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 2000.
3er Curso de I.T. Informática de Sistemas.

Apellidos, Nombre: _____ ***Calificación:*** _____

PRÁCTICA

Se desea construir un intérprete de un lenguaje de programación que permite utilizar variables de dos tipos diferentes:

- Cadena de caracteres.
- Lógico.

En nuestro lenguaje **no** hay zona de declaraciones, de manera que el tipo de una variable se sabe por un carácter especial que se le añade al final de su identificador. Los caracteres especiales son los siguientes:

- Cadena de caracteres: $. ej. de identificadores: a$, h0$, auxiliar_000_$, aux$, etc.
- Lógico: @. Ej. de identificadores válidos: j@, jjj__jj@, j0igge6@, etc.

Se tienen las siguientes consideraciones:

- No se permiten variables con el mismo nombre y con diferente terminador. P.ej., no se admite el tener dos variables como aux$ y aux@. Se recomienda que en la tabla de símbolos se introduzcan los nombres de los identificadores sin su carácter terminador.
- Nuestro lenguaje permite constantes de ambos tipos. Las constantes de tipo *cadena* van siempre entre comillas dobles, mientras que las de tipo lógico puede ser únicamente los literales *TRUE* y *FALSE*.
- Los comentarios deben ignorarse. Un comentario comienza siempre por el símbolo // (doble barra), y dura hasta el final de la línea.
- Se permiten asignaciones múltiples, de manera que una asignación también se considera una expresión cuyo tipo es el de su l-valor (esto es, el del identificador a la izquierda de la asignación).
- Hay dos tipos de sentencias: las que son expresiones, que simplemente calculan valores pero no sacan por pantalla ningún resultado, y la sentencia *PRINT* que saca el valor de una expresión por pantalla.

La gramática que reconoce nuestro lenguaje es la siguiente:

```
prog    :       prog sent ';'
        |       prog error ';'
        |       /* Epsilon */
        ;
sent    :       expr
        |       PRINT expr
        ;
expr    :       ID
        |       CTE_LOGICA
        |       CTE_CADENA
        |       expr '+' expr
        |       '(' expr '?' expr ':' expr ')'
        |       expr '>' expr
        |       expr '=' expr
        |       expr AND expr
        |       NOT expr
        |       ID ASIG expr
        ;
```

La semántica de las diferentes operaciones es la siguiente:

- +: Si los dos operandos son de tipo *lógico*, se produce el OR lógico. Si uno de los operandos es del tipo *cadena*, entonces:
  - Si el otro es de tipo *cadena*, se produce una concatenación.
  - Si el otro es de tipo *lógico*, el *lógico* se traduce a la cadena “FALSE” o “TRUE” según su valor, y se concatenan.
- **?**: La primera expresión debe ser de tipo *lógico*. Si ésta vale *TRUE* entonces se devuelve la segunda expresión y s

- >: Se aplica lo dicho para el +, aunque si los dos son de tipo *lógico*, se supone que *FALSE* es menor que *TRUE*.
- =: Se aplica lo dicho para el +.
- **AND y NOT**: Los operadores sólo pueden ser de tipo *lógico*.
- **ASIG**: La asignación se representa por ":=". Si el *ID* es de tipo *cadena* se le puede asignar tanto una expresión de tipo *cadena* como de tipo *lógico*, siendo de aplicación lo ya dicho para el +. Si el *ID* es de tipo *lógico*, sólo puede asignársele otro valor de tipo *lógico*.
- Si se viola alguno de lospreceptos anteriores, se produce una expresión de tipo indefinido.

La tabla de símbolos es accedida a través de un t.a.d. (almacenado en el fichero **tabsim00.c**) como el visto comúnmente en clase, en el que se tienen los siguientes tipos:

```
typedef struct _tipoYValor{
        char    tipo; /* '$'-string, '@'-boolean, 'u'-undefined */
        union {
                char            valorCadena[50];
                short int       valorLogico;
        }       info;
} tipoYValor;
typedef struct _simbolo {
    struct _simbolo *       sig;
    char                nombre[20];
    tipoYValor          datos;
   } simbolo;
```

Las funciones de la tabla de símbolos son:

```
simbolo * crear();
void insertar(simbolo * * p_t,simbolo * s);
simbolo * buscar(simbolo * t, char nombre[20]);
```

Además se suministra la función **cambiaACadena**:

```
char * cambiaACadena(tipoYValor * y){
#define x (*y)
        if (x.tipo == 'u')
                strcpy(x.info.valorCadena, "Undefined");
        else if (x.tipo == '@')
                strcpy(x.info.valorCadena, (x.info.valorLogico)?"TRUE":"FALSE");
        x.tipo = '$';
        return x.info.valorCadena;
#undef x
}
```

Se pide:

- Construir los programas Lex y Yacc que permitan reconocer como entrada un texto con las características mencionadas y se coporte tal y como se ha explicado anteriormente. Nótese que la gramática dada es ambigua y es necesario eliminarle la ambigüedad mediante las directivas que Yacc posee a tal efecto. Debe construirse un intérprete que emita mensajes de error en los que se indique el número de línea del error, así como una correcta descripción del error producido.

Para aclarar dudas, a continuación se expone un ejemplo de entrada y la salida correspondiente:

// este es un programa de prueba para un examen.
 	
a$:="pepe";
d@:=TRUE;
 	
PRINT "a$ vale ";
PRINT a$;
 	
PRINT "d@ vale ";
PRINT d@;
 	
PRINT "a$+d@ vale ";
PRINT a$+d@;
 	
PRINT "d@+TRUE+FALSE vale";
PRINT d@+TRUE+FALSE;
 	
PRINT "' Adios'+'Hola' vale ";
PRINT " Adios"+"Hola";
 	
PRINT "(a$?'uno':'dos') vale ";
PRINT (a$?"uno":"dos");
 	
PRINT "(NOT d@?'Hola ':TRUE)+FALSE vale ";
PRINT (NOT d@?"Hola ":TRUE)+FALSE;
 	
PRINT "d@ * TRUE vale ";
PRINT d@ * TRUE;
 	
PRINT "a$ * TRUE vale ";
PRINT a$ * TRUE;
 	
PRINT "d@ AND TRUE vale ";
PRINT d@ AND TRUE;
 	
PRINT "NOT d@ vale ";
PRINT NOT d@;
 	
PRINT "NOT a$ vale ";
PRINT NOT a$;
 	
PRINT "v@ := w@ := d@ vale ";
PRINT v@ := w@ := d@;
 	
PRINT "w@ vale ";
PRINT w@;
 	
PRINT "v@ vale ";
PRINT v@;
 	
PRINT PRINT;
d$:="mal";
PRINT d@;
	a$ vale
	pepe
	
	d@ vale
	TRUE
	
	a$+d@ vale
	pepeTRUE
	
	d@+TRUE+FALSE vale
	TRUE
	
	' Adios'+'Hola' vale
	AdiosHola
	
	(a$?'uno':'dos') vale
	Undefined
	
	(NOT d@?'Hola ':TRUE)+FALSE vale
	TRUE
	
	d@ * TRUE vale
	TRUE
	
	a$ * TRUE vale
	Undefined
	
	d@ AND TRUE vale
	TRUE
	
	NOT d@ vale
	FALSE
	
	NOT a$ vale
	Undefined
	
	v@ := w@ := d@ vale
	TRUE
	
	w@ vale
	TRUE
	
	v@ vale
	TRUE
	
	syntax error
	d con un tipo diferente
	TRUE