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

Examen de Traductores, Intérpretes y Compiladores.
Convocatoria extraordinaria de febrero de 2001
3er Curso de I.T. Informática de Sistemas.

Apellidos, Nombre: _____ ***Calificación:*** _____

TEORÍA

1.- Partiendo del enunciado de apartado práctico, proponer una gramática que reconozca cualquier programa válido.

2.- Dada la siguiente gramática:

```
ld    ->   d ;
       |   ld, d ;
d     ->   t lid
t     ->   I
       |   R
       |   D
       |   t [ ]
       |   ( t )
       |   t ( )
       |   * t
lid   ->   ID
       |   lid , ID
```

construir manualmente el árbol sintáctico que reconoce la siguiente sentencia:

I [ ] ID ; * (* R [ ]) ( ) ID , ID , ID ;

3.- El siguiente programa Lex no hace lo que se espera de él. Contiene 8 fallos que hay que corregir para que funcione. Indicar cuáles son.

Solución:

El orden de los literales está mal.
El orden de la condición start está mal.
Falta reconocer el "*" sólo en el COMENT.
Yytext[1] no tiene nada en ese momento.
Las comillas simples no se utilizan para entrecomillar literales.
La barra de la t y de la n está al revés.
Ylex() es con una sola X.

El literal TRES no empieza en la columna 0.

```
%start COMENT
%%
[A-Z]*  {return ID}
[ /t/n]              {;}
.*          {return yytext[1];}
"(*"     {BEGIN COMENT;}
<COMENT>[^\*]* {;}
<COMENT>"*)"   {BEGIN 0;}
'UNO'  {return UNO;}
'DOS'   {return DOS;}
 'TRES' {return TRES;}
%%
void main(){
        yylexx();
}
```

4.- Responder brevemente a las siguientes preguntas:

a) ¿Para que se utilizan los atributos en una compilación dirigida por sintaxis?

b) ¿Qué utilidad tienen los llamados “recolectores de basura”?

c) ¿Qué sentido tiene la utilización de registros de activación?

d) La fase de optimización, actúa sobre la salida de otra/s fase/s, ¿de cuáles? ¿Por qué?

PRÁCTICA

Se pretende realizar la generación de código de tercetos para el lenguaje LISP (LISt Processing), para lo cual vamos a partir de un prototipo con las siguientres características:

- Cada sentencia va englobada entre paréntesis.
- La sentencia de asignación es de la forma (**SETQ** *ID expr*).
- Además de la asignación, se tiene una sentencia condicional, de la forma (**LOOP** (**UNTIL** *cond*) *listaSentencias*), de manera que *listaSentencias* se ejecutan hasta que se cumpla la condición. Si la *condición* es verdad al principio, la *listaSentencias* no se ejecuta ninguna vez.
- Una condición aparece también entre paréntesis, y es de la forma (*opRelacional expr expr*), donde *opRelacional* es una palabra que indica si se quiere preguntar por > (mayor que), < (menor que), o 0 (igual que). Las palabras son, respectivamente, **GREATERP**, **LESSP**, y **EQ**.
- Las expresiones más sencillas son las que vienen dadas directamente por un identificador o por un número.
- Las expresiones se pueden combinar mediante operadores aritméticos, que también se representan mediante palabras: **PLUS**, **DIFFERENCE**, **TIMES** y **QUOTIENT**, que representan respectivamente la suma, resta, producto y cociente entre otras dos expresiones. Este tipo de expresiones también ha de ir entre paréntesis.
- Las expresiones más complejas son las condicionales y que, de alguna forma, son parecidas a las sentencias CASE, aunque más sencillas de implementar. Una expresión condicional se coloca entre paréntesis, y está formada por la palabra clave **COND** seguida de una lista de pares de la forma (*condición expr*), de manera que la expresión representada por **COND** toma el valor de la *expr* asociada a la primera *condición* que sea verdadera. Ej. la expresión:
  (**COND** ((**GREATERP** ***X 11***) ***7***) ((**EQ** ***X 11***)(**PLUS** ***Y 10***)) ((**LESSP** ***X 11***) ***7***)
  toma el valor ***7*** si ***X*** es mayor que ***11***; toma el valor de ***Y*** más ***10*** si ***X*** vale ***11***, y el valor ***7*** si ***X*** es menor que ***11***. Si se cumple más de una condición, se asocia la expresión de la primera que se cumpla.
- Todas las variables son de tipo entero.

El código de tercetos que se desea generar tiene las características que se han visto en clase.

La figura **1** ilustra un ejemplo de programa de entrada, y la figura **2** la salida asociada.

Se pide:

Construir los programas Lex y Yacc necesarios para solucionar el enunciado propuesto. Para ello se suministra la gramática necesaria.

```
(SETQ X 15)
(SETQ X (PLUS X 8))
(SETQ X (COND ((GREATERP X 20) 10) ((LESSP X 10) 20) ((EQ 1 1) 0)))
(SETQ Y 0)
(LOOP (UNTIL (EQ X 100))
        (SETQ X (PLUS X 1))
        (SETQ Y (TIMES Y X))
)
(SETQ Z 0)
(SETQ X 1)
(LOOP (UNTIL (EQ X 200))
        (SETQ Y 1)
        (LOOP (UNTIL (EQ Y 200))
                (SETQ Z (COND ((LESSP X 50) (PLUS X (PLUS Y 1)))
                              ((EQ X 50) (PLUS X Y))
                              ((GREATERP X 50) (PLUS X (DIFFERENCE Y 1)))
                        )
                )
        )
)
```

1. Ejemplo de entrada

2. Resultado de salida:

```
        X=15
        tmp1=X+8
        X=tmp1
        if X>20 goto etq2
        goto etq3
label etq2
        tmp2=10
        goto etq1
label etq3
        if X<20 goto etq4
        goto etq5
label etq4
        tmp2=20
        goto etq1
label etq5
        if 1=1 goto etq6
        goto etq7
label etq6
        tmp2=0
        goto etq1
label etq7
label etq1
        X=tmp2
        Y=0
label etq11
        if X=100 goto etq13
        goto etq12
label etq12
        tmp7=X+1
        X=tmp7
        tmp8=Y*X
        Y=tmp8
        goto etq11
label etq13
        Z=0
        X=1
label etq101
        if X=200 goto etq102
        goto etq18
label etq18
        Y=1
label etq201
        if Y=200 goto etq202
        goto etq22
label etq22
        if X<50 goto etq30
        goto etq31
label etq30
        tmp20=Y+1
        tmp21=X+tmp20
        tmp10=tmp21
        goto etq80
label etq31
        if X=50 goto etq40
        goto etq41
label etq40
        tmp22=X+Y
        tmp10=tmp22
        goto etq80
label etq41
        if X>50 goto etq50
        goto etq51
label etq50
        tmp23=Y-1
        tmp24=X+tmp23
        tmp10=tmp24
label etq51
label etq80
        Z=tmp10
        goto etq201
label etq202
```

goto etq101 label etq102

La gramática necesaria es la siguiente:

```
listaSent   :   /*Epsilon */
            |   listaSent ‘(‘ sent ‘)’
            |   listaSent error ‘)’
            ;
sent        :   SETQ ID expr
            |   LOOP '(' UNTIL expr ')' listaSent
            ;
expr        :   '(' opArit expr expr ')'
            |   ID
            |   NUM
            |   ‘(‘ COND listaPares ‘)’
            ;
opArit      :   PLUS
            |   DIFFERENCE
            |   QUOTIENT
            |   TIMES
            ;
listaPares  :   par
            |   listaPares par
            ;
par         :   ‘(‘ cond expr ‘)’
            ;
cond        :   ‘(‘ opBool expr expr ‘)’
            ;
opBool      :   GREATERP
            |   LESSP
            |   EQ
            ;
```