Trabajo Práctico Nº 4

Tema: ESTRUCTURA DEL COMPILADOR

1. Identifíquense los lexemas que forman los componentes léxicos en los siguientes programas. Poner valores razonables de atributos para los componentes léxicos:

   a) Pascal

```
function max(i, j: integer) :integer;
  {devuelve el maximo de los enteros i y j }
begin
    if  i > j then  max :=  i
            else   max := j
end;
```

   b) C

```
int max (i, j) int i, j;
  /* devuelve el maximo de los enteros i y j */
{     return  i>j?i:j;
 }
```

2. Escriba un scanner en Pascal o C para los componentes léxicos de la siguiente tabla:

Componente léxico	token	Valor atributo
if	pr	1
then	pr	2
else	pr	3
id	id	Nº posición TS
num	num	Nº ordinal
<	oprel	1
>	oprel	2
>=	oprel	3
<=	oprel	4
=	oprel	5
<>	oprel	6
:=	asig	

Las palabras reservadas se escriben en mayúsculas, minúsculas o combinación de estas. Los identificadores tienen una longitud máxima de 5 caracteres y comienzan con una letra. Los números son enteros.
A medida que el scanner reconoce un lexema se visualizará por pantalla: lexema, token y atributo.

3. Modifique el programa de (2) para que el scanner detecte e informe los errores léxicos (indicar número de línea)

4. Modifique el programa de (3) para que acepte comentarios (cadenas de caracteres delimitadas por {}) y no los tome como errores.

5. Escribir en C un programa que se ejecute: *Lex Archivo lexema* y devuelva los números de líneas de Archivo donde esta lexema (Archivo es un fichero de texto y lexema es una cadena de caracteres)

6. Dada la siguiente gramática para expresiones:

Trabajo Práctico Nº 4

Tema: ESTRUCTURA DEL COMPILADOR

E ::== E + T | E - T | T
T ::== T * F | T div F | F
F ::== (E) | numero

Para cada una de las siguientes cadenas dibuje el árbol de análisis sintáctico:

a) 2 + 3
b) (2 + 3)
c) 2 + 3 * 5
d) (2 + 3) * 5
e) 2 + (3 * 5)

7. Escriba gramáticas para describir la sintaxis de cada una de las siguientes casos:
   a) Secuencias de letras o dígitos, comenzando con una letra
   b) Numero reales tanto la parte entera como la fraccionaria pueden estar vacías pero no ambas

8. Completar la siguiente gramática en BNFE para expresiones aritméticas:

   EXPRESIÓN ::== TERMINO {(+ | -) TERMINO}
   TERMINO ::== FACTOR {(* | div) FACTOR}
   FACTOR ::== '(' EXPRESION')' | VARIABLE | CONSTANTE

9. Proponga una gramática para las siguientes sentencias en Pascal
   a) For
   b) Repeat
   c) Case

10. Dada la siguiente gramática simple
    EXPRESIÓN ::== DIGITO {[(+ | -)] DIGITO}
    DIGITO ::== 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Donde las expresiones: 4+5-7; 3-1+8-9 son expresiones válidas y 4+-7;33-8; p+5 y -9 no lo son.

Escribir un parser en Pascal o C que compruebe la correctitud sintáctica de estas cadenas.

11. En el esquema de traducción dirigido por sintáxis:

Producción	*Reglas semánticas*
$P \rightarrow D; E$	
$D \rightarrow D; D$	
$D \rightarrow id : T$	*{ añadetipo (id.entrada, T.tipo)}*
$T \rightarrow$ char	*{ T.tipo := char}*
$T \rightarrow$ integer	*{ T.tipo := integer}*
$T \rightarrow \uparrow T_1$	*{ T.tipo := pointer (T_1.tipo)}*
$T \rightarrow$ array [num] of T_1	*{ T.tipo := array (1..num.val, T_1.tipo)}*

a) Agregar el tipo conjunto (set) del lenguaje Pascal.
b) Definir las expresiones de tipo para las operaciones entre conjuntos: union, intersección y diferencia.
c) Definir las expresiones de tipo para el operador "in" utilizado para comprobar la pertenencia.

Trabajo Práctico Nº 4

Tema: ESTRUCTURA DEL COMPILADOR

12. Escribir las reglas de comprobación de tipos para la sentencia Case del lenguaje Pascal.

13. Traducir las siguientes expresiones aritméticas a:
    a) un árbol sintáctico
    b) código de tres direcciones

13.1 a * - (b+c) a

13.2 –(a+b) * (c+d) + (a+b+c) a

14. Dada la siguiente proposición:

```
while  a <  b  do
   if  c  <  d  then
                x := y + z
            else
             x := y – z
```

Expresar en código de tres direcciones.
¿Cuáles son las reglas para generar el código intermedio de las proposiciones while e if?

15. Genere código objeto para las siguientes proposiciones en C:
    a) x = 1
    b) x = y
    c) x = x + 1
    d) x = a + b * c
    e) x = a / (b+c) d * (e+f)

16. Las siguientes secuencias de código de tres direcciones corresponden a la misma expresión: (a + b) – (e – (c + d)) y hacen lo mismo:

a)	*b)*
$t_1 := a + b$	$t_2 := c + d$
$t_2 := c + d$	$t_3 := e – t_2$
$t_3 := e – t_2$	$t_1 := a + b$
$t_4 := t_1 – t_3$	$t_4 := t_1 – t_3$

Sin embargo, ¿producen el mismo código objeto?

17. Investigue
    a) El lenguaje de programación Java, es compilado o interpretado?
    b) Cómo Java logra ser multiplataforma?