Lenguajes Imperativos

Moisés García Villanueva

24 de Agosto de 2012

1 Variables y asignación

En programación, las variables son espacios reservados en la memoria que, como su nombre indica, pueden cambiar de contenido a lo largo de la ejecución de un programa. Una variable corresponde a un área reservada en la memoria principal de la computadora pudiendo ser de longitud:

- Fija.- Cuando el tamaño de la misma no variará a lo largo de la ejecución del programa. Todas las variables, sean del tipo que sean tienen longitud fija, salvo algunas excepciones — como las colecciones de otras variables (arrays) o las cadenas.
- Variable.- Cuando el tamaño de la misma puede variar a lo largo de la ejecución. Típicamente colecciones de datos.

1.1 Tipos de datos

Debido a que las variables contienen o apuntan a valores de tipos determinados, las operaciones sobre las mismas y el dominio de sus propios valores están determinadas por el tipo de datos en cuestión. Algunos tipos de datos usados:

- Tipo de dato lógico.
- Tipo de dato entero.
- Tipo de dato de coma flotante (real, con decimales).
- Tipo de dato carácter.
- Tipo de dato cadena

1.2 Ejercicios

1. Defina tipos de datos para almacenar:
   (a) La cantidad de habitantes en China.
   (b) Operaciones de gran precisión al utilizar funciones trigonómetricas.
   (c) Los nombres de personas
   (d) El número de caracteres de un documento.
2. Defina un nuevo tipo de dato en un lenguaje de programación.

2 Goto, comandos no estructurados

GOTO o GO TO (**ir a** en inglés) es una instrucción muy común en los lenguajes de programación con el objetivo de controlar el flujo del programa. El efecto de su versión más simple es transferir sin condiciones la ejecución del programa a la etiqueta o número de línea especificada. Es una de las operaciones más primitivas para traspasar el control de una parte del programa a otra; tal es así que muchos compiladores traducen algunas instrucciones de control como GOTO.

La instrucción se puede encontrar en muchos lenguajes; uno de los primeros lenguajes de alto nivel que lo incluyeron fue el FORTRAN, desarrollado en 1954. También se encuentra en: Algol, COBOL, SNOBOL, BASIC, Lisp, C, C#, C++, Pascal y Perl entre otros, especialmente el lenguaje ensamblador. En este último se lo puede encontrar como BRA (de branch: ramificar en inglés), JMP o JUMP (saltar o salto en inglés) y es, generalmente, el único modo de organizar el flujo del programa.

Existe incluso en lenguajes usados para la enseñanza de programación estructurada, como Pascal. Sin embargo, no está en todos los lenguajes de programación, en algunos (como Java) es una palabra reservada y en el paródico lenguaje INTERCAL se utiliza COME FROM (venir de en inglés).

Se pueden encontrar también variaciones de la instrucción GOTO. En BASIC, la instrucción ON GOTO puede seleccionar de una lista de diferentes puntos del programa a los que saltar. Podría ser interpretado como un antecesor de la instrucción switch/case.

2.0.1 Etiquetas y goto en el lenguaje C

En el lenguaje C, la estructura para utilizar la instrucción `goto`, es la siguiente:

```
goto <nombre_de_la_etiqueta>;
```

Como se mencionó anteriormente, esta instrucción de salto, se puede usar en un programa para transferir incondicionalmente el control del mismo a la primera instrucción después de una etiqueta, o dicho de otra forma, al ejecutar una instrucción goto, el control del programa se transfiere (salta) a la primera instrucción después de una etiqueta. Una etiqueta se define mediante su nombre (identificador) seguido del carácter dos puntos (:).

2.0.2 Ejemplo de uso de la etiqueta goto

En el siguiente ejemplo se observa como se utiliza la instrucción `goto` en lugar de la instrucción `break`:

```
#include <stdio.h>

int main()
{

   int n, a;

   a = 0;
   do
   {
     printf( "Introduzca un numero entero: " );
     scanf( "%d", &n );

     if ( n == 0 )
     {
      printf( "ERROR: El cero no tiene opuesto.\n" );
      goto etiqueta_1;
      /* En el caso de que n sea un cero,
```

```
            el control de programa salta a la
            primera instrucción después de
            etiqueta_1. */
        }
        printf( "El opuesto es: %d\n", -n );
        a += n;
    } while ( n >= -10 && n <= 10 );


    etiqueta_1:
    printf( "Suma: %d", a );

    return 0;
}
```

2.0.3 Ejercicios

1. Hacer un programa que implemente las funciones aritméticas básicas de una calculadora, utilizando la instrucción `goto`.

2. Implementar un ciclo utilizando la instrucción `goto`.

2.1 Comandos no estructurados

La programación no estructurada está compuesta de secuencias de instrucciones en donde los saltos y el fin de programa no siguen ninguna estructura. Los saltos pueden apuntar a cualquier punto del código lo que ocasiona que el algoritmo termine siendo indecifrable. Además de que tampoco se puede saber cuando termina.

Este tipo de programación persiste en lenguajes como el Assembler (Ensamblador).

```
10 size 600,600
 20 a=0
 30 b=0
 40 if a<300 then goto 70
 50 rect a,b,40,40
 60 goto 80
 70 ellipse a,b,40,40
 80 b = b + 50
 90 if b>600 then goto 110
 100 goto 40
 110 a = a + 50
 120 if a>600 then end
 130 goto 30
```

El programa de arriba es un ejemplo de programación no estructurada. Este programa lo que hace es llamar la función `rect` en el caso de que la variable `a` tenga un valor menor a 300, caso contrario llamará la función `ellipse`. El problema principal de este tipo de programación son las sentencia goto que permiten saltar a cualquier línea del programa. Por ejemplo, la línea 40 (que dice if a¡300 then goto 70) se puede traducir cómo "si la variable a tiene un valor mayor a 300 entonces hay que saltar a la línea 70). En este programa existen 5 sentencias goto, lo que complica bastante la lectura de este muy simple algoritmo. También es muy difícil determinar cuando terminará el programa, si bien la sentencia end se encuentra en la línea 120, no podemos saber cuándo es que el algoritmo llegará (y si llegará realemente) a esa línea.

3 Programación estructurada

La programación estructurada es una técnica para escribir programas (programación de computadora). Para ello se utilizan únicamente tres estructuras: secuencia, selección e iteración; siendo innecesario el uso de la instrucción o instrucciones de transferencia incondicional (GOTO, EXIT FUNCTION, EXIT SUB o múltiples RETURN).

Hoy en día las aplicaciones informáticas son mucho más ambiciosas que las necesidades de programación existentes en los años 1960, principalmente debido a las aplicaciones gráficas, por lo que las técnicas de programación estructurada no son suficientes. Ello ha llevado al desarrollo de nuevas técnicas, tales como la programación orientada a objetos y el desarrollo de entornos de programación que facilitan la programación de grandes aplicaciones.

3.1 Orígenes de la programación estructurada

A finales de los años 1970 surgió una nueva forma de programar que no solamente daba lugar a programas fiables y eficientes, sino que además estaban escritos de manera que facilitaba su comprensión posterior.

El teorema del programa estructurado, propuesto por Böhm-Jacopini, demuestra que todo programa puede escribirse utilizando únicamente las tres instrucciones de control siguientes:

1. Secuencia
2. Instrucción condicional.
3. Iteración (bucle de instrucciones) con condición al principio.

Solamente con estas tres estructuras se pueden escribir todos los programas y aplicaciones posibles. Si bien los lenguajes de programación tienen un mayor repertorio de estructuras de control, éstas pueden ser construidas mediante las tres básicas citadas.

Estructura secuencial

Una estructura de programa es secuencial si las instrucciones se ejecutan una tras otra, a modo de secuencia lineal, es decir que una instrucción no se ejecuta hasta que finaliza la anterior, ni se bifurca el flujo del programa.

Ejemplo:

```
INPUT x
INPUT y
auxiliar= x
x= y
y= auxiliar
PRINT x
PRINT y
```

Esta secuencia de instrucciones permuta los valores de x e y, con ayuda de una variable auxiliar, intermedia.

1. Se guarda una copia del valor de x en auxiliar.
2. Se guarda el valor de y en x, perdiendo su valor anterior, pero se mantiene una copia del contenido en auxiliar.
3. Se copia a y el valor de auxiliar, que es el valor inicial de x.

El resultado es el intercambio de los valores entre x e y, en tres operaciones secuenciales.

Estructura selectiva o de selección

La estructura selectiva permite que la ejecución del programa se bifurque a una instrucción (o conjunto) u otra/s, según un criterio o condición lógica establecida, sólo uno de los caminos en la bifurcación será el tomado para ejecutarse.

Ejemplo:

```
IF a > b THEN
   PRINT a ; " es mayor que " ; b
ELSE
   PRINT a ; " no es mayor que " ; b
END IF
```

La instrucción selectiva anterior puede presentar uno de dos mensajes: a es mayor que b o a no es mayor que b, según el resultado de la comparación entre a y b; si el resultado de $a > b$ es verdadero, se presenta el primer mensaje, si es falso se exterioriza el segundo. Las palabras clave IF, THEN, ELSE, y END IF; constituyen la propia estructura de la instrucción condicional (palabra reservadas), proporcionada por el lenguaje, el usuario no debe utilizar sus nombres salvo para este fin. El caso ejemplo se ha codificado en BASIC.

- IF señala el comienzo de la instrucción condicional, y se espera que después siga la condición lógica de control de la instrucción.
- THEN señala el fin de la condición, y después estará la instrucción a ejecutar si la condición es verdadera.
- ELSE es opcional, le sigue la instrucción que se ejecutará si la condición es falsa.
- END IF indica el final de la estructura, luego de ésta el programa seguirá su curso.

Ampliando un poco el ejemplo anterior, con estructuras anidadas:

```
IF a > b THEN
   PRINT a ; " es mayor que " ; b
ELSEIF a < b THEN
   PRINT a ; " es menor que " ; b
ELSE
   PRINT a ; " es igual que " ; b
END IF
```

Este ejemplo permite considerar situaciones en las que se tiene más de dos alternativas. En este caso se ha considerado tres, pero hay situaciones en las que deben considerarse más casos y para ellos se puede repetir las veces que sea necesario la opcional ELSEIF.

Estructura iterativa

Un bucle iterativo o iteración de una secuencia de instrucciones, hace que se repita su ejecución mientras se cumpla una condición, el número de iteraciones normalmente está determinado por el cambio en la condición dentro del mismo bucle, aunque puede ser forzado o explícito por otra condición.

Ejemplo:

```
a= 0
b= 7
DO WHILE b > a
   PRINT a
   a= a + 1
LOOP
```

Esta instrucción tiene tres palabras reservadas WHILE, DO y LOOP.

- DO WHILE: señala el comienzo del bucle ("haga mientras") y después de estas palabras se espera la condición lógica de repetición, si la condición es verdadera pasa el control al cuerpo del bucle, en caso contrario el flujo salta directamente al final de la estructura, saliendo de la misma.
- LOOP: señala el final del cuerpo de la estructura de bucle.

El bucle mientras, se repite mientras la condición sea verdadera, esta condición se comprueba o chequea antes de ingresar al cuerpo del bucle, por lo que el mismo puede que no se ejecute nunca (cuando la condición es falsa desde un principio) o bien que se repita tantas veces como resulte y mientras la condición sea cierta.

3.1.1 Caracteristicas de la programación estructurada

1. Los programas son más fáciles de entender, pueden ser leídos de forma secuencial, no hay necesidad de hacer engorrosos seguimientos en saltos de línea (GOTO) dentro de los bloques de código para intentar entender la lógica.
2. La estructura de los programas es clara, puesto que las instrucciones están más ligadas o relacionadas entre sí.
3. Reducción del esfuerzo en las pruebas y depuración. El seguimiento de los fallos o errores del programa ("debugging") se facilita debido a su estructura más sencilla y comprensible, por lo que los errores se pueden detectar y corregir más fácilmente.
4. Reducción de los costos de mantenimiento. Análogamente a la depuración, durante la fase de mantenimiento, modificar o extender los programas resulta más fácil.
5. Programas son más sencillos y más rápidos de confeccionar (y se facilita su optimización).
6. Los bloques de código son casi auto-explicativos, lo que reduce y facilita la documentación.
7. Las instrucciones de salto, GOTO, quedan reservadas para construir las instrucciones básicas, si fuera realmente imprescindible. Aunque no se usan de forma directa, por estar prohibida su utilización, están incluidas implícitamente en las instrucciones de selección e iteración.
8. Un programa escrito de acuerdo a los principios de programación estructurada no solamente tendrá una mejor estructura sino también una excelente presentación.
9. Se incrementa el rendimiento de los programadores, comparada con la forma tradicional que utiliza GOTO.

La programación estructurada ofrece estos beneficios, pero no se la debe considerar como una panacea ya que el desarrollo de programas es, esencialmente, una tarea de dedicación, esfuerzo y creatividad; programar es casi un arte.

Inconvenientes de la programación estructurada

El principal inconveniente de este método de programación es que se obtiene un único bloque de programa, que cuando se hace demasiado grande puede resultar problemático el manejo de su código fuente; esto se resuelve empleando conjuntamente la programación modular, es decir, si es necesario, se definen módulos independientes, programados y compilados por separado (en realidad esto no es necesario, pero sí es recomendable para su mejor mantenimiento y depuración).

En realidad, cuando se programa hoy en día (inicios del siglo XXI) se utilizan normalmente, tanto las técnicas de programación estructurada como las de programación modular, de forma conjunta y por lo tanto es muy

común que cuando se hace referencia a la programación estructurada muchos entiendan que ella incluye también las técnicas modulares, estrictamente no es así.

Un método un poco más sofisticado es la programación por capas, en la que los módulos tienen una estructura jerárquica en la que se pueden definir funciones dentro de funciones o de procedimientos.

Si bien las metodologías en cuestión ya son de antigua data ("en plazos informáticos"), aun en la actualidad la conjunción "Programación estructurada" y "programación modular" es una de la más utilizadas, juntamente con un más moderno paradigma, en pleno auge, completamente distinto, llamado programación orientada a objetos.

4 Composición secuencial, condicionales y ciclos

4.1 Ejercicios

1. Convertir un ciclo for a un ciclo while y viceversa.

5 Variables globales y efectos colaterales

Una variable global es aquella que se define fuera del cuerpo de cualquier función, normalmente al principio del programa, después de la definición de los archivos de biblioteca (#include), de la definición de constantes simbólicas y antes de cualquier función. El ámbito de una variable global son todas las funciones que componen el programa, cualquier función puede acceder a dichas variables para leer y escribir en ellas. Es decir, se puede hacer referencia a su dirección de memoria en cualquier parde del programa.

El uso de variables globales no es aconsejable a pesar de que aparentemente nos parezca muy útil, esto se debe a varias razones fundamentales:

- Legibilidad menor.
- Nos condiciona en muchos casos que el programa sólo sirva para un conjunto de casos determinados.
- El uso indiscriminado de variables globales produce efectos colaterales. Esto sucede cuando existe una alteración no deseada del contenido de una variable global dentro de una función, bien por invocación, bien por olvidar definir en la función una variable local o un parámetro formal con ese nombre. La corrección de dichos errores puede ser muy ardua.
- Atenta contra uno de los principios de la programación, la modularidad. El bajo acoplamiento supone no compartir espacios de memoria con otras funciones, y potenciar el paso de información (llamadas) para que la función trate la información localmente.

5.1 Ejercicios

1. Escriba un programa que contenga solamente variables globales y llamadas a funciones al implementar el ordenamiento de números en un arreglo.
2. Escribir un programa que contenga solamente variables locales y llamadas a funciones al implementar el ordenamiento de números en un arreglo.

6 Aliases y apuntadores perdidos (dangling pointers)

```
Tipo variable, valor;
Tipo * puntero;
puntero = &variable;
*puntero = valor; /* Equivale a variable = valor; */
```

Las expresiones `variable` y `*puntero` de la sección anterior son equivalentes. Se dice entonces que `p` es un **alias** de `variable`. Nada impide crear múltiples alias de una variable, que podrían utilizarse desde distintos lugares de un programa:

```
float valor;
float *alias_1, *alias_2, *alias_3;
alias_1 = alias_2 = alias_3 = &valor;
```

El uso de alias es potencialmente peligroso, pues permite modificar el valor de una variable empleando cualquiera de esos alias.

7 Subrutinas, subprogramas, procedimientos y funciones

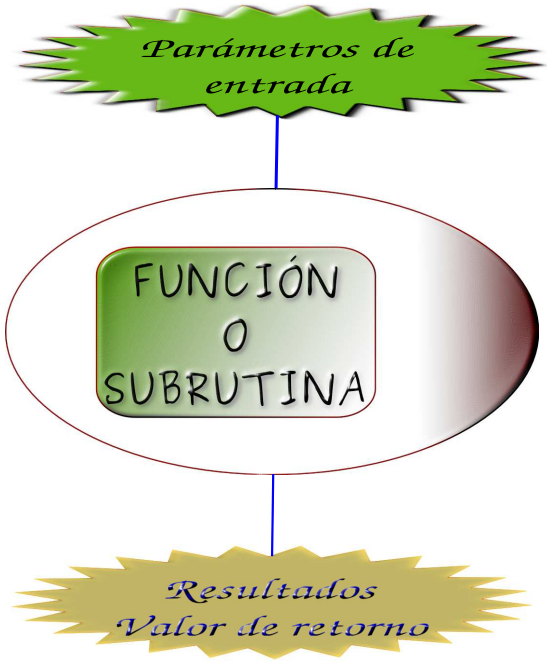

Figure 1: Diagrama que representa en forma general una subrutina, función, procedimiento o rutina.

En computación, una subrutina o subprograma (también llamada procedimiento, función o rutina), ver figura 1, como idea general, se presenta como un subalgoritmo que forma parte del algoritmo principal, el cual permite resolver una tarea específica. Algunos lenguajes de programación, como Visual Basic .NET o Fortran, utilizan el nombre función para referirse a subrutinas que devuelven un valor.

Una subrutina al ser llamada dentro de un programa hace que el código principal se detenga y se dirija a ejecutar el código de la subrutina, en cambio cuando se llama a una macro, el compilador toma el código de la macro y lo implanta donde fue llamado, aumentando así el código fuente y por consiguiente el objeto.

- Los subprogramas o rutinas son unidades más pequeñas en las que se puede dividir un programa.
- Procedimientos: Son rutinas que no regresan valor o regresan un valor vacío.
- Funciones: Son rutinas que regresan un valor.

Ejemplos

```
PROGRAMA principal
    instrucción 1
                  instrucción 2
    ...
    instrucción N
    ...
    SUBRUTINA NombreX
       .......
    FIN SUBRUTINA
    ...
FIN PROGRAMA principal.
```

La siguiente función en C es la analogía al cálculo del promedio matemático. La función "Promedio" devuelve un valor decimal correspondiente a la suma de 2 valores enteros de entrada (A y B):

```
float Promedio(int A, int B){
   float r;
   r=(A+B)/2.0;
   return r;
}
```

Así una llamada "Promedio(3, 5)" devolverá el valor de tipo real (float) 4,0.

8 Paso de parámetros (valor, referencia, valor-resultado y nombre)

8.1 Parámetros por valor y por referencia

El paso por valor significa que al compilar la función y el código que llama a la función, ésta recibe una copia de los valores de los parámetros que se le pasan como argumentos. Las variables reales no se pasan a la función, sólo copias de su valor.

```
#include<stdio.h>
int operaciones(int x, int y);
main(){
   int a=10,b=20;
   int resultado = operaciones(a,b);
}
int operaciones(int A, int B){
   return A + B;
}
```

Cuando una función debe modificar el valor de la variable pasada como parámetro y que esta modificación retorne a la función de donde se realizó la invocación, se debe pasar el parámetro por referencia. En este método, el compilador no pasa una copia del valor del argumento; en su lugar, pasa una referencia, que indica a la función dónde existe la variable en memoria.

La referencia que una función recibe es la dirección de la variable. Es decir, pasar un argumento por referencia es, simplemente, indicarle al compilador que pase la dirección de memoria del argumento.

```
#include<stdio.h>
int operaciones(int x, int y);
```

```
main(){
   int a=10,b=20;
   int resultado;
   operaciones(a,b,&resultado);
}
void operaciones(int A, int B, int *RESULTADO){
   *RESULTADO = A + B;
}
```

8.2 Parámetros por valor-resultado

Es una generalización de la llamada por valor, de forma que su efecto es similar a la llamada por referencia. Antes de la llamada, se evalúan los valores de los parámetros actuales, pasándolos a la rutina llamada de la misma forma que en la llamada por valor, pero con la diferencia que además se han guardado las direcciones de los parámetros actuales de forma que a la vuelta de la rutina llamada, los valores que en ese momento tengan los parámetros formales se copiarán en las direcciones de los parámetros actuales.

Es un tipo poco usado en los lenguajes de programación actuales. Se basa en que dentro de la función se trabaja como si los argumentos hubieran sido pasados por valor pero al acabar la función los valores que tengan los argumentos serán copiados a las variables que pertenecían.

Este tipo puede ser simulado en cualquier lenguaje que permita el paso de valores por referencia de la siguiente forma:

```
void EjemploValorRes(int& a1, int& a2, int& a3) {
    int aux1 = a1, aux2 = a2, aux3 = a3;
    // código trabajando con aux1, aux2 y aux3
    a1 = aux1; a2 = aux2; a3 = aux3; // Dependiendo del compilador la copia se realiza en un sentid
}
```

Tal y como indica el ejemplo de simulación de valor-resultado, el orden de copia depende del compilador, lo que implica que la misma función pueda dar resultados diferentes según el compilador usado.

8.3 Parámetros por nombre

Este es el método principal especificado por ALGOL. Es un mecanismo de gran potencia e interés teórico, pero desafortunadamente es difícil de implementar e incluso de usar. La idea básica es dejar los parámetros actuales sin evaluar hasta que se necesiten. En ese preciso momento se evaluán, es decir, cada vez que se necesitan los parámetros se procede a evaluarlos. La implementación más usual de este método es pasar al procedimiento llamado unas rutinas sin parámetros (denominadas 'thunks'), que serán las encargadas de evaluar los parámetros en los momentos que se precisen.

8.4 Ejercicios

1. Hacer un programa que contenga una función que regrese por referencia el resultado de las 5 operaciones aritméticas básicas del lenguaje C de dos números.

2. Hacer un programa que implemente una estructura de datos con la información poersonal de alumnos y por medio de la invocación a una función se lean los datos al pasarlos por referencia.

3. Hacer un programa que implemente una estructura de datos para almacenar una imagen formato `ppm`. EL programa debe tener las funciones de leer y escribir la imagen en otro archivo.

En algunos lenguajes de programación el paso de parámetros que tenemos es:

- Paso por referencia es el estándar en FORTRAN.
- Paso por nombre es el estándar en ALGOL 60, pero es opcional el paso por valor.
- SIMULA 67 permite paso por valor, referencia y por nombre.
- C++, PASCAL y MODULA-2 permiten pasar por valor o por referencia.

9 Ligas estáticas y dinámicas a registros de activación

Los lenguajes de programación que permiten subprogramas anidados como: Pascal, Modula, Ada y aún aquellos que no como C, C++ y Java, tienen la activación en tiempo de ejecución de subprogramas que son administrados con una pila de ARIs, por sus siglas en inglés de Activation Record Instances. Dos apuntadores son utilizados en los registros de activación para habilidar la administración de la pila y acceso eficiente a variables no locales.

1. La liga dinámica apunta a la cima del ARI de las llamadas.
2. La liga estática apunta a la parte más baja del ARI de las llamadas del padre estático.