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

Apellidos, Nombre: _____

Calificación: _____

TEÓRICO

1.- Dibujar un único diagrama de Conway (diagrama de sintaxis) que reconozca el mismo lenguaje que la siguiente gramática:

```
sent  ->    GOTO listaNum CORRESPONDING ON e PYC
listaNum->        NUM
            |     listaNum COMA NUM
e     ->    ID
      |     NUM
      |     e MAS e
      |     e MENOS e
      |     e POR e
      |     e ENTRE e
```

2.- A partir del diagrama de sintaxis creado en el apartado anterior se pide codificar en C la función recursiva que implementa su análisis sintáctico descendente, suponiendo la existencia de una función main como:

```
void main(){
      getToken();
      sent();
      if (token != EOF) ¡Error!;
}
```

3.- Sea la gramática:

```
prog  ->    ε
      |     asig ';' prog
asig  ->    mat '=' mat
mat   ->    ID '[' listaId ']'
listaId ->        ID listaId'
listaId' ->       ε
            |     ',' ID listaId'
```

cuya tabla de análisis LL(1) es:

T / N	ID	;	=	[	]	,	$
prog	prog –> asig ';' prog						prog –> ε
asig	asig –> mat '=' mat						
mat	mat –> ID '[' listaId ']'						
listaId	listaId –> ID listaId'						
listaId'					listaId' –> ε	listaId' –> ',' ID listaId'	

Reconocer o rechazar la secuencia:

ID [ID , ID] = ID [ID , ID , ID] ; ID [ID ID] = ID [ID ID ID] ;

indicando en cada paso el estado de la pila y la parte consumida de la entrada.

4.- Describir brevemente (no más de cinco líneas por concepto) las fases de:

a) Compilar

b) Enlazar

c) Ejecutar

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

Apellidos, Nombre: _____

Calificación: _____

PRÁCTICO

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

Para ello se suministra una gramática que reconoce un expresión cuya derivada se quiere calcular en los puntos que la siguen. Por ejemplo, una entrada podría ser:

x*x ; 2.0, 3.7, 6.3

y el intérprete debe devolver:

4.0
7.4
12.6

y finalizar.

La gramática que permite reconocer las expresiones que forman las funciones de la tabla adjunta y la secuencia de números reales en que calcular su derivada es:

```
s        :      e ';' listaNumR
         ;
e        :      NUMR
         |      'x'
         |      e '+' e
         |      e '-' e
         |      e '* 'e'
         |      e '/' e
         |      '(' e ')'
         |      SENO '(' e ')'
         |      COSENO '(' e ')'
         |      ARCTG '(' e ')'
         |      LN '(' e ')'
         |      EXP '(' e ')'
         ;
listaNumR :     NUMR
         |      listaNumR ',' NUMR
         ;
```

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.

El token NUMR representa al un número real, esto es, con coma decimal obligatoria (léxicamente se utilizará punto decimal), 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)**

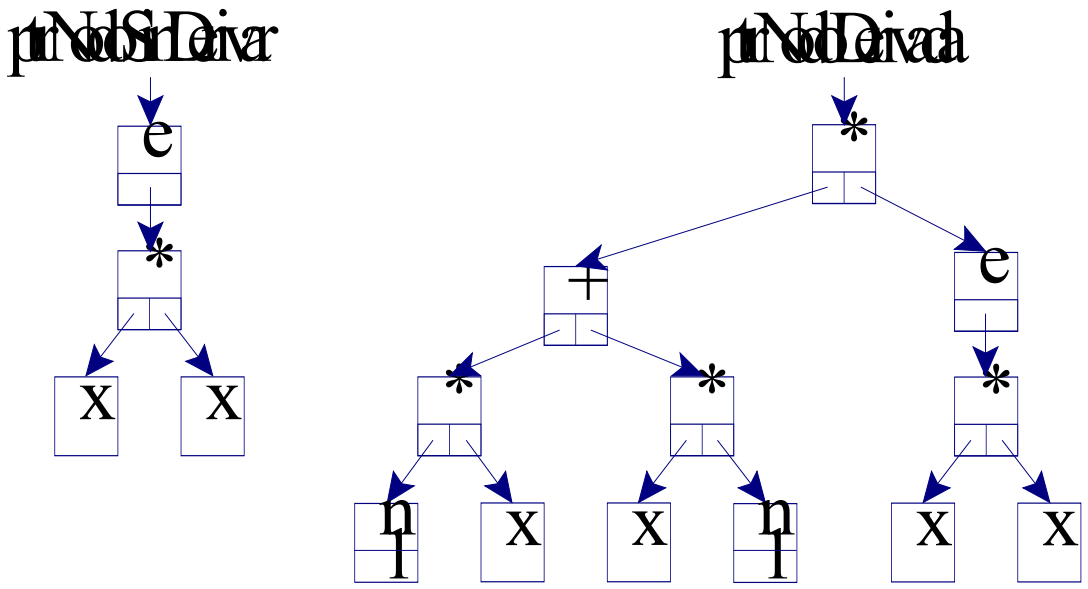
Los valores **double** se emiten por pantalla con **printf** mediante el modificador **%lf**.

Para solucionar este ejercicio, la idea consiste en asociar a una expresión un par de árboles formados por nodos con distinto contenido y que, una vez finalizado el análisis de la expresión, puedan ser evaluados para obtener la derivada:
- Uno de los árboles representa a la expresión derivada.
- El otro representa a la expresión sin derivar.

Un nodo tiene la siguiente estructura:

```
typedef struct _nodo{
        char tipo; // 'n'umero, 'x', 's'eno, 'c'oseno, 'e'xponencial, 'l'n, 'a'rctg, '/', '+', '-', '*'
        union {
                double valor;
                struct _nodo * hijo;
                struct {
                        struct _nodo * hijo1;
                        struct _nodo * hijo2;
                } hijos;
        } contenido;
} nodo;
```

Así, una expresión como:
exp(x*x)
tendrá asociados los árboles:
Nota: Los nodos que tiene una **e** vienen de **exponenciación**, y NO de **expresión**.



Una vez obtenidos los árboles de una expresión completa, se utilizará el árbol apuntado por ptrNodoDerivada para evaluar cada uno de los puntos dados por números reales.

Se pide:

Construir, a partir del esqueleto dado, los programas Lex y Yacc completos que, haciendo uso de la gramática anterior, proporcione los valores de la derivada de la función en los puntos dados. Asimismo, se pide crear el cuerpo de la función **evaluar()** que evalúa un árbol en el punto especificado.

Fichero exsep07l.lex

```
%%
```

Fichero exsep07y.yac

```
%{
#include <stdlib.h>
#include <math.h>
#include <stdarg.h>
typedef struct _nodo{
    char tipo; // 'n'umero, 'x', 's'eno, 'c'oseno, 'e'xponencial, 'l'n, 'a'rctg, '/', '+', '-', '*'
    union {
        double valor;
        struct _nodo * hijo;
        struct {
            struct _nodo * hijo1;
            struct _nodo * hijo2;
        } hijos;
    } contenido;
} nodo;

// Esta función toma un número variable de argumentos después del tipo:
// - Ninguno si el tipo es 'x'
// - Un double si el tipo es 'n'
// - Un puntero a nodo si el tipo es una función matemática
// - Dos punteros a nodo si el tipo es una operación aritmética
nodo * crearNodo(char tipo, ...){
    va_list ap;
    nodo * retorno = (nodo *)malloc(sizeof(nodo));
    va_start(ap, tipo);
    retorno->tipo=tipo;
    switch(retorno->tipo){
        case 'n':    {  retorno->contenido.valor=va_arg(ap, double); break; }
```

```
        case 'x':     {   break; }
        case 's':     {   ; }
        case 'c':     {   ; }
        case 'e':     {   ; }
        case 'l':     {   ; }
        case 'a':     {   retorno->contenido.hijo=va_arg(ap, nodo *); break; }
        case '/':     {   ; }
        case '+':     {   ; }
        case '-':     {   ; }
        case '*':     {
                          retorno->contenido.hijos.hijo1=va_arg(ap, nodo *);
                          retorno->contenido.hijos.hijo2=va_arg(ap, nodo *);
                          break;
                      }
    }
    va_end(ap);
    return retorno;
}

// Evaluación de un árbol de nodos en un punto
double evaluar(nodo * arbol, double punto){
    if (arbol == NULL) return 0.0;
```



```
}

    nodo * estructuraAEvaluar;

%}

%union{
```

```
}
%token
%type
%token SENO COSENO LN EXP ARCTG
%left ‘+’ ‘-‘
%left ‘*’ ‘/’
%%
s   :   e ';' {                                         } lp
    ;
lp  :   NUMR        {                                                   }
    |   lp ';' NUMR {                                                   }
    ;
e   :   NUMR        {

                    }
    |   'x'         {

                    }
    | SENO '(' e ')'    {

                    }
    | COSENO '(' e ')'  {

                    }
    | LN '(' e ')'      {

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

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

                    }
    | e '-' e                 {

                    }
    | e '*' e                 {

                    }
    | e '/' e                 {

                    }
    | '(' e ')'               {

                    }
    ;
    %%

    void main(){
        yyparse();
```

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