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 Junio de 2007
3er Curso de I.T. Informática de Sistemas.

Apellidos, Nombre: _____

Calificación: _____

TEÓRICO

1.- Sea la gramática:

① P –> n
② | P P O
;
③ O –> m
④ | s
⑤ | d
;

cuya tabla de análisis LALR(1) es:

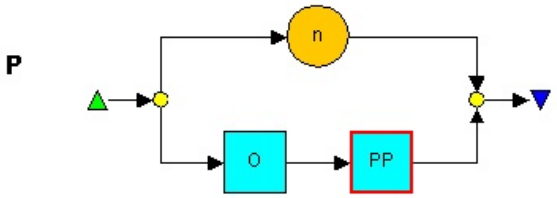
E \ T∪N	n	s	m	d	$	P	O
0	D-2					1	
1	D-2				Aceptar	3	
2	R1	R1	R1	R1	R1		
3	D-2	D-6	D-5	D-7		3	4
4	R2	R2	R2	R2	R2		
5	R3	R3	R3	R3	R3		
6	R4	R4	R4	R4	R4		
7	R5	R5	R5	R5	R5		

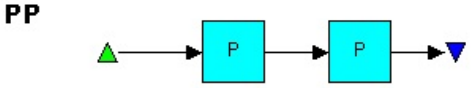
Reconocer o rechazar la secuencia:

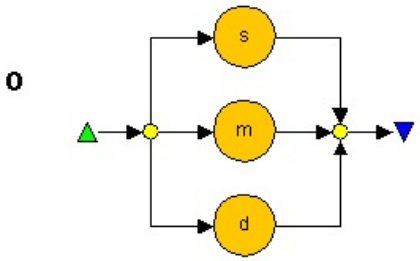
n n n m n s d

indicando en cada paso el estado de la pila-α, de β, y la acción a aplicar.

2.- Dados los siguientes diagramas de sintaxis:







se pide codificar en C las funciones recursivas que implementan su análisis sintáctico descendente, suponiendo la existencia de una función main como:

```
void main(){
      getToken();
      P();
}
```

3.- Indicar gráficamente la secuencia de fases que componen un compilador, así como su relación con cualesquiera otros componentes del compilador. Indicar qué fases se corresponden con la etapa de análisis y cuáles con la de síntesis, así como el porqué. De manera similar, indicar qué fases componen el *front-end* y el *back-end* y en que consisten estos conceptos.

4.- Indicar para qué se usan en Lex los siguientes símbolos:

a) Comillas dobles: “
b) Barra invertida: \
c) Barra inclinada: /
d) Circunflejo: ^

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 Junio de 2007
3er Curso de I.T. Informática de Sistemas.

Apellidos, Nombre: _____

Calificación: _____

PRÁCTICO

Se desea calcular la derivada de una función en un punto. La derivada de una función puede calcularse según la tabla adjunta.

Para ello se suministra una gramática que reconoce un número real seguido de punto y coma y la expresión cuya derivada se quiere calcular en dicho número. Por ejemplo, una entrada podría ser:

2,0 ; x*x

y el intérprete debe devolver:

4

y finalizar.

La gramática que permite reconocer las expresiones que forman las funciones de la tabla adjunta es:

```
s       :       NUMR ';' e
        ;
e       :       NUMR
        |       'x'
        |       e '+' e
        |       e '-' e
        |       e '* 'e'
        |       e '/' e
        |       '(' e ')'
        |       SEN '(' e ')'
        |       COS '(' e ')'
        |       ATAN '(' e ')'
        |       LN '(' e ')'
        |       EXP '(' e ')'
        ;
```

Función en x	Derivada
$constante$	0
x	1
$u \pm v$	$u' \pm v'$
$u \cdot v$	$u' \cdot v + u \cdot v'$
$\frac{u}{v}$	$\frac{u' \cdot v - u \cdot v'}{v^2}$
$\sin(u)$	$u' \cdot \cos(u)$
$\cos(u)$	$-u' \cdot \sin(u)$
$\arctan(u)$	$\frac{u'}{1+u^2}$
$\ln(u)$	$\frac{u'}{u}$
$\exp(u)$	$u' \cdot \exp(u)$

donde u es cualquier función de x.

Para solucionar este ejercicio, la idea consiste en asociar a una expresión un par de atributos de tipo **double**, de tal manera que:

- Uno es el valor de la expresión en el punto dado.
- El otro es el valor de la expresión derivada en el punto dado.

El token NUMR representa al un número real, esto es, con coma decimal obligatoria, y su atributo es de tipo double.

Las funciones en C que permiten realizar las operaciones anteriores se encuentran en la biblioteca **math.h**, y son:

- Seno: . . . . . **double sin(double n)**
- Coseno: . . . . . **double cos(double n)**
- Arcotangente: . . . . . **double atan(double n)**
- Logaritmo neperiano: . . . . . **double log(double n)**
- Exponenciación: . . . . . **double exp(double n)**

Por último, los valores **double** se emiten por pantalla con **printf** mediante el modificador **%lf**.

Se pide:

Construir los programas Lex y Yacc completos que, haciendo uso de la gramática anterior, proporcione el valor de la derivada de una función en un punto dado.

Fichero exjun07l.lex

```
%%
```

Fichero exjun07y.yac

```
%{
#include <stdlib.h>
#include <math.h>

%}

%union{

}
%token
%type
%token
%left '+' '-'
%left '*' '/'
%right MENOS_UNARIO
```

```
%%
s       :       NUMR ';'        {

                                }
                e               {

                                }
        ;
e       :       NUMR            {

                                }
        |       'x'             {

                                }
        | e '+' e               {

                                }
        | e '-' e               {

                                }
        | e '*' e               {

                                }
        | e '/' e               {

                                }
        | '-' e %prec MENOS_UNARIO      {

                                }
        | '(' e ')'             {

                                }
        | SENO '(' e ')'        {
```

```
                                }
        | COSENO '(' e ')'       {

                                }
        | LN '(' e ')'           {

                                }
        | EXP '(' e ')'          {

                                }
        | ARCTG '(' e ')'        {

                                }
        ;
%%
#include "exjun07l.c"

```