Programación

Unidad 2: Variables y Operadores

“Un nombre largo y descriptivo es mejor que un nombre corto y enigmático. Un nombre largo y descriptivo es mejor que un comentario largo y descriptivo ".

Robert C. Martin

Variables y Constantes

Variables

Previamente, cuando hablamos sobre cómo analizar un problema y definir sus datos de entrada y salida, mencionamos que algunos datos de entrada se clasifican como *Variables* y otros como *Constantes*.

Ahora bien ¿Qué es una Variable?

Una variable es una posición, un espacio, de memoria (memoria RAM) a la cual desde nuestro programa le asociamos un nombre para poder accederla, leer y/o modificar su valor.

¿Y para qué me sirve una variable? Bien, una variable la podemos usar para guardar valores que se ingresan al programa como puede ser el nombre de una persona, la edad; para guardar resultados de cálculos que deba realizar el programa como puede ser un promedio, el resultado de una operación aritmética como una suma o una resta, etc.

Toda variable tiene un *tipo de dato* y un nombre o identificador. La persona que desarrolla un programa debe procurar que este nombre sea lo más representativo posible para facilitar el uso de la variable y la lectura del código.

¿Cómo definimos el nombre de una variable?

Cada vez que necesitemos utilizar el valor que guarda una variable, o bien asignarle un valor, lo haremos a través de su nombre. Ahora bien, para definir el nombre de una variable debemos seguir ciertas reglas que a continuación mencionamos:

- ✓ Debe comenzar con una letra (cualquiera entre la 'a' y la 'z') o el caracter _ (guión bajo).
- ✓ Luego del primer caraceter **sólo** puede contener letras (mayúsculas y/o minúsculas), números o _ (guión bajo).
- ✗ No puede contener espacios en blanco, o cualquier otro caracter que no sean letras (mayúsculas y/o minúsculas), números o _ (guión bajo).

Basándonos en las 3 reglas mencionadas, se suele utilizar una convención llamada *"camelCase"* que es un estilo de escritura que utilizamos cuando el nombre de nuestra variable está formado por varias palabras. En este estilo de escritura la primer letra del nombre de la variable será siempre una letra minúscula y las siguientes primeras letras de las otras palabras serán una letra mayúscula.

Veamos algunos ejemplos

Formas correctas de definir el nombre de una variable	Formas incorrectas de definir el nombre de una variable
✓ `Definir _salario como Real;` ✓ `Definir nombre_alumno como Cadena;` ✓ `Definir letra_4 como Caracter;` Ahora algunos ejemplos utilizando camelCase: ✓ `Definir cuentaBancaria como Entero;` ✓ `Definir nombreAlumno como Cadena;` ✓ `Definir esVocal como Booleano;`	✗ `Definir el salario como Real;` ✗ `Definir nombre alumno como Cadena;` ✗ `Definir 4_letra como Caracter;` ✗ `Definir esVocal? como Booleano:` ✗ `Definir legajo-empleado como Entero;`

¿Por qué una variable tiene que tener un tipo de dato?

El tipo de dato de una variable define 2 cosas, en primer lugar define el rango de valores posibles que puede guardar una variable, así si una variable la defino como de *tipo Entero* sólo podrá almacenar números enteros, por ejemplo: -1040, -2, 0, 10, 20, 130...
En segundo lugar, el tipo de dato de una variable, define qué cosas puedo hacer con esa variable, es decir, qué operaciones puedo ejecutar sobre esa variable. De nuevo si una variable la defino como de *tipo Entero* sólo voy a poder realizar operaciones que estén definidas para los números enteros como ser suma, resta, multiplicación, división, etc. Pero si tuviese una variable de *tipo Cadena* claramente no voy a poder ejecutar una división de 2 cadenas porque la operación no está definida para el tipo de dato Cadena.

Toda variable debe ser definida antes de poder ser utilizada, es decir antes de poder asignarle un valor o de intentar leer el valor que tiene, la variable debe ser definida.

Veamos algunos ejemplos

Ejemplo correcto	**Ejemplo incorrecto**

`// Define la variable edad de tipo Entero`
`Definir edad como Entero;`
 	
`// Asigna un valor a la variable edad`
`edad = 18;`
	`// Trata de asignar un valor a la variable edad, pero la variable edad no fue definida.`
	`edad = 18;`
	`----------`

```
// Define la variable nombre de tipo Cadena
Definir nombre como Cadena;

// Asigna un valor a la variable nombre
nombre = “Pepe”;
```

```
Definir resultado como Entero;
Definir nombre como Cadena;

nombre = “Pepe”;

// Trata de realizar una multiplicación con
una variable de tipo cadena
resultado = nombre * 2;
```

Constantes

Al igual que las variables, una constante es una posición, un espacio, de memoria (memoria RAM) a la cual desde nuestro programa le asociamos un nombre para poder accederla y leer su valor pero lo que nunca vamos a poder hacer es cambiarle el valor que le asignemos inicialmente.

Las constantes también poseen un tipo de dato y debemos definirle un nombre siguiendo las mismas reglas que utilizamos para definir el nombre de una variable.

Tipos de datos

El concepto de tipo de dato no es propio de la programación estructurada, sino que aplica a cualquier tipo de programación que utilicemos.

Los lenguajes de programación permiten trabajar con tipos de datos básicos (primitivos) como los que mencionamos al describir cómo analizar un problema.

Los tipos de datos los utilizamos para definir datos de entrada (variables, constantes) y datos de salida.

Tipo de Dato	Valor posible	Ejemplo
Entero (Integer)	Número Entero	Definir **num** como Entero; La variable **num** puede guardar valores como -10, 3, 9, 100. La variable **num** no puede guardar valores como 3,14
Real (Float)	Número Real	Definir **precio** como Real; La variable **precio** puede guardar valores como -10, 3, 3.5, 9, 9.9, 100, 100.2.
Caracter (Character)	Un solo símbolo (letra, dígito, signo)	Definir **letra** como Caracter; La variable **letra** puede guardar valores como “a” o signo “$”.

		La variable **letra** no puede guardar valores como “Pepe”
Cadena (String)	Conjunto de caracteres (palabra, frases)	Definir **nombre** como Cadena; La variable **nombre** puede guardar valores como “Pepe” y también como “a”, “$”
Lógico (Boolean)	Verdadero o Falso	Definir **esNumero** como Lógico; La variable **esNumero** puede guardar valores como Verdadero o Falso únicamente.

Más adelante veremos que tomando como base los tipos de datos primitivos podemos crear tipos de datos que llamaremos *tipo de dato definido por el usuario*.

Operadores y Expresiones

Los operadores nos permiten realizar cálculos aritméticos, asignaciones, evaluar relaciones y expresiones lógicas

Asignación

El operador de asignación es el caracter “=” y lo utilizamos para asignar un valor a una variable.

Ejemplo:

```
Definir numero Como Entero;  // Declara la variable numero

numero = 4; // y asigna el valor 4.
```

¿Qué es una asignación? En programación, usamos el término asignación para indicar que estamos dando valor a una variable.

Para realizar una asignación utilizamos el signo “=” y una sintaxis (forma de escribir) similar a la matemática donde ponemos a izquierda del signo igual la variable y a derecha del signo igual el valor o expresión. De la siguiente manera:

“variable” = “valor a asignar”;

Se debe tener en cuenta que la variable y el valor a asignar deben ser de tipos “compatibles” o asignables

Ejemplo:

```
Definir numero como Entero; // Define la variable numero de tipo Entero
numero = 3; // Le asigna el número 3 a la variable numero
```

```
Definir promedio como Real; // Define la variable promedio de tipo Real
promedio = (7 + 8) / 2
```

Aritméticos

Las operaciones aritméticas se llevan a cabo utilizando los operadores **+, -, *, /** que se corresponden con las operaciones **suma, resta, multiplicación y división**.

También existe el operador **%** que nos permite obtener el resto de una división.

Estos operadores sólo se pueden usar entre variables, constantes o expresiones que sean de tipo numérico: Entero o Real.

Ejemplos:

```
Definir suma Como Entero;  // Declara la variable suma
suma = 4 + 5;              // A la variable suma le asigna el resultado de sumar 4 + 5.

Definir resta Como Entero; // Declara la variable resta
resta = 3 - 2;           // A la variable resta le asigna el resultado de restar 3 - 2.

Definir producto Como Entero; // Declara la variable producto
producto = 4 * 5; // A la variable producto le asigna el resultado de multiplicar 4 * 5

Definir cociente Como Entero; // Declara la variable cociente

// A la variable cociente le asigna el resultado de dividir 19 / 5. Esto es 3
cociente = 19 / 5;

Definir resto Como Entero; // Declara la variable resto

// A la variable resto le asigna el resto de dividir 19 / 5. Esto es 4
cociente = 19 % 5;
```

Lógicos y Relacionales

Tanto los operadores relacionales como los lógicos se presentan, generalmente, en expresiones que devuelven un valor Booleano, es decir un valor que puede ser Verdadero (true) o Falso (false).

Los operadores relacionales son utilizados para realizar comparaciones entre valores de 2 o más variables mientras que los operadores lógicos nos sirven de soporte para armar expresiones que vamos a relacionar.

Estos operadores sólo se pueden usar entre variables, constantes o expresiones que sean del tipo Lógico

Operadores Relacionales

Operador	Acción
>	mayor que
<	menor que
>=	mayor o igual que
<=	menor o igual que
==	igual a
!=	distinto de (no igual)

Operadores Lógicos

Operador	Acción
Y	conjunción
O	disyunción
NO	negación

Ejemplos:

```
Definir resultado Como Logico; // Declara la variable resultado;
// La variable resultado guardará el valor Falso (false) ya que 4 no es mayor que 5
resultado = 4 > 5;
// La variable resultado guardará el valor Verdadero (true) ya que 4 es menor (o
// igual) que 5
resultado = 4 <= 5;
// La variable resultado guardará el valor Verdadero (true) ya que 4 es mayor que 3
// Y 2 es mayor (o igual) a 2
resultado = 4 > 3 Y 2 >= 2;
```

A continuación mostramos las operaciones lógicas y cuál es su tabla de verdad

p	q	p Y q	p O q	NO q

F	F	F	F	V
F	V	F	V	F
V	F	F	V	V
V	V	V	V	F

Expresiones Combinadas

Las operaciones y expresiones mencionadas en los puntos anteriores pueden combinarse de forma que se realicen varias operaciones en una misma sentencia. Al escribir una expresión que combine varios operadores se debe tener en cuenta las reglas de precedencia de los mismos, la siguiente tabla muestra el orden de precedencia siendo los de mayor nivel los de más arriba:

- multiplicación y división: * / %
- suma y resta: + -
- operadores relacionales: < > <= =>
- equivalencia: == !=
- conjunción: Y
- disyunción: O
- operadores de asignación: =

Cabe destacar que **lo último que se hará en una sentencia es realizar la asignación**, por eso está al final de la tabla. Esto tiene sentido dado que antes de poder asignar un valor se debe resolver toda la expresión a calcular.

Veamos unos ejemplos de cómo actúa esta regla.

```
j = 1 + 3 * 4; // resultado j = 13
```

Desde que aprendimos aritmética básica, conocemos la regla que nos obliga a calcular la multiplicación antes de una suma.

```
j = 1 + 3 - 4;  resultado j= 0;
```

Si todos los operadores tienen un nivel idéntico de precedencia **se evalúa la expresión de izquierda a derecha**.

En caso que se desee forzar una precedencia se deben utilizar paréntesis

```
bol = (8 + 10) / 2 > 6;  resultado bol= Verdadero;
```

Primero resuelve la suma por el uso de los paréntesis, luego la división y luego resuelve la comparación con 6, resultando en verdadero

Como en matemáticas, podemos anidar los paréntesis. Se comenzará a evaluar los internos hasta llegar a los externos.

Cabe agregar que los paréntesis no disminuyen el rendimiento de los programas. Por lo tanto, agregar paréntesis no afecta negativamente al programa.

```
k = ((12 - 2) * ( 21 - 11)) / ((1+1)*(15-10)) + 1 ;
```

Bibliografía utilizada

Luis Joyanes Aguilar, Ignacio Zahonero Martínez. Programación en C. Segunda Edición. España: McGRAW-HILL/INTERAMERICANA DE ESPAÑA. S.A.U., 2005. ISBN-84: 481-9844-1.

● Brian W. Kernighan, Rob Pike. La práctica de la programación. Pearson Educación. Méjico (2000)

● Debugging with CodeBlocks: http://wiki.codeblocks.org/index.php?title=Debugging_with_Code::Blocks