7. FUNCIONES Y PROCEDIMIENTOS

7.1 FUNCIONES

El concepto de función en programación se fundamenta en el concepto de función matemática[1]

Una función, desde el punto de vista de la programación, se define como un proceso que recibe valores de entrada (llamados argumentos) y el cual retorna un valor resultado. Adicionalmente, las funciones son sub-programas dentro de un programa que se pueden invocar (ejecutar), desde cualquier parte del programa, es decir, desde otra función, desde la misma función o desde el programa principal[2], cuantas veces sea necesario.

Las funciones se usan cuando existen dos o más porciones de algoritmo dentro de un programa que son iguales o muy similares, por ejemplo, en un algoritmo se puede emplear varias veces una porción de algoritmo que eleva a una potencia dada un número real. De esta manera se define una función que al ser invocada ejecute dicho código, y en el lugar donde estaba la porción de algoritmo original, se hace un llamado (ejecución) de la función creada.

Una función se declara de la siguiente manera:

```
funcion nombre( arg1 : tipo1 , ..., argn : tipon ) : tipo
variables
    <declaraciones>
inicio
    <instrucciones>
    retornar <valor>
fin_funcion
```

donde,

- nombre: es el nombre de la función
- arg_i: es nombre del argumento i-esimo de la función.
- $tipo_i$: es el tipo del i-esimo argumento de la función.
- tipo: es el tipo de dato que retorna la función.
- <declaraciones>: es el conjunto de variables definidas para la función (diferentes a los argumentos).

- <instrucciones>:es el conjunto de instrucciones que realiza la función.
- <valor>:es el valor que retorna la función, puede ser una variable del tipo que retorna la función o una expresión que de cómo resultado un dato del tipo de retorno.

EJEMPLOS

Ejemplo 1. h: Reales x *Reales* $\Rightarrow$ *Reales*

$(a \ , \ b) \quad \Rightarrow \ a^2+2*b$

```
funcion h ( a : real, b : real): real
variables
inicio
    retornar  a*a + 2*b
fin_funcion
```

Ejemplo 2. minimo: *Reales* x *Reales* x *Reales* $\Rightarrow$ *Reales*

(a , b , c) $\Rightarrow$ a si a $\leqslant$ b y a $\leqslant$ c

$\Rightarrow$ b si b $\leqslant$ a y b $\leqslant$ c

$\Rightarrow$ c si c $\leqslant$ a y c $\leqslant$ b

```
funcion minimo( a : real, b : real , c : real ):
real
variables
   m : real
inicio
    si (a <=b & a <=c) entonces
       m :=a
    sino
        si (b <=a & b<=c)entonces
         m:= b
        sino
         m :=c
        fin_si
    fin_si
    retornar  m
fin_funcion
```

7.2 FUNCIONES RECURSIVAS

Una **función recursiva** es una función que se define en términos de si misma, es decir, que el resultado de la función depende de resultados de la misma función en otros valores.

Se debe tener mucho cuidado en la definición de funciones recursivas, pues si no se hace bien, la función podría requerir de un cálculo infinito o no ser calculable. Observe las siguientes definiciones, una correcta y la otra incorrecta:

Definición Recursiva correcta

$$f(x) = \begin{cases} 1 & \text{si } x \leq 1 \\ f(x-1) * x & \text{otro caso.} \end{cases}$$

Esta bien definida porque se puede calcular el valor de la función para cualquier valor que tome *x*. Por ejemplo si **x = 3.5** se tiene que **f(3.5) = 13.125** ya que:

f(3.5) = f(3.5-1.0)* 3.5 = f(2.5) * 3.5
f(2.5) = f(2.5-1.0)*2.5 = f(1.5) * 2.5
f(1.5) = f(1.5-1.0)*1.5 = f(0.5) * 1.5
f(0.5) = 1.0 (pues 0.5 <= 1.0) y de esta manera, el cálculo de la función se devuelve.
f(1.5) = f(0.5)*1.5 = 1.0 * 1.5 = 1.5
f(2.5) = f(1.5)*2.5 = 1.5 * 2.5 = 3.75
f(3.5) = f(2.5)* 3.5 = 3.75 * 3.5 = 13.125

Definición Recursiva Incorrecta

$$f(x) = \begin{cases} 1 & \text{si } x \leq 1 \\ f(x+1) * x & \text{otro caso} \end{cases}$$

Esta mal definida porque no se puede calcular el valor de la función para cualquier valor que tome *x*. Por ejemplo, si **x = 3.5** se tiene que **f(3.5)** no se puede calcular ya que:

f(3.5) = f(3.5+1.0)* 3.5 = f(4.5) * 3.5

f(4.5) = f(4.5+1.0)*4.5 = f(5.5) * 4.5
f(5.5) = f(5.5+1.0)*5.5 = f(6.5) * 5.5
f(6.5) = ... y nunca se termina este proceso

Una función recursiva es aquella que para calcular su valor en un dato dado, generalmente necesita ser calculada en uno u otros valores. Un **punto de ruptura** de la recursión es un valor del argumento para el cual la función no tiene que ser calculada de nuevo en otros valores.

EJEMPLOS

Ejemplo 1. factorial : Entero ⇒ Entero
n ⇒ 1 si n ≤1

⇒ n * factorial(n-1) en otro caso

Punto de ruptura: cuando n es igual a uno (1)
Algoritmo:

```
funcion factorial ( n :
entero ) : entero
variables
  m : entero
inicio
   si (n <=1) entonces
       m :=1
  sino
       m :=factorial(n-1) *
n
   fin_si
   retornar m
fin_funcion
```

Ejemplo 2. Fibonacci : Entero → *Entero*
n ⇒ *1 si n* ≤1
⇒ *fibonacci(n-1)+fibonacci(n-2) otro caso*

Punto de ruptura: cuando n es menor o igual a uno (1)

Algoritmo:

```
funcion fibonacci( n : entero ) :
entero variables
  m : entero
inicio
   si (n <=1 entonces)
      m :=1
   sino
      m ←fibonacci(n-1) +
fibonacci(n-2)
   fin_si
   retornar m
fin_funcion
```

Ejemplo 3. suma_rara : Entero x *Entero* ⇒ *Entero*
(n , m) ⇒ *n si n*>0 y *m* ≤0
⇒ *m si n* ≤0 y 0 ≤m
⇒ suma_rara(n-2,m-3)+ suma_rara(m-2,n-3)+ m + n

Puntos de ruptura: cuando m es menor o igual a cero (0), y cuando n es menor o igual a cero (0).

Algoritmo:

```
funcion suma_rara( n : entero, m : entero )
: entero
variables
  s : entero
inicio
   si (n > 0 &m <=0) entonces
       s :=n
   si_no
        si (n <=0 & 0 <=m) entonces
     s :=m
        si_no
         s :=suma_rara(n-2,m-3)+
suma_rara(m-2,n-3)+ m + n
        fin_si
   fin_si
```

```
    retornar s
fin_funcion
```

7.3 PROCEDIMIENTOS

En muchos casos existen porciones de código similares que no calculan un valor si no que por ejemplo, presentan información al usuario, leen una colección de datos o calculan mas de un valor. Como una función debe retornar un único valor[3] este tipo de porciones de código no se podrían codificar como funciones. Para superar este inconveniente se creó el concepto de procedimiento. Un procedimiento se puede asimilar a una función que no retorna un resultado, y que puede retornar más de un valor mediante el uso de parámetros o argumentos por referencia[4].

Los procedimientos son muy utilizados en los efectos laterales que se pueden llevar a cabo: imprimir en pantalla, modificar variables, leer datos, etc.

Un procedimiento se define de la siguiente manera

```
procedimiento nombre (
arg1: tipo1, ...)
      <declaraciones
variables>
inicio
    <instrucciones>
fin_procedimiento
```

7.4 ARGUMENTOS POR VALOR Y POR REFERENCIA

7.4.1 Argumentos por valor

Los argumentos convencionales son por valor, es decir a la función o procedimiento se le envía un valor que almacena en la variable correspondiente al argumento, la cual es local, de manera que su

modificación no tiene efecto en el resto del programa.

7.4.2 Argumentos por referencia

Sí un procedimiento tiene un argumento por referencia, este no recibe un valor, sino una referencia a una variable, es decir la misma variable (posición en memoria y valor) que envía el algoritmo que hace el llamado al procedimiento con un alias (el nombre de la variable del argumento que se recibe por referencia), por lo tanto cualquier modificación del argumento tiene efectos en el algoritmo que realizó el llamado del procedimiento. Cuando se ejecuta un procedimiento con uno o varios argumentos por referencia, ni un literal ni una constante se pueden poner en la posición de alguno de estos argumentos, es decir, ni las constantes ni los literales pueden ser pasados a un procedimiento por referencia. Un argumento por referencia se especifica anteponiendo la palabra **ref** a su definición.

En el siguiente ejemplo el argumento **A** es pasado por referencia y el argumento **B** es pasado por valor:

```
procedimiento Proc ( ref A:
entero, B: entero)
                .
inicio
                .
fin_procedimiento
```

Al realizar un llamado al procedimiento Proc, como el siguiente Proc(suma, dato), si el procedimiento modifica el argumento A que recibe por referencia, también se está modificando el contenido de la variable suma, pues suma y A en este momento son la misma variable pero con dos nombres. Mientras que si el procedimiento modifica el argumento que recibe por valor B, no está modificando la variable dato, ya que al realizar el llamado, el procedimiento crea una variable nueva para el argumento B, distinta de dato, con diferente posición de memoria, pero copiando el contenido de la variable dato en el argumento B.

EJEMPLOS.

Ejemplo 1. Desarrollar un procedimiento que intercambie los valores de dos variables enteras, es decir, que implemente el intercambio para

variables enteras.

```
procedimiento intercambio (ref x : entero, ref y
: entero)
variables
   /* variable auxiliar para realizar el intercambio
*/
    aux : entero
inicio
   /* se almacena el valor de x en la variable aux
*/
   aux:=x
   /* se almacena el valor de y en la variable x */
   x :=y
   /* se almacena el valor original de x en la
variable y */
   y :=aux
fin_procedimiento
```

Ejemplo 2. Desarrollar un procedimiento que lea una colección de hasta cien (100) números reales.

```
procedimiento leer_arreglo (ref n : entero, ref A : arreglo[100] de
real)
variables
    i : entero
inicio
   /*   las siguientes cuatro lineas son para obtener el numero de
   datos a leer. Se controla que no sea un numero invalido    */
    repetir
      escribir "Ingrese el numero de reales a operar"
       leer (n)
    hasta (0 < n I n <=100)
   /* las siguientes seis lineas leen los n datos a procesar */
    para ( i :=0 hasta n – 1) hacer
     escribir ("Ingrese el dato ")
     escribir (i)
     escribir ("-esimo")
     leer (A[i])
    fin_para
fin_procedimiento
```

7.5 INTERACCIÓN DEL PROGRAMA CON LAS FUNCIONES Y PROCEDIMIENTOS

La estructura de un programa que utiliza funciones y/o procedimientos es la siguiente:

```
<declaración de funciones y/o procedimientos>
procedimiento principal()
variables
    <declaración de variables programa>
inicio
    <instrucciones>
Fin_procedimiento
```

Donde:

- <declaración de funciones y/o procedimientos>: Es el conjunto de funciones y procedimientos declarados que se usarán en el programa. Cada función y procedimiento se definen como se mencionó en las anteriores secciones.
- <declaración de variables programa>: Es el conjunto de variables que son usadas únicamente por el programa principal, es decir, por el programa que hace llamados a las funciones y/o procedimientos.
- <instrucciones>: Es el programa principal.

7.6 VARIABLES LOCALES Y GLOBALES

7.6.1 Variables Globales

Variables definidas al comienzo del programa (antes de cualquier función), que se pueden usar a lo largo de todo el programa, es decir, dentro del algoritmo principal y en cada función definida en el programa.

7.6.2 Variables Locales

Variables que son definidas dentro de cada función y/o procedimiento, y que solo se pueden usar en la función y/o procedimientos en la que son declaradas.

Una buena técnica de programación es no usar, o usar la menor cantidad de variables globales, de tal forma que las funciones y/o procedimientos que se creen no dependan de elementos externos, en este caso las variables globales, para realizar su proceso. El no usar variables globales dentro de una función y/o procedimiento garantiza su fácil depuración y seguimiento.

```
<definición de variables
globales>
<declaración de funciones y/o
procedimientos>
procedimiento principal()
variables
    <declaración de variables
programa>
inicio
    <instrucciones>
fin_procedimiento
```

EJEMPLOS.

Ejemplo 1. Desarrollar un programa que calcule la serie de Laurent de la función exponencial en un valor x dado por, con una cantidad de n+1 de términos.

ALGORITMO:

```
Funcion factorial( n: entero ) : entero
variables
 m : entero
inicio
   si (n <=1) entonces
      m :=1
   sino
      m :=factorial(n-1) * n
  fin_si
   retornar m
fin_funcion
funcion elevar ( x : real, n : entero ) : real
variables
 y : real
 i : entero
inicio
    y :=1.0
     para (i :=1 hasta n) hacer
        y :=y * x
     fin_para
     retornar y
fin_funcion
funcion e( x : real, n : entero ) : real
variables
 i : entero
 suma : real
inicio
       suma :=0.0
       para (i :=0 hasta n) hacer
         suma :=suma + elevar( x, i ) /
factorial( i )
       fin_para
       retornar suma
fin_funcion
procedimiento principal()
variables
 n : entero
 x : real
 y : real
inicio
       escribir ("Digite el numero de
terminos a calcular de       e(x)")
       leer (n)
```

```
        escribir ("Digite el valor sobre el que
quiere calcular e(x)")          leer( x)
        y :=e( x, n )
        escribir ("El valor de e(" )
        escribir (x)
        escribir (") es ")
        escribir (y)
fin_procedimiento
```

Ejemplo 2. Desarrollar un programa que permita ingresar una colección de máximo mil (1000) números enteros y la imprima en orden ascendente y descendente.

ALGORITMO:

```
procedimiento intercambio( ref x: entero, ref y:
entero )
 variables
  /* variable auxiliar para realizar el intercambio */
  aux : entero
 inicio
      /* se almacena el valor de x en la variable aux
*/
      aux :=x
      /* se almacena el valor de y en la variable x
*/
      x :=y
      /* se almacena el valor original de x en la
variable y */
      y :=aux
fin_procedimiento
procedimiento ordenar (n: entero, ref A:
arreglo[1000] de entero)
variables
 i : entero
 j:entero
inicio
      para (i desde 0 hasta n-2) hacer
         para (j desde i+1 hasta n-1) hacer
           si (A[i]>A[j]) entonces
           intercambio ( A[i], A[j] )
           sino
          fin_si
```

```
        fin_para
      fin_para
fin_procedimiento
procedimiento imprimir_ascendente ( n : entero,
                  ref A : arreglo[1000] de entero )
variables
 i : entero
inicio
      para ( i :=0 hasta n-1) hacer
        escribir (A[i])
        escribir (" ")
      fin_para
fin_procedimiento
procedimiento imprimir_descendente ( n : entero,
referencia A : arreglo[1000] de entero )
variables
 i : entero
inicio
      para (i :=0hasta n-1) hacer
        escribir (A[n-1-i])
        escribir (" ")
    fin_para
fin_procedimiento
procedimiento leer_arreglo ( ref n : entero,
referencia A : arreglo[1000] de entero )
variables
 i : entero
inicio
    /* las siguientes cuatro lineas son para obtener
el numero de
    datos a leer. Se controla que no sea un numero
invalido */
    repita
      escribir ("Ingrese el numero de reales a
operar")
      leer (n)
    hasta (0 < n & n <=1000)
    /* las siguientes seis lineas leen los n datos a
procesar */
    para (i :=0hasta n-1) hacer
        escribir("Ingrese el dato " )
        escribir (i)
        escribir ("-esimo")
        leer (A[i])
```

```
    fin_para
fin_procedimiento
procedimiento principal()
variables
n : entero
 A : arreglo[1000] de entero
inicio
       leer_arreglo( n, A )
       ordenar( n, A )
       escribir “Esta es la colección
ascendentemente:”
       imprimir_ascendente( n, A )
       escribir (cambio_linea)
       escribir (“Esta es la colección
descendentemente:”)
       imprimir_descendente( n, A )
       escribir (cambio_linea)
fin_procedimiento
```

---

[1] Una **función** es una relación que asocia con cada elemento de un conjunto llamado el **dominio**, uno y solo un elemento de otro conjunto llamado el **codominio**. La relación puede ser establecida mediante una tabla, un proceso o un cálculo.

f:Dom ⇒ Codom

x ⇒ f(x)

Ejemplo 1. f:{a,b,c} ⇒ {0,1,2}
a ⇒ 1
b ⇒ 0
c ⇒ 2
Ejemplo 2. g: Naturales ⇒ Naturales
x ⇒ x^2
Ejemplo 3. h: Reales x Reales ⇒ Reales
(a , b) ⇒ a^2+2*b.

[2] En la siguiente sección se dirá que es el programa principal.

[3] Una función puede retornar más de un valor si ella usa argumentos por referencia. En este texto los argumentos por referencia sólo se usarán en los procedimientos ya que, es una muy mala técnica de programación el uso de argumentos por referencia en funciones. Esta consideración se hace pues, desde el punto de vista matemático, una función no puede modificar los valores de los argumentos.

[4] Los argumentos por referencia se tratarán en el siguiente aparte