* Ejercicio 26 *
Modifique la escritura de la BNF para if y para while de tal forma que no requieran una aclaración en lenguaje natural.

3.6 BNF Y EL ANSI C

El Lenguaje de Programación C fue desarrollado entre 1969 y 1972 por Dennis Ritchie, con el objetivo principal de implementar el sistema operativo UNIX en un lenguaje de alto nivel.

A mediados de la década de los 80, el ANSI (*American National Standards Institute*) comenzó a desarrollar una estandarización de este LP. En diciembre de 1989, esta estandarización fue aprobada y se publicó en 1990 como "American National Standard for Information Systems – Programming Language C"; lo llamaremos *Manual de Referencia Oficial del ANSI C* [MROC].

A partir de ese momento, este LP es conocido como ANSI C, aunque en la actualidad ya se utilizan C y ANSI C como sinónimos.

En esta sección se describe y se analiza la sintaxis de un subconjunto del ANSI C, de acuerdo a su definición oficial. Tengamos en cuenta, entonces, cómo es el BNF que utiliza el MROC:

1º Los noterminales (categorías léxicas y sintácticas) están escritos en itálica, eliminándose los metasímbolos < y >.

2º El "operador es" está representado por : ("dos puntos"). Como dice el MROC, "un : después de un noterminal introduce su definición."

3º No existe el metasímbolo | ("ó"), sino que cada lado derecho de un determinado noterminal se escribe en una línea separada.

4º En algunos casos se usa el metasímbolo uno de para representar varios lados derechos que son caracteres para un noterminal; es cuando un noterminal representa un conjunto de caracteres.

5º Los terminales se escriben en negritas.

6º Un símbolo opcional está indicado con un subíndice $_{op}$.

7º El lado derecho de una regla recursiva se representa como en el BNF original; no utiliza el metasímbolo incorporado por Wirth para indicar una secuencia de "0 o más veces".

Ejemplo 22
Un token o categoría léxica se define se esta manera, con cada lado derecho en una línea separada:

token: *palabraReservada*
identificador
constante
literalCadena
operador
carácterPuntuación

Con el solo fin de escribir estas definiciones en forma más compacta, en algunos casos ampliaremos el uso del metasímbolo uno de, empleándolo así:

Ejemplo 23
La definición del Ejemplo 22 la podemos escribir en forma más compacta de esta manera:

token: uno de *palabraReservada identificador constante literalCadena operador carácterPuntuación*

* Ejercicio 27 *
¿El Ejemplo 23 define LRs o LICs? Si son LRs, indique cuáles son finitos y cuáles son infinitos. Si son LICs, justifique su afirmación.

Ejemplo 24
Con respecto al uso de un símbolo opcional, el MROC, presenta el siguiente ejemplo:

{ *expresión*$_{op}$ **}**

que indica una expresión opcional encerrada entre llaves; esto es:

{ *expresión*$_{op}$ **}** equivale a: **{** *expresión* **}** ó **{ }**

A continuación veremos un ejemplo de un programa simple – pero completo – en ANSI C, que será utilizado luego como referencia. Las líneas del programa serán numeradas al solo efecto de facilitar su identificación cuando nos referimos a ellas en las diversas explicaciones; esta numeración nunca forma parte de un programa en ANSI C.

3.6.1 UN PROGRAMA EN ANSI C

Ejemplo 25

```
 1  /* Programa en ANSI C – J.Muchnik – feb'2010 */
 2  #include <stdio.h>
 3  int Mayor (int, int);
 4  int main (void) { /* comienza main */
 5     int n1, n2, max;
 6     printf ("Ingrese dos números enteros: ");
 7     scanf ("%d %d", &n1, &n2);
 8     max = Mayor (n1, n2);
 9     printf ("El mayor entre %d y %d es %d\n", n1, n2, max);
10     return 0;
11  } /* fin main */
12  int Mayor (int a, int b) { /* comienza Mayor */
13     if (a > b)
14        return a;
15     else
16        return b;
17  } /* fin Mayor */
```

Análisis:
Este programa en ANSI C está compuesto por:
Linea 1: un comentario (no forma parte del lenguaje ANSI C; es analizado por el preprocesador)
Linea 2: una directiva al preprocesador (no forma parte del lenguaje ANSI C)
Linea 3: el prototipo de la función Mayor (es la declaración de esta función)
Lineas 4 a 11: la definición de la función main, función principal de todo programa ANSI C
Lineas 12 a 17: la definición de la función Mayor, que es utilizada por la función main.

* Ejercicio 28 *
(a) Investigue e informe qué es el PREPROCESADOR.
(b) Investigue e informe cada uno de los constructos que componen el programa del Ejemplo 25.
(c) Investigue e informe qué hace el programa C del ejemplo anterior.

A continuación definiremos la sintaxis de este simple programa C, en BNF. Recordemos que los comentarios y las directivas al preprocesador no pertenecen al ANSI C, como se indicó en el Análisis del Ejemplo 25; por ello, en el próximo ejemplo aparece el noterminal *noC* para representar los contenidos de las líneas 1 y 2.

Ejemplo 26

programaC: *noC prototipo main función*
prototipo: *tipoFunción nombreFunción* **(** *tiposParámetros* **) ;**
tiposParámetros: *tipoUnParámetro*
tiposParametros **,** *tipoUnParámetro*
main: *encabezamiento cuerpo*
encabezamiento: **int** main **(void)**
cuerpo: **{** *declaración sentencias* **}**
declaración: *tipoVariables variasVariables ;*
variasVariables: variable
variasVariables variable
sentencias: *sentencia*
sentencias sentencia
. . .

* Ejercicio 29 *
Analice cada noterminal definido en el Ejemplo 26 y escriba a qué líneas del programa C del Ejemplo 25 corresponde.

* Ejercicio 30 *
Sea un programa C cuya función main es la siguiente:

```
int main (int argc, char *argv[]) {
   char vec [10+1];
   int i;
   if (argc > 10) {
      printf ("No se puede\n");
   else
      for (i=0; i < argc; i++)
         vec[i] = argv[i][0];
      vec[i] = '\0';
      printf ("La cadena creada es %s\n", vec);
   return 0;
}
```

Investigue e informe el significado de cada componente de esta función y el de la función en su totalidad. Escriba una BNF que describa la estructura general de esta función main.

3.6.2 SINTAXIS DE UN SUBCONJUNTO DE ANSI C

En el MROC, cada una de las secciones está dividida en tres partes:

1) "Sintaxis", con una BNF que define la sintaxis del constructo analizado;
2) "Restricciones", en la que figuran ciertas restricciones existentes sobre el constructo definido en "Sintaxis" y que la BNF no expresa totalmente (como ya hemos visto en el caso de Pascal);
3) "Semántica", donde se explica el significado de lo definido en "Sintaxis", lo veremos en el capítulo 4.

En esta sección analizaremos la primera parte, "Sintaxis", y, en algunas ocasiones, haremos referencia a la segunda parte, "Restricciones", para completar la comprensión de la sintaxis.

La tercera parte, "Semántica", será analizada en el próximo capítulo.

3.6.2.1 LAS CATEGORÍAS LÉXICAS

Esta sección se refiere a los TOKENS o CATEGORÍAS LÉXICAS del ANSI C, que ya han sido enumerados en el Ejemplo 22: las palabras reservadas, los identificadores, las constantes, las cadenas literales, los operadores y los caracteres de puntuación. Estos tokens son LRs.

Los elementos que componen un LR se denominan palabras en el área de los Lenguajes Formales. Aquí los llamaremos LEXEMAS, término utilizado en el área de los compiladores que cubriremos en el Volumen 2.

Nota 4
Siendo rigurosos, un "lexema" puede ser tanto una palabra de un LR (token), como puede ser una cadena que no pertenece a ningún LR. El último caso, si aparece, será debidamente aclarado.

A continuación analizaremos el programa del Ejemplo 25 y construiremos una tabla en la que figuren algunos de los lexemas de esta codificación en ANSI C y a qué CATEGORÍA LÉXICA o TOKEN pertenecen. En otras palabras, trabajaremos con los LRs que componen la sintaxis del ANSI C.

Ejemplo 27
Como los comentarios y las directas al preprocesador no pertenecen al lenguaje ANSI C, los eliminamos en el proceso de detección de lexemas:

```
 3  int Mayor (int, int);
 4  int main (void) {
 5     int n1, n2, max;
 6     printf ("Ingrese dos numeros enteros: ");
 7     scanf ("%d %d", &n1, &n2);
 8     max = Mayor (n1, n2);
 9     printf ("El mayor entre %d y %d es %d\n", n1, n2, max);
10     return 0;
11  }

/* continúa */
```

```
12  int Mayor (int a, int b) {
13     if (a > b)
14        return a;
15     else
16        return b;
17  }
```

NRO.LÍNEA	LEXEMA	TOKEN
3	int	*palabraReservada*
3	Mayor	*identificador*
3	(	*carácterPuntuación*
3	;	*carácterPuntuación*
4	main	*identificador*
4	void	*palabraReservada*
4	{	*carácterPuntuación*
5	n1	*identificador*
6	printf	*identificador*
6	(	*operador*
6	"Ingrese dos numeros enteros: "	*literalCadena*
7	&	*operador*
8	=	*operador*
10	return	*palabraReservada*
10	0	*constante*
13	if	*palabraReservada*
..	..	..
..	..	..

* Ejercicio 31 *
Complete la tabla del Ejemplo 25 e indique cuántos lexemas tiene en total el programa.

* Ejercicio 32 *
Sea la siguiente función en ANSI C:

```
double XX (double a, int b) {
   while (a > b) b++;
   return b;
}
```

Construya una tabla, como la del ejemplo anterior, con <u>todos</u> los lexemas que hay en esta función y la CATEGORÍA LÉXICA a la que pertenece cada uno.

Habíamos dicho que los tokens del ANSI C son: *palabraReservada*, *identificador*, *constante*, *literalCadena*, *operador* y *carácterPuntuación*.

Algunos de estos tokens representan LRs finitos y otros representan LRs infinitos; pero todos representan LRs. Veamos cada uno de ellos, teniendo en cuenta que <u>las BNFs para los tokens siempre son precisas y completas</u>.

3.6.2.1.1 LAS PALABRAS RESERVADAS

El ANSI C posee 32 palabras reservadas que constituyen los elementos de un LR finito: el token palabraReservada. Mediante BNF haremos una lista de algunas de ellas:

palabraReservada: una de **char do double else float for if int long return sizeof struct typedef void while**

* Ejercicio 33 *
Investigue cada una de esas palabras reservadas; defínala y muestre ejemplos de su uso.

* Ejercicio 34 *
Investigue e informe si main es una palabra reservada. Si no lo es, ¿la puede utilizar como nombre de una variable? Justifique su afirmación.

3.6.2.1.2 LOS IDENTIFICADORES

Otro de los tokens es identificador, que constituye un LR infinito. Siguiendo la línea histórica de ALGOL y de PASCAL, los identificadores en C tampoco pueden comenzar con un dígito. ¿Por qué? Este LR infinito es definido en BNF de la siguiente manera:

identificador: noDígito
identificador noDígito
identificador dígito
noDígito: uno de **_ 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
dígito: uno de **0 1 2 3 4 5 6 7 8 9**

Nota 5
Observe que el carácter "guión bajo" forma parte del conjunto definido por noDígito. Por lo tanto, en ANSI C, un identificador puede comenzar con letra o con "guión bajo".

* Ejercicio 35 *
(a) Investigue e informe si _ (un guión bajo) y __ (dos guiones bajos) son identificadores válidos en ANSI C.
(b) Investigue e informe si los identificadores `ab3` y `AB3` representan el mismo objeto. Justifique su respuesta.

3.6.2.1.3 LAS CONSTANTES

Continuamos con la descripción de los tokens. Las constantes son definidas en el MROC de la siguiente manera:

constante: una de *constanteReal constanteEntera constanteEnumeración constanteCarácter*

Ahora veremos una BNF parcial de las constantes: nos ocuparemos de las constantes numéricas, tanto enteras como reales. Ambas representan diferentes LRs infinitos. Las constantes numéricas se definen "sin signo" porque, en ANSI C, el signo es un operador. Entonces,

constanteNumérica: una de *constanteEntera constanteReal*

Comenzamos con la definición de la constanteEntera teniendo en cuenta que, en ANSI C, existen tres tipos de constantes enteras:

(a) las decimales, o sea, en base 10;
(b) las octales (en base 8); y
(c) las hexadecimales (en base 16).

En consecuencia, La BNF para las constantes enteras en ANSI C es la siguiente:

constanteEntera: *constanteDecimal sufijoEntero*$_{op}$
constanteOctal sufijoEntero$_{op}$
constanteHexadecimal sufijoEntero$_{op}$
constanteDecimal: *dígitoNoCero*
constanteDecimal dígito
constanteOctal: **0**
constanteOctal dígitoOctal
constanteHexadecimal: *ceroX dígitoHexa*
constanteHexadecimal dígitoHexa
ceroX: uno de **0x 0X**
dígitoNoCero: uno de **1 2 3 4 5 6 7 8 9**
dígito: uno de **0 1 2 3 4 5 6 7 8 9**
dígitoOctal: uno de **0 1 2 3 4 5 6 7**
dígitoHexa: uno de **0 1 2 3 4 5 6 7 8 9 a b c d e f A B C D E F**
<*sufijoEntero*>: . . .

* Ejercicio 36 *
(a) Investigue e informe qué significa el noterminal sufijoEntero.
(b) Sea la constante 425. ¿Es una constante octal correcta? Justifique su respuesta.

* Ejercicio 37 *
(a) De acuerdo al BNF del MROC, ¿cuál es la mínima constanteDecimal?
(b) ¿Cómo se representa el número entero decimal 0 en ANSI C?

* Ejercicio 38 *
(a) Para cada una de las siguientes constantes enteras, escriba a cuál de los tres tipos pertenece:
00 123U 0x4A2F2 0654 99LU 02L
(b) Derive verticalmente por izquierda la constante 0Xa4b8.

Continuamos con la BNF de las constantes. Ahora corresponde definir la sintaxis de las constantes reales, que se pueden representar en punto fijo (como 2.14) o en punto flotante (como 3E18, que significa $3x10^{18}$). La BNF para las constantes reales en ANSI C es la siguiente:

constanteReal: *constanteFraccionaria parteExponente*$_{op}$ *sufijoReal*$_{op}$
secuenciaDígitos parteExponente sufijoReal$_{op}$
constanteFraccionaria: *secuenciaDígitos*$_{op}$ **.** *secuenciaDígitos*
secuenciaDígitos **.**
parteExponente: *operadorE signo*$_{op}$ *secuenciaDígitos*
operadorE: uno de **e E**
signo: uno de **+ -**

(continúa en la página siguiente)

secuenciaDígitos: dígito
secuenciaDígitos dígito
dígito: uno de **0 1 2 3 4 5 6 7 8 9**
sufijoReal: . . .

* Ejercicio 39 *
(a) Investigue e informe qué significa el noterminal sufijoReal.
(b) Sea la constante 425. ¿Es una constante real? Justifique su respuesta.

* Ejercicio 40 *
(a) De acuerdo al BNF del MROC, ¿cuál es la mínima constanteReal? Escríbala en todos los formatos posibles.
(b) Construya una tabla con todos los formatos de constantes reales existentes en ANSI C y un ejemplo para cada uno de ellos.

* Ejercicio 41 *
Derive verticalmente por izquierda la constante real 4.6E–23

* Ejercicio 42 *
Una constanteCarácter se representa así: 'a'; es decir, el carácter a, en este caso, se escribe entre DELIMITADORES apóstrofos. Investigue e informe cómo se representa la constanteCarácter apóstrofo (').

* Ejercicio 43 *
Investigue e informe qué es y cómo se representa una constanteEnumeración.

* Ejercicio 44 *
El token literalCadena corresponde a todas las cadenas de caracteres que se pueden representar en ANSI C, como "abc"; como se observa, los DELIMITADORES son comillas.
(a) Investigue e informe cómo se representa el literalCadena "".
(b) Investigue e informe cómo se representa el literalCadena formado por estos tres caracteres: "A".
(c) Investigue e informe si un literalCadena puede ser vacío y, en tal caso, cómo se representa. Justifique su respuesta.
(d) Investigue e informe sobre la diferencia existente entre "234" y 234.
(e) Escriba una BNF que defina el subconjunto de cadenas formadas solo por dos o más dígitos, más la cadena vacía.

3.6.2.1.4 LOS OPERADORES Y LOS CARACTERES DE PUNTUACIÓN

Veamos, a continuación, las BNF para algunos operadores y para algunos caracteres de puntuación del ANSI C; ambos son LRs finitos:

operador: uno de **++ * + & ! sizeof / % < <= == != && || ?: = +=**
carácterPuntuación: uno de **( ) { } , ; [ ]**

* Ejercicio 45 *
(a) Investigue e informe sobre cada uno de los operadores mencionados arriba. Tenga en cuenta que, en algunos casos, un símbolo puede representar más de un operador.
(b) Brinde diferentes ejemplos de aplicación de esos operadores.

* Ejercicio 46 *
Investigue e informe sobre cada uno de los caracteres de puntuación mencionados arriba.

* Ejercicio 47 *
Existen caracteres en ANSI C que son operadores y también son caracteres de puntuación. Esto es lo que ocurre con los paréntesis, los corchetes y con la "coma". Investigue e informe, con ejemplos, cuándo estos caracteres representan caracteresPuntuación y cuándo representan operadores.

3.6.2.2 LAS CATEGORÍAS GRAMATICALES

En esta sección analizaremos la sintaxis de tres constructos más complejos:
(a) las expresiones,
(b) las declaraciones; y
(c) las sentencias.

3.6.2.2.1 LAS EXPRESIONES

Las expresiones constituyen un importantísimo LIC en cualquier Lenguaje de Programación. En forma genérica, podemos decir que, sintácticamente, una EXPRESIÓN es una secuencia de operandos y operadores más el posible uso de paréntesis. Téngase en cuenta que siempre una constante, una variable o la invocación a una función son casos particulares y mínimos de una expresión.

Muchas veces, no es suficiente con definir una BNF para las expresiones, sino que, además, debe haber restricciones que agregan importantes características que la GIC o la BNF no pueden representar adecuadamente. Esto ya lo hemos visto con las expresiones Booleanas de Pascal y también lo veremos, ahora, con las expresiones en ANSI C.

Otro propiedad importante a tener en cuenta es que en ANSI C no existen las constantes booleanas, como TRUE y FALSE en Pascal, y tampoco existe el tipo Boolean. La resolución a este problema es muy simple.

* Ejercicio 48 *
(a) Investigue e informe cómo el ANSI C representa el valor verdadero y el valor falso.
(b) Investigue e informe cuál es el resultado de una expresión booleana en ANSI C. Escriba diversos ejemplos.

A continuación, analizaremos un subconjunto reducido de las expresiones del ANSI C. Se debe tener en cuenta que el ANSI C tiene más de 40 operadores distribuidos en 15 niveles de precedencia. Un análisis de la BNF total para expresiones en ANSI C traería más complicaciones que beneficios; por ello, trabajaremos sobre un subconjunto de 33 reglas que representarán todas las características importantes de la BNF de expresiones en ANSI C. En estas producciones encontraremos operadores

binarios (operan sobre dos operandos), operadores unarios (operan sobre un solo operando) y el único operador ternario (opera sobre tres operandos).

Recordemos que una adecuada BNF de expresiones debe representar correctamente tanto la precedencia (o prioridad) como la asociatividad de los operadores. En el caso de ANSI C, veremos que hay operadores asociativos a izquierda y operadores asociativos a derecha, diferencia que es determinada por el formato de la producción: recursiva a izquierda para el primer caso y recursiva a derecha para el segundo caso.

Introducimos dos definiciones que brinda el MROC y que serán de gran utilidad:

– Objeto: es una región de la memoria de la computadora, compuesta por una secuencia de uno o más bytes consecutivos, que tiene la capacidad de contener la representación de valores. Por ejemplo: una variable es un objeto; una expresión no es un objeto.

– Lvalue (traducido, habitualmente, como ValorL; la "L" proviene de "left", palabra inglesa que significa "izquierda" o "izquierdo"): es una expresión que designa un objeto. Su nombre proviene de la operación de asignación, donde la expresión que se encuentra a la izquierda del operador = debe ser un ValorL. Ejemplos de ValorL: variable a, elemento de un vector v[3], expresión *(p+2); en cambio, una constante y una expresión aritmética no son ValoresL.

* Ejercicio 49 *
Investigue e informe qué significa la expresión *(p+2).

Presentamos, entonces, el siguiente subconjunto de la BNF de expresiones en ANSI C con 17 operadores distribuidos en 11 niveles de precedencia. Analizaremos la BNF en la medida que se va desarrollando.

1 *expresión: expAsignación*
2 *expAsignación: expCondicional*
3 *expUnaria operAsignación expAsignación*
4 *operAsignación:* uno de **= +=**

La producción 1 presenta al axioma, expresión, en una forma simplificada; no consideramos la otra producción para el axioma, con el operador de mínima prioridad (el operador "coma"), para simplificar este análisis.

Las producciones 2 y 3 desarrollan expresión de Asignación, una BNF recursiva a derecha, como se ve en la producción 3, que nos informa que operAsignación (el operador de asignación) es asociativo a derecha y que, además, tiene una prioridad muy baja porque está muy cerca del axioma. La realidad indica que solo el operador "coma" tiene menor prioridad que el operador de asignación.

Por último, la producción 4 muestra dos de los operadores de Asignación existentes: la asignación simple (=) y uno (+=), a modo de representante de los 10 operadores de asignación compuestos que tiene el ANSI C.

5 *expCondicional: expOr*
6 *expOr* **?** *expresión* **:** *expCondicional*

Aquí aparece el único operador ternario, conocido como operador condicional. La producción 6 es recursiva a derecha, por lo que se trata de un operador asociativo a derecha. Si el valor de expOr (expresión "or") es distinto de 0 (esto es: verdadero), entonces el resultado de esta operación será el

valor de expresión; en cambio, si el valor de expOr es 0 (esto es: falso), entonces el resultado de esta operación será el valor de expCondicional.

```
 7 expOr: expAnd
 8         expOr || expAnd
```

La producción 8 define el operador booleano "or" (||) que, como se ve, es asociativo a izquierda.

```
 9 expAnd: expIgualdad
10          expAnd && expIgualdad
```

La producción 10 introduce el operador booleano "and" (&&), que también es asociativo a izquierda.

```
11 expIgualdad: expRelacional
12               expIgualdad == expRelacional
```

La producción 12 incorpora el operador de "igualdad", asociativo a izquierda. En el mismo nivel se encuentra también el operador de "desigualdad" (**!=**).

```
13 expRelacional: expAditiva
14                 expRelacional >= expAditiva
```

La producción 14 introduce el operador relacional >=. En el mismo nivel se hallan los operadores **>**, **<** y **<=**, que tienen igual precedencia y son asociativos a izquierda. En el caso de todos los operadores relacionales, si la expresión evaluada es verdadera, el valor de verdad producido será 1, y si es falso, será 0.

```
15 expAditiva: expMultiplicativa
16              expAditiva + expMultiplicativa
```

La producción 16 presenta al operador de suma; también en este nivel de precedencia se encuentra el operador de resta y ambos son asociativos a izquierda, como lo muestra la producción recursiva a izquierda.

```
17 expMultiplicativa: expUnaria
18                     expMultiplicativa * expUnaria
```

La producción 18 presenta al operador de multiplicación; también en este nivel de precedencia hay producciones smilares para los operadores **/** (división) y **%** (operador módulo). La producción es recursiva a izquierda y, por lo tanto, estos tres operadores son asociativos a izquierda.

```
19 expUnaria: expPostfijo
20             ++ expUnaria
21             operUnario expUnaria
22             sizeof (nombreTipo)
23 operUnario: uno de & (dirección) * (puntero) – (signo) ! ("not")
```

Las producciones 20 a 22 introducen varios operadores unarios, aunque no son todos los que ofrece el ANSI C. En la producción 20 se introduce el operador de preincremento, que debe aplicarse a un ValorL modificable. La producción 21 introduce los operadores unarios detallados en la producción 23. Observe que las producciones 20 y 21 son recursivas a derecha. La producción 22 introduce un operador inusual que determina una cantidad de bytes; aquí su operando es un Tipo pero también puede ser una variable.

* Ejercicio 50 *
Investigue e informe, con ejemplos, qué es un ValorL modificable.

```
24 expPostfijo: expPrimaria
25              expPostfijo [ expresión ]
26              expPostfijo ( listaArgumentos_op )
27 listaArgumentos: expAsignación
28                      listaArgumentos , expAsignación
```

En este segmento de producciones incorporamos dos operadores. En la linea 25 se introduce el operador representado por el par de corchetes, utilizado para acceder a un elemento de un arreglo de cualquier dimensión. En la línea 26 se introduce el operador representado por un par de paréntesis, utilizado en la invocación a una función. Como se ve, la lista de Argumentos es opcional, pero nunca lo son los paréntesis.

```
29 expPrimaria: identificador
30              constante
31              literalCadena
32              ( expresión )
```

Las líneas 29 y 30 muestran los elementos básicos que intervienen en la escritura de una expresión; ambos están definidos anteriormente, entre los tokens. La línea 32 presenta una recursividad indirecta que nos lleva al axioma.

```
33 nombreTipo: uno de char int double
```

Finamente, la regla 33 presenta algunos tipos de datos primitivos del ANSI C.

A continuación habrá una serie de Ejemplos y de Ejercicios sobre las Expresiones en ANSI C.

Ejemplo 28
Sea la siguiente derivación:

```
expPostfijo
expPostfijo [expresión]
expPostfijo [expresión][expresión]
. . .
```

* Ejercicio 51 *
Continúe la derivación del ejemplo anterior para obtener la expresión: `matriz12[2+3][4*6]`.

* Ejercicio 52 *
Según la BNF de las Expresiones, ¿cuántas dimensiones puede tener un arreglo?

Ejemplo 29
El operador preincremento **++** debe tener un operando ValorL modificable. Por ejemplo, ++a (siendo a una variable entera) es correcto, pero ++12 es incorrecto.

* Ejercicio 53 *
(a) Investigue e informe el significado del operador preincremento **++** y cómo actúa.
(b) Investigue e informe el operador **&** (dirección) y cómo actúa.
(c) Investigue e informe el operador unario `*` (puntero) y cómo actúa.
(d) Investigue e informe el operador unario `!` (“not”) y cómo actúa.
(e) Investigue e informe el operador unario **sizeof** y cómo actúa.

Si nos atenemos estrictamente a la BNF de las expresiones y no tenemos en cuenta ciertas restricciones que se expresan en lenguaje natural, podemos llegar a conclusiones absurdas.

Ejemplo 30
A nadie se le ocurre pensar que la expresión de asignación `1 = 2` es sintácticamente correcta. Sin embargo . . .

* Ejercicio 54 *
Derive verticalmente la expresión del Ejemplo 30.

Conclusión:

En algunos casos, la BNF describe situaciones que son sintácticamente incorrectas. Esto se debe a que la BNF también es utilizada para la construcción del compilador y ciertas detecciones las hace este programa para mayor eficiencia, como veremos en el Volumen 2.

En consecuencia, es indispensable leer las restricciones que figuran en los manuales, acompañando a las descripciones en BNF. Y si no son claras o suficientes, uno mismo debe agregarlas.

Por caso, cuando el ANSI C describe los operadores de asignación, agrega una restricción muy poderosa que dice: "Un operador de asignación debe tener un ValorL modificable como su operando izquierdo." Obviamente, una constante no es un ValorL modificable y, por lo tanto, la expresión de asignación del Ejemplo 30 es incorrecta (aunque se pueda derivar).

* Ejercicio 55 *
Construya una Tabla de Derivación para la expresión `a = b = c = 2`.

* Ejercicio 56 *
(a) Según la BNF de expresiones, ¿es `2 <= 5 <= 4` sintácticamente correcta? Si responde que no, justifique su respuesta.
(b) Si responde que sí, escriba la Tabla de Derivación para esa expresión y luego escriba la Tabla de Evaluación.

* Ejercicio 57 *
(a) Según la BNF de expresiones, ¿es `2 && 3 || 4 && 5 || 6` sintácticamente correcta? Si responde que no, justifique su respuesta.
(b) Si responde que sí, escriba la Tabla de Derivación para esa expresión y luego escriba la Tabla de Evaluación.

* Ejercicio 58 *
Compare la precedencia, la asociatividad y la cantidad de operadores del lenguaje Pascal con los de ANSI C.

3.6.2.2.2 DECLARACIONES Y DEFINICIONES

Una definición es como una declaración, pero provoca una reserva de memoria para el objeto o la función definidos.

Ejemplo 31
Recordemos la parte de ANSI C del programa del Ejemplo 25:

```
 3  int Mayor (int, int);
 4  int main (void) { /* comienza main */
 5     int n1, n2, max;
    . . .
11  } /* fin main */
12  int Mayor (int a, int b) { /* comienza Mayor */
    . . .
17  } /* fin Mayor */
```

En este programa, el prototipo de la linea 3 es una declaración de la función Mayor, mientras que en las líneas 12 a 17 se define esta función. Asimismo, en la línea 5 se definen tres variables.

Ejemplo 32
Otra situación se presenta con la declaración y la definición de una estructura en ANSI C:

```
. . .
typedef struct {
           int a;
           char vec[20+1];
           int mat[4][18];
        } REGISTRO;
. . .
int main (void) {
   REGISTRO x;
    . . .
}
```

Primero se declara REGISTRO como una estructura con tres campos. Luego, en la función main, se define una variable (un objeto) x de tipo REGISTRO.

* Ejercicio 59 *
Investigue e informe, acompañado de ejemplos:
(a) qué es typedef, y
(b) qué es struct.

A continuación, veamos un subconjunto elemental de la BNF para las declaraciones y definiciones de variable simples:

declaVarSimples: tipoDato listaVarSimples ;
tipoDato: uno de **`int double char`**
listaVarSimples: unaVarSimple
listaVarSimples , unaVarSimple
unaVarSimple: variable inicial$_{op}$
variable: identificador
inicial: **=** *constante*
identificador: . . .
constante: . . .

Ejemplo 33

```
int a, b = 10, c, d = 4;
```

* Ejercicio 60 *
Derive la definición del Ejemplo 33 partiendo de la BNF dada.

* Ejercicio 61 *
Escriba una BNF que represente la declaración de un typedef struct, como se mostró en el Ejemplo 32, pero que solo contenga variables simples.

Debemos diferenciar dos conceptos:
(1) DERIVABLE DE UNA BNF DADA, y
(2) SINTÁCTICAMENTE CORRECTO.

* Ejercicio 62 *
(a) Sea el siguiente fragmento de una función:

```
void XX (void) {
   int a;
   double a;
   . . .
}
```

Investigue los dos conceptos incorporados e indique cuál se aplica en este caso. Justifique su respuesta.
(b) ¿A cuál de los dos conceptos responde la expresión 12 = 23+6?
(c) ¿Y la expresión 4/3?

3.6.2.2.3 SENTENCIAS

Como se sabe, las sentencias especifican las acciones que llevará a cabo la computadora en tiempo de ejecución. Seguidamente, describiremos y analizaremos un subconjunto de las sentencias en ANSI C.

sentencia: una de *sentCompuesta sentExpresión sentSelección sentIteración sentSalto*

El axioma, sentencia, presenta cinco tipos de sentencias que analizaremos a continuación.

sentCompuesta: **{** *listaDeclaraciones*$_{op}$ *listaSentencias*$_{op}$ **}**
listaDeclaraciones: declaración
listaDeclaraciones declaración
listaSentencias: sentencia
listaSentencias sentencia

La sentencia compuesta (o bloque) queda definida por las llaves y lo que ellas encierran. Note que las declaraciones (y/o definiciones), si existen, van al comienzo de una sentencia compuesta.

* Ejercicio 63 *
(a) Una sentencia compuesta, ¿puede ser vacía? Justifique su respuesta.
(b) ¿Qué es **{** `/* Hola */` **}** ?

sentExpresión: expresión$_{op}$ **;**

Una sentencia expresión es una expresión que termina con un "punto y coma".

* Ejercicio 64 *
Investigue e informe qué es la sentencia nula.

* Ejercicio 65 *
Investigue e informe si existe la "sentencia expresión" en Pascal.

* Ejercicio 66 *
Sea la función ANSI C:

```
void XX (void) { 14; ; }
```

(a) ¿Es derivable de la BNF?;
(b) ¿Es sintácticamente correcta?
(c) ¿Cuántas sentencias componen su cuerpo?

* Ejercicio 67 *
Investigue e informe sobre la siguiente sentencia compuesta:

```
{ 22; int a; a = 14; }
```

sentSelección: **`if (`** *expresión* **`)`** *sentencia*
`if (` *expresión* **`)`** *sentencia* **`else`** *sentencia*
`switch (` *expresión* **`)`** *sentencia*

* Ejercicio 68*
Investigue e informe sobre la sentencia switch. Escriba ejemplos de su uso.

* Ejercicio 69 *
Compare la sintaxis de la sentencia if de ANSI C con la sintaxis de la sentencia if de Pascal.

sentIteración: **`while (`** *expresión* **`)`** *sentencia*
`do` *sentencia* **`while (`** *expresión* **`) ;`**
`for (` *expresión*$_{op}$ **`;`** *expresión*$_{op}$ **`;`** *expresión*$_{op}$ **`)`** *sentencia*

* Ejercicio 70 *
Investigue e informe sobre la sentencia do while. Escriba ejemplos de su uso.

* Ejercicio 71 *
(a) Investigue e informe sobre la sentencia for. Escriba ejemplos de su uso.
(b) ¿Qué significa `for(;;);` ?

sentSalto: **`return`** *expresión*$_{op}$ **`;`**

* Ejercicio 72 *
Investigue e informe sobre la sentencia return y escriba varios ejemplos de su uso.

* Ejercicio 73 *
(b) ¿Qué ocurre si una sentencia compuesta solo tiene declaraciones? Informe.

* Ejercicio 74 *
Sea la sentencia: for (; –4;) ;
(a) ¿Es derivable de la BNF de ANSI C? Justifique su respuesta.

(b) ¿Es sintácticamente correcta? Justifique su respuesta.
(c) Si lo es, describa qué hace en ejecución.

* Ejercicio 75 *
¿Qué acción realiza la sentencia do 22; while (3); ?

* Ejercicio 76 *
Sea la sentencia return a = 8;
(a) ¿Es derivable de la BNF de ANSI C? Justifique su respuesta.
(b) ¿Es sintácticamente correcta?
(c) Si lo es, ¿qué acción lleva a cabo?

3.7 OTROS USOS DE BNF

Si bien el metalenguaje BNF fue diseñado para describir la sintaxis de los LPs, también puede ser aplicado a otras áreas.

Ejemplo 34
Necesitamos describir un listado de los alumnos de la UTN FRBA en los que consten los siguientes datos: (a) nombre y apellido; (b) legajo; (c) código de carrera. Podemos hacerlo de la siguiente manera:

```
listadoUTN: alumno
            listadoUTN alumno
alumno: nombreApellido legajo códigoCarrera
nombreApellido: apellido , nombre
. . .
```

* Ejercicio 77 *
(a) Sea un archivo de texto que contiene secuencias de uno o más dígitos decimales, cada una terminada con un #. Defina su contenido en una BNF completa.
(b) ¿Cómo cambia la BNF si la secuencia de dígitos puede ser vacía?

* Ejercicio 78 *
Se diseña un LP con las palabras reservadas en castellano. Este LP tiene una única sentencia de selección que condensa los dos tipos de sentencias que aparecen habitualmente. Esta sentencia de selección comienza con la palabra reservada SI y termina con la palabra reservada FIN. Entre estos delimitadores puede haber un número indeterminado de constructos, cada uno de los cuales comienza con una expresión booleana seguida de **:** seguida de una secuencia compuesta encerrada entre "llaves". Antes de FIN, siempre habrá un constructo que comienza con la palabra reservada OTRO seguida de **:** seguida de una secuencia compuesta encerrada entre "llaves".
(a) Escriba las hipótesis que considere necesarias para definir estrictamente esta sentencia de selección, (b) Escriba varios ejemplos de esta sentencia; y c) Escriba la sintaxis en BNF de esta sentencia de selección.