Departamento de Lenguajes y Ciencias de la Computación
UNIVERSIDAD DE MÁLAGA

Apuntes para la asignatura

Informática

Facultad de Ciencias (Matemáticas)

http://www.lcc.uma.es/personal/pepeg/mates

Tema 3. Introducción al lenguaje de programación Modula-2

3.1 Valores y tipos 3
3.2 Representación de constantes 3
3.2.1 Valores numéricos enteros 4
3.2.2 Valores numéricos reales 4
3.2.3 Caracteres 5
3.2.4 Cadenas de caracteres (Strings) 5
3.3 Tipos predefinidos 6
3.3.1 El tipo INTEGER 6
3.3.2 El tipo CARDINAL 7
3.3.3 El tipo REAL 7
3.3.4 El tipo LONGREAL 8
3.3.5 El tipo CHAR 9
3.4 Expresiones aritméticas 9
3.5 Identificadores 10
3.6 Constantes 12
3.7 Variables 13
3.7.1 Declaración de variables 143
3.7.2 Sentencia de asignación 14
3.8 Operaciones de Entrada y Salida. 16
3.8.1 Operaciones de Salida 16
3.8.2 Operaciones de Entrada 18
3.9 Estructura general de un programa 20

Bibliografía

- *Programación 1*. José A. Cerrada y Manuel Collado. Universidad Nacional de Educación a Distancia.

- *Programming with TopSpeed Modula-2.* Barry Cornelius. Addison-Wesley.

Introducción

Modula-2 fue diseñado por Niklaus Wirth y es uno de los lenguajes de programación de alto nivel más utilizado actualmente para la enseñanza de la programación de ordenadores. Existen varias versiones de este lenguaje. Nos centraremos en el resto del curso en el estudio de una de las versiones más populares: *TopSpeed Modula-2*.

Una de las características que diferencia a un lenguaje de programación de otras notaciones para expresar algoritmos, como el pseudocódigo, es la menor flexibilidad del primero. En efecto, un programa escrito en un lenguaje de programación de alto nivel consta de una serie de instrucciones (o *sentencias*) que indican al ordenador los pasos a seguir para resolver cierto problema. Al conjunto de reglas que dictan cuáles son las instrucciones posibles y el modo de combinarlas se denomina *sintaxis del lenguaje*. En este tema comenzaremos el estudio de la sintaxis del lenguaje *Modula-2*. Para definir cada una de las reglas utilizaremos la notación BNF. Finalizaremos el tema con un ejemplo completo de un programa escrito en *Modula-2* que el alumno podrá probar en la primera práctica de la asignatura.

3.1 Valores y tipos

Todo programa de ordenador manipula una serie de datos para obtener unos resultados. Definimos *dato* como un elemento de información que puede tomar un valor entre varios posibles.

En programación, un dato puede tomar valores de una sola clase. No tiene sentido decir que el nombre de una persona es “24”. Tampoco lo tiene decir que el número de personas en una clase es “Pedro”. A las distintas clases de valores se denominan *tipos*. Así, un dato tiene asociado un tipo que representa cual es la clase de valores que puede tomar. Algunos ejemplos de tipos son:

- Valores numéricos enteros positivos y negativos: 100, -24, 36, ...
- Valores numéricos enteros y positivos: 0, 1, 2, 3, ...
- Valores numéricos reales: 3.1415, -16.18,...
- Valores carácter: ‘a’, ‘b’, ‘c’,...

Para cada tipo habrá una serie de operaciones que se le pueda aplicar. Así, tiene sentido sumar dos valores numéricos, pero no dos letras. Se llama *tipo abstracto de datos* a un conjunto de valores junto con las operaciones que permiten manipular éstos.

Dentro de un programa existirán dos tipos de valores:

- Valores *constantes*: se trata de valores cuya magnitud no cambia durante cualquier ejecución del programa.
- Valores *variables*: la magnitud puede tener distintos valores en distintos puntos del programa.

3.2 Representación de constantes

Uno de los objetivos de los lenguajes de programación es evitar las ambigüedades o imprecisiones que existen en los lenguajes naturales. Por ejemplo, los anglosajones utilizan la coma para separar millares, mientras que nosotros la usamos para separar la parte entera de la decimal:

348,536	anglosajones	trescientos cuarenta y ocho mil quinientos treinta y seis
	España	trescientos cuarenta y ocho con quinientas treinta y seis

Estudiaremos a continuación las distintas reglas que sigue el lenguaje Modula-2 para representar valores de tipo numérico y de tipo carácter.

3.2.1 Valores numéricos enteros

Los valores numéricos enteros representan un número exacto de unidades y no pueden tener parte fraccionaria. Un valor entero en Modula-2 se escribe con una secuencia de 1 o más dígitos del 0 al 9 no separados entre ellos y precedido opcionalmente de los signos mas (+) o menos (-). Ejemplos de valores válidos son:

2
+56
0
-2345
1000

Ejemplos de valores no válidos son:

123,32	NO se pueden insertar comas
22.56	NO se pueden insertar puntos
13B7	NO se pueden insertar letras

Usando la notación BNF:

Valor_Entero	::=	[+ | -] Secuencia_Digitos
Secuencia_Digitos	::=	Digito { Digito }
Digito	::=	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

3.2.2 Valores numéricos reales

Los valores numéricos reales permiten expresar cualquier cantidad, incluyendo fracciones de unidad. En *Modula-2*, se pueden representar de dos modos: notación *decimal habitual* o notación *científica*.

En la notación decimal habitual, un valor real se escribe con una parte entera terminada siempre en un punto (.) seguida, opcionalmente, por una secuencia de dígitos que constituyen la parte decimal. Ejemplos de valores válidos son:

-0.78
5.
+234.53

En la notación científica, un valor real se escribe como una mantisa, que es un número real en notación decimal habitual, seguido de la letra E y un exponente entero que indica la potencia de 10 por la que se multiplica la mantisa. Ejemplos de valores válidos son:

-0.78E+2	equivale a -0.78×10^{2}
5.E-3	equivale a $5.0 \times 10^{-3} = 0.005$
+234.53E4	equivale a 234.53×10^{4}

Ejemplos de valores no válidos son:

.234	Es necesario al menos un dígito en la parte entera

2,56	NO se pueden insertar comas
13.4B7	NO se puede insertar la letra B

Obsérvese que podemos escribir un mismo valor de distintos modos:

45.6 456.E-14.56E+1 45.60E+0 456000.00E-4

La definición usando notación BNF es:

Valor_Real	**::=**	Valor_Real_Decimal **|** Valor_Real_Científica
Valor_Real_Decimal	**::=**	Valor_Entero . **[** Secuencia_Digitos **]**
Valor_Real_Científica	**::=**	Valor_Real_Decimal Factor_Escala
Factor_Escala	**::=**	E Valor_Entero

3.2.3 Caracteres

Además de tipos numéricos, el lenguaje de programación *Modula-2* permite operar valores de tipo carácter. Se trata de las letras y signos que constituyen un texto. El valor correspondiente a un carácter concreto se expresa en Modula-2 escribiendo dicho carácter entre comillas (" situado en la tecla correspondiente al número 2 del teclado de un PC) o apóstrofos (' situado en la tecla correspondiente al cierre de interrogación). Ejemplos de valores válidos son:

'a'	Representa a la letra a minúscula
"A"	Representa también a la letra a minúscula
"?"	Representa el carácter correspondiente al cierre de interrogación

Ejemplos de valores no válidos son:

a	Deben ir entre comillas o apóstrofes.
'ab'	Solo puede ir una letra
'a"	Debe empezar y acabar con el mismo delimitador

Algunas observaciones son:

- El espacio en blanco (' ') es un carácter válido. A veces lo representaremos con '~~b~~'.
- Hay que distinguir entre el número entero 7 y la letra '7'.
- El valor correspondiente al carácter apóstrofo solo se puede representar entre comillas (" ' ") y viceversa (' " ')

Usando notación BNF:

Valor_Caracter	**::=**	' Caracter_menos_apostrofo ' **|** " Caracter_menos_comillas "
Caracter_menos_comillas	**::=**	a **|** b **|** c **|** ... **|** 1 **|** 2 **|** 3 **|** ... símbolos
Caracter_menos_comillas	**::=**	a **|** b **|** c **|** ... **|** 1 **|** 2 **|** 3 **|** ... símbolos

3.2.4 Cadenas de caracteres (Strings)

Normalmente no es suficiente trabajar con caracteres sueltos. Suele interesar trabajar con palabras o frases. Estos valores se representan con cadenas de caracteres o *Strings*. Una cadena de caracteres se escribe como una secuencia de caracteres entre apóstrofos o comillas. Ejemplos válidos son:

"Coche"	Representa la palabra Coche
'Coche'	Idem al anterior

“La casa es verde” Representa una frase

Algunas observaciones son:

- Si una cadena incluye apóstrofos en su interior sólo podrá escribirse entre comillas y viceversa. Ejemplo “Incluir entre ‘apóstrofos’ texto”.
- Es posible definir una cadena que no contenga ningún carácter. A ésta se llama cadena vacía y se escribe como “” o ‘’.

Usando notación BNF:

Valor_Cadena	**::=**	‘ **{** Caracter_menos_apostrofo **}** ‘ **|** “ **{** Caracter_menos_comillas **}** “

3.3 Tipos predefinidos

En programación, un dato puede tomar valores de una sola clase y que a dicha clase se llama tipo del dato. En *Módula-2* existen tipos de datos predefinidos y sentencias para que el programador defina nuevos tipos. Algunos de los tipos de datos predefinidos se designan con los siguientes nombres: INTEGER, CARDINAL, REAL, LONGREAL, CHAR y BOOLEAN. Estudiaremos los cinco primeros a continuación. Estudiaremos el tipo BOOLEAN y los mecanismos para definir nuevos tipos en temas posteriores.

3.3.1 El tipo INTEGER

En *TopSpeed Modula-2* los valores de este tipo son los valores numéricos enteros positivos y negativos comprendidos en el rango [-32768 , 32767]. La representación de cualquiera de estos valores en la memoria del ordenador es exacta (no se producen errores de precisión al operarlos).

En caso de no recordar estos valores, podemos hacer referencia a ellos con las expresiones **MIN**(**INTEGER**) y **MAX**(**INTEGER**).

Las operaciones definidas para valores de este tipo son las siguientes:

- a + b. Suma de los enteros a y b
- a – b. Diferencia de los enteros a y b
- a * b. Producto de los enteros a y b
- a **DIV** b. Cociente resultante de dividir el entero a por el entero b.
- a **MOD** b. Resto resultante de dividir el entero a por el entero b.
- **ABS**(a). Valor absoluto del entero a.
- + a. Devuelve el valor del entero a inalterado.
- - a. Devuelve el valor del entero a negado.

Como puede observarse, los signos + y – tienen un doble significado según se usen como operadores *infijos* entre dos enteros o *prefijos* frente a un único entero.

El operador **DIV** devuelve la parte entera resultante de dividir un número entero por otro. **MOD** devuelve el resto correspondiente. Cuando el divisor es cero se produce un error. Se cumple siempre la siguiente regla:

a **MOD** b = a – b (a **DIV** b)

Algunos ejemplos de uso de estos operadores son los siguientes:

Operación	Resultado
10 **DIV** 3	3
10 **MOD** 3	1
(-20) **DIV** (-7)	2
(-20) **MOD** (-7)	-6
17 **DIV** (-3)	-5
17 **MOD** (-3)	2
10 **DIV** 0	Error
ABS (-2)	2

Cuando se están operando datos de tipo INTEGER hay que tener en cuenta el rango de valores válidos. Si durante algún momento se produce un resultado fuera de dicho rango, el programa acabará con un error (suponemos que la opción de chequeo de rangos del compilador está activada). Debido a esto, la propiedad asociativa no es válida para los datos del tipo INTEGER y ciertos operadores. Por ejemplo

Operación	Resultado
(32767 + 1) –10	Error
32767 + (1-10)	32758

ya que en la primera expresión la suma se sale del rango válido.

3.3.2 El tipo CARDINAL

En *TopSpeed Modula-2* los valores de este tipo son los valores numéricos enteros positivos comprendidos en el rango [0 , 65535]. La representación de cualquiera de estos valores en la memoria del ordenador es exacta (no se producen errores de precisión al operarlos).

En caso de no recordar el valor máximo, podemos nombrarlo con la expresión **MAX**(**CARDINAL**).

Con excepción de la operación de cambio de signo o negación las operaciones son las mismas que para el tipo INTEGER.

Es un error intentar operar un dato de tipo INTEGER con otro de tipo CARDINAL. Sólo se pueden operar datos del mismo tipo.

3.3.3 El tipo REAL

En *TopSpeed Modula-2* los valores de este tipo son los valores numéricos reales positivos y negativos comprendidos en el rango [-3.4×10^{38}, $+3.4 \times 10^{38}$]. La representación de estos valores en la memoria del ordenador no es exacta (se producen errores de precisión al operarlos). Se manejan valores aproximados.

Las operaciones definidas para valores de este tipo son las siguientes:

- a + b. Suma de los reales a y b
- a – b. Diferencia de los reales a y b
- a * b. Producto de los reales a y b
- a / b. Cociente de los reales a y b
- **ABS**(a). Valor absoluto del real a.

- + a. Devuelve el valor del real a inalterado
- - a. Devuelve el valor del real a negado.

En el caso de los reales la división no devuelve el cociente entero, sino el valor real resultante de dividir el cociente por el divisor. Para el tipo REAL NO existen las operaciones **`DIV`** y **`MOD`**.

Operación	Resultado
10.5 / 0.2	52.5
-15.4E2 + 450.0	-1090.0

3.3.4 El tipo LONGREAL

En *TopSpeed Modula-2* los valores de este tipo son los valores numéricos reales positivos y negativos comprendidos en el rango [$-1.7x10^{308}$, $+1.7x10^{308}$]. La representación de estos valores en la memoria del ordenador no es exacta (se producen errores de precisión al operarlos). Se manejan valores aproximados.

Las operaciones definidas para valores de este tipo son las mismas que para el tipo REAL. Para valores de este tipo están también definidas las siguientes operaciones:

- `Sin`(a): Devuelve el seno del valor LONGREAL a como un valor de tipo LONGREAL.
- `Cos`(a): Devuelve el coseno del valor LONGREAL a como un valor de tipo LONGREAL.
- `Tan`(a): Devuelve la tangente del valor LONGREAL a como un valor de tipo LONGREAL.
- `ASin`(a): Devuelve el arco-seno del valor LONGREAL a como un valor de tipo LONGREAL.
- `ACos`(a): Devuelve el arco-coseno del valor LONGREAL a como un valor de tipo LONGREAL.
- `ATan`(a): Devuelve la arco-tangente del valor LONGREAL a como un valor de tipo LONGREAL.
- `Sqrt`(a): Devuelve la raíz cuadrada del valor LONGREAL a como un valor de tipo LONGREAL.
- `Exp`(a): Devuelve la función exponencial del valor LONGREAL a como un valor de tipo LONGREAL.
- `Log`(a): Devuelve el logaritmo en base e del valor LONGREAL a como un valor de tipo LONGREAL.
- `Log10`(a): Devuelve el logaritmo en base 10 del valor LONGREAL a como un valor de tipo LONGREAL.

En el caso de las funciones trigonométricas, todos los ángulos que se manejan están expresados en radianes, NO en grados.

Estas operaciones, a diferencia de todas las vistas hasta el momento, no están disponibles de modo directo. Se encuentran en un *módulo de biblioteca* (MATHLIB) y es necesario que sean *importadas* por el programa antes de ser usadas. Veremos posteriormente como se importan operaciones de una biblioteca.

NOTA: Estas operaciones NO están definidas para valores del tipo REAL.

3.3.5 El tipo CHAR

Este tipo incluye como valores todos los caracteres disponibles en el ordenador. Además de los caracteres alfabéticos (letras), se incluyen caracteres numéricos y de puntuación.

Existen tres operaciones definidas para valores de este tipo:

- **ORD**(x): Toma como argumento un carácter y devuelve su posición dentro de la tabla ASCII.
- **CHR**(x): Toma como argumento un valor entre 0 y 255 devuelve el carácter que ocupa dicha posición dentro de la tabla ASCII.
- **CAP**(x): Toma como argumento un carácter y si éste es una letra minúscula devuelve la letra mayúscula correspondiente. En otro caso, devuelve el mismo carácter que toma como argumento.

Nótese que las funciones CHR y ORD son inversas la una de la otra:

CHR(**ORD** (x)) = x **ORD**(**CHR** (x)) = x.

Ejemplos de uso son:

Operación	Resultado
ORD('A')	65
CHR(65)	'A'
CHR(66)	'B'
CAP('q')	'Q'
CAP('Q')	'Q'
CAP('?')	'?'

3.4 Expresiones aritméticas

Una expresión aritmética representa un cálculo a realizar con valores. Para ello se combinan operandos y operadores.

Si no se utilizan paréntesis, el orden en que se realizan las operaciones es el siguiente:

1º Cambios de signos: + y – como prefijos

2º Operadores multiplicativos *, /, **DIV**, y **MOD**

3º Operadores aditivos: + y – como infijos

Si aparece más de un operador del mismo nivel, las operaciones se realizan de izquierda a derecha. Si una expresión va precedida de un cambio de signo se entiende que solamente afecta al próximo operando. Si se quiere que éste afecte a toda la expresión, ésta deberá incluirse entre paréntesis.

Algunos ejemplos son:

Expresión	Equivale a
5 * 30 + 5	(4*30)+5
334 DIV 6 MOD 4 * 5	((334 DIV 6) MOD 4) * 5
-5 * 10 MOD 3	((-5) * 10) MOD 3
(5 + 10) * 4 + 5	((5 + 10) * 4) + 5

En caso de duda es mejor utilizar los paréntesis.

En *Modula-2*, es un error operar datos de tipos distintos con los operadores aritméticos. En caso de que sea necesario operar valores de tipos distintos utilizaremos las funciones de conversión **REAL** y **TRUNC**:

- **REAL**(a): toma un valor de tipo INTEGER o CARDINAL y devuelve el REAL equivalente.
- **TRUNC**(a): toma un valor de tipo REAL y devuelve el INTEGER o CARDINAL equivalente. Para ello elimina la parte decimal del número original.

Expresión	Equivale a
5.5 + **REAL**(10)	15.5
5.0 + **REAL** (10)	15.0
5.0 + 10	ERROR
5 + **TRUNC**(10.5)	15
5 + **TRUNC**(10.0)	15

Obsérvese que el tipo del resultado siempre coincide con el tipo de los operandos (REAL en los dos primeros casos e INTEGER o CARDINAL en los dos últimos).

3.5 Identificadores

En cualquier programa aparece una serie de datos que son manipulados por éste. La manera de hacer referencia a los diferentes elementos que intervienen en un programa es darles un nombre particular a cada uno de ellos. En programación, se llaman *identificadores* a los nombres usados para identificar cada elemento del programa.

En *Modula-2*, un identificador es una palabra formada por caracteres alfabéticos o numéricos seguidos (sin espacios en blanco o signos de puntuación intercalados) y que debe comenzar por una letra. Pueden usarse las letras mayúsculas y minúsculas del alfabeto inglés y los dígitos del 0 al 9. No se puede usar ni la eñe ni vocales acentuadas en un identificador, aunque sí se pueden usar en valores de tipo carácter y cadenas de caracteres.

Ejemplos de identificadores válidos son:

```
Indice DiaDelMes Ejemplo1
IdentificadorMuyLargo
```

Ejemplos de identificadores no válidos:

`3Ejemplo`	No pueden comenzar por un dígito
`Ejemplo$`	No se pueden usar signos de puntuación
`Dia Del Mes`	No se pueden intercalar espacios
`Año`	No se puede usar la eñe

En Modula-2, se distinguen las letras mayúsculas y minúsculas, por lo que son identificadores diferentes `DiaSemana`, `Diasemana` y `DIASEMANA`.

La definición de un identificador usando la notación BNF es:

```
Identificador  ::=  Letra { Letra | Dígito }
Dígito         ::=  0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
Letra          ::=  A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
                    P | Q | R | S | T | U | V | W | X | Y | Z |
                    a | b | c | d | e | f | g | h | i | j | k | l | m | n | o |
                    p | q | r | s | t | u | v | w | x | y | z |
```

Sin embargo, no todos los identificadores construidos usando la definición anterior son válidos. Existen una serie de palabras que tienen un significado especial y que, por tanto, no pueden ser usadas como identificadores. Se trata de las *palabras reservadas* y los *identificadores predefinidos*.

La lista de palabras reservadas en *TopSpeed Modula-2* es la siguiente:

AND	**FORWARD**	**PROCEDURE**
ARRAY	**FROM**	**QUALIFIED**
BEGIN	**GOTO**	**RECORD**
BY	**IF**	**REPEAT**
CASE	**IMPLEMENTATION**	**RETURN**
CONST	**IMPORT**	**SET**
DEFINITION	**IN**	**THEN**
DIV	**LABEL**	**TO**
DO	**LOOP**	**TYPE**
ELSE	**MOD**	**UNTIL**
ELSIF	**MODULE**	**VAR**
END	**NOT**	**WHILE**
EXIT	**OF**	**WITH**
EXPORT	**OR**	
FOR	**POINTER**	

La lista de identificadores predefinidos en *TopSpeed Modula-2* es la siguiente:

ABS	**FLOAT**	**ODD**
ADDRESS	**HALT**	**ORD**
ADR	**HIGH**	**PROC**
BITSET	**INC**	**REAL**
BOOLEAN	**INCL**	**SHORTADDR**
BYTE	**INTEGER**	**SHORTCARD**
CAP	**LONGCARD**	**SHORTINT**
CARDINAL	**LONGINT**	**SIZE**
CHAR	**LONGREAL**	**TRUE**
CHR	**LONGWORD**	**TRUNC**
DEC	**MAX**	**VAL**
DISPOSE	**MIN**	**VSIZE**
EXCL	**NEW**	**WORD**
FALSE	**NULLPROC**	

El programador puede redefinir los identificadores predefinidos, aunque esto no es una buena práctica de programación, ya que confundiría a otros programadores que leyesen el programa y esperasen el significado habitual. Por otro lado, no es posible redefinir las palabras reservadas.

3.6 Constantes

Una *constante* es un valor fijo que se utiliza en un programa. El valor debe ser siempre el mismo en cualquier ejecución del programa.

Algunos ejemplos de constantes son el número de meses del año, el número de días de una semana y la constante matemática π.

Es posible asociar un nombre a una constante mediante una *declaración de constante*. Así podemos asignar al identificador `PI` el valor 3.141592 mediante la siguiente declaración:

```
CONST PI = 3.141592;
```

La declaración se inicia con la palabra reservada **`CONST`**, seguida del nombre de la constante que se está definiendo, el signo igual y por último, el valor asociado. La declaración termina con un punto y coma.

Una vez definida la constante puede utilizarse en expresiones. Las siguientes expresiones son equivalentes:

```
2. * 3.141592     2. * PI
```

También es posible declarar más de una constante a la vez en un programa. Para ello se comienza con la palabra reservada CONST y se prosigue con las distintas declaraciones separadas por punto y coma:

```
CONST
   DIASSEMANA  = 7;
   MESESANYO   = 12;
   NUMEROE     = 2.718281;
   LETRAPUNTO  = '.';
```

Como puede observarse, es habitual definir una constante en cada línea de texto y situar el comienzo de cada declaración de constante a la misma altura horizontal dentro de la línea. A esta práctica se llama *indentación*. No es obligatoria pero si bastante conveniente ya que mejora la legibilidad del programa.

Cada constante tiene asociado de modo automático un tipo:

`DIASSEMANA`	Tipo INTEGER o CARDINAL
`MESESANYO`	Tipo INTEGER o CARDINAL
`NUMEROE`	Tipo REAL o LONGREAL
`LETRAPUNTO`	Tipo CHAR

Es posible declarar el valor de una constante mediante una expresión siempre que todos los operandos que intervengan en ésta sean valores constantes y que las operaciones sean predefinidas del lenguaje:

```
CONST
   RADIO     = 5.3;
   DIAMETRO  = 2.0*RADIO;
```

La notación BNF para las declaraciones de constantes es la siguiente:

Declaración_de_Constantes	**::=**	CONST **{** Asociación_de_Constante ; **}**
Asociación_de_Constante	**::=**	Identificador = Expresión _Constante
Expresión_Constante	**::=**	**...**

Definiremos qué es una expresión constante en temas posteriores.

3.7 Variables

El concepto de variable en programación es distinto al concepto de variable algebraica. Las variables algebraicas representan valores de modo que una vez asociada la variable a un valor determinado dicho valor no cambia.

❏ Ejemplo: Sea la siguiente ecuación:

$$2x = 1$$

el valor asociado a la variable *x* en este caso es *x* = ½.

❏

En programación el concepto de variable es distinto y está asociado a la memoria del ordenador. La memoria del ordenador mantiene, además de las instrucciones del programa, una serie de datos. Una vez que el programa almacena un dato en una de las posiciones de la memoria su valor se mantiene. Sin embargo, es posible modificar el valor cuantas veces se desee.

Una *variable* representa un valor almacenado en la memoria del ordenador, el cual puede ser usado tantas veces como se desee. El valor de la variable puede ser modificado en cualquier momento, y será el nuevo valor el que estará almacenado en ella a partir de entonces.

Podemos ver una variable como una caja en la cual es posible almacenar un único valor en un momento dado. Es posible en todo momento consultar cuál es el valor almacenado en la caja. También es posible almacenar un nuevo valor en la caja de modo que el valor antiguo se pierde y el valor de la caja pasa a ser desde ese momento el nuevo valor almacenado.

Las variables de un programa se designan mediante nombres o identificadores. El identificador de una variable representa el valor almacenado en dicha variable. El programador puede elegir los nombres más apropiados para las variables que utilice en su programa.

3.7.1 Declaración de variables

En un programa en *Modula-2* es necesario declarar cada una de las variables que se utilice. Para cada variable usada es necesario especificar su nombre y el tipo de valores que puede tomar. Esto quiere decir que si una variable se declara con tipo INTEGER, por ejemplo, podrá almacenar en un momento un único valor entero (un 3, un –20 ó 15000) pero no podrá almacenar un valor de otro tipo (CHAR o REAL, por ejemplo).

❏ Ejemplo: Para usar una variable cuyo nombre sea `Edad` y que almacene un valor de tipo entero es necesario declararla del siguiente modo:

```
VAR Edad : INTEGER;
```

❏

Como puede observarse, una declaración de variable consta de la palabra reservada **VAR** seguida del nombre de la variable, un signo de dos puntos y el nombre del tipo de valor que puede tomar la variable. La declaración termina con un punto y coma.

Es posible declarar más de una variable usando una única palabra **VAR**:

```
VAR  Dia    : INTEGER;
     Mes    : INTEGER;
     Anyo   : INTEGER;
     Letra  : CHAR;
```

Si varias variables tienen el mismo tipo pueden declararse conjuntamente escribiendo sus nombres seguidos, separados por comas. El ejemplo anterior puede escribirse de forma abreviada:

```
VAR  Dia, Mes, Anyo : INTEGER;
     Letra          : CHAR;
```

La descripción BNF de la declaración de variables es la siguiente:

Declaración_de_Variables	**::=**	VAR **{** Lista_de_Variables ; **}**
Lista_de_Variables	**::=**	Lista_de_Identificadores : Tipo
Lista_de_Identificadores	**::=**	Identificador **{** , Identificador **}**
Tipo	**::=**	INTEGER **|** CARDINAL **|** REAL **|** LONGREAL **|** CHAR **|** ...

La definición de Tipo se completará en temas posteriores.

3.7.2 Sentencia de asignación

La forma de conseguir que una variable guarde un nuevo valor es mediante la *sentencia de asignación*. Mediante esta sentencia es posible:

- Inicializar una variable: una variable declarada contiene un valor indeterminado hasta que explícitamente se le da uno. Al hecho de dar un valor inicial a una variable se le llama *inicialización* de la variable.
- Modificar el valor que tenga: es posible modificar el valor que tenga una variable de modo que desde ese momento pase a contener un nuevo valor.

La forma de la sentencia de asignación es la siguiente:

Sentencia_de_Asignación	**::=**	Nombre_de_Variable := Expresión
Nombre_de_Variable	**::=**	Identificador
Expresión	**::=**	...

Una sentencia de asignación consta del nombre de la variable que se quiere modificar, el operador de asignación (los caracteres dos puntos e igual seguidos y sin espacios entre ellos) y una expresión que indica el nuevo valor de la variable.

El modo en el cual se ejecuta una sentencia de asignación es el siguiente:

1) Se calcula el resultado correspondiente a la expresión que aparece a la derecha del operador de asignación.
2) Se almacena el valor calculado en la variable cuyo nombre aparece a la izquierda del operador de asignación.

Para que una sentencia de asignación sea correcta es imprescindible que el tipo de la expresión a la derecha del operador de asignación sea compatible con el tipo declarado para la variable cuyo nombre aparece a la izquierda. Suponiendo las siguientes declaraciones de variables:

```
VAR  Num1           : INTEGER
     Num2           : CARDINAL;
     Num3           : REAL;
     Letra1, Letra2 : CHAR;
```

son válidas las siguientes sentencias de asignación:

```
Num1 := -10;
```

```
Num2 := 32+24;
Num3 := 1.5E2;
Letra1 := '1';
Letra2 := 'Ñ';
```

NOTA: En *Modula-2* es necesario separar las distintas instrucciones del programa con el símbolo punto y coma (;). Además es habitual escribir cada instrucción en una línea. Esto último no es obligatorio.

Sin embargo, las siguientes asignaciones no son válidas:

`Num1 := 40000;`	El valor máximo para un dato INTEGER es 32767
`Num2 := -20;`	Un dato CARDINAL no puede tomar valores negativos.
`Num3 := 10;`	La constante 10 es un entero y `Num3` es REAL;
`Letra1 := 1;`	`Letra1` debe almacenar un carácter, no un número
`Letra2 := 'ab';`	`Letra2` puede almacenar un único carácter

Si intervienen variables o constantes en la expresión a la derecha de una sentencia de asignación, se usará el valor que tengan en dicho momento. Por ejemplo la siguiente secuencia de asignaciones

```
CONST
  DIASSEMANA = 7;
VAR
  a, b, c : INTEGER;

a := DIASSEMANA;
b := 2 * a;
c := a + b;
```

dejará como valores almacenados en las variables `a`, `b` y `c` los valores 7, 14 y 21 respectivamente.

Un caso que requiere cierta atención es aquél en el que la variable que está siendo modificada forma parte de la expresión. Por ejemplo:

```
a := DIASSEMANA;
a := a + 1;
```

A la hora de ejecutar la segunda sentencia de asignación se siguen los pasos que vimos al principio de este apartado:

1) Se calcula el valor de la expresión a la derecha del operador de asignación: a + 1 = 7 + 1 = 8
2) Se almacena dicho valor en la variable que aparece a la izquierda: el nuevo valor de la variable `a` pasa a ser 8.

Como puede observarse, la sentencia `a := a + 1` tiene como resultado que la variable `a` vea su valor incrementado en una unidad. También es posible decrementar el valor de una variable en un unidad (`a := a – 1`), o en cualquier número de unidades (`a := a + 20` hace que el valor almacenado en la variable `a` aumente en veinte unidades).

Dado que es muy habitual incrementar y decrementar el valor de variables *Modula-2* cuenta con dos instrucciones para ello **`INC`** y **`DEC`**:

```
INC(x)      ≡     x := x + 1
DEC(x)      ≡     x := x + 1
INC(x, 10)  ≡     x := x + 10
```

```
DEC(x, 10)  ≡     x := x - 10
```

Veamos un ejemplo de cómo van modificándose los valores de las variables durante la ejecución de un pequeño programa:

```
VAR  Posi         : CARDINAL;
     Dato         : INTEGER;
     EjeX, EjeY   : REAL;
```

EjeX	EJeY	Dato	Posi	Sentencias
?	?	?	?	Ejex := 34.89;
34.89	?	?	?	Dato := 67;
34.89	?	67	?	Posi := **TRUNC**(EjeX) + Dato;
34.89	?	67	101	Dato := **TRUNC**(EjeY) + Posi;
34.89	?	?	101	EjeY := EjeX +**REAL**(Posi);
34.89	135.89	?	101	Dato := Posi**DIV** 5;
34.89	135.89	20	101	

3.8 Operaciones de Entrada y Salida

La mayoría de los programas necesitan una serie de datos. Estos datos suelen ser introducidos por una persona usando alguna unidad de entrada (generalmente el teclado del ordenador). Se necesitan instrucciones que permitan a un programa escrito en *Modula-2* pedir un dato desde el teclado (o desde otra unidad de entrada) y guardarlo en una variable. A este tipo de instrucciones se les llama *operaciones de entrada*.

De un modo similar, la mayoría de los programas suelen calcular unos resultados que hay que emitir al exterior a través de una unidad de salida (generalmente la pantalla del ordenador). Necesitamos instrucciones que permitan a un programa escrito en *Modula-2* enviar resultados al exterior. A este tipo de instrucciones se les denomina *operaciones de salida*.

En *Módula-2* las instrucciones para efectuar E/S se encuentran en módulos de biblioteca, es decir, módulos preprogramados disponibles para el programador. La biblioteca donde están implementadas estas instrucciones depende del compilador de *Modula-2* que se utilice. En el caso de *Top Speed Modula-2* están contenidas en la biblioteca IO (Input-Output) y se describen a continuación.

3.8.1 Operaciones de Salida

Las principales operaciones de salida definidas en el modulo de biblioteca IO son las siguientes:

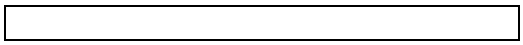
- `WrChar(c)`: escribe en pantalla el valor de `c` que debe ser una expresión de tipo CHAR.

  El siguiente trozo de programa:

  ```
  WrChar('h');
  WrChar('o');
  WrChar('l');
  WrChar('a');
  WrChar(CHR(ORD('!')));
  ```

  producirá la siguiente salida en la pantalla del ordenador:

  ```
  hola!
  ```



- `WrLn`: hace que el resultado correspondiente a la próxima operación de salida se muestre en la siguiente línea de pantalla.

  El siguiente trozo de programa:

```
WrChar('h'); WrLn;
WrChar('o'); WrLn;
WrChar('l'); WrLn;
WrChar('a'); WrLn;
WrChar(CHR(ORD('!')));
```

  producirá la siguiente salida en la pantalla del ordenador:

```
h
o
l
a
!
```

- `WrStr(s)`: escribe la cadena de caracteres `s` en pantalla.

  El siguiente trozo de programa:

```
WrStr('hola¿qué hay?'); WrLn;
WrStr('entre "comillas"');
```

  producirá la siguiente salida en la pantalla del ordenador:

```
hola¿qué hay?
entre "comillas"
```

- `WrInt(i,a)`: escribe en pantalla el valor de `i` que debe ser una expresión de tipo INTEGER. El valor de `a` indica el número de posiciones reservadas para escribir `i` en la pantalla. En caso de que `i` requiera un número de posiciones menor del reservado, las posiciones sobrantes se rellenarán con espacios en blanco. Si el número de posiciones indicado por `a` es menor que el número de caracteres necesarios para imprimir `i`, se imprimirá `i` ignorando lo indicado por `a`.

  El siguiente trozo de programa:

```
WrInt(123,5); WrLn;
WrInt(45,1); WrLn;
WrInt(67,0); WrLn;
WrInt(-89,6); WrLn;
WrInt(1000+500, 6);
```

  producirá la siguiente salida en la pantalla del ordenador:

```
bb123
45
67
bbb-89
bb1500
```

  Es habitual utilizar un ancho cero, de este modo el valor ocupará tantas posiciones en pantalla como necesite.

- `WrCard(c,a)`: idéntico al anterior, pero el valor c debe ser de tipo CARDINAL.
- `WrReal(r,p,a)`: Escribe el valor de la expresión `r`, que debe ser de tipo REAL, en pantalla. Los números reales siempre se muestran usando notación científica y con un único dígito en la parte entera. El significado de `a` es el mismo que en `WrInt`. Se mostrarán `p-1` cifras en la parte fraccionaria.

  El siguiente trozo de programa:

```
WrReal(1.2345, 1, 10); WrLn;
WrReal(1.2345, 2, 10); WrLn;
WrReal(12.345, 5, 0); WrLn;
WrReal(12.345, 6, 0); WrLn;
```

  producirá la siguiente salida en la pantalla del ordenador:

```
bbbbb1.E+0
bbbb1.2E+0
1.2345E+1
1.23450E+1
```

- `WrLngReal(r,p,a)`: igual al anterior pero `r` debe ser una expresión de tipo LONGREAL.

NOTA: El símbolo b representa un espacio en blanco.

3.8.2 Operaciones de Entrada

Las principales operaciones de entrada definidas en el módulo de biblioteca IO son las siguientes:

- `RdChar()`: lee un dato de tipo CHAR del teclado.
- `RdInt()`: lee un número de tipo INTEGER del teclado.
- `RdCard()`: lee un número de tipo CARDINAL del teclado.
- `RdReal()`: lee un número de tipo REAL del teclado.
- `RdLngReal()`: lee un número de tipo LONGREAL del teclado.

Estas instrucciones se usan de la misma forma que un valor dentro de programa, por lo que no pueden aparecer como instrucciones aisladas dentro del mismo (cosa que ocurría con las de salida vistas anteriormente) sino sólo formando parte de una expresión.

El modo usual de utilizar estas operaciones es mediante una sentencia de asignación, en cuya parte izquierda aparece el nombre de la variable en la cual se quiere guardar el valor leído del teclado. En la parte derecha de la sentencia de asignación aparece la operación de entrada. El tipo de la variable en la parte izquierda debe coincidir con el tipo de dato que lee la operación.

Tras cada llamada a una de estas funciones, el programa se detiene a la espera de que el usuario introduzca un valor. Cuando el usuario introduce el dato y pulsa la tecla <ENTER>, el programa continúa ejecutándose y el valor es asignado a la variable en la parte izquierda de la sentencia de asignación.

Tras cada una de estas operaciones de entrada es conveniente ejecutar la operación `RdLn` (que debe importarse también del módulo de biblioteca IO) para evitar complicaciones.

El siguiente trozo de programa (suponiendo que la variable `Num` está declarada con tipo INTEGER) lee un valor entero del teclado, lo almacena en la variable `Num`, suma 10 al valor leído y muestra el resultado por pantalla:

```
Num := RdInt();
RdLn;
Num := Num + 10;
WrInt(Num, 0);
```

Al ejecutar el programa anterior, el ordenador se detiene al llegar a la instrucción `RdInt()` y muestra un cursor parpadeante para indicar al usuario que está a la espera de que introduzca un dato:

```
_
```

En este punto el usuario debe introducir el dato y pulsar la tecla <ENTER>:

```
120_
```

Al pulsar la tecla <ENTER> el dato 120 es asignado a la variable `Num`.. Además, el cursor pasa a la siguiente línea de pantalla de modo que la próxima operación de salida comenzará en dicha línea:

```
120

_
```

La asignación `Num := Num + 10` hace que el valor de la variable `Num` pase a ser 130. Por último, la operación `WrInt(Num,0)` muestra el valor de la variable `Num` con el ancho que sea necesario:

```
120
130
```

Es normal preceder las operaciones de entrada con operaciones de salida que indiquen al usuario qué dato debe introducir. El siguiente programa resulta más claro para el usuario:

```
WrStr("Introduzca un número entero: ");
Num := RdInt();
RdLn;
Num := Num + 10;
WrStr("El número introducido más diez es: ");
WrInt(Num, 0);
```

Al ejecutar este último programa, el ordenador muestra el mensaje `"Introduzca un número entero: "` y un cursor parpadeante, lo cual indica que está a la espera de que el usuario introduzca un dato entero:

```
Introduzca un número entero: _
```

Una vez que el usuario introduce el número y pulsa la tecla <ENTER> el cursor pasa a la siguiente línea:

```
Introduzca un número entero: 120

_
```

```
```

Por último, se incrementa en diez el valor de la variable y ejecutan las dos operaciones de salida finales:

```
Introduzca un número entero: 120
El número introducido más diez es: 130
```

3.9 Estructura general de un programa. Ejemplo

Con los elementos introducidos en este tema es posible escribir programas completos en *Modula-2*. El siguiente ejemplo muestra un programa que lee el radio de un círculo desde el teclado, calcula el área de dicho círculo y lo escribe por pantalla:

```
MODULE Circulo;
FROM IO IMPORT WrStr, RdReal, RdLn, WrReal;
CONST
  PI        = 3.1416; (* El número PI *)
  PRECISION = 5;      (* Nº de dígitos al escribir reales *)
  ANCHO     = 0;      (* Calcular ancho necesario automáticamente *)
VAR
  Radio, Area :REAL;
BEGIN
  WrStr('Introduce el radio: ');
  Radio := RdReal();
  RdLn;
  Area := PI*Radio*Radio;
  WrStr('El área del círculo es ');
  WrReal(Area,PRECISION,ANCHO)
END Circulo.
```

Podemos distinguir las siguientes partes dentro del programa:

- *Cabecera*: todo programa debe comenzar con la palabra reservada MODULE seguida de un identificador, que indica el nombre del programa, y un signo punto y coma. En el presente ejemplo se ha elegido como nombre del programa el identificador `Circulo`. El programa debe finalizar con la palabra reservada END seguida del nombre del programa (el mismo identificador usado en la cabecera) y un punto.
- *Listas de importación*: a continuación es necesario enumerar los módulos de biblioteca usados por el programa, especificando cada una de las operaciones utilizadas. Los módulos de bibliotecas contienen elementos que se utilizan con frecuencia en el desarrollo de distintos programas, por lo que están disponibles en el lenguaje, y por lo tanto, no es necesario programarlos. Para utilizarlos hay que indicarlo en la lista de importación. Esto se puede hacer de dos formas:

  1. Forma implícita:

     ```
     IMPORT NombreBiblioteca;
     ```

     En este caso se puede utilizar cualquier elemento definido en la biblioteca refiriéndonos a él mediante su nombre precedido por el del módulo.

  2. Forma explícita:

     ```
     FROM NombreBiblioteca IMPORTListaElementos;
     ```

En este caso sólo se pueden utilizar los elementos especificados en `ListaElementos` refiriéndonos a él sólo por su nombre. Este es el caso del ejemplo.

Cada compilador de *Módula-2* tiene sus propias bibliotecas y define las características de los procedimientos que contiene. Este es un aspecto a revisar siempre que, aun usando el mismo lenguaje, cambiemos de compilador.

- *Declaraciones*: en esta sección se declaran las distintas constantes y variables que son utilizadas en el programa. El programa ejemplo usa tres valores constantes (`PI`, `PRECISION` y `ANCHO`) y dos variables de tipo REAL (`Radio` y `Area`).
- *Parte Ejecutiva*: comienza con la palabra reservada BEGIN y en ella aparecen las instrucciones que constituyen el programa, separadas entre sí con el signo punto y coma.

Es posible incluir en cualquier punto del programa un comentario. Un *comentario* es una secuencia de caracteres que es ignorada por el compilador. Se usan para documentar el programa, de manera que aunque no contribuyan a resolver el problema, sí ayudan a mejorar la comprensión del programa. Se insertan en el programa escribiendo un texto entre los símbolos `(*` y `*)`.

Usando la notación BNF, la estructura de un programa en *Modula-2* es la siguiente:

Modulo_de_Programa	**::=**	Cabecera_de_Programa
		{ Lista_de_Importación **}**
		{ Declaraciones **}**
		Parte_Ejecutiva
		Fin_de_Programa
Cabecera_de_Programa	**::=**	MODULE Nombre_de_Programa ;
Lista_de_Importación	**::=**	FROM Nombre_de_Biblioteca IMPORT Lista_de_Operaciones ; **|**
		IMPORT Nombre_de_Biblioteca ;
Lista_de_Operaciones	**::=**	Nombre_de_Operación **{** , Nombre_de_Operación **}**
Declaraciones	**::=**	Declaración_de_Constante **|** Declaración_de_Variable
Parte_Ejecutiva	**::=**	BEGIN Secuencia_de_Sentencias
Secuencia_de_Sentencias	**::=**	Sentencia **{** ; Sentencia **}**
Sentencia	**::=**	Sentencia_de_Asignación **|** ...
Fin_de_Programa	**::=**	END Nombre_de_Programa **.**
Nombre_de_Programa	**::=**	Identificador
Nombre_de_Biblioteca	**::=**	Identificador
Nombre_de_Operación	**::=**	Identificador

Podemos observar que tanto la lista de importación como las declaraciones pueden aparecer cero, una o más veces en un programa. Completaremos la definición de sentencia a lo largo del curso.

Relación de Problemas (Tema 3)

1. Escribir un programa que lea desde el teclado dos valores de tipo INTEGER y escriba en pantalla su suma, su diferencia, su producto, su cociente y su resto.

2. Escribir un programa que pida una letra desde el teclado, la pase a mayúscula e imprima el resultado por pantalla.

3. Escribir un programa que lea el radio y la altura de un cilindro desde el teclado y escriba por pantalla el área y volumen correspondiente.

NOTA: El área y volumen del cilindro de altura *h* y radio *R* es:

$$\text{Area} = 2\pi R^2 + 2\pi Rh$$

$$\text{Volumen} = \pi R^2 h$$

4. Escribir un programa que lea desde el teclado una cantidad de segundos y muestre por pantalla el número de horas, minutos y segundos equivalentes.

5. Escribir un programa que lea un carácter del teclado y escriba por pantalla su código dentro de la tabla ASCII.

6. a) ¿Qué ocurre al ejecutar el siguiente programa?

```
MODULE Prueba;
FROM IO IMPORT WrInt;
VAR
  Num : INTEGER;
BEGIN
  Num := MAX(INTEGER);
  Num := Num + 10 – 100;
  WrInt (Num, 0)
END Prueba.
```

b) ¿Cómo puede solucionarse el problema?

c) ¿Qué ocurre si cambiamos el programa por el siguiente?

```
MODULE Prueba;
FROM IO IMPORT WrCard;
VAR
  Num : CARDINAL;
BEGIN
  Num := MAX(INTEGER);
  Num := Num + 10 – 100;
  WrCard (Num, 0)
END Prueba.
```

7. Escribe un programa que lea del teclado la longitud de los dos catetos de un triángulo rectángulo, calcule la longitud de la hipotenusa y la escriba por pantalla.

8. Escribe un programa que lea la longitud del lado de un cuadrado y escriba en pantalla su área y su perímetro.

9. Escribe un programa que lea un carácter del teclado y escriba en pantalla el carácter que le sigue en el código ASCII.

10. Encuentra los errores en el siguiente programa:

```
MODULE Erroneo;
FROM IO IMPORT WrInt;
CONST
  PRECISION = 2;
  ANCHO     = 0;
VAR
  Num1 : INTEGER;
  Num2 : CARDINAL;
  Num3 : REAL;
BEGIN
  Num1 := 10;
  Num2 := -100;
  Num3 := 13.;
  Letra := 'A'
  Num3 := Num3 + 1;
  WrStr("Valor número 1: ");
  WrInt(Num1, ANCHO);
  WrLn;
  WrStr("Valor número 2: ");
  WrCard(Num2, ANCHO);
  WrLn;
  WrStr(Valor número 3:);
  WrReal(Num3, PRECISION, ANCHO);
  WrLn;
  WrStr("Valor letra: ");
  WrChar(Letra);
  WrLn
END Erroneo.
```