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

Examen de Traductores, Intérpretes y Compiladores.
Convocatoria ordinaria de Febrero de 2005
3er Curso de I.T. Informática de Sistemas.

Apellidos, Nombre: _____

Calificación: _____

TEORÍA

1.- Sea la gramática:

S → P n n n n C g
P → f | h
C → ε | f C

cuya tabla de reconocimiento LL(1) es:

N \ T	f	h	n	g	$
S	S → P n n n n C g	S → P n n n n C g			
P	P → f	P → h			
C	C → f C			C → ε	

se pide:

a) reconocer o rechazar la cadena:
f n n n n f f f g
b) Construir un único diagrama de Conway para reconocer el no terminal S.

2.- A partir del diagrama de Conway construído en el paso anterior, codificar en C o en pseudocódigo, una función que reconozca el no terminal S.

3.- Dada la expresión regular PCLex:

[a-z]{3, 5}[qr]"77 "

indicar cuáles de las siguientes cadenas se reconocen y cuáles se rechazan (responder equivocadamente una opción resta puntos; no responder una opción ni suma ni resta puntos):

a) aaq77
b) abcq77
c) aaaqr77
d) azazqr77
e) abcqr77

4.- La gramática:

S → S m A | S m m B
A → m a
B → m b

¿tiene algún problema a la hora de ser reconocida por PCYacc? Y si es así, ¿es posible solucionarlo?

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

Examen de Traductores, Intérpretes y Compiladores.
Convocatoria ordinaria de Febrero de 2005
3er Curso de I.T. Informática de Sistemas.

Apellidos, Nombre: _____

Calificación: _____

PRÁCTICA

El presente ejercicio pretende transformar una gramática de contexto libre (independiente del contexto) en otra equivalente pero expresada en forma normal de Chomsky extendida (FNCE). Una gramática se encuentra en FNCE si y solo si todas sus reglas están en formato FNCE; y una regla está en formato FNCE si y solo si:

- El antecedente está formado por un único no terminal.
- El consecuente está formado por una de las siguientes posibilidades:
  - Un único terminal. Ej: S → t
  - Un único no terminal. Ej: S → N
  - Dos no terminales. Ej: S → N M

De esta explicación se desprende que una gramática en FNCE no admite reglas épsilon.

Características de la gramática de entrada.

La entrada a nuestro traductor será una gramática expresada en notación PCYacc a la que se le ha extirpado todo aquello que no tenga que ver exclusivamente con la gramática expresada. De esta manera, la gramática que reconoce una gramática extirpada en PCYacc es:

```
gramatica     :      secuenciaTokens SEP secuenciaReglas
              ;
secuenciaTokens      :      listaToken
                     |      secuenciaTokens listaToken
                     ;
secuenciaReglas      :      regla
                     |      secuenciaReglas regla
                     ;
listaToken    :      TOKEN ID
              |      listaToken ID
              ;
regla  :      ID ':' listaOpciones ';'
              ;
listaOpciones        :      opcion
                     |      listaOpciones '|' opcion
                     ;
opcion        :      simbolo
              |      opcion simbolo
              ;
simbolo       :      ID
              |      CHAR
              ;
```

Como se desprende de esta gramática, vamos a admitir gramáticas cualesquiera divididas en dos bloques: a) uno de declaración de *tokens* y b) otro de declaración de reglas; ambos están separados por **%%** representado por el *token* **SEP**.

Cada **listaTokens** representa una línea de la forma:

%token TOK_1 TOK_2 ... TOK_n

en la que se declaran TODOS los terminales de la gramática.

Las reglas tiene la ya consabida forma tradicional de PCYacc, admitiéndose en el consecuente la existencia de *tokens*, no terminales, *tokens* representados mediante un solo carácter entre los símbolos ‘’ e incluso acciones que DEBEN SER IGNORADAS por el analizador léxico

(aunque este hecho provoca que una regla pueda aparecer en cualquier lugar de la entrada, no se tendrá en cuenta esta circunstancia). Téngase en cuenta que una acción viene delimitada por los símbolos **{** y **}** y que, a su vez puede contener más pares de llaves en su interior. Por otro lado, las reglas que poseen un antecedente común pueden separarse por '|'. El final de una regla viene dado por el símbolo ';'.

Para facilitar la construcción del traductor, no obligaremos a que los diferentes componentes de la gramática tengan que comenzar en la columna 0. Por tanto, los espacios, retornos de carro y tabuladores se ignorarán.

Características de la gramática de salida.

Como ya se ha comentado, el propósito del ejercicio consiste en producir una gramática de salida equivalente a la de entrada, pero que esté en FNCE. Para ello, se ignorarán las acciones que pueda contener la gramática de entrada, y la conversión pasa por crear nuevos no terminales intermedios que permitan realizar la transformación. Estos no terminales se crearán en tres situaciones:

- Para cada *token* TOK se creará una regla de la forma:

  _pre_TOK → TOK

  de tal manera que, en el resto de la gramática, cada vez que aparezca TOK, se utilizará en su lugar _pre_TOK que es un no terminal. Con esto se consigue que un terminal sólo aparezca en reglas en las que él es el único elemento en el consecuente.
- Para cada *token* 't' expresado entre comillas simples se creará una regla de la forma:

  _pre_xxx → 't'

  donde **xxx** es el código ASCII del carácter entrecomillado ('t' en este caso).
- Cada vez que en un consecuente aparezcan dos no terminales juntos N_1 N_2 se creará una regla de la forma:

  _nt_xx → N_1 N_2

  donde **xx** es un número secuencial que se va incrementado a partir de 0. Por ejemplo, la regla:

  expr : expr '+' expr ;

  se traduce a:

  _nt_0 : expr _pre_43 ;
  _nt_1 : _nt_0 expr ;
  expr : _nt_1 ;

  ya que el 43 es el código ASCII del +.

En la gramática de salida sólo son requeridas las reglas de producción. No es necesario sacar los *tokens*. A continuación se ilustra un ejemplo de salida con una entrada relativamente grande:

```
%token DECLARE BEGINN END PRINT
%token  ID
%token  NUM
%%
prog   : DECLARE l_decl BEGINN l_sent END
       ;
l_decl: decl ';'
             | l_decl decl ';'
             ;
decl   : NUM '*' NUM ID          { accion1 }
             | decl ',' ID       { accion2 }
             ;
l_sent: sent
       | l_sent sent ';'
       ;
sent   : PRINT mtrz {
                          if ($2.ptrInt != NULL){
                                accion3
                          }
```

```
                    }
       | ID '=' mtrz        { accion4 }
       | ID '[' NUM ',' NUM ']' '=' NUM        { accion5 }
       | ID '[' NUM ']' '=' '{' { accion6 } l_num '}' { accion7 }
       ;
l_num  : NUM
       | l_num ',' NUM
       ;
mtrz   : ID
       | mtrz '*' mtrz
       | mtrz '+' mtrz
       | '(' mtrz ')'
       ;
```

Ante esta entrada, la salida debe ser:

```
_NT_0  :      _PRE_DECLARE l_decl;
_NT_1  :      _NT_0 _PRE_BEGINN;
_NT_2  :      _NT_1 l_sent;
_NT_3  :      _NT_2 _PRE_END;
prog   :      _NT_3;
_NT_4  :      decl _PRE_59;
l_decl        :      _NT_4;
_NT_5  :      l_decl decl;
_NT_6  :      _NT_5 _PRE_59;
l_decl        :      _NT_6;
_NT_7  :      _PRE_NUM _PRE_42;
_NT_8  :      _NT_7 _PRE_NUM;
_NT_9  :      _NT_8 _PRE_ID;
decl   :      _NT_9;
_NT_10        :      decl _PRE_44;
_NT_11        :      _NT_10 _PRE_ID;
decl   :      _NT_11;
l_sent        :      sent;
_NT_12        :      l_sent sent;
_NT_13        :      _NT_12 _PRE_59;
l_sent        :      _NT_13;
_NT_14        :      _PRE_PRINT mtrz;
sent   :      _NT_14;
_NT_15        :      _PRE_ID _PRE_61;
_NT_16        :      _NT_15 mtrz;
sent   :      _NT_16;
_NT_17        :      _PRE_ID _PRE_91;
_NT_18        :      _NT_17 _PRE_NUM;
_NT_19        :      _NT_18 _PRE_44;
_NT_20        :      _NT_19 _PRE_NUM;
_NT_21        :      _NT_20 _PRE_93;
_NT_22        :      _NT_21 _PRE_61;
_NT_23        :      _NT_22 _PRE_NUM;
sent   :      _NT_23;
_NT_24        :      _PRE_ID _PRE_91;
_NT_25        :      _NT_24 _PRE_NUM;
_NT_26        :      _NT_25 _PRE_93;
_NT_27        :      _NT_26 _PRE_61;
_NT_28        :      _NT_27 _PRE_123;
_NT_29        :      _NT_28 l_num;
_NT_30        :      _NT_29 _PRE_125;
sent   :      _NT_30;
l_num  :      _PRE_NUM;
_NT_31        :      l_num _PRE_44;
_NT_32        :      _NT_31 _PRE_NUM;
l_num  :      _NT_32;
mtrz   :      _PRE_ID;
_NT_33        :      mtrz _PRE_42;
_NT_34        :      _NT_33 mtrz;
mtrz   :      _NT_34;
_NT_35        :      mtrz _PRE_43;
_NT_36        :      _NT_35 mtrz;
mtrz   :      _NT_36;
_NT_37        :      _PRE_40 mtrz;
_NT_38        :      _NT_37 _PRE_41;
mtrz   :      _NT_38;
_PRE_41       :      ')';
_PRE_40       :      '(';
_PRE_43       :      '+';
_PRE_125      :      '}';
_PRE_123      :      '{';
_PRE_93       :      ']';
_PRE_91       :      '[';
_PRE_61       :      '=';
_PRE_44       :      ',';
_PRE_42       :      '*';
_PRE_59       :      ';';
_PRE_NUM      :      NUM;
_PRE_ID       :      ID;
_PRE_PRINT    :      PRINT;
_PRE_END      :      END;
_PRE_BEGINN   :      BEGINN;
_PRE_DECLARE  :      DECLARE;
```

Se pide, **como requisito indispensable para obtener un APROBADO**:

- Realizar un programa Lex/Yacc basado en las especificaciones anteriores que realice las acciones deseadas. Para ello puede suponerse la existencia de una tabla de símbolos con las siguientes características:

```
typedef struct _simbolo {
      struct _simbolo * sig;
      char nombre[20];
             char tipo; // 't'oken, 'n'o terminal
             short definido, usado;
} simbolo;
simbolo * buscarToken(simbolo * t, char nombre[20]){
       while( (t != NULL) && ( (strcmp(nombre, t->nombre)) || (t->tipo != 't') ) )
             t = t->sig;
       return (t);
}
void insertarToken(simbolo * * p_t, char nombre[20]){
```

```
        if (buscarToken((*p_t), nombre))
                printf("Error: %s redeclarado.\n", nombre);
        else {
                simbolo * s = (simbolo *) malloc(sizeof(simbolo));
                strcpy(s->nombre, nombre);
                s->tipo = 't';
                s->definido = 1;
                s->usado = 0;
                s->sig = (*p_t);
                (*p_t) = s;
        }
}
void sacarReglasTokens(simbolo * t, char * prefijo){
    while(t != NULL) {
        if (t->tipo == 't')
            if (t->nombre[0]=='\'')
                printf("%s%d \t:\t%s;\n", prefijo, t->nombre[strlen(t->nombre)-2], t->nombre);
            else
                printf("%s%s \t:\t%s;\n", prefijo, t->nombre, t->nombre);
       t = t->sig;
    }
}
void generarNoTerminal(char * dest){
        static num=0;
        sprintf(dest, "_NT_%d", num++);
}
```

Para obtener una calificación **superior al APROBADO**, se propone:
- Controlar que todos los *tokens* declarados se utilicen en la zona de reglas.
- Controlar que todos los no terminales que aparezcan en el antecedente de una regla se utilicen a la derecha de alguna otra.
- Controlar que todos los no terminales que se utilicen a la derecha de alguna regla hayan sido declarados, esto es, que aparezcan como antecedente de laguna regla de producción.

Para esto debe dotarse de cuerpo y usarse las siguientes funciones C:

```
simbolo * buscarNoTerminal(simbolo * t, char nombre[20]){




}
void insertarNoTerminal(simbolo * * p_t, char nombre[20], short definido, short usado){
simbolo * s;
        if (! (s = buscarNoTerminal((*p_t), nombre))) {




        }
        s->definido = definido | s->definido;
        s->usado = usado | s->usado;
}
void recorrer(simbolo * t, char tipo){
```

}

El esqueleto que se suministra es el siguiente:

Parte Lex:

```
%{
        int cont = 0;
%}
%start ACTION
%%
```

Parte Yacc:

```
%{
// Inclusión de la tabla de símbolos
#include "tabsim00.c"


%}
%union {
    char nombre[50];
}



%%
gramatica :  secuenciaTokens SEP secuenciaReglas
          ;
secuenciaTokens  :  listaToken
                 |  secuenciaTokens listaToken
                 ;
secuenciaReglas  :  regla
                 |  secuenciaReglas regla
                 ;
listaToken    :  TOKEN ID        {                          }
              |  listaToken ID    {                          }
              ;
regla  :  ID    {
```

```
                }
        ':' listaOpciones ';'
        ;
listaOpciones    :  opcion    {

                              }
                 |  listaOpciones '|' opcion
                              {

                              }
                 ;
opcion     :  simbolo          {                }
           |  opcion simbolo {




                              }
           ;
simbolo    :  ID {









                      }
                }
           |  CHAR   {




                      }
           ;

%%
#include "exdi04l.c"
void main(){
   yyparse();




}
void yyerror(char * s){ printf("%s\n", s); }
```