13048 - LABORATORIO DE PROCESADORES DE LENGUAJE

Curso: 2005-2006

Práctica 3: Analizador léxico/sintáctico y generación de código intermedio con PCCTS

Esta práctica consta de dos partes diferenciadas. En la primera parte se trata de generar un analizador léxico, sintáctico usando la herramienta PCCTS para la gramática de la práctica 2 en la que se reconocía una simplificación del lenguaje de alto nivel MODULA-2 y, de forma opcional, se puede incluir la Tabla de Símbolos (TS) desarrollada en la práctica anterior así como las comprobaciones semánticas. La segunda parte se trata de implementar un generador de código intermedio.

Parte 1: Generación de un analizador léxico y sintáctico en PCCTS

1- Construir el analizador léxico usando las mismas especificaciones de la práctica anterior.
2- Dada la gramática de la práctica anterior, realizar lo siguiente:
   - Compactar la gramática usando la notación EBNF (en la que no aparecen producciones-ε).
   - Generar el árbol de análisis sintáctico y realizar un recorrido postfijo para visualizarlo.

Importante: Habrá que tener en cuenta la precedencia entre operadores y el cambio de signo, tanto el menos unario como el positivo, en las expresiones que trabajan con dichos operadores. Por otra parte, deberemos indicar la fila y columna de los posibles errores, tanto léxicos como sintácticos (y, opcionalmente, de las comprobaciones semánticas que incluyamos).

Consideraciones sobre la generación del árbol sintáctico

Dado que la generación del **árbol sintáctico** se usará para la generación de código, no es necesario generar la parte del árbol correspondiente a las declaraciones de variables y a las declaraciones de las funciones, sino solamente a las definiciones de las funciones, que contienen las acciones a ejecutar. En ciertos casos se podrán considerar otras simplificaciones del árbol sintáctico, teniendo en cuenta que no se altere el significado del código que representan. Por ejemplo, no será necesario crear nodos para los paréntesis en el caso de la llamada a funciones o procedimientos. De todas formas en caso de duda es preferible abstenerse de simplificaciones.

Para después poder realizar la segunda parte de la práctica, se recomienda compactar la gramática de manera que nos permita generar un árbol del estilo al que se muestra en los ejemplos siguientes (que se corresponden con los de la práctica 2). Notar que aparecen nuevos nodos que nos hemos creado adicionalmente y que se le pueden dar el nombre que se quiera.

> **NOTA IMPORTANTE:** Para simplificar la identificación del uso de procedimientos, vamos a añadir la restricción de que: "TODA LLAMADA A UN PROCEDIMIENTO/FUNCIÓN AL QUE NO SE LE PASAN ARGUMENTOS DEBE VENIR DADA, AL MENOS, POR EL NOMBRE DEL PROCEDIMIENTO/FUNCIÓN Y LOS PARÉNTESIS DE APERTURA Y CIERRE". Es decir, no podemos llamar al procedimiento solo por su nombre, por ejemplo, WriteLn, deberíamos escribir WriteLn(). Si no llevase los paréntesis, no sería considerado como un procedimiento/función sino como un identificador cualquiera.
> Esto no supone modificación en la gramática sino solo que, en los ejemplos de la practica deberemos añadir esta consideración.

Ejemplo de árbol generado para el fichero ejem1.txt

(MODULE (IMPORT WriteInt)(BEGIN (LLAMADA_PROCEDIMIENTO WriteInt 111111 5))END)

Ejemplo de árbol generado para el fichero ejem2.txt (sin tener en cuenta la consideración de los procedimientos)

(MODULE (IMPORT ReadInt WriteLn WriteInt)(BEGIN (LLAMADA_PROCEDIMIENTO ReadInt numero_1)WriteLn (LLAMADA_PROCEDIMIENTO ReadInt numero_2)WriteLn (LLAMADA_PROCEDIMIENTO WriteInt (+ numero_1 numero_2)10))END)

Ejemplo de árbol generado para el fichero ejem2.txt (teniendo en cuenta la consideración de los procedimientos- fichero ejem2b.txt)

(MODULE (IMPORT ReadInt WriteLn WriteInt)(BEGIN (LLAMADA_PROCEDIMIENTO ReadInt numero_1)(LLAMADA_PROCEDIMIENTO WriteLn)(LLAMADA_PROCEDIMIENTO ReadInt numero_2)(LLAMADA_PROCEDIMIENTO WriteLn)(LLAMADA_PROCEDIMIENTO WriteInt (+ numero_1 numero_2)10))END)

Ejemplo de árbol generado para el fichero ejem3.txt

(MODULE (IMPORT WrInteger)(PROCEDURE EscribeSumatorio (TKN_ARGUMENTOS N)(BEGIN (:= Suma 0)(FOR (:= i 1)N (DO (:= Suma (+ Suma i)))END)(LLAMADA_PROCEDIMIENTO WrInteger Suma 0)END))(BEGIN (LLAMADA_PROCEDIMIENTO EscribeSumatorio 6))END)

Ejemplo de árbol generado para el fichero ejem4.txt

(MODULE (IMPORT WrReal)(IMPORT TRUNC)(PROCEDURE Redondea (TKN_ARGUMENTOS Num)(BEGIN (:= Num (LLAMADA_FUNCION TRUNC (+ Num 0.5)))END))(BEGIN (:= x 10.7)(LLAMADA_PROCEDIMIENTO Redondea x)(LLAMADA_PROCEDIMIENTO WrReal x 5 0))END)

Ejemplo de árbol generado para el fichero ejem5b.txt

(MODULE (IMPORT ReadInt WriteLn WriteInt)(BEGIN (LLAMADA_PROCEDIMIENTO ReadInt multiplicando)(LLAMADA_PROCEDIMIENTO WriteLn)(:= multiplicador 1)(WHILE (< multiplicador 11)(DO (:= resultado (* multiplicando multiplicador))(LLAMADA_PROCEDIMIENTO WriteInt resultado 5)(LLAMADA_PROCEDIMIENTO WriteLn)(:= multiplicador (+ multiplicador 1)))END))END)

Ejemplo de árbol generado para el fichero ejem6b.txt

(MODULE (IMPORT WriteLn WriteInt)(PROCEDURE Repetir (TKN_ARGUMENTOS numero)(TKN_ARGUMENTOS veces)(BEGIN (FOR (:= k 1)veces (DO (LLAMADA_PROCEDIMIENTO WriteInt k 3)(LLAMADA_PROCEDIMIENTO WriteInt numero 5)(LLAMADA_PROCEDIMIENTO WriteLn))END)END))(BEGIN

(LLAMADA_PROCEDIMIENTO Repetir 0 1)(LLAMADA_PROCEDIMIENTO Repetir 10 10))END)

Ejemplo de árbol generado para el fichero ejem7.txt

(MODULE (IMPORT WrReal)(PROCEDURE EscribirDivCuadrados (TKN_ARGUMENTOS x y)(BEGIN (:= Cociente (/ (* x x)(* y y)))(LLAMADA_PROCEDIMIENTO WrReal Cociente PRECISION ANCHO)END))(BEGIN (:= x 1.0)(:= y 2.0)(LLAMADA_PROCEDIMIENTO EscribirDivCuadrados y x))END)

Ejemplo de árbol generado para el fichero masejem1b.txt

(MODULE (IMPORT WrReal WrLn)(IMPORT Sqrt)(PROCEDURE EscribirModulo (TKN_ARGUMENTOS Real Imag)(BEGIN (LLAMADA_PROCEDIMIENTO WrReal Real 5 0)(LLAMADA_PROCEDIMIENTO WrReal Imag 5 0)(:= Real (* Real Real))(:= Imag (* Imag Imag))(:= Modulo (LLAMADA_FUNCION Sqrt (+ Real Imag)))(LLAMADA_PROCEDIMIENTO WrReal Modulo 5 0)(LLAMADA_PROCEDIMIENTO WrLn)END))(BEGIN (:= x (UNARIO 5.0))(:= y (POSITIVO 6.0))(LLAMADA_PROCEDIMIENTO EscribirModulo x y)(LLAMADA_PROCEDIMIENTO WrReal x 5 0)(LLAMADA_PROCEDIMIENTO WrLn)(LLAMADA_PROCEDIMIENTO WrReal y 5 0)(LLAMADA_PROCEDIMIENTO WrLn))END)

Ejemplo de árbol generado para el fichero masejem2b.txt

(MODULE (IMPORT WrInt)(PROCEDURE Intercambiar (TKN_ARGUMENTOS Var1 Var2)(BEGIN (:= Temp Var1)(:= Var1 Var2)(:= Var2 Temp)END))(BEGIN (:= x 10)(:= y 20)(LLAMADA_PROCEDIMIENTO Intercambiar x y)(LLAMADA_PROCEDIMIENTO WrInt x 0)(LLAMADA_PROCEDIMIENTO WrLn)(LLAMADA_PROCEDIMIENTO WrInt y 0))END)

Ejemplo de árbol generado para el fichero masejem3b.txt

(MODULE (IMPORT WrReal)(PROCEDURE VolumenCubo (TKN_ARGUMENTOS l)(BEGIN (RETURN (* (* l l)l))END))(PROCEDURE Potencia (TKN_ARGUMENTOS x)(TKN_ARGUMENTOS n)(BEGIN (:= Pot 1.0)(FOR (:= i 1)n (DO (:= Pot (* Pot x)))END)(RETURN Pot)END))(PROCEDURE Maximo (TKN_ARGUMENTOS a b)(BEGIN (IF (> a b)(THEN (:= Max a))(ELSE (:= Max b))END)(RETURN Max)END))(BEGIN (:= x (LLAMADA_FUNCION VolumenCubo 3.25))(:= y (LLAMADA_FUNCION Potencia x 2))(:= z (LLAMADA_FUNCION Potencia x y))(LLAMADA_PROCEDIMIENTO WrReal x 5 0)(LLAMADA_PROCEDIMIENTO WrLn)(LLAMADA_PROCEDIMIENTO WrReal y 5 0)(LLAMADA_PROCEDIMIENTO WrLn)(LLAMADA_PROCEDIMIENTO WrReal z 5 0)(LLAMADA_PROCEDIMIENTO WrLn))END)

Parte 2: Generación de código intermedio en PCCTS

En este apartado se trata de generar un fichero de texto (haciendo uso de la función de generación de código que se ha de implantar) con un formato determinado y que se corresponde con la gramática y especificaciones que se comentan más adelante.

Para la generación de código nos basaremos en el árbol sintáctico que hemos generado y seguiremos las indicaciones dadas en la teoría de forma que convertiremos nuestras expresiones a otras escritas en otro lenguaje.

El código a producir deberá contemplar:

- Expresiones aritméticas.
- Sentencias de asignación.
- Estructuras de control: selectivas y repetitivas.
- Llamadas a función o procedimiento. Es recomendable probar primero que todo funciona cuando sólo existe una función en el programa y posteriormente añadir la gestión necesaria para el caso en que haya más de una función.
- Uso de las funciones importadas del MODULA-2: WrReal, WrLn, …

Gramática correspondiente al código intermedio a generar:

Números enteros o reales con o sin signo y valores booleanos NUM_TKN
Nombres de variables (identificador): ID_TKN

Podemos escribir líneas de tipo comentario que empiezan por el carácter '#'. La gramática del código a generar es:

// indicación del número de instrucciones y variables utilizadas + Lista Instruciones.

Nota: las variables temporales empezarán en el índice 0 llamándose t0, t1, … y el índice de la instrucción también comenzará en cero.

Programa ::= **N_Instr** NUM_TKN **N_Vars** NUM_TKN ListaInstrs
ListaInstr ::= Instrucción ListaInstr | ε
Instrucción ::=

ASIG_C ID_TKN NUM_TKN | *// asignación de una cte: id ← num*
ASIG_V ID_TKN ID_TKN | *// copia de una variable: id1 ← id2*
SUMA ID_TKN ID_TKN ID_TKN | *// id1 ← id2 + id3*
RESTA ID_TKN ID_TKN ID_TKN | *// id1 ← id2 - id3*
MULT ID_TKN ID_TKN ID_TKN | *// id1 ← id2 * id3*
DIV ID_TKN ID_TKN ID_TKN | *// id1 ← id2 / id3*
AND ID_TKN ID_TKN ID_TKN | *// id1 ← id2 AND id3*
OR ID_TKN ID_TKN ID_TKN | *// id1 ← id2 OR id3*
ES_POSITIVO ID_TKN | *// Le añade un signo + al id*
ES_NEGATIVO ID_TKN | *// Le añade un signo - al id*
ES_MENOR ID_TKN ID_TKN ID_TKN | *// id1← 1 si id2 < id3, 0 sino*
ES_MAYOR ID_TKN ID_TKN ID_TKN | *// id1 ← (id2 > id3)*
ES_MEN_IG ID_TKN ID_TKN ID_TKN | *// id1 ←(id2 <= id3)*
ES_MAY_IG ID_TKN ID_TKN ID_TKN | *// id1 ←(id2 >= id3)*
ES_IGUAL ID_TKN ID_TKN ID_TKN | *// id1 ←(id2 = id3)*
ES_DIFER ID_TKN ID_TKN ID_TKN | *// id1 ←(id2 != id3)*

IR_A_REL NUM_TKN | *// salto incondicional relativo (a la instrucción cuyo índice es el de la instrucción actual más el número indicado)*

IR_SI_REL ID_TKN NUM_TKN | *// salto condicional relativo: si la variable ID es verdadera (diferente de 0) se hace un salto relativo (ver IR_A_REL)*

IR_SINO_REL ID_TKN NUM_TKN | *// salto condicional relativo: si la variable ID es falsa (igual a 0) se hace un salto relativo (ver IR_A_REL)*

PUSH ID_TKN | *// apilar el valor de la variable*

POP ID_TKN | *// desapilar y guardar en la variable*

CALL NUM_TKN | *// saltar a la instrucción de índice dado para ejecutar una subrutina (debe apilarse automáticamente el índice de instrucción actual)*

RET ID_TKN ID_TKN | *// retornar de la llamada a una subrutina hasta la instrucción de índice ID1, devolviendo el valor de la variable ID2 como valor de retorno*

IMPORTAR ID_TKN | *//busca la función externa especificada en el id*

FIN *// fin de la ejecucion*

Pasos y consideraciones de este apartado:

- Generar el código intermedio al recorrer el árbol en forma postfija, llamando a la función recursiva de generación de código que deberá contemplar todos los tipos de nodos asociados a las sentencias del lenguaje. Las instrucciones se almacenarán en una lista de cuádruplos (usando como operadores los de la gramática especificada arriba). Cuando todo el árbol haya sido recorrido, se escribirá dicha lista en el fichero.
- Dado que la generación de código se realiza a partir del árbol sintáctico, se supone que el código a generar será correcto léxica y sintácticamente.
- Las variables temporales empezarán siempre por la letra "t" seguida de un número entero. Opcionalmente se puede aplicar el método de reutilización de variables temporales.
- Opcionalmente se pueden introducir las comprobaciones semánticas, tanto las que no dependen de la tabla de símbolos como añadir la construcción de dicha tabla y las comprobaciones que de ella se deriven.

Comentarios sobre las llamadas y retornos de funciones

La secuencia que debe esperarse para una llamada a una función es la siguiente:

```
ASIG_C      t0    147   // asigna la dirección de retorno 147 (la siguiente al CALL)
PUSH        t0    // apila la dirección de retorno
PUSH        t1    // apila el valor de t1 que será el primer argumento de la función
PUSH        t2    // apila el valor de t2 que será el segundo argumento de la función
CALL        X     // salta a la función que hay en la dirección X
POP         t3    // Instrucción 147: recoger el valor devuelto por la función y meterlo en t3
…
```

A partir de la dirección X tendríamos la definición de la función:

```
POP         t10   // recoge el segundo argumento y lo mete en la variable 'local' t10
POP         t11   // idem con el primer argumento
… // operaciones de la función que almacena su resultado en, digamos, la variable t20
POP         t15   // recoge de la pila la dirección de retorno (147) y la pone en t15
```

RET t15 t20 // salida de la función: el entorno de ejecución deberá primero saltar a la instrucción cuyo índice es el valor de t15 y apilar el valor de t20 como valor de retorno.

Comentarios sobre las llamadas a procedimientos

En este caso, la secuencia que debe esperarse para una llamada a un procedimiento es una simplificación de la llamada a función, ya que no debemos apilar la dirección de retorno, y es la siguiente:

```
PUSH    t0    // apila el valor de t0 que será el primer argumento del procedimiento
PUSH    t1    // apila el valor de t1 que será el segundo argumento del procedimiento
CALL    X     // salta al procedimiento que hay en la dirección X
…
```

A partir de la dirección X tendríamos la definición de la función:

```
POP     t10   // recoge el segundo argumento y lo mete en la variable 'local' t10
POP     t11   // idem con el primer argumento
… // operaciones del procedimiento…
```

Ejemplos de generación de código en estructuras de control:

Estructura selectiva.

```
….
IF a>b THEN
        Max =a;
ELSE
        MAX =b;
END;
…..

……
ES_MAYOR        t0      a       b       //evalúa la condición del IF (en este caso si a>b) y
guarda el resultado en una vble temporal,  un 1 si es cierto y 0 si no
IR_SINO_REL     t0      +2      //si el resultado de la condición es falso (t0 = 0), saltará hasta la
primera instrucción del ELSE, si no, continua.
#Ahora hacemos las operaciones de dentro del THEN
ASIG_V          Max     a       //Copia el valor de a en la variable Max
IR_A_REL        +2              //Al terminar las instrucciones del THEN debemos saltarnos las del
ELSE. Saltamos 2 instrucciones hacia delante para continuar después del else
###en este caso hay ELSE por lo que pondremos el código de las instrucciones del ELSE
ASIG_V          Max     b       // Copia el valor de b en la variable Max
…..
```

Estructura repetitiva: FOR.

```
….
FOR i := 1 TO N DO
        Suma := Suma + i;
END;
….
```

```
…
ASIG_C      t0      1       //guardamos el numero en variable temporal
ASIG_V      i       t0      //inicializamos el contador
ASIG_C      t1      1       //usamos una variable auxiliar para el incremento de 1 en 1
###hacemos la comprobación del bucle
ES_MAYOR    t2      i       N       //si i>N guardamos en t2 un 1, si no, un 0
IR_SI_REL   t2      +6      //si t2 es verdadera, salta 6 instrucciones más abajo (no debe entrar en el buble), si no, continua
#Ahora hacemos las operaciones de dentro del bucle
SUMA        t3      Suma    i       //Guarda en t3 el valor de la suma de variables
ASIG_V      Suma    t3              //Copia el valor de t3 en la variable Suma
#finalmente incrementamos el contador
###El incremento en los bucles FOR siempre va de 1 en 1
SUMA        t4      i       t1      //incrementa en t1 el valor de la i (a traves de t4)
ASIG_V      i       t4              //Copia el valor de t4 en la i

IR_A_REL    -6              //Saltamos 6 instrucciones hacia atrás para comprobar el bucle
…..
```

Estructura repetitiva: WHILE.

```
….
WHILE i > 1 DO
        i := i - 1;
END;
….

…
ASIG_C      t0      1       //guardamos el numero en variable temporal
###hacemos la comprobación del bucle
ES_MAYOR    t1      i       t0      //si i>t0 guardamos en t1 un 1, si no, un 0
IR_SINO_REL t1      +4      //si t1 es falso, salta 5 instrucciones más abajo (no debe entrar en el buble), si no, continua
#Ahora hacemos las operaciones de dentro del bucle
ASIG_C      t2      1       //guardamos el numero en variable temporal
RESTA       t3      i       t2      //Resta a i el valor de t2 (a través de t3)
ASIG_V      i       t3      /asigna a i el valor de t3l
#después de ejecutar las instrucciones volvemos para ver si debe volver a entrar en el bucle
IR_A_REL    -5              //Saltamos 5 instrucciones hacia atrás para comprobar el bucle
…..
```

Estructura repetitiva: REPEAT.

```
….
REPEAT
        i := i + 1;
UNTIL i > 0;
….

…
#### primero se ejecutan las instrucciones del bule
ASIG_C      t0      1       //guardamos el numero en variable temporal
SUMA        t1      i       t0      //Resta a i el valor de t0, guarda el resultado en t1
ASIG_V      i       t1      //guardamos el numero en variable temporal
###hacemos la comprobación del bucle
ASIG_C      t2      0       //guardamos el numero en variable temporal
ES_MAYOR    t3      i       t2      //si i>t2 guardamos en t3 un 1, si no, un 0
```

IR_SINO_REL t3 -4 *//si t2 es falso, salta 4 instrucciones hacia arriba (debe volver a ejecutar las instrucciones del bucle), si no, continua*
…..

Ejemplos de ficheros de salida de código intermedio correspondientes a los ejemplos de la práctica 2:

```
MODULE primero;
FROM InOut IMPORT WriteInt;
BEGIN
   WriteInt ( 111111, 5 );
END primero.

#
###El N_Intstr hace referencia a las líneas de código intermedio generadas sin contar los comentarios
N_Instr   9
#
### El N_Vars es la suma del número de variables declaradas en el código fuente más el número de
### variables temporales que se han debido generar para la creación de código intermedio
N_Vars   2
#
### llamar al Procedimiento WriteInt con dos argumentos (dos números)
#
ASIG_C      t0      111111 // guardamos el numero en una variable temporal
PUSH        t0                  // apilar el primer argumento
ASIG_C      t1      5           // guardamos el numero en una variable temporal
PUSH        t1                  // apilar el segundo argumento
CALL        8                   // llamar al procedimiento WriteInt que está en la dirección 8
### Fin del programa
FIN                             // fin de la ejecucion
#
# FUNCION IMPORTADA WriteInt: la primera instrucción tiene el índice 8
#
IMPORTAR    WriteInt            // importa la función de la librería desafilando argumentos si los
                                // tiene y devolviendo la dirección de retorno si fuese necesario


MODULE suma_dos_numeros;
FROM InOut IMPORT ReadInt, WriteLn, WriteInt;
VAR
   numero_1, numero_2: INTEGER;

BEGIN
   ReadInt(numero_1);
   WriteLn();
   ReadInt(numero_2);
   WriteLn();
   WriteInt( numero_1+numero_2, 10 );
END suma_dos_numeros.

N_Instr   17
N_Vars   7
#
### llamar al Procedimiento ReadInt con un argumento (la variable numero_1)
#
PUSH        numero_1            // apilar el argumento
```

```
CALL        14                  // llamar al procedimiento ReadInt que está en la dirección 16
### llamar al Procedimiento WriteLn sin argumentos
CALL        15          // llamar al procedimiento WriteLn que está en la dirección 15
### llamar al Procedimiento ReadInt con un argumento (la variable numero_2)
#
PUSH        numero_2            // apilar el argumento
CALL        14                  // llamar al procedimiento ReadInt que está en la dirección 14
### llamar al Procedimiento WriteLn sin argumentos
CALL        15          // llamar al procedimiento WriteLn que está en la dirección 15
### llamar al Procedimiento WriteInt con dos argumentos (una operación y un número)
#
SUMA        t0      numero_1    numero_2    // sumar los elementos y los guarda en t0
PUSH        t0                  // apilar el primer argumento
ASIG_C      t1      10          // guardamos el numero en una variable temporal
PUSH        t1                  // apilar el segundo argumento
CALL        16                  // llamar al procedimiento WriteInt que está en la dirección 16
### Fin del programa
FIN                             // fin de la ejecucion
#
# FUNCION IMPORTADA ReadInt: la primera instrucción tiene el índice 14
#
IMPORTAR    ReadInt                 // importa la función de la librería
# FUNCION IMPORTADA WriteLn: la primera instrucción tiene el índice 15
#
IMPORTAR    WriteLn             // importa la función de la librería
# FUNCION IMPORTADA WriteInt: la primera instrucción tiene el índice 16
#
IMPORTAR    WriteInt            // importa la función de la librería
```

```
MODULE Ejemplo1;
IMPORT WrInteger;
PROCEDURE EscribeSumatorio ( N : INTEGER );
VAR Suma, i : INTEGER;
BEGIN
      Suma := 0;
      FOR i := 1 TO N DO
            Suma := Suma + i;
      END;
      WrInteger (Suma, 0);
END EscribeSumatorio;
BEGIN
      EscribeSumatorio ( 6 );
END Ejemplo1.
```

```
N_Instr  25
N_Vars  12
#
### llamar al Procedimiento EscribeSumatorio con un argumento (un número)
#
ASIG_C      t7      6       // guardamos el número en una variable temporal
PUSH        t7              // apilar el primer argumento
CALL        7               // llamar al procedimiento EscribeSumatorio, está en la dirección 7
FIN                         // fin de la ejecucion
#
# FUNCION IMPORTADA WrInteger: la primera instrucción tiene el índice 6
#
IMPORTAR    WrInteger           // importa la función de la librería
#
# PROCEDIMIENTO EscribeSumatorio: la primera instrucción tiene el índice 7
#
```

```
POP             N               // desapilar el argumento y lo guarda en la variable N
ASIG_C          t0      0       // guardamos el numero 0 en una variable temporal
ASIG_V          Suma    t0      // copia en la variable Suma el valor de t0
#
###Instrucciones para el bucle FOR
###empezamos por los cúadruplos de la instrucción i := 1
ASIG_C          t1      1       //guardamos el numero en variable temporal
ASIG_V          i       t1      //inicializamos el contador
###El incremento en los bucles FOR siempre va de 1 en 1
ASIG_C          t2      1       //usamos una variable auxiliar para el incremento de 1 en 1
###hacemos la comprobación del bucle
ES_MAYOR        t3      i       N       //si i>N guardamos en t3 un 1, si no, un 0
IR_SI_REL       t3      +5       //si t3 es verdadera, salta 5 instrucciones más abajo, si no, continua
#Ahora hacemos las operaciones de dentro del bucle
SUMA            t4      Suma    i       //Guarda en t4 el valor de la suma de variables
ASIG_V          Suma    t4              //Copia el valor de t4 en la variable Suma
#finalmente incrementamos el contador
SUMA            t5      i       t2      //incrementa en t2 el valor de la t5
ASIG_V          i       t5      //copia el valor de t5 en la variable i
IR_A_REL        -5              //Saltamos 5 instrucciones hacia atrás para comprobar el bucle
#Aqui ya se ha salido del bucle
### llamar al Procedimiento WrInteger con dos argumentos (una variable y un número)
#
PUSH            Suma            // apilar el primer argumento
ASIG_C          t6      0       // guardamos el numero en una variable temporal
PUSH            t6              // apilar el segundo argumento
CALL            6               // llamar al procedimiento WrInteger, está en la dirección 6
# como EscribeSumatorio es procedimiento no hay que devolver resultado
FIN                             //fin de la ejecución del procedimiento
```

```
MODULE tabla_mult;

FROM InOut IMPORT ReadInt, WriteLn,WriteInt;

VAR
   multiplicando, multiplicador, resultado: INTEGER;

BEGIN
   ReadInt( multiplicando );
   WriteLn;
   multiplicador:=1;
   WHILE multiplicador < 11 DO
       resultado:=multiplicando * multiplicador;

       WriteInt( resultado, 5 );
       WriteLn;

       multiplicador:=multiplicador+1;
   END;
END tabla_mult.
```

```
N_Instr   25
N_Vars   13
#
### llamar al Procedimiento ReadInt con un argumento (la variable multiplicando)
#
PUSH            multiplicando           // apilar el argumento
CALL            22                      // llamar al procedimiento ReadInt que está en la dirección 22
### llamar al Procedimiento WriteLn sin argumentos
```

```
CALL        23              // llamar al procedimiento WriteLn que está en la dirección 23
ASIG_C      t0      1       //guarda el valor 1 en una variable temporal
ASIG_V      multiplicando   t0      //copia el calor de t0 en la variable multiplicando
#
###Empieza el bucle WHILE
###generamos el código de "la condición del bucle" que es multiplicador < 11
ASIG_C      t1      11      //guardamos el número en la variable temporal
ES_MENOR    t2      multiplicador   t1      //comprobamos la condición y guardamos 1 o 0 en t2
IR_SINO_REL t2      +12     //si t2 es falsa, vamos 12 instrucciones más abajo que es el final del bucle
#
###hacemos las operaciones del while
MULT        t3      multiplicando   multiplicador   //multiplico las variables y lo guardo en t3
ASIG_V      resultado       t3      //copio la variable t3 en la variable resultado
### llamar al Procedimiento WriteInt con dos argumentos (una variable y un número)
PUSH        resultado       // apilar el primer argumento
ASIG_C      t4      5       //guardo el 5 en una variable temporal
PUSH        t4              // apilar el segundo argumento almacenado en t4
CALL        24              // llamar al procedimiento WriteInt que está en la dirección 24
### llamar al Procedimiento WriteLn sin argumentos
CALL        23              // llamar al procedimiento WriteLn que está en la dirección 23
#hacemos la operación del igual multiplicador:=multiplicador+1
ASIG_C      t5      1       //guardo el 1 en una variable temporal
SUMA        t6      multiplicador   t5      //sumo las dos variables y lo almaceno en t6
ASIG_V      multiplicador   t6      //copio la variable t6 en multiplicador
IR_REL      -12     //retrocedo al principio del bucle, es decir, voy 12 instrucciones atrás
###termina el bucle
### Fin del programa
FIN             // fin de la ejecucion
#
# FUNCION IMPORTADA ReadInt: la primera instrucción tiene el índice 22
#
IMPORTAR    ReadInt             // importa la función de la librería
# FUNCION IMPORTADA WriteLn: la primera instrucción tiene el índice 23
#
IMPORTAR    WriteLn             // importa la función de la librería
# FUNCION IMPORTADA WriteInt: la primera instrucción tiene el índice 24
#
IMPORTAR    WriteInt            // importa la función de la librería
```

```
MODULE Matematicas;
FROM IO IMPORT WrReal, WrLn;
PROCEDURE VolumenCubo (l : REAL) : REAL;
BEGIN
      RETURN (l*l*l);
END VolumenCubo;
PROCEDURE Potencia (x : REAL; n : INTEGER) : REAL;
VAR i : INTEGER;
Pot : REAL;
BEGIN
      Pot := 1.0;
      FOR i := 1 TO n DO
            Pot := Pot * x;
      END;
      RETURN Pot;
END Potencia;
PROCEDURE Maximo (a, b : REAL) : REAL;
VAR Max : REAL;
      BEGIN
            IF (a>b) THEN
                  Max := a;
            ELSE
```

```
                    Max := b;
      END;
      RETURN Max;
END Maximo;
VAR x, y , z: REAL;
BEGIN
      x := VolumenCubo (3.25);
      y := Potencia( x , 2 );
      z := Potencia( x , y );
      WrReal (x, 5, 0); WrLn();
      WrReal (y, 5, 0); WrLn();
      WrReal (z, 5, 0); WrLn();
END Matematicas.
```

```
N_Instr  80
N_Vars  39
#
### llamar a la Función VolumenCubo con un argumento (un número)
ASIG_C      t12     7       //asigna la dirección de retorno 7 (la siguiente al CALL)
PUSH        t12             //apila la dirección de retorno
ASIG_C      t13     3.25    // guardamos el número en una variable temporal
PUSH        t13             // apilar el primer argumento
CALL        48              // llamar a la Función VolumenCubo, está en la dirección 48
POP         t14     //Desapilar el resultado de la función y guardarlo en t14 (esta es la instrucción 7)
### cuádruplo correspondiente a la instrucción =
ASIG_V      x       t14             // copia en la variable x el valor de t14
#
### llamar a la Función Potencia con dos argumentos (una variable y un número)
ASIG_C      t15     15      //asigna la dirección de retorno 15 (la siguiente al CALL)
PUSH        t15             //apila la dirección de retorno
PUSH        x               // apilar el primer argumento
ASIG_C      t16     2       // guardamos el número 2 en una variable temporal
PUSH        t16             // apilar el segundo argumento
CALL        54              // llamar al procedimiento Potencia, está en la dirección 54
POP         t17     //Desapilar el resultado de la función y guardarlo en t17 (esta es la instrucción 15)
ASIG_V      y       t17             // copia en la variable y el valor de t17
#
### llamar a la Función Potencia con dos argumentos (dos variables)
ASIG_C      t18     22      //asigna la dirección de retorno 22 (la siguiente al CALL)
PUSH        t18             //apila la dirección de retorno
PUSH        x               // apilar el primer argumento
PUSH        y               // apilar el segundo argumento
CALL        54              // llamar al procedimiento Potencia, está en la dirección 54
POP         t19     //Desapilar el resultado de la función y guardarlo en t19 (esta es la instrucción 22)
ASIG_V      z       t19             // copia en la variable z el valor de t19
#
### llamar al Procedimiento WrReal con tres argumentos (una variable y dos números)
PUSH        x               // apilar el primer argumento
ASIG_C      t20     5       //guardo el 5 en una variable temporal
PUSH        t20             // apilar el segundo argumento almacenado en t20
ASIG_C      t21     0       //guardo el 0 en una variable temporal
PUSH        t21             // apilar el tercer argumento almacenado en t21
CALL        46              // llamar al procedimiento WrReal que está en la dirección 46
### llamar al Procedimiento WrLn sin argumentos
CALL        47              // llamar al procedimiento WrLn que está en la dirección 47
### llamar al Procedimiento WrReal con tres argumentos (una variable y dos números)
PUSH        y               // apilar el primer argumento
ASIG_C      t22     5       //guardo el 5 en una variable temporal
PUSH        t22             // apilar el segundo argumento almacenado en t22
ASIG_C      t23     0       //guardo el 0 en una variable temporal
```

```
PUSH        t23             // apilar el tercer argumento almacenado en t23
CALL        46              // llamar al procedimiento WrReal que está en la dirección 46
### llamar al Procedimiento WrLn sin argumentos
CALL        47              // llamar al procedimiento WrLn que está en la dirección 47
### llamar al Procedimiento WrReal con tres argumentos (una variable y dos números)
PUSH        z               // apilar el primer argumento
ASIG_C      t24     5       //guardo el 5 en una variable temporal
PUSH        t24             // apilar el segundo argumento almacenado en t24
ASIG_C      t25     0       //guardo el 0 en una variable temporal
PUSH        t25             // apilar el tercer argumento almacenado en t25
CALL        46              // llamar al procedimiento WrReal que está en la dirección 46
### llamar al Procedimiento WrLn sin argumentos
CALL        47              // llamar al procedimiento WrLn que está en la dirección 47
FIN                         // fin de la ejecución y del programa
#
# FUNCION IMPORTADA WrReal: la primera instrucción tiene el índice 46
IMPORTAR    WrReal          // importa la función de la librería
#
# FUNCION IMPORTADA WrLn: la primera instrucción tiene el índice 47
IMPORTAR    WrLn            // importa la función de la librería
#
#### Función VolumenCubo: la primera instrucción tiene el índice 48
#
POP     l               //desapila el argumento y lo guarda en la variable l
MULT    t0      l       l       //realiza la operación l*l y la guarda en t0
MULT    t1      t0      l       //realiza la operación t11*l y la guarda en t1
POP     t2                      //desapila la dirección de retorno y la guarda en t2
RET     t2      t1              //vuelve a la dirección especificada en t2 apilando el resultado t1
FIN
#
#### Función Potencia: la primera instrucción tiene el índice 54
#
POP     n               //desapila el 2º argumento y lo guarda en la variable n
POP     x               //desapila el 1º argumento y lo guarda en la variable x
ASIG_C      t3      1.0     //guarda en t3 el número 1.0
ASIG_V      Pot     t3      //copia en Pot el valor de t3
#realizamos el bucle FOR
###empezamos por los cúadruplos de la instrucción i := 1
ASIG_C      t4      1       //guardamos el numero 1 en variable temporal
ASIG_V      i       t4      //inicializamos el contador copiando la variable t4 en i
###El incremento en los bucles FOR siempre va de 1 en 1
ASIG_C      t5      1       //usamos una variable auxiliar para el incremento de 1 en 1
###hacemos la comprobación del bucle
ES_MAYOR    t6      i       n       //si i>n guardamos en t6 un 1, si no, un 0
IR_SI_REL   t6      +5       //si t6 es verdadera, salta 5 instrucciones más abajo, si no, continua
#Ahora hacemos las operaciones de dentro del bucle
MULT        t7      Pot     x       //Guarda en t7 el valor de la multiplicación de las variables
ASIG_V      Pot     t7              //Copia el valor de t7 en la variable Pot
#finalmente incrementamos el contador
SUMA        t8      i       t5      //incrementa en t5 el valor de la i
ASIG_V      i       t8              //Copia el valor de t8 en la variable i
IR_A_REL    -5              //Saltamos 5 instrucciones hacia atrás para comprobar el bucle
#Aqui ya se ha salido del bucle
POP     t9                      //desapila la dirección de retorno y la guarda en t9
RET     t9      Pot             //vuelve a la dirección especificada en t9 apilando el resultado Pot
FIN
#
```

```
#### Función Máximo: la primera instrucción tiene el índice 71
#
POP     b               //desapila el 2º argumento y lo guarda en la variable b
POP     a               //desapila el 1º argumento y lo guarda en la variable a
#Vamos a las instrucciones del IF-ELSE
###empezamos por el IF y su condición de salto
###como son variables, hacemos la comprobación directamente (no hay instrucciones previas)
ES_MAYOR        t10     a       b       //si a>b guardamos en t10 un 1, si no, un 0
IR_SINO_REL     t10   +2        //si t9 no es verdadera, salta 2 instrucciones más abajo, si no, continua
#Ahora hacemos las operaciones de dentro del IF
ASIG_V          Max     a       //Copia el valor de a en la variable Max
IR_A_REL        +2              //Saltamos 2 instrucciones hacia delante para continuar después del else
###en este caso hay ELSE por lo que pondremos el código de las instrucciones del ELSE
ASIG_V          Max     b       // Copia el valor de b en la variable Max
###si hubiesen más instrucciones en la función se pondrían aquí, pero en el ejemplo no las hay
POP     t11                     //desapila la dirección de retorno y la guarda en t11
RET     t11     Max             //vuelve a la dirección especificada en t10 apilando el resultado Max
FIN
```

Cosas a tener en cuenta en la práctica:

1.- Se deberá compactar la gramática de forma que quede en notación EBNF (que es con la que trabaja el PCCTS).
2.- Se deberá indicar la fila y columna de los posibles errores tanto léxicos como sintácticos.
3.- Se deberá poder visualizar el árbol sintáctico generado.
4.- El código intermedio generado deberá escribirse en un fichero.
5.- Opcionalmente se puede incluir la tabla de símbolos y las comprobaciones semánticas de la práctica anterior.
6.- PUNTUACION:
    a) Análisis léxico (1punto).
    b) Compactación de la gramática y análisis sintáctico (2 punto).
    b) Creación correcta del árbol (1.5 puntos).
    c) Generación de código intermedio (5.5 puntos).

Fecha de entrega:

Todos los grupos deberán entregar la práctica con fecha máxima el 5 de Junio.

DURACIÓN: 3 sesiones