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 2006
3er Curso de I.T. Informática de Sistemas.

Apellidos, Nombre: _____

Calificación: _____

TEÓRICO

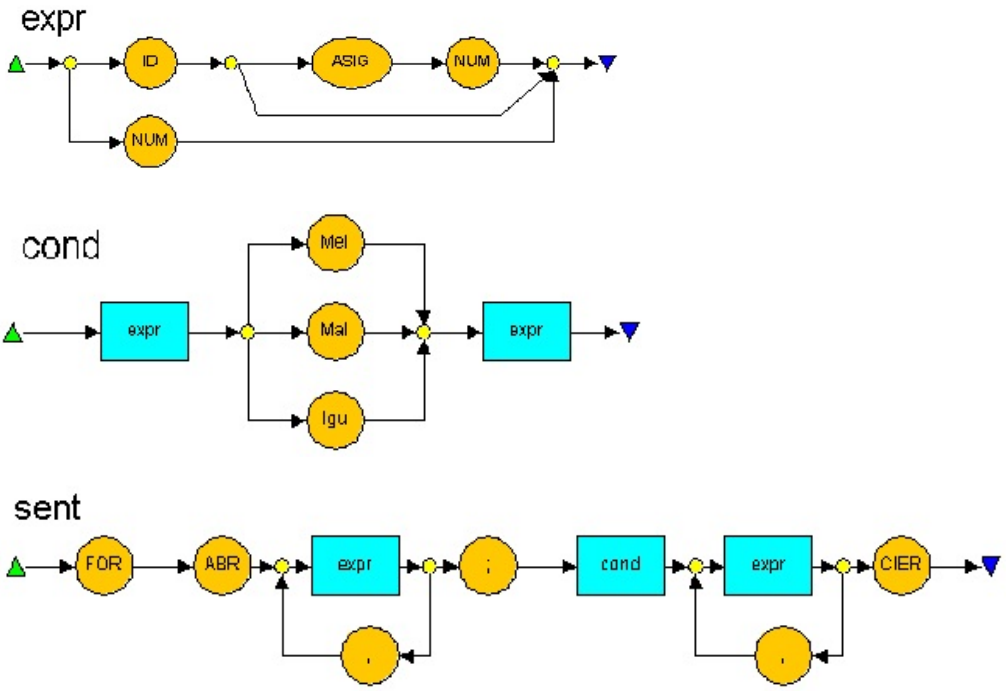
1.- Dada la gramática:

```
s   :  I B s E s
    |  I B s
    |  a
    ;
a   :  a O a O K
    |  a O a
    |  K
    ;
```

escribir todos los árboles sintácticos diferentes que reconocen la sentencia:

I B I B K O K O K E I B K

2.- Dados los diagramas de sintaxis siguientes



se pide codificar en C las funciones recursivas que implementan su análisis sintáctico descendente.

3.- Dada la gramática:

```
❶ expr    :  ID ASIG NUM
❷         |  NUM
❸         |  ID
          ;
❹ cond    :  expr MaI expr
❺         |  expr MeI expr
❻         |  expr Igu expr
          ;
❼ sent    :  FOR '(' listaExpr ';' cond ';' listaExpr ')'
          ;
❽ listaExpr     :  expr
❾               |  listaExpr ',' expr
                ;
```

se pide suministrar el *parse* derecho que reconoce la sentencia (suponiendo que el axioma inicial es **sent**):

```
FOR ( ID ASIG NUM , NUM ; NUM MaI ID ; ID , ID )
```

4.- Sea la gramática:

```
decl   :  ID declP
       ;
declP  :  ':' tipo ';'
       |  ',' decl
       ;
tipo   :  INT
       |  REAL
       |  RECORD listaDecl END RECORD
       ;
listaDecl    :     decl  listaDeclP
             ;
listaDeclP   :     ε
             |  decl ',' listaDecl
             ;
```

cuya tabla LL(1) es:

	ID	INT	REAL	RECORD	END	:	;	.	$
decl	decl --> ID declP								
declP						declP --> : tipo ;		declP --> , decl	
tipo		tipo --> INT	tipo --> REAL	tipo --> RECORD listaDecl END RECORD					
listaDecl	listaDecl --> decl listaDeclP								
listaDeclP	listaDeclP --> decl , listaDecl				listaDeclP --> EPS				

Se pide, mediante un análisis LL(1) con esta tabla, aceptar o rechazar la cadena:

ID , ID , ID : RECORD ID : INT ; ID , ID : REAL ; END RECORD ; $

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 2006
3er Curso de I.T. Informática de Sistemas.

Apellidos, Nombre: _____

Calificación: _____

PRÁCTICO

Partiendo del ejemplo de generación de código de tercetos en sentencias complejas visto en clase, se desean incluir dos modificaciones. La primera se refiere a PCYACC, y la segunda a Lex.

1) La primera modificación consiste en incluir una nueva sentencia de control, la sentencia LOOP, que tiene la siguiente sintaxis:

```
sent :    LOOP
                sent ';'
          EXIT WHEN cond ';'
               sent ';'
          END LOOP
    ;
```

Básicamente se trata de una sentencia iterativa en la que la primera sentencia se ejecuta siempre, al menos una vez; luego se evalúa la condición del **EXIT WHEN** que, si es cierta, finaliza el bucle pasando a la siguiente sentencia tras el **END LOOP**, y si es falsa continúa la ejecución de la segunda sentencia. Este proceso se repite hasta que la condición sea cierta momento en que se finaliza el bucle.

La ventaja de este tipo de bucles radica en que facilita la escritura de iteraciones sin repetir partes del código fuera del bucle en sí. Así, por ejemplo, el bucle:

```
LOOP {
     a := a+3;
}; EXIT WHEN a < 8; {
     a := a -2;
     b:=5;
}; END LOOP;
```

produciría el siguiente código de tercetos:

```
     label etq1
             tmp2 = 3;
             tmp3 = a + tmp2;
             a = tmp3
             tmp4 = 8;
             if a < tmp4 goto etq2
             goto etq3
     label etq3
             tmp5 = 2;
             tmp6 = a - tmp5;
             a = tmp6
             tmp7 = 5;
             b = tmp7
             goto etq1
     label etq2
```

En este apartado se pide incluir las acciones semánticas oportunas en las reglas de la sentencia **LOOP** para que se genere el código pedido.

2) La segunda modificación respecta al código Lex. Lo que se pretende es permitir directivas de compilación que permitan definir *flags* y luego compilar código o no en función de si tales *flags* están activos o no. Las directivas que se permiten son:

`#define flag` Crea el flag de nombre *flag*.

`#undef flag` Destruye el flag de nombre *flag*.

`#ifdef flag` Si *flag* está creado entonces compila todo lo que viene a continuación hasta su correspondiente `#fi`. Si *flag* no está creado entonces se ignora todo lo que haya a continuación hasta el correspondiente `#fi`.

`#fi` Cierra un `#ifdef`.

Deben tenerse en cuenta las siguientes consideraciones:

- Un *flag* es, sencillamente, un identificador formado sólo por letras minúsculas.
- Se permite el uso de directivas `#ifdef` anidadas.
- Las directivas de compilación deben comenzar siempre en la primera columna.
- Para ignorar determinadas partes del código, lo más sencillo es que yylex() no envíe *tokens* a yyparse().

Por ejemplo, la siguiente tabla ilustra un código fuente y lo que debe generarse.

Código fuente	Código generado y comentarios

`k := 11;`
`w := 22;`
`******`
`******`
`#define tita`
`#ifdef tita`
`k:= 7;`
`#fi`
`#undef tita`
`#ifdef tita`
`k := 12;`
`#fi`
`******`
`#ifdef tita`
`#define phi`
`k := 56;`
`#fi`
`#ifdef phi`
`k := 63;`
`#fi`
`******`
`#define gamma`
`#ifdef tita`
`k := 108;`
`#ifdef gamma`
`k := 99;`
`#fi`
`k := 207;`
`#fi`
`******`
`k := 199;`
	`tmp20 = 11;`
	`k = tmp20;`
	`tmp21 = 22;`
	`w = tmp21;`
	`* tita está definida`
	
	`tmp22 = 7;`
	`k = tmp22;`
	`* tita ya no está definida`
	`* este código no se compila`
	
	`* este código no se compila`
	`* phi tampoco se define`
	
	`* este código no se compila`
	
	`* gamma está definida`
	`* este código no se compila...`
	
	`* ... ni nada de lo que hay dentro`
	`* hasta el correspondiente #fi`
	
	`* Aquí acaba el #ifdef y`
	`* lo siguiente sí se compila`
	`tmp23 = 199;`
	`k = tmp23;`

En este apartado se pide modificar el fichero Lex visto en clase para adaptarlo a las directivas de compilación propuestas en este enunciado. El comportamiento debe ser el aquí indicado.