Universidad de Málaga
Departamento de Lenguajes y
Ciencias de la Computación
Campus de Teatinos, 29071 MÁLAGA

TRADUCTORES, COMPILADORES E INTÉRPRETES.
EJERCICIOS TEMA 5, 6, 7 y 8.

1.) (HOLUB). Escribir un traductor que reconozca definiciones de C siguiendo el estilo antiguo, tal como:

```
prueba(uno, dos, tres, cuatro, cinco)
char *uno;
long dos;
double tres;
{
}
```

y las traduzca a definiciones del estilo C++, como

```
prueba(char *uno, long dos, double tres, int cuatro, int cinco)
{
}
```

Nótese que las variables a las que no se les ha indicado tipo, se suponen de tipo entero.

2.) Suponer un lenguaje que nos permitirá declarar variables de tipo subrango de enteros, de la forma *var1, var2 : [límite1..límite2];*. Además permite efectuar asignaciones a las variables previamente declaradas. Hacer un intérprete que controle lo siguiente:

- No se puede declarar una variable dos o más veces.
- Sólo se pueden hacer asignaciones a variables previamente declaradas.
- Toda asignación a una variable debe controlar que el valor entero que se le asigne esté entre los límites del rango con que se declaró.
- Las operaciones deberán ser: *, /, + y -.

3.) Es una buena intención el permitir al programador que exprese condiciones complejas como (a < b) AND (b < c), de una forma mucho más intuitiva, tal como a < b < c. Hacer un esquema de traducción con LEX/YACC que permita condiciones como la propuesta con objeto de generar código de tercetos.

4.) Según el esquema de traducción de LEX/YACC visto en clase para estructuras complejas de control, indicar el código de tercetos que se obtendría como resultado de traducir el siguiente trozo de código:

```
SUMA := OPERANDO1;
WHILE OPERANDO2 > 0 DO
        SUMA := SUMA + 1;
FIN WHILE;
```

5.) Suponiendo que un puntero se almacena en 4 bytes, un entero en 2 bytes, un real en 5 bytes, y un lógico en un byte, modificar adecuadamente el esquema LEX/YACC visto en clase sobre sistemas de tipos complejos, de manera que para cada identificador, junto con su tipo, se almacene el tamaño que ocupa. Ignorar la existencia del tipo función, y considerar que un array implica el almacenamiento repetido del tipo a que referencia.

6.) En el esquema visto en clase sobre estructuras complejas de control, se generan numerosas variables temporales. Modificar adecuadamente las acciones para que se reutilicen las

variables que ya no se van a necesitar más.

P.ej., el siguiente código:

```
IF A > B THEN
  {
        A := B + A;
        B := A - B;
        A := A - B;
  };
FIN IF;
```

que en condiciones normales se traduciría a como indica la columna 1:

```
        if A > B goto etq1
        goto etq2
label etq1
        tmp1 = B + A;
        A = tmp1
        tmp2 = A - B;
        B = tmp2
        tmp3 = A - B;
        A = tmp3
        goto etq3
label etq2
label etq3
```

```
        if A > B goto etq1
        goto etq2
label etq1
        tmp1 = B + A;
        A = tmp1
        tmp1 = A - B;
        B = tmp1
        tmp1 = A - B;
        A = tmp1
        goto etq3
label etq2
label etq3
```

debería traducirse a como indica la columna 2.

Por cierto, ¿qué hace el trozo de código indicado como ejemplo?.

7.) (Examen Diciembre del 92) Construir en LEX/YACC una calculadora que realice las siguientes operaciones:

1.- Suma, resta, multiplicación, división entera, y menos unario.

2.- Las constantes son enteros, y pueden introducirse en:

- Formato decimal. Ej.: 85, 85d, 85D.
- Formato octal. Ej.: 76o, 76O.
- Formato hexadecimal. Ej.: a8fh, A8FH.
- Formato binario. Ej.: 011100b, 011100B.

3.- El resultado se puede obtener también en cualquiera de estos formatos según lo requiera el usuario. Si al principio de la línea aparece la cadena:

- D. ó d.: El resultado se visualizará en decimal.
- H. ó h.: El resultado se visualizará en hexadecimal.
- O. u o.: El resultado se visualizará en octal.
- B. ó b.: El resultado se visualizará en binario.
- Por defecto se visualizará en decimal.

4.- La calculadora supone que cada expresión acaba con un retorno de carro.

5.- El último cálculo obtenido se guardará en la variable u (que inicialmente valdrá 0). Por tanto, ese último cálculo se puede usar en la siguiente expresión.

Hacer el programa LEX/YACC que se comporte de la forma indicada.

Ej.:

```
35+2*10
Igual a 55
```

4/3	B.35+1010b*2o	D.35+2*10
Igual a 1	IGual a 110111B	Igual a 55
u*75	B.u	H.u
Igual a 75	IGual 110111B	Igual a 37H
	D.35+ah*2	
	Igual a 55	

8.) (Examen Junio del 92). Se considera el lenguaje L, cuyo alfabeto es {a, b, c, ...z, <, -, 0, 1, 2, ..., 9, .}. Los nombres de las variables de este lenguaje empiezan por una letra minúscula seguida posiblemente por otras letras o dígitos. Las constantes pueden ser una secuencia de dígitos decimales o una secuencia de dígitos hexadecimales (0, 1, 2, 3..., 9, a, b, ..., f), terminada en h. Las sentencias de este lenguaje son sentencias de asignación (posiblemente múltiple), que usan el operador de asignación <-. Todas las sentencias terminan con punto (.).

Ej. válidos:

a<-3.
a<-b1d<-4bh.
a1<-bc3<-a.
4.
a.

Ej. no válidos:

1a<-3.
a<-3
A<-4.
a<-1g2h.

Se pide:

1.- Construir una gramática para L.

2.- Construir un programa LEX que realice el análisis lexicográfico.

3.- Construir un programa YACC para hacer el análisis sintáctico.. Este programa debe conseguir que en la tabla de símbolos, al final del proceso de compilación, las variables contengan su valor correspondiente, según las asignaciones.

Pista: Como tabla de símbolos se puede emplear una tabla estática de MAX_SIMB elementos, cada uno de los cuales es de tipo SIMB, que se define como:

```
typedef struct simb
{
        char * nombre;
        long valor;
} SIMB;
```

Para el tipo de yylval, se puede definir en YACC la siguiente unión:

```
%union
{
        short indice;
        long valor;
}
```

9.) (Examen Diciembre del 93). Construir un programa LEX/YACC para calcular el valor de números binarios que poseen parte entera y parte decimal. El programa debe aceptar un número binario e imprimir su valor real por pantalla. Ej.:

101.10	5.5
111	7

11.101 3.625
110. 6.

Aunque este problema podría ser resulto completamente en LEX, se pide que sea resuelto mediante un análisis ascendente (implementado con YACC). Esto implica que existirá un token que llamaremos DÍGITO, responsable de detectar un dígito binario.

10.) (Examen Diciembre del 94) Construir un programa LEX/YACC que pase una descripción sintáctica BNF a una definición regular. La yuxtaposición en la notación BNF se notará mediante un ".", lo cual facilitará la implementación. Ej.:

letra ::= 'a' | 'b' | 'c' | 'A' | 'B' | 'Z'
dígito ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
identificador ::= letra.{letra|dígito}
sentencia ::= "una sentencia"
condición ::= "una condición"
sentif ::= "if".condición."then".sent. ["else".sent]
desc ::= "algo". {"varios"} | "todos" | ["esto".{ninguno"}]

se deberá traducir a:

letra ---> 'a' | 'b' | 'c' | 'A' | 'B' | 'Z'
dígito ---> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
identificador ---> letra (letra | dígito)*
sentencia ---> "una sentencia"
condición ---> "una condición"
sentif ---> "if" condición "then" sent (tira_nula | "else" sent)
desc ---> "algo" ("varios")* | "todos" | (tira_nula | "esto" (ninguno")*)

Comentarios:

1.- El programa debe ser capaz de aceptar varias líneas, tal como se muestra en el ejemplo.
2.- Un nombre válido para una metanoción (la cadena que aparece delante del operador ***::=***), consiste en una secuencia de letras y/o dígitos, que empieza por una letra.
3.- Ejemplos válidos para un terminal (algo que no sepuede describir más), son los siguientes: 0, 8, "hola", 'a', etc..
4.- Las metanociones se deben instalar en una tabla de símbolos, ue vendrá dada por una tabla estática de elementos de tipo SIMBOL.
5.- La tira nula (también llamada épsilon, o denotada por ε), se representará con la cadena tira_nula.

11.) Sea la siguiente gramática, que permite definir textualmente las fórmulas que aparecerán gráficamente en un procesador de textos:

```
formula -->     NUM
        |       ID
        |       formula OP formula
        |       ENTRE formula
        |       formula PARTIDO formula
        |       MATRIZ lista_listas '.'
        ;
lista_listas --> lista
        |       lista_listas ',' lista
        ;
```

lista --> formula
| lista formula
;

Se pretende hacer un programa LEX/YACC que para una formula nos diga sus dimensiones de alto y ancho (en puntos), sabiendo que:

- Cualquier carácter tiene 3 puntos de alto y 2 puntos de ancho.
- ENTRE hace que la altura de la fórmula se incremente en 2 puntos, y su anchura en 4.
- PARTIDO obtiene una fórmula cuya anchura es 2 puntos más que la máxima anchura de las dos fórmulas que intervienen, y cuya altura es 3 más la suma de las alturas de dichas fórmulas.
- MATRIZ tiene a continuación una lista de listas de fórmulas. Cada lista tendrá de ancho la suma de los anchos de las fórmulas que la integran, más 1 punto entre cada 2 fórmulas; y un alto igual a la máxima altura de ellas. La lista de listas tendrá una altura suma de las alturas de cada una de estas listas más 1 punto entre cada par de listas; y la anchura será la máxima de cada una de las listas. Al resultado habrá que sumar 2 de altura y 4 de anchura.

Comprobar asimismo que todas las listas de una misma matriz tengan el mismo número de fórmulas.

P.ej., la siguiente expresión, que representa a la fórmula:

MATRIZ
1 5 PARTIDO 8 ENTRE 4+2,
528 7 PARTIDO 8+2 6,
MATRIZ 1 72. 4 1 PARTIDO MATRIZ 3 7 11.
.

$$\begin{matrix} 1 & \frac{5}{8} & (4+2) \\ 528 & \frac{7}{8+2} & 6 \\ [1\ \ 72] & 4 & \frac{1}{[3\ \ 7\ \ 11]} \end{matrix}$$

debe dar como resultado:
Alto: 33 puntos.
Ancho: 35 puntos.