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 Septiembre de 2004
3er Curso de I.T. Informática de Sistemas.

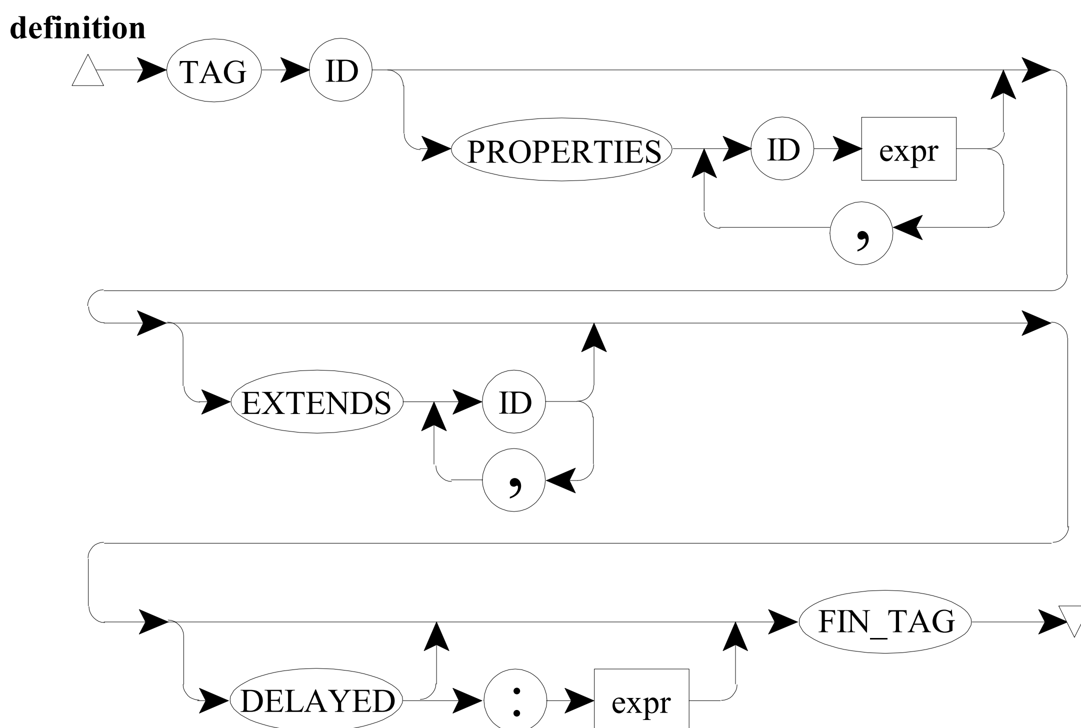
Apellidos, Nombre: ____________________________

Calificación: ______________

TEORÍA

1.- Escribir una gramática al estilo Yacc (mediante reglas de producción) que tenga por objetivo reconocer gramáticas expresadas en notación BNF. En otras palabras, se pide escribir una gramática que, en lugar de reconocer un lenguaje de programación, reconozca una secuencia de reglas expresadas en notación BNF. (**3 ptos.**)

2.- Escribir en C la función recursiva que se corresponde al siguiente diagrama de sintaxis (se supone la existencia de una función recursiva que reconoce **expr**) (**2'5 ptos.**):



3.- Escribir una gramática mediante reglas de producción que, con el menor número de no terminales posible, reconozca lo mismo que el diagrama de sintaxis anterior. Puede suponerse la existencia de reglas que reconocen una **expr**. En base a dicha gramática dibujar el árbol de sintaxis que reconoce la siguiente secuencia de *tokens* (**3 ptos.**):

TAG ID PROPERTIES ID expr , ID expr DELAYED : expr FIN_TAG

4.- Indicar brevemente 4 diferencias entre un compilador y un intérprete. (**1'5 ptos.**)

Open Course Ware
Gálvez Rojas, S.
(2011). Traductores, Compiladores e Intérpretes
OCW- Universidad de Málaga http://ocw.uma.es
Bajo licencia Creative Commons Attribution-Non-Comercial-ShareAlike
cc BY NC SA

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 Septiembre de 2004
3er Curso de I.T. Informática de Sistemas.

Apellidos, Nombre: ___________________________

Calificación: ______________

EJERCICIO PRÁCTICO

La empresa Soyuz Compilators S.A. se dedica a la creación masiva de compiladores y gramáticas de contexto libre. La definición de las gramáticas suele hacerse en una notación BNF propia en la que:
- Los corchetes [] indican opcionalidad.
- Las llaves {} indican repetición cero o más veces.
- La barra vertical | sirve para separar distintas opciones.
- La yuxtaposición equivale a la concatenación y tiene prioridad sobre la barra vertical.

Con el objetivo de aumentar la documentación disponible sobre cada proyecto, Soyuz Compilators S.A. se plantea la necesidad de construir un conversor automático de gramáticas expresadas en notación BNF a gramáticas expresadas en reglas de producción, al estilo de Yacc. Y en eso consiste este ejercicio. Un ejemplo de conversión puede observarse en el siguiente caso práctico.

Gramática BNF de entrada:

```
prog ::= sent <punto> { sent <punto> };
sent ::= sentIf | sentFor | sentWhile;
sentIf ::= <if> cond <then> prog [ <else> prog ] <finIf>;
cond ::= expr (<mayor>|<menor>|<igual>|<diferente>) expr;
expr ::= <num> | <id> ;
sentFor ::= <for> <id> <igual> expr <do> prog <finFor>;
sentWhile ::= <while> cond <do> prog <finWhile>;
prueba1 ::=   <if> cond <then> prog [ <else> prog ] <finIf> |
              <for> <id> <igual> expr <do> prog <finFor> |
              <while> cond <do> prog <finWhile> ;
prueba3 ::= <uno> [ <dos> { <coma> <dos> [ <as> <name> ] } ] ;
prueba3 ::= prueba4 ;
```

Resultado del conversor:

```
NoTerminal_0      :       NoTerminal_0  sent punto
          |
          ;
prog      :       sent punto NoTerminal_0
          ;
sent      :       sentIf
          |       sentFor
          |       sentWhile
          ;
```

Open Course Ware
Gálvez Rojas, S.
(2011). Traductores, Compiladores e Intérpretes
OCW- Universidad de Málaga http://ocw.uma.es
Bajo licencia Creative Commons Attribution-Non-Comercial-ShareAlike
cc BY NC SA

```
NoTerminal_1    :    else prog
        |
        ;
sentIf  :    if cond then prog NoTerminal_1 finIf
        ;
NoTerminal_2    :    mayor
        |    menor
        |    igual
        |    diferente
        ;
cond    :    expr NoTerminal_2 expr
        ;
expr    :    num
        |    id
        ;
sentFor     :    for id igual expr do prog finFor
        ;
sentWhile   :    while cond do prog finWhile
        ;
NoTerminal_3    :    else prog
        |
        ;
prueba1     :    if cond then prog NoTerminal_3 finIf
        |    for id igual expr do prog finFor
        |    while cond do prog finWhile
        ;
NoTerminal_4    :    as name
        |
        ;
NoTerminal_5    :    NoTerminal_5  coma dos NoTerminal_4
        |
        ;
NoTerminal_6    :    dos NoTerminal_5
        |
        ;
prueba3     :    uno NoTerminal_6
        ;
prueba3     :    prueba4
        ;
```

Los tokens son: name, as, coma, dos, uno, finWhile, while, finFor, do, for, id, num, diferente, igual, menor, mayor, finIf, else, then, if, punto,

prueba4 no ha sido definida.
prueba3 ha sido definida mas de una vez.
prueba3 no ha sido usada.
prueba1 no ha sido usada.

Open Course Ware
Gálvez Rojas, S.
(2011). Traductores, Compiladores e Intérpretes
OCW- Universidad de Málaga http://ocw.uma.es
Bajo licencia Creative Commons Attribution-Non-Comercial-ShareAlike

Como puede observarse, el conversor debe comprobar:
1.- Que todo no terminal se use.
2.- Que todo no terminal que se use se declare antes o después.
3.- Que un mismo no terminal no se defina más de una vez.

Para realizar la conversión hay que tener en cuenta:
1.- Que cada regla BNF finaliza en punto y coma.
2.- Que las reglas de producción que se generan tienen la misma estructura que en Yacc.
3.- Que para la yuxtaposición se debe generar una sola regla de producción. Por ejemplo, si la entrada es:

    p1 ::= a1 a2 a3 ;

debe generarse la regla de producción:

    p1 : a1 a2 a3 ;

y no:

    NoTerminal_0 : a1 a2 ;
    NoTerminal_1 : NoTerminal_0 a3 ;
    p1 : No_terminal_1 ;

4.- Que los *tokens* de la notación BNF están delimitados por corchetes angulares: < y >. En las reglas de producción equivalentes estos angulitos se deben quitar.
5.- Que los *tokens* y los no terminales deben tener el mismo nombre tanto en BNF como en las reglas de producción generadas.
6.- Que suele ser necesario generar nuevos no terminales auxiliares. Estos no terminales auxiliares serán de la forma NoTerminal_*xx*, tal y como se muestra en el ejemplo anterior.

Se debe:

- Realizar los programas Lex y Yacc que produzcan los objetivos propuestos a partir de la gramática que se proporciona. Dicha gramática es ambigua y se debe desambiguar convenientemente mediante las directivas %left, %right, %nonassoc y %prec que se estimen oportunas.
- Rellenar las funciones cuyas cabeceras se suministran.
- Tras la gramática formal producida debe suministrarse la lista de los terminales (*tokens*) que forman parte de la gramática (sin los angulitos);
- Al final de la gramática equivalente y de la lista de *tokens* debe suministrarse la lista de los no terminales no usados, no definidos o definidos más de una vez.

Nota: El ejercicio debe resolverse en los huecos al efecto dentro del propio enunciado.

Esqueleto Lex:

%% // (0,75 ptos.)

Open Course Ware
Gálvez Rojas, S.
(2011). Traductores, Compiladores e Intérpretes
OCW- Universidad de Málaga http://ocw.uma.es
Bajo licencia Creative Commons Attribution-Non-Comercial-ShareAlike

Esqueleto Yacc:

```
%{
#include "stdio.h"
#include "stdlib.h"
	void generarNT(char * s){
		static int noTerminal = 0;
		sprintf(s, "NoTerminal_%d", noTerminal++);
	}
	// La T es de 'T'erminal.
	struct T{
		char nombre[50];
		struct T * sig;
	};
	struct T * tablaT = NULL;
	void registrarT(char * nombre){
		struct T * t = tablaT;
		while((t!=NULL)&&(strcmp(t->nombre, nombre)))
			t = t->sig;
		if (t == NULL){
			struct T * aux = (struct T *) malloc(sizeof(struct T));
			strcpy(aux->nombre, nombre);
			aux->sig = tablaT;
			tablaT = aux;
		}
	}
	// La NT es de 'N'o 'T'erminal.
	struct NT{
		char nombre[50];
		unsigned short usada;
		int definida;
		struct NT * sig;
	};
	struct NT * tablaNT = NULL;
	struct NT * crearNT(char * nombre){
	// Aquí se debe crear un no terminal y meterlo en la pila tablaNT. (0,25 ptos.)







	}
	struct NT * buscarOCrearNT(char * nombre){
	// Aquí se debe buscar un no terminal. Si no existe se lo crea. (0,50 ptos.)
	// En cualquier caso se debe devolver un puntero al no terminal encontrado o recién creado.
```

Open Course Ware

Gálvez Rojas, S.
(2011). Traductores, Compiladores e Intérpretes
OCW- Universidad de Málaga http://ocw.uma.es
Bajo licencia Creative Commons Attribution-Non-Comercial-ShareAlike

cc BY NC SA

```
        }
%}

%union{
        char expresion[256];
        char texto[50];
}
        // Aquí van los %token, %type, etc. (1,5 ptos.)





%%
gram    :       regla
        |       gram regla
        ;
regla   :       ID ASIG exprBNF ';' { // (1,25 ptos.)


                                  }
        |       error ';'
        ;
exprBNF         :       /* Epsilon */           { // (0,25 ptos.)

                                                }
                |       exprBNF TOKEN           { // (0,50 ptos.)

                                                }
                |       exprBNF ID              { // (0,625 ptos.)

                                                }
                |       exprBNF '[' exprBNF ']' { // (0,75 ptos.)





                                                }
                |       exprBNF '(' exprBNF ')'{ // (0,625 ptos.)
```

Open Course Ware

Gálvez Rojas, S.
(2011). Traductores, Compiladores e Intérpretes
OCW- Universidad de Málaga http://ocw.uma.es
Bajo licencia Creative Commons Attribution-Non-Comercial-ShareAlike

```
                                        }
                    |       exprBNF '{' exprBNF '}'{ (1 pto.)



                                        }
                    |       exprBNF '|' exprBNF     { (1,25 ptos.)

                                        }
                    ;
%%
#include "ExSep04L.c" // Inclusión del fichero Lex
void main(){
        // Declaraciones necesarias (0,25 ptos.)


        yyparse();
        // Listado de tokens (0,25 ptos.)







        printf("\n");
        // Listado de errores en la gramática BNF de entrada (0,25 ptos.)









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

Open Course Ware
Gálvez Rojas, S.
(2011). Traductores, Compiladores e Intérpretes
OCW- Universidad de Málaga http://ocw.uma.es
Bajo licencia Creative Commons Attribution-Non-Comercial-ShareAlike
BY NC SA