Guía práctica de estudio 12: Funciones

Elaborado por:
Ing. Jorge A. Solano Gálvez
Guadalupe Lizeth Parrales Romay

Revisado por:
M.C. Edgar E. García Cano

Autorizado por:
M.C. Alejandro Velázquez Mena

Guía práctica de estudio 12: Funciones

Objetivo:

Elaborar programas en lenguaje FORTRAN que permitan dividir la solución del problema en funciones.

Actividades:

- Distinguir lo que es el prototipo de una función y la implementación o definición de ella.
- Utilizar parámetros tanto en la función principal como en funciones secundarias.

Introducción

Como ya se mencionó, un programa en lenguaje FORTRAN consiste en una o más funciones. FORTRAN permite tener dentro de un archivo fuente varias funciones, esto con el fin de dividir las tareas y que sea más fácil la depuración, la mejora y el entendimiento del código.

Licencia GPL de GNU

El software presente en esta guía práctica es libre bajo la licencia GPL de GNU, es decir, se puede modificar y distribuir mientras se mantenga la licencia GPL.

```
/*
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
 *
 * Author: Jorge A. Solano
 *
*/
```

Funciones y subrutinas

FORTRAN provee dos maneras de implementar subprogramas: creando funciones o creando subrutinas. Estos subprogramas pueden ser de dos tipos intrínsecos y externos. Los subprogramas intrínsecos son aquellos que provee el compilador, algunos de los más comunes son:

abs	valor absoluto
min	valor mínimo
max	valor máximo
sqrt	raíz cuadrada
sin	seno
cos	coseno
tan	tangente
atan	arco tangente
exp	exponente (natural)
log	logaritmo (natural)

Por otro lado, los subprogramas externos son aquellos escritos por el usuario (o bien pueden formar parte de una biblioteca desarrollados por terceros). Estos son los que se abordarán en este tema.

Función

La sintaxis básica de una función (también conocida como definición de la función) es la siguiente:

```
valorRetorno nombre (parámetros)
        ! bloque de código de la función
END
```

Por otro lado, el prototipo de una función está compuesto por tres elementos: el nombre de la función, los parámetros que recibe la función y el valor de retorno de la función.

El nombre de la función se refiere al identificador con el cual se ejecutará la función; es recomendable seguir la notación de camello.

Una función puede recibir parámetros de entrada, dichos parámetros se deben definir dentro de los paréntesis de la función y separados por comas, es decir:

```
(nom1, nom2,  nom3…)
  tipoDato nom1, nom2,  nom3…
```

El tipo de dato puede ser cualquiera de los vistos hasta el momento (entero, lógico, real, carácter, complejo o arreglo). Los parámetros de una función son opcionales.

El valor de retorno de una función indica el tipo de dato que va a regresar la función al terminar el bloque de código de la misma. El valor de retorno puede ser cualquiera de los tipos de datos vistos hasta el momento (entero, lógico, real, carácter, complejo o arreglo).

Código (función)

```
      program principal

! Programa que permite calcular los valores de un polinomio
! de grado 2, es decir, del tipo A*x^2 + B*x + C

      real sum, x, coef
      integer m

      write(*,*) 'Escriba el valor de x'
      read(*,*) x

      sum = 0.0
      do 10 m = 0, 2
         write(*,*) 'Escriba el valor del coeficiente de x', m
         read(*,*) coef
         sum = sum + (coef * potencia(x, m))
  10  continue
      write (*,*) 'El valor del polinomio valuado en ', x, 'es', sum

      stop
      end

      real function potencia(x,m)
         real x
         integer m
         potencia = 1.0
         do i = 1, m
            potencia = potencia * x
         enddo

         return
       end
```

La función *principal* hace una llamada a la función *potencia* pasando como parámetro los valores x y m. La función *potencia* ejecuta su bloque de código y cuando termina regresa el valor almacenado en potencia, que es de tipo real.

Subrutina

Una función en FORTRAN solo puede regresar un valor. Algunas ocasiones es necesario regresar dos o más valores, en este caso se puede hacer uso de las subrutinas de FORTRAN. Su sintáxis básica es la siguiente:

```
SUBROUTINE nombre (parámetros)
      ! bloque de código de la función
END
```

Como se puede observar, las subrutinas no poseen un valor de retorno explícito en la declaración de la misma. Además, para invocar una subrutina se debe hacer uso de la palabra reservada CALL antes del nombre y los parámetros de la misma.

Código (subrutina)

```
      program principal
! Programa que realiza el ordenamiento descendente de 2 números dados
      integer m, n
      m = 1
      n = 2
      if (m .LE. n) then
         call intercambiar(m, n)
      endif
      write(*,*) m, n
      stop
      end

      subroutine intercambiar (a,b)
         integer a, b
! Variables locales a la subrutina intercambiar
         integer tmp
         tmp = a
         a = b
         b = tmp
         return
       end
```

La función *principal* manda llamar a la subrutina *intercambiar* pasando como parámetros las variables m y n. La función *intercambiar* recibe como parámetros los valores y los guarda en las variables a y b, los cuales intercambia dentro del bloque de código de la subrutina. Cuando la subrutina llega a su fin, los valores que se intercambiaron son, en

realidad, los que envió la función principal y, por ende, se ven afectados directamente, es decir, los valores que se intercambiaron fueron m y n.

Código (subrutina)

```
      program principal

      integer ene
      parameter (ene = 5)
      real equis(ene), ye(ene), escalar
      parameter (escalar = 4)
      do 10 i = 1, ene
         equis(i) = i * 2
         ye(i) = i * 1
   10 continue

      call saxpy(ene, escalar, equis, ye)

      do 11 i = 1, ene
         write(*,*) ye(i)
   11 continue

      stop
      end

      subroutine saxpy (a,b)
! La operación SAXPY se calcula de la siguiente manera:
! y = alfa*x + y,
! donde ‘x’ y ‘y’ son vectores de longitud n.
         integer n, i
         real alfa, x(*), y(*)
         do 10 i = 1, n
            y(i) = alfa*x(i) + y(i)
   10    continue
         return
       end
```

Ámbito o alcance de las variables

Las variables declaradas dentro de un programa tienen un tiempo de vida que depende de la posición donde se declaren, es decir, las variables en FORTRAN sólo existen dentro de las funciones o subrutinas donde se declaran y la única manera de enviar el o los valor(es) a otro subprograma es pasarlo(s) como parámetro(s). Sin embargo, esto puede ser inconveniente cuando el número de parámetros es muy grande. Para ello FORTRAN cuenta con lo que se conoce como un bloque común (common block), que se puede compartir con varios subprogramas.

Como ya se mencionó, un programa en FORTRAN puede contener varios subprogramas. Las variables que se declaren dentro de cada subprograma se conocen como variables locales (le pertenecen únicamente a la función o subrutina). Estas variables existen al momento de que el subprograma es llamado y desaparecen cuando éste llega a su fin.

```
      integer function sumar()
         integer x
         !ámbito de la variable x desde aquí hasta el final de la función
```

Las variables que se desean compartir entre varios subprogramas se llaman variables comunes. Las variables comunes existen durante la ejecución de todo el programa y pueden ser utilizadas por cualquier función o subrutina.

Código (Bloque común)

```
      program commonBlock

       integer a, b
! Se comparten las variables a y b en un bloque común llamado global
       common /global/ a, b
       a = 5
       b = 4
       write(*,*) multiplicar()
       stop
       end

       integer function multiplicar()
         integer a, b
```

```
! Cualquier función puede ver el bloque común y puede utilizar los valores
! o asignar valores a las variables del bloque.
         common /global/ a, b
         multiplicar = a*b
         return
       end function
```

En el ejemplo anterior, para comprobar que las variables que se están utilizando son, en efecto, las variables del bloque común, se puede comentar la invocación a este bloque dentro de la función multiplicar. Si no se utilizan las variables del bloque común entonces se utilizarán las variables a y b declaradas dentro de la función multiplicar, las cuales no han sido asignadas a valor alguno, dando como resultado cero en la operación multiplicación.

Código (Bloque común)

```
      program incrementaVariable

c La función incremento aumenta el valor de la
c variable enteraGlobal cada vez que es invocada.

c Solo la variable enteraGlobal es visible para
c otras funciones, ya que se encuentra en un bloque común

       integer enteraGlobal, enteraLocal
       common /global/ enteraGlobal

       do enteraLocal = 1,5,1
          write (*,*) incremento(),'+',enteraLocal,'='
          write (*,*) enteraGlobal + enteraLocal
       enddo

       stop
       end

      integer function incremento()
         integer enteraGlobal
         common /global/ enteraGlobal

         enteraGlobal = enteraGlobal + 2
         incremento = enteraGlobal

      end function
```

Bibliografía

- Oracle (2010). Fortran 77 Languaje Reference. Consulta: Julio de 2015. Disponible en: http://docs.oracle.com/cd/E19957-01/805-4939/

- Stanford University (1995). Fortran 77 Tutorial. Consulta: Julio de 2015. Disponible en: http://web.stanford.edu/class/me200c/tutorial_77/