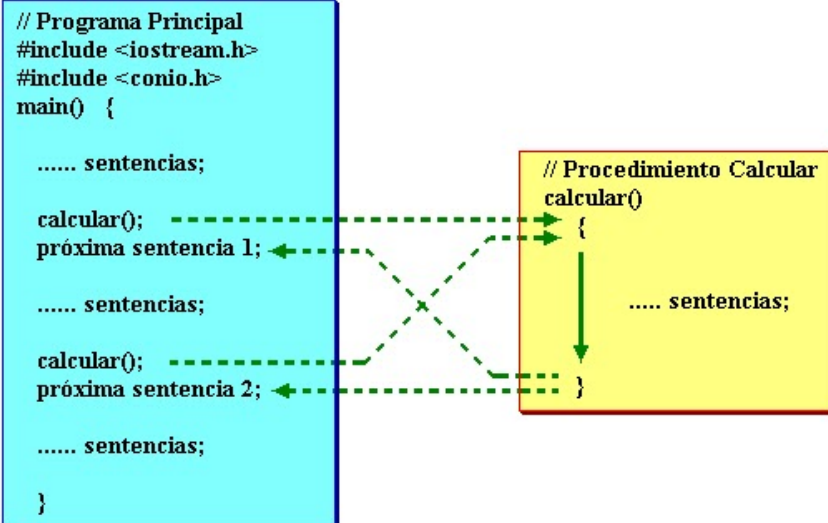




UNIVERSIDAD DISTRITAL
FRANCISCO JOSE DE CALDAS



Funciones y Procedimientos

2013

Transversal de Programación Básica

Proyecto Curricular de Ingeniería de Sistemas

Objetivos

- Aprender a resolver problemas grandes y complejos dividiendo un problema en subproblemas a través del uso de procedimientos y funciones.
- Distinguir entre parámetros de entrada (valor) y parámetros de salida (referencia).
- Analizar las diferencias entre funciones y procedimientos.
- Aprender a escribir funciones recursivas.
- Distinguir entre variables locales y variables globales.

Introducción

El concepto de función en programación se fundamenta en el concepto de función matemática. 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: Dominio	\Rightarrow	Codominio
	x	\Rightarrow	f(x)
Ejemplo 1.	f: {a,b,c}	\Rightarrow	{0,1,2}
	a	\Rightarrow	1
	b	\Rightarrow	0
	c	\Rightarrow	2
Ejemplo 2.	g: Naturales	\Rightarrow	Naturales
	x	\Rightarrow	x²
Ejemplo 3.	h: Reales x Reales	\Rightarrow	Reales
	(a , b)	\Rightarrow	a²+2*b.

Una función, desde el punto de vista de la programación, se define como un proceso que recibe valores de entrada (llamados parámetros) y el cual retorna un valor resultado. Adicionalmente, las funciones son subprogramas 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, 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 conviene definir 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.

En el pseudo-lenguaje una función se declara de la siguiente manera:

```

funcion <nombre> ( param1 : tipo1 , ..., paramn : tipon ) : tipo
variables
  <declaraciones>
inicio
  <instrucciones>
  retornar <expresión>
fin_funcion

```

Donde:

- <nombre>: representa el nombre de la función
- param_i: representa el parámetro i-ésimo de la función.
- tipo_i: representa el tipo del i-ésimo parámetro de la función.
- tipo: representa el tipo de dato que retorna la función.
- <declaraciones>: representa el conjunto de variables definidas para la función (diferentes a los parámetros).
- <instrucciones>: representa el conjunto de instrucciones que realiza la función.
- <expresión>: representa el valor que retorna la función.

Nota: Ver ejemplos 1 y 2

1. FUNCIONES RECURSIVAS

Una **función recursiva** es una función que se define en términos de sí misma, es decir, que el resultado de la función depende de resultados obtenidos de evaluar la misma función con 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

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

Está 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:

$$\begin{aligned} f(3.5) &= f(3.5-1.0) * 3.5 = f(2.5) * 3.5 \\ f(2.5) &= f(2.5-1.0) * 2.5 = f(1.5) * 2.5 \\ f(1.5) &= f(1.5-1.0) * 1.5 = f(0.5) * 1.5 \\ f(0.5) &= 1.0 \text{ (pues } 0.5 \leq 1.0) \end{aligned}$$

y de esta manera, el cálculo de la función se devuelve.

$$\begin{aligned}f(1.5) &= f(0.5) * 1.5 = 1.0 * 1.5 = 1.5 \\f(2.5) &= f(1.5) * 2.5 = 1.5 * 2.5 = 3.75 \\f(3.5) &= f(2.5) * 3.5 = 3.75 * 3.5 = 13.125\end{aligned}$$

Definición Recursiva Incorrecta

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

Está 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:

$$\begin{aligned}f(3.5) &= f(3.5+1.0) * 3.5 = f(4.5) * 3.5 \\f(4.5) &= f(4.5+1.0) * 4.5 = f(5.5) * 4.5 \\f(5.5) &= f(5.5+1.0) * 5.5 = f(6.5) * 5.5 \\f(6.5) &= \dots \text{ y nunca se termina este proceso}\end{aligned}$$

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 parámetro para el cual la función no tiene que ser calculada de nuevo en otros valores.

Nota: Ver ejemplos del 3 al 5

2. 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 más de un valor. Como una función debe retornar un único valor¹ 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 puede retornar más de un valor mediante el uso de parámetros por referencia².

Los procedimientos se usan para evitar duplicación de código y conseguir programas más cortos. Son también una herramienta conceptual para dividir un problema en subproblemas logrando de esta forma escribir más fácilmente programas grandes y complejos.

¹ Una función puede retornar más de un valor si ella usa parámetros por referencia. En este texto los parámetros por referencia sólo se usarán en los procedimientos ya que, es una muy mala técnica de programación el uso de parámetros por referencia en funciones. Esta consideración se hace pues, desde el punto de vista matemático, una función no puede modificar los valores de los parámetros.

² Los parámetros por referencia se tratarán en el siguiente aparte

En el seudolenguaje un procedimiento se define de la siguiente manera

```
procedimiento <nombre> ( param1: tipo1, ..., paramn : tipon)  
variables  
    <declaraciones>  
inicio  
    <instrucciones>  
fin_procedimiento
```

Donde:

- <nombre>: representa el nombre del procedimiento.
- param_i: representa el parámetro i-ésimo del procedimiento.
- tipo_i: representa el tipo del i-ésimo parámetro del procedimiento.
- <declaraciones>: representa el conjunto de variables definidas para el procedimiento (diferentes a los parámetros).
- <instrucciones>: representa el conjunto de instrucciones que realiza el procedimiento.

3. PARAMETROS POR VALOR Y POR REFERENCIA

3.1 Parámetros por valor

Los parámetros convencionales son por valor, es decir, a la función o procedimiento se le envía un valor que almacena en la variable correspondiente al parámetro, la cual es local, de manera que su modificación no tiene efecto en el resto del programa.

3.2 Parámetros por referencia

Si un procedimiento tiene un parámetro por referencia quiere decir que no está recibiendo 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 parámetro que se recibe por referencia). Por lo tanto, cualquier modificación al parámetro que se haga dentro del procedimiento, tiene efectos en el algoritmo que realizó el llamado al procedimiento. Cuando se ejecuta un procedimiento con uno o varios parámetros por referencia, ni un literal ni una constante se pueden poner en la posición de alguno de estos parámetros, es decir, ni las constantes ni los literales pueden ser pasados por referencia a un procedimiento. Un parámetro por referencia se especifica en pseudo-lenguaje, anteponiendo la palabra **ref** a su definición.

En el siguiente ejemplo el parámetro **A** es recibido por referencia y el parámetro **B** es recibido 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 parámetro **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 parámetro 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 parámetro **B**, distinta de **dato**, con diferente posición de memoria, pero copiando el contenido de la variable dato en el parámetro **B**.

Nota: Ver ejemplos 6 y 7

4. 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>: Representa 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>: Representa 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>: Representa el programa principal.

Nota: Ver ejemplos del 8 al 12

5. VARIABLES LOCALES Y GLOBALES

5.1 Variables Globales

Son 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.

5.2 Variables Locales

Son variables definidas dentro de cada función y/o procedimiento, y que solo se pueden usar en la función y/o procedimiento, 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.

Entonces, el esquema general de un programa es:

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

Nota: Ver ejemplos del 13 al 15

Ejemplos

A continuación se muestran algunos ejemplos con el objeto de esclarecer la teoría antes presentada

- **Ejemplo Uno**

La función h que en matemáticas se define como sigue:

$$h: \text{Reales} \times \text{Reales} \Rightarrow \text{Reales}$$

$$(a, b) \Rightarrow a^2 + 2 \cdot b$$

En programación se define así:

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

• Ejemplo Dos

La función mínimo que en matemáticas se define como sigue:

minimo: Reales x Reales x Reales \Rightarrow Reales
 minimo(a , b , c) = a , si a \leq b y a \leq c
 minimo(a , b , c) = b , si b \leq a y b \leq c
 minimo(a , b , c) = c , si c \leq a y c \leq b

En programación se define así:

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

• Ejemplo Tres

La función factorial que en matemáticas se define como sigue:

factorial : Entero + \mathbb{N} Entero +
 factorial(n) = 1, si n \leq 1
 factorial(n) = n * factorial(n-1), si n > 1

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

funcion factorial (n : entero) : entero
inicio
si (n <=1) entonces
retornar 1
sino

retornar factorial(n-1) * n
fin_si
fin_funcion

• **Ejemplo Cuatro**

La función fibonacci que en matemáticas se define como sigue:

fibonacci : Entero → Entero
 fibonacci(0)=0
 fibonacci(1)=1
 fibonacci(n)=fibonacci(n-1)+fibonacci(n-2), para n>1

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

funcion fibonacci(n : entero) : entero
inicio
si (n =0 n = 1) entonces
retornar n
sino
retornar fibonacci(n-1) + fibonacci(n-2)
fin_si
fin_funcion

• **Ejemplo Cinco**

Teniendo en cuenta la siguiente función:

suma_rara : Entero x Entero = Entero
 suma_rara(n , m) = n , si n>0 y m ≤0
 suma_rara(n , m) = m , si n ≤0 y 0 ≤m
 suma_rara(n , m) = suma_rara(n-2,m-3)+suma_rara(m-2,n-3)+ m + n, en otro caso.

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

funcion suma_rara(n : entero, m : entero) : entero
inicio
si (n > 0 & m <=0) entonces
retornar n
sino
si (n <=0 & 0 <=m) entonces
retornar m
sino
retornar suma_rara(n-2,m-3)+suma_rara(m-2,n-3)+m+n
fin_si

fin_si
fin_funcion

- **Ejemplo Seis**

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 Siete**

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 líneas 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 & n <=100)
/* las siguientes seis líneas leen los n datos a procesar */
para (i :=0 hasta n – 1) hacer
escribir ("Ingrese un dato ")
leer (A[i])
fin_para
fin_procedimiento

- **Ejemplo Ocho**

Construir un programa que determine la suma de los números desde 1 hasta un límite que lee. Se usa una función para calcular la suma de los números desde 1 hasta el límite.

funcion SumaHasta(limite : entero) : entero
variables
numero : entero
suma : entero
inicio
suma:=0
para (numero:=1 hasta limite) hacer
suma := suma + numero
fin_para
retornar suma
fin_funcion
procedimiento principal()
variables
x : entero
y : entero
inicio
escribir("Ingrese el límite para suma")
leer(x)
y := SumaHasta(x)
escribir("La suma de los números desde 1 hasta")
escribir(x)
escribir("es:")
escribir(y)
fin_procedimiento

Nota: Observe que el cálculo de la función SumaHasta se guarda en el programa principal en la variable **y**, que es del mismo tipo de la función. La función solo puede retornar un valor.

- **Ejemplo Nueve**

Calcular el área y la circunferencia de un círculo cuyo radio se lee en el programa principal.

procedimiento circulo(rad : real, ref ar : real, ref circun : real)
constantes
pi = 3.14159
inicio
ar := pi*rad*rad
circun := 2*pi*rad
fin_procedimiento
procedimiento principal()
variables
radio : real
area : real
circunferencia : real
inicio
escribir("Ingrese el valor del radio")
leer(radio)
circulo(radio, area, circunferencia)
escribir("Para un círculo de radio ")
escribir(radio)
escribir(cambio_linea)
escribir("El área es:")
escribir(area)
escribir(cambio_linea)
escribir("La circunferencia es:")
escribir(circunferencia)
fin_procedimiento

Nota: Observe que el llamado al procedimiento no es una proposición de asignación como lo es la función; en su lugar se reciben los valores en los parámetros referencia **area** y **circunferencia**, y en este sentido se dice que el procedimiento puede devolver mas de un valor.

- **Ejemplo Diez**

Construir un programa que reciba 3 valores y devuelva el mínimo valor. El programa pregunta al usuario si quiere entrar más datos y se repita mientras el usuario quiera. Utilizar una función.

funcion minimo (a : real, b : real , c : real): real
inicio
si (a <=b & a <=c) entonces
retornar a
sino
si (b <=a & b<=c)entonces
retornar b
sino
retornar c
fin_si
fin_si
fin_funcion
procedimiento principal()
variables
x : entero
y : entero
z : entero
menor : entero
c : caracter
inicio
haga
escribir("Digite el primer número:")
leer(x)
escribir("Digite el segundo número:")
leer(y)
escribir("Digite el tercer número:")
leer(z)
menor:= minimo(x,y,z)
escribir(cambio_linea)
escribir("El menor valor entre:")
escribir(x)
escribir(y)
escribir(z)
escribir("es")
escribir(menor)
escribir("Desea entrar más datos? s/S")
leer(c)
mientras(c='s' c='S')
fin_procedimiento

- **Ejemplo Once**

Construir un programa que reciba 3 valores y devuelva el mínimo valor. El programa pregunta al usuario si quiere entrar más datos y se repita mientras el usuario quiera. Utilizar un procedimiento.

procedimiento minimo(a : real , b : real , c : real , ref 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
fin_procedimiento
procedimiento principal()
variables
x : entero
y : entero
z : entero
menor : entero
c : caracter
inicio
haga
escribir("Digite el primer número:")
leer(x)
escribir("Digite el segundo número:")
leer(y)
escribir("Digite el tercer número:")
leer(z)
minimo(x,y,z,menor)
escribir(cambio_linea)
escribir("El menor valor entre:")
escribir(x)
escribir(y)
escribir(z)
escribir("es")

escribir(menor)
escribir("Desea entrar más datos? s/S")
leer(c)
mientras(c='s' c='S')
fin_procedimiento

Algoritmo En Código C

#include <iostream.h>
#include <stdlib.h>
void minimo(float a, float b, float c, float &m)
{
if (a<=b&& a<=c)
{
m=a;
}
else
{
if (b<=a&&b<=c)
{
m=b;
}
else
{
m=c;
}
}
}
int main()
{
float x,y,z,menor;
char car;
do
{
cout<<"digite el primer numero: " ;
cin >> x;
cout<<"digite el segundo numero: " ;
cin >> y;
cout<<"digite el tercer numero: " ;
cin >> z;

minimo(x,y,z,menor);
cout <<"\n";
cout << "El menor entre \n" << x <<"\t" <<y <<"\t" <<z <<" es " <<menor <<"\n";
cout <<"Desea entrar mas datos ? " ;
cin >>car;
}
while (car=='s' car=='S');
system("PAUSE");
return 0;
}

Nota: Observe las diferencias del algoritmo escrito mediante un procedimiento y el ejemplo anterior escrito mediante una función.

- **Ejemplo Doce**

Escribir un programa que lea 3 números y los clasifique en orden ascendente.

procedimiento intercambio(ref x : entero, ref y : entero)
variables
aux : entero
inicio
aux := x
x := y
y := aux
fin_procedimiento
procedimiento clasificacion (ref primero : entero, ref segundo : entero, ref tercero : entero)
inicio
si(primeros > segundo)entonces
intercambio(primeros,segundo)
fin_si
si(segundo > tercero)entonces
intercambio(segundo , tercero)
si(primeros > segundo)entonces
intercambio(primeros , segundo)
fin_si
fin_si
fin_procedimiento
procedimiento principal()

variables
a, b, c : entero
inicio
escribir("Digite el primer número :")
leer(a)
escribir("Digite el segundo número :")
leer(b)
escribir("Digite el tercer número :")
leer(c)
clasificacion(a,b,c)
escribir("Los números en orden ascendente son:")
escribir(a)
escribir(" ")
escribir(b)
escribir(" ")
escribir(c)
fin_procedimiento

Nota: Observe que el procedimiento intercambio es llamado desde el procedimiento clasificación.

- **Ejemplo Trece**

Se quiere escribir un programa que imprima 10 triángulos, alternando triángulos que tienen 6 renglones de asteriscos con otros que tienen 7 renglones de x así:

```

*
**
***
****
*****
*****

X
XX
XXX
XXXX
XXXXX
XXXXXX
XXXXXXX

```

En el pseudo lenguaje el algoritmo es:

constantes
numTriang = 10
variables
i : entero
procedimiento Dibujar(numFila : entero, car : caracter)
variables
columna : entero
inicio
para(i := 1 hasta numFila) hacer
para(columna := 1 hasta i) hacer
escribir(car)
fin_para
escribir(cambio_linea)
fin_para
fin_procedimiento
procedimiento principal()
inicio
variables
limite : entero
para(limite= 1 hasta numTriang) hacer
Dibujar(6,'*')
escribir(cambio_linea)
Dibujar(i,'x')
escribir(cambio_linea)
fin_para
fin_procedimiento

En código de lenguaje C el mismo algoritmo es:

<code>#include <iostream.h></code>
<code>#include <stdlib.h></code>
<code>int numTriang = 10 ;</code> <i>/*constantes*/</i>
<code>int i;</code> <i>/*variable global*/</i>
<code> <i>/****** definición del procedimiento******/</i></code>
<code>void Dibujar(int numFila, char car)</code>
<code>{</code>
<code> int columna;</code> <i>/*variable local del procedimiento*/</i>
<code> for (i = 1;i<= numFila; i++)</code>

{
for (columna = 1; columna<= i; columna++)
{
cout<< car;
}
cout<< "\n";
}
}
int main() /*programa principal */
{
int limite; /*variable procedimiento principal*/
for(limite= 1; limite<= numTriang; limite++)
{
Dibujar(6,'*');
cout<< "\n";
Dibujar(i,'x');
cout<< "\n";
}
system("PAUSE");
return 0;
}

Nota: La variable *i* es global por estar definida al comienzo del programa y por lo tanto puede ser usada dentro de cualquier procedimiento.

Note en este ejemplo que para evitar duplicación de código se usa el procedimiento Dibujar.

• **Ejemplo Catorce**

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

funcion factorial(n: entero) : entero
inicio
si (n <=1) entonces
retornar 1
sino
retornar factorial(n-1) * n
fin_si
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, y : real
inicio
escribir ("Digite el numero de términos 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 Quince**

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

procedimiento intercambio(ref x: entero, ref y: entero)
variables
aux : entero /* variable auxiliar para realizar el intercambio */
inicio
aux :=x /* se almacena el valor de x en la variable aux */
x :=y /* se almacena el valor de y en la variable x */
y :=aux /* se almacena el valor original de x en y */
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, ref A : arreglo[1000] de entero)
variables
i : entero
inicio
para (i :=0 hasta n-1) hacer
escribir (A[n-1-i])
escribir (" ")

fin_para
fin_procedimiento
procedimiento leer_arreglo (ref n : entero,
ref A : arreglo[1000] de entero)
variables
i : entero
inicio

Nota: Puesto que en los dos ejemplos anteriores no hay variables definidas al comienzo del programa, las variables son locales al procedimiento o función donde fueron definidas y solo pueden ser usados dentro de estos.

Ejercicios

1. Elaborar un programa que reciba tres datos y los ordene descendentemente. El programa debe elaborarse en base a procedimientos. Utilice el procedimiento **intercambio** ya presentado en este capítulo.
2. Elaborar un programa que calcule la siguiente suma mediante el uso de una función,

$$s = \sum_{i=1}^n i^2$$

3. Elaborar una función que reciba un número entero y retorne -1 si el número es negativo. Si el número es positivo debe devolver una clave calculada de la siguiente manera: Se suma cada dígito que compone el número y a esa suma se le calcula el modulo 7. Por ejemplo: para la cifra 513, la clave será $5+1+3=9$; $9 \bmod 7 = 2$. Utilice la función para construir un programa que lea una secuencia de valores y determine si el número leído fue negativo o si fue positivo que clave le corresponde.
4. Desarrolle el punto 3 pero en lugar de utilizar una función utilice un procedimiento.
5. Construir un programa que reciba 3 valores y devuelva el valor mínimo y máximo. El programa debe resolverse utilizando funciones.
6. Resolver el punto 5 pero utilizando un solo procedimiento que devuelva el máximo y el mínimo.
7. Elaborar un programa que lea **n** caracteres y los almacene en un arreglo. Se cumple siempre que el número de caracteres es menor a 15. Cada carácter debe almacenarse en un elemento del arreglo. El conjunto de caracteres forma una palabra. El programa debe escribir si la palabra es palíndromo o no. Utilizar una función que tome el arreglo y devuelva 0 si la palabra no es un palíndromo o 1 si lo es.
8. Elaborar el programa del punto 7 pero con un procedimiento.

Resumen

Los procedimientos y funciones son una herramienta conceptual para dividir un problema en subproblemas logrando de esta forma escribir más fácilmente programas grandes y complejos. Además evitar la duplicación de código, consiguiendo escribir programas más cortos.

Las funciones solo pueden retornar un valor único y aquí se utilizarán solamente con parámetros de entrada por valor.

Los procedimientos pueden contener parámetros de entrada o valor y parámetros de salida o referencia. En el sentido del uso de los parámetros por referencia se dice que un procedimiento puede retornar más de un valor, tantos como parámetros referencia se coloquen.

Las variables globales son las que se definen al comienzo del programa y pueden ser usadas por cualquier procedimiento o función.

Referencias

1. Becerra, César. Algoritmos. Ed. César Becerra, 1993.
2. Séller Arthur. Programación en Pascal. Ed Mc Graw Hill, 1982
3. Konvalina John Stanley Wileman. Programación con Pascal. Ed Mc Graw Hill, 1989.

Lectura de Profundización:

- Texto tomado de:
<http://aplicaciones.virtual.unal.edu.co/drupal/files/Funciones%20y%20Procedimientos%20-%20Programacion%20de%20computadores.pdf>
- Profundizar: <http://www.itescam.edu.mx/principal/sylabus/fpdb/recursos/r62707.PDF>

Imágenes

Las imágenes fueron tomadas de www.google.com

Fuentes:

- http://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n
- http://sisbib.unmsm.edu.pe/bibvirtual/publicaciones/indata/v04_n1/lenguajes.htm
- <http://ocw.unican.es/enseanzas-tecnicas/fortran-para-la-simulacion-electromagnetica/clase/capitulo9.pdf>
- <http://ocw.upm.es/ciencia-de-la-computacion-e-inteligencia-artificial/fundamentos-programacion/contenidosteoricos/ocwfundamentosprogramaciontema7.pdf>