Examen de Traductores, Intérpretes y Compiladores.
Convocatoria ordinaria de Junio de 1998.
3 Curso de I.T. Informática de Sistemas.

Apellidos, Nombre: _____ Calificación: _____

TEORÍA

1.- Sea la siguiente tabla LL(1):

T / N	id	^	.	[	]	$
e	e --> id e'					
e'		e'--> ^e'	e'-->. id e'	e'--> [e] e'	e'--> ε	e'--> ε

asociada a la gramática:

e --> id e'
e'--> ε | ^ e' | . id e' | [e] e'

Proceder al reconocimiento o rechazo de la sentencia:
id^.id[id^]^$
mostrando en todo momento el estado de la pila, y de la cadena de entrada.

2.- Explicar, con extensión no superior a media página por punto, los siguientes conceptos:

a) Función del enlazador (linker) y del cargador (loader).
b) *Front-end* y *back-end*.
c) Partes del registro de activación.

3.- Justificar si la siguiente gramática es ambigua o no:

l_decl --> ε | l_decl decl
decl --> **ID :** tipo | **ID ,** decl
tipo --> **INT | REAL | POINTER TO** tipo | **RECORD OF** l_decl

Si lo es, sustituir la gramática por la que se considere más oportuna, y que solucione la ambigüedad.

4.- Escribir una gramática que permita reconocer la zona de reglas de YACC.

Examen de Traductores, Intérpretes y Compiladores.
Convocatoria ordinaria de Junio de 1998.
3 Curso de I.T. Informática de Sistemas.

Apellidos, Nombre: _____ Calificación: _____

PRÁCTICA

Se desea hacer un lenguaje de programación, de manera que como prototipo se desean las siguientes características, y con objeto de generar código de tercetos:

a) Permitir el uso de variables de tipo entero, sin necesidad de declaraciones.
b) Eliminar los comentarios. A este efecto, un comentario está formado por dos barras de división, y tiene validez hasta el fin de línea.
c) Permitir asignaciones simples a un identificador. En una expresión sólo intervienen +,-,* y *menos unario*.
d) Permitir que una expresión sea un pre/post-incremento o un pre/post-decremento de un identificador.
e) Permitir los bucles FOR. A estos efectos se requiere lo siguiente:
   e.1) El bucle FOR tiene la siguiente estructura:
   FOR ID := $expr_1$ TO $expr_2$ [STEP $expr_3$] DO
   sent;
   FIN FOR
   en el que el STEP es opcional. Si no existe el paso ($expr_3$) se supone +1.
   e.2) Si el paso es positivo, el bucle pasará cuando el ID tenga un valor superior al de $expr_2$.
   e.3) Es posible que no se llegue a entrar nunca en el bucle.
   e.4) En esta primera aproximación, **no** se contemplará la posibilidad de bucles FOR anidados.
   e.5) Dentro de un FOR podemos encontrarnos las siguientes sentencias:
   CONTINUE: Corta el ciclo actual, e inicia la siguiente iteración.
   BREAK: Cancela el bucle, y pasa a la primera sentencia tras el FIN FOR.
h) Un número entero puede intervenir como operando en cualquier terceto.
i) Se pueden emplear cuantas variables temporales se considere necesario.
j) Los identificadores que a lo largo de la compilación, sólo hayan aparecido como r-valores o sólo como l-valores, se sacarán por pantalla, al final del análisis, con el correspondiente mensaje de advertencia o error.
k) No existe ningún terceto que pueda multiplicar, luego una multiplicación del programa fuente debe traducirse a un trozo de código en tercetos.

Se pide:

- Definir la estructura de cada símbolo en la tabla de símbolos.
- Hacer el programa LEX correspondiente, que efectúa el punto b), y además devuelve los tokens necesarios.
- Hacer el programa YACC correspondiente, que hace los puntos a) y del c) al i), y el k).
- Implementar la función ts_comprobar(), que efectúa el punto j).
- Como último punto, y sólo para obtener Matrícula de Honor, explicar textualmente y detalladamente qué modificaciones son necesarias para

permitir la existencia de FOR anidados con el correspondiente buen funcionamiento de las sentencias BREAK y CONTINUE.

Para ello se da el siguiente esqueleto:

Fichero TabSimb98.c

```
#include <stdlib.h>
#include <stdio.h>
typedef struct_simbolo
  {
   struct_simbolo * sig;
```

```
  } simbolo;

símbolo *ts_crear()

void ts_insertar(simbolo * * p_t, simbolo *s)

simbolo * ts_buscar(simbolo *t, char nombre[21])
```

```
void ts_comprobar
```

Fichero Jun98L.lex

```
%%
```

Fichero Jun97Y.yac

```
%{
#include “tabsimb.c”
```

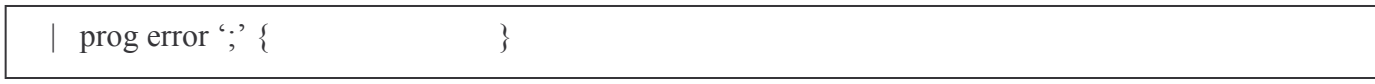
```
simbolo * t;
}

%union
{
```

}

%%

prog : prog sent ';'



| prog error ';' { }

|

;

sent : ID ASIG expr {

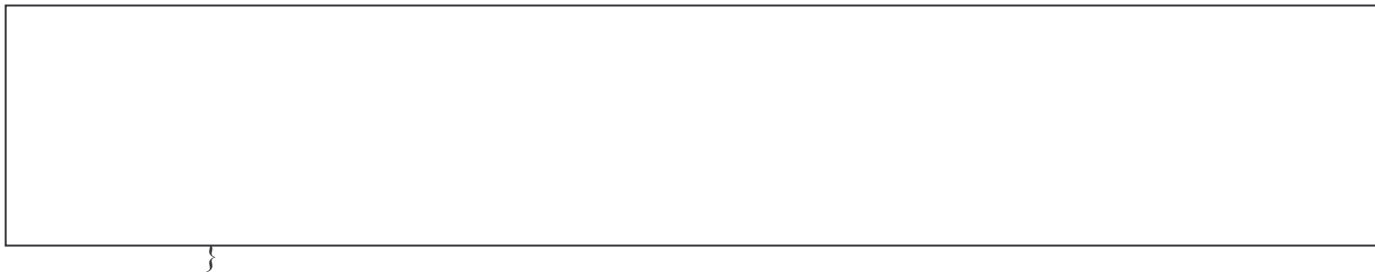
}

| FOR ID ASIG expr {

}

TO expr

opcional {



}

DO

sent ';'

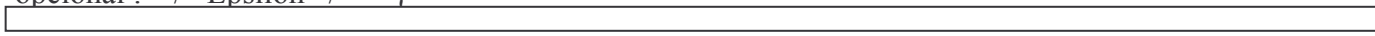
FIN FOR {

}

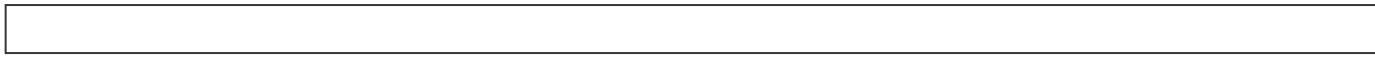
| CONTINUE {

}
| BREAK {

}
| '{' lista_sent '}'
;
lista_sent : /* Epsilon */
| lista_sent sent ';'
;
opcional : /* Epsilon */ {



}
| STEP expr {



}
;
expr : NUMERO {

}
| ID {

}
| MASMAS ID {

}
| ID MASMAS {

}
| MENOSMENOS ID {

```
                }
  | ID MENOSMENOS        {
```

```
                }
  |  expr '+' expr      {
```

```
                }
  |  expr '-' expr      {
```

```
                }
  |  expr '*' expr      {
```

```
                }
  |  '-' expr %prec MENOS_UNARIO
                {
```

```
                }
  |  '(' expr ')'       {
```

```
                }
  ;

%%
#include "exaju98l.c"
```

#include “errorlib.c”