

```
Program SumaFraccion;

Function SumaF (N1, D1, N2, D2 : shortint) : real;
Begin
   SumaF := (N1 * D2 + N2 * D1)  /  (D1 * D2);
End;
```

```
Var
   N1, D1, N2, D2 : shortint;
Begin
Write(´ingrese  numerador y denominador de la 1ra fracción´); Readln (N1, D1);
Write(´ingrese  numerador y denominador de la 2da fracción´); Readln (N2, D2);
If  D1 * D2 <> 0   entonces
     Writeln(´La suma es ´ , SumaF (N1, D1, N2, D2):8:2)
Else
     Writeln(´No son datos válidos´)
End.
```

Ej4-Desarrollar una función booleana que a partir de la medida de tres segmentos devuelva verdadero si forman triángulo. Utilizar esta función en un programa que ingrese N triplas (cada una consta de tres segmentos que pueden o no constituir un triángulo) calcule e imprima % de trángulos sobre el total de datos (triplas) leídos.
Ejemplo : (3, 4, 6), (10, 12, 7), (2, 3, 8), (8, 8, 45), (16, 9, 20) → 60%

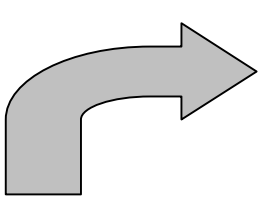
```
Program Triángulos;

Function EsTriangulo (A,B,C : real): boolean;
Begin
  EsTriangulo := (A < B+C) and (B < A+C) and ( C < A+B);
end;

Var
   A, B, C : real;
   i,  N,  ContTrian : byte;
Begin
Readln(N);
ContTrian := 0;
For i:=1 to N do
     Begin
          Readln(A, B, C);
           If EsTriangulo (A, B, C) then
          ContTrian := ContTrian + 1;
     End;
    Writeln ('El porcentaje es =', ContTrian * 100/N : 6 : 2) ;
End.
```

3. Procedimiento

Un procedimiento es un módulo de código independiente, que se ejecuta cuando es invocado o llamado por otro proceso para realizar una tarea específica, que 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.

A continuación se analiza un programa que invoca algunos procedimientos predefinidos de Pascal:

```
Program EjemploProcedimientos;
Uses
        Crt, Dos;
Var
        Anio, Mes, Dia, DS : word;
        Palabra: string;
Begin
ClrScr;  {limpia la pantalla, Crt}
GetDate(Anio, Mes, Dia, DS );       {devuelve la fecha del sistema, Dos}
Writeln(Dia,'/', Mes'/', Anio);
Palabra := 'tejido';
Delete(Palabra, 3, 2);          {borra 2 caracteres de Palabra comenzando en la posición 3}
Writeln(Palabra);
Insert('cha', Palabra, 3);       { inserta la cadena 'cha' a partir de la posición 3 de Palabra}
Writeln(Palabra);
End.
```



```
15/5/2015
tedo
techado
```

Notar que:

a. Las invocaciones a procedimientos constituyen en sí una sentencia ejecutable, a diferencia de las funciones que forman parte de una expresión y son reemplazadas por el resultado que devuelven en el mismo lugar de la invocación.
b. ClrScr es un procedimiento que no tiene parámetros ni devuelve resultados
c. GetDate es un procedimiento que devuelve 4 resultados. (parámetros de Entrada, comunicación bidireccional)
d. Delete e Insert son procedimientos que reciben y transforman valores sobre los mismos parámetros. (parámetros de Entrada/Salida, comunicación bidireccional)

3.1. Sintaxis de la declaración

Consta de un encabezamiento y un cuerpo.

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

Ejemplos:

Ej5-Leer dos valores reales, mostrar su suma y su producto, utilizar :

5_a - Procedimiento que a partir de dos valores reales imprima su suma y su producto

5_b - Procedimiento que a partir de dos valores reales devuelva su suma y su producto, el programa invocante imprime los resultados.

```
Program Ej5_a;
Var
  X,Y : real;
Procedure CalculaImprime ( W, Z : real);          {define procedimiento,
    Begin                                          W y Z parámetros formales de entrada}
    Writeln ('La suma es = ', W + Z);
    Writeln ('El producto es = ', W * Z);
    End;
Begin
Write('Ingese dos numeros'); Readln (X, Y);
CalculaImprime (X, Y);                    {invoca el procedimiento, X e Y parámetros actuales}
End.
```

Para la solución del inciso 5_b es necesario que el procedimiento comunique resultados a través de parámetros, la comunicación debe ser Bidireccional

3.2.Parámetros VARIABLE o REFERENCIA

- ✓ La comunicación es bidireccional, se produce intercambio de información, se pueden obtener a través de los parámetros resultados de cualquier tipo.
- ✓ 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.
- ✓ El parámetro formal VARIABLE es siempre un identificador y el actual debe ser variable (ya que representa una dirección donde se almacenará un resultado).
- ✓ En la definición del subprograma a este tipo de parámetro, se le antepone la palabra VAR.

Como las funciones devuelven un solo resultado (de tipo simple en el nombre), no utilizan este tipo de parámetros.

```
Program Ejemplo5_b ;
Var
  X,Y , Sum, Prod: real;

Procedure Calcula   ( W, Z : real; Var Suma, Producto: real);
{define procedimiento,  W y Z parmetros 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.
```

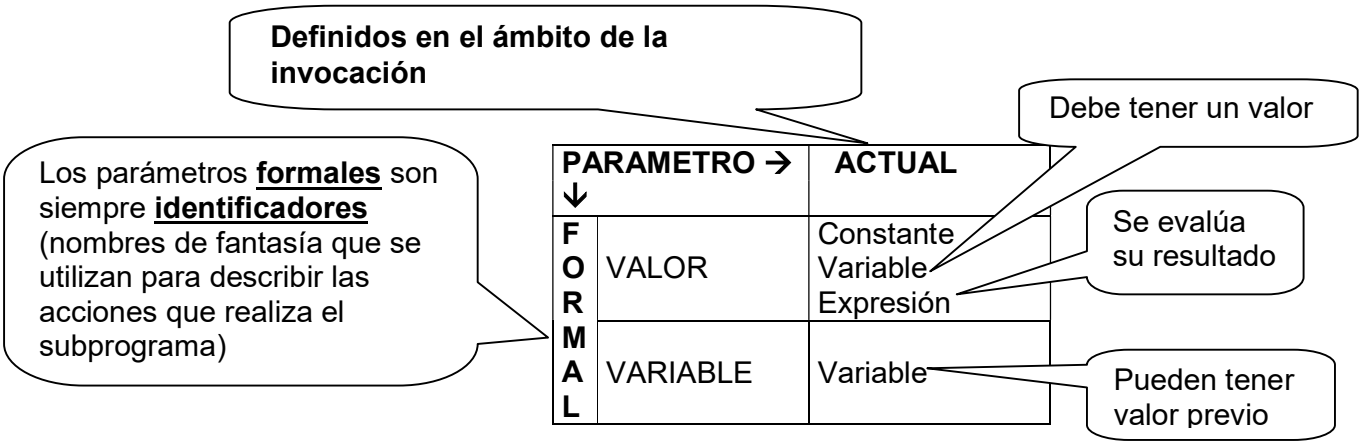
Ej6 - Procedimiento que a partir de dos variables reales intercambia el contenido

```
.....
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;
```

De lo visto se resume que los parámetros se pueden clasificar por su:

a. intervención en la definición – invocación : **formales** – **actuales**
b. forma de comunicación unidireccional – bidireccional: **valor** - **variable o referencia**
c. rol como ingreso de datos – entrega de resultados – ambos aspectos : **E** – **S** – **E/S**

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



¿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 se requiere lectura ni escritura, en caso contrario un procedimiento.
Siempre es posible escribir una función como procedimiento, pero no siempre es posible escribir un procedimiento como función.

Ej7- Se requiere calcular e imprimir las boletas de N clientes de una empresa de electricidad, por cada cliente se conoce:

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

Desarrollar un programa que lea de teclado los datos de los clientes, calcule el consumo, el importe e imprima las boletas. La boleta deberá contener el número de usuario, la cantidad de Kw consumida y el importe. Para este último considerar básico fijo de $50 y más un monto variable que depende del consumo, se establece una escala de valores para el precio por kw según rangos de consumo:

- $5 si consumo <=100
- $3.5 si 100 < consumo <=250
- $2.7 si 250 < consumo

Se presentan una solución implementada a través de tres subprogramas:

- Función **Precio** que calcula el precio del KW según el consumo
- Procedimiento **Boleta** que calcula el costo de la electricidad a partir del consumo y el precio
- Procedimiento **Imprime** que imprime la boleta

```
Prrogram Ej7;

Function Precio (Consum : word) : real; {define función, Consum parámetro de entrada}
Begin
If Consum <= 100 then
    Precio := 5
  else
    if Consum <=250 then
       Precio := 3.5
    else
       Precio :=2.7;
End;

Procedimiento Boleta( EstAct, Est Ant : word ; Var  Imp : real;  Var Cons : word);
Begin
Cons := EstAct - EstAnt;
 Imp := 50 + Cons * Precio(Cons);
End;   {Cons es parámetro actual de la función y  parámetro formal del procedimiento (*)}

Procedure Imprime( NroUsu: String ; Imp : real;  Cons : word);
begin
Writeln('Usuario', NroUsu);
Writeln('Consumo', Cons);
Writeln('Importe', Imp);
End;
```

{como se verá mas adelante, la function Precio podría haber sido declarada localmente dentro del procedimiento Boleta, en dicho caso sería local a Boleta y solo reconocida y utilizada en ese ámbito}

```
{Programa Principal}
Var
  NroUsu: string [10];
  EstAct, EstAnt, Cons, N, i : word;
  Imp : real;
Begin
Write('Ingrese cantidad de clientes'); Readln ( N);
For i:= 1 to N do
     Begin
      Readln ( NroUsu);    Readln ( EstAct, EstAnt);
      Boleta (EstAct, EstAnt,  Imp, Cons);
      Imprime( NroUsu, Imp, Cons);
      end;
End.
```

(*) Boleta invoca a la función Precio. En un llamado anidado, el parámetro actual puede ser variable local o parámetro formal del subprograma que realiza la invocación

Ej 8. Dado un archivo de texto que contiene palabras separadas por uno o más blancos terminando con un punto. Se pide leerlo e informar la/s palabra/s más larga/s y su longitud.

Ejemplo:
www dd aaa tttt fff ssss dddd.

Escribe:
estas son las palabras mas largas
tttt ssss dddd
tienen 4 caracteres

```
Program MasLargas;
Type
  St20=string[20];

Procedure Evalua(Var Todas: string; Var Max: byte);
Var
  Arch: Text;
  Car: char;
  Pal: St20;
  i: byte;
Begin
Assign (Arch, 'texto.txt');  Reset(Arch);
Max:=0;
Read(Arch, Car);
While Car <> '.'  do
  If  Car = ' ' then
    Read(Arch, Car)  {si es blanco lee otro caracter}
  else
```

```
    begin     { es el comienzo de palabra}
    Pal:= '';
    i:=0;
    While (Car <> '.') and  (Car <> ' ') do
       begin   {cuenta los caracteres y arma la palabra}
        i:=i+1;
       Pal:= Pal + Car;
       Read(Arch, Car)  ;
       End;
    If i>Max then {evalua si es m s larga}
       begin
       Todas :=  Pal;
       Max := i;
       End
    else
       if i= Max then {evalua si es igual}
          Todas:= Todas + ' '+ Pal;
    end;
Close(Arch);
End;

Var
  Todas: string;
  Max : byte;
Begin
Evalua(Todas, max);
Writeln ('estas son las palabras mas largas') ;
writeln (Todas);
Writeln ('Tienen ', Max, '  caracteres') ;
Readln;
end.
```

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
- Es necesario declarar previamente la función o procedimiento invocado (Llamados1) puede ser FORWARD), de otra forma desconoce el identificador y da error.

Estas invocaciones se pueden realizar en forma sucesiva. Ejemplo:

Program Llamados1;	Program Llamados2;

Procedure A(….);
Begin
……..
End;
 	
Procedure B(….);
Begin
A (....);
……..
End;
 	
Begin {programa principal}
.....
B(.....);
End.
	Procedure A(….); FORWARD; *{ solo declara la cabecera, más adelante, se declara el cuerpo }*
	Procedure B(….);
	Begin
	A (....);
	……..
	End;
	
	Procedure A; *{declaración de A sin parámetros}*
	Begin
	……..
	End;
	
	Begin {programa principal}
	.....
	B(.....);
	End.

4. Ámbito de alcance (Local- 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 sean 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 subprograma permite a través de su nombre asociar la **tarea que realiza** (Suma, Escribe) y los parámetros informan que **información recibe y que resultados devuelve**.

5. Definiciones anidadas

Un subprograma, en su parte declarativa, puede definir otro procedimiento y/o función, se dice que hay un anidamiento y el subprograma interno es privado del subprograma y solo reconocido dentro de su ámbito. Estas definiciones se pueden realizar en forma sucesiva. Ejemplo:

```
Program Anidamientos;
  Procedure C(….);
          Procedure B(….);
                Procedure A(….);
                Begin{cuerpo del proc. A}
                 ……..
                 End;
          Begin {cuerpo del proc. B}
           A (....);
            ……..
           End;
   Begin
    B (....);     {cuerpo del proc. C}
     ……..
    End;
Begin  {programa principal }
.....
C(......);
End.
```

Los procedimientos A y B son desconocidos en el programa principal y no pueden ser invocados por éste.

6.Tipo Subprograma

Es posible definir el tipo procedimiento y función, y luego declarar variables de dichos tipos.
Una aplicación es utilizar las variables tipo función o procedimiento como parámetros de otro subprograma.
También las componentes de una tabla (vector) pueden ser tipo procedimiento o función.
Ventaja:
permite flexibilizar el código que toma su forma definitiva en la ejecución.

Sintaxis :

```
Type
      Id_procedure  = procedure(lista de parámetros);
      Id_function     = function(lista de parámetros): tipo;
```

Ejemplos:

```
Program TiposSubprograma;
      Type
            Tfunc   = function( m: integer): real;
            Tproce  = procedure(var x, y : real);

      function F1( m: integer) : real;
                  begin
                  F1 := m*(m - 0.5);
                  End;
      function F2( m: integer) : real;
                  begin
                  F2 := m / (m - 0.5)
                  End;
```

```
function F3( m: integer) : real;
        begin
        F3 := (m - 1) / m
        End;
procedure intercambia (var x, y : real);
        var
            aux:real;
        begin
        aux := x;    x := y;    y := aux
        end;
procedure transforma (var x, y : real);
        begin
        if x < y then
             x := x/y
        else
             y := y/x
        end;
Procedure Cualquiera( m: integer; f: Tfunc);
Var
    i : integer;
begin
    for i := 1 to m do
        write ( f ( i ) : 6: 2)
end;

Var         {programa principal}
    Proc : Tproce;
    x, y : real;
    i, m: integer;
begin
  readln (x, y);
  if  (x<>0) and (y <> 0) then
        proc:= @transforma
  else
        proc:= @intercambia;
  proc(x, y);
  writeln(x, y);   readln (m);
  Write('seleccione una funcion : 1-F1; 2-F2; 3-F3'); readln( i );
  Case i of
        1:  Cualquiera(m, @F1);
        2: Cualquiera(m, @F2);
        3: Cualquiera(m, @F3);
  end;
End.
```

Consideraciones

Las funciones y procedimientos que se corresponderán con variables o parámetros de este tipo deben coincidir en la cantidad y el tipo de parámetros, y en el tipo de resultado en el caso de funciones.

Las variables de tipo función o procedimiento, almacenan las direcciones de memoria del código de un procedimiento o función compatible (son punteros).