PROCESADORES DE LENGUAJE (LABORATORIO) *Junio-2009*
4º Curso Ingeniería Informática Tiempo: 2 horas
Nombre: Apellidos: Grupo:

(3 ptos) Pregunta 1:

Dado el conjunto de producciones de una gramática reducida de la práctica 1 que permitía generar cabeceras del protocolo de comunicación de mensajes de correo electrónico entre máquinas:

```
Conexion → MailFrom Recpto Data Quit
MailFrom → TKN_MAIL TKN_FROM Dir
Recpto → TKN_RCPT TKN_TO Dir
Data → TKN_DATA TKN_ID
Quit → TKN_QUIT TKN_221
Dir → TKN_ID TKN_@ TKN_ID TKN_PTO TKN_ID
```

Se pide:

a. (1 ptos) Modifica la gramática para que podamos tener una lista de direcciones a la que enviar el correo. Cada dirección vendrá precedida por **TKN_RCPT TKN_TO**. Amplíala para que las direcciones puedan estar formadas por dos o tres campos después del TKN_@. Recuerda que la gramática resultante debe ser LL(1).

Sobre la gramática modificada haz los siguientes apartados:

b. (1,5 ptos) Implementa los procedimientos según el método de análisis descendente recursivo predictivo. Incluyendo la recuperación en modo de pánico. En caso de error, se debe indicar el token que se ha encontrado y lo que se esperaba en ese momento.
c. (0,5 ptos) Implementa un filtro que vuelque a un fichero texto la dirección de quién lo envía y las direcciones de destino, de forma que ignore todos los destinatarios cuya dirección sea gmail.com.

(3 ptos) Pregunta 2:

Dada la siguiente gramática extraída de la práctica 2 donde se ha considerado sólo la parte de declaraciones de procedimientos y de funciones.

```
Programa    → Subrutinas Bloque
Subrutinas  → Funcion Subrutinas | Proced Subrutinas | ε
Proced      → TKN_SUB TKN_ID TKN_( Args TKN_) Bloque TKN_SUB
Funcion     → TKN_FUNCT TKN_ID TKN_( Args TKN_) TKN_AS Tipo Bloque TKN_FUNCT
Args        → TKN_ID DeclaraVector TKN_AS Tipo MasArgs | ε
DeclaraVector     → TKN_( TKN_) | ε
MasArgs     → TKN_, TKN_ID DeclaraVector TKN_AS Tipo MasArgs | ε
Tipo        → TKN_INTEGER | TKN_SINGLE | TKN_BOOLEAN
```

a. (0,5 ptos.) Escribe la **estructura** de tabla de símbolos que vas a necesitar aquí (y que será una versión reducida de la utilizada en la práctica) y los **prototipos** de las funciones para el manejo de la tabla de símbolos.

b. (1 ptos.) Inserta en la gramática las acciones necesarias para rellenar la tabla de símbolos con la información sobre las funciones y procedimientos. Se debe incluir: el número y nombre de parámetros, tipo de los parámetros, si se trata de vectores o variables simples y sobre el tipo de valor de retorno.

c. (1 ptos.) Realiza las siguientes comprobaciones semánticas antes de insertar símbolos en la tabla de símbolos:

   i. Los identificadores de funciones/procedimientos son únicos. No puede haber ningún argumento con el mismo nombre que una función/procedimiento, independientemente del ámbito que tenga.

   ii. No se puede insertar un identificador de un argumento dos veces con el mismo nombre y ámbito (aunque sí con ámbitos distintos).

d. (0,5 ptos.) Haciendo uso de las variables dólares $ de Bison para poder pasar información entre producciones, implementa una comprobación semántica que obligue a que las funciones/procedimientos no tengan más de 10 parámetros.

NOTA: No hay que implementar las funciones de manejo de la tabla de símbolos, únicamente sus prototipos.

(4 ptos) Pregunta 3:

Dada la versión muy reducida de la gramática de la práctica 3:

```
Programa        → Bloque
Bloque          → Bdeclaraciones Bsentencias
Bdeclaraciones  → Declara Bdeclaraciones | ε
Declara         → TKN_DIM TKN_ID MasVariables TKN_AS Tipo
Tipo            → TKN_INTEGER | TKN_STRING
MasVariables    → TKN_, TKN_ID MasVariables | ε
Bsentencias     → TKN_BEGIN Instruccion TKN_; MasInstruccion TKN_END
Instruccion     → TKN_ID Opcion |
                  TKN_IF BoolExp TKN_THEN Bsentencias TKN_ENDIF |
                  TKN_IF BoolExp TKN_THEN Bsentencias TKN_ELSE Bsentencias
                TKN_ENDIF
MasInstruccion  → Instruccion TKN_; MasInstruccion | ε
Opcion          → TKN_= Exp
Exp             → TKN_ID | TKN_NUMENT | TKN_MENOS TKN_NUMENT | TKN_MENOS TKN_ID
                  | Exp TKN_ADD Exp | Exp TKN_MULT Exp
BoolExp         → Exp TKN_< Exp
```

a) (1,0 ptos.) Compacta la gramática (teniendo en cuenta el apartado b) y modifícala si es necesario para tener en cuenta la precedencia de operadores escribiéndola dentro de la clase del PCCTS MiParser. El operador * tiene mayor prioridad que el operador +.

b) (1,0 ptos.) Introduce los operadores para la creación del árbol sintáctico de forma que, para el siguiente ejemplo, se genere exactamente el árbol que se muestra a continuación. Modifica la forma de las producciones según sea necesario.

```
DIM a,b,c AS INTEGER
BEGIN
IF a < b  THEN
    BEGIN
              a =  a + b*2;
    END
ELSE
    BEGIN
            a= -a;
             a = b + a;
            a = a * -1;
    END
ENDIF;
END
```

Recorrido en pre-orden del árbol:

```
(INICIO (BEGIN (IF (< a b )(THEN (BEGIN (= a (+ a (* b 2 ))))))(ELSE
(BEGIN (= a (UNARIO a ))(= a (+ b a ))(= a (* a (UNARIO 1 )))))))END )
```

c) (2 ptos.) Implementa toda la función `genera_codigo` en PCCTS de forma que saque el siguiente código intermedio (el mismo que en la práctica excluyendo la cuenta de cuádruplos y variables):

IS_LESS t0 a b //corresponde al <
NOT t1 t0 NULL //esto es del IF
GOTO_REL t1 6 NULL //salta 6 instrucciones para ir al ELSE
ASSIG_C t2 2 NULL //aquí ya empiezan las sentencias del IF
MULT t3 b t2
ADD t4 a t3
ASSIG_V a t4 NULL
GOTO_REL 9 NULL NULL //salta 9 instrucciones para salir del IF_ELSE y solo estaba si había ELSE
NEG t5 a NULL //aquí empiezan las sentencias del ELSE
ASSIG_V a t5 NULL
ADD t6 b a
ASSIG_V a t6 NULL
ASSIG_C t7 1 NULL
NEG t8 t7 NULL ////corresponde al menos unario
MULT t9 a t8
ASSIG_V a t9 NULL
HALT NULL NULL NULL