



Apellidos, Nombre: \_\_\_\_\_

Calificación: \_\_\_\_\_

## PRÁCTICA

El lenguaje **Protolen** posee las siguientes características:

- Es un lenguaje compilado, aunque en nuestro caso nos limitaremos sólo a controlar los posibles errores semánticos..
- Dispone de los tipos entero y cadena de caracteres.
- Las expresiones de tipo entero pueden sumarse, restarse, multiplicarse y dividirse.
- Las expresiones de tipo cadena pueden concatenarse. Para ello se emplea el mismo símbolo que para sumar enteros. Es posible encadenar una cadena con un entero, pero no al revés.
- Un programa en lenguaje **Protolen** solo permite construir funciones, que deben constar de:
  - Nombre que puede estar sobrecargado.
  - Cero, uno o varios parámetros formales de los tipos arriba indicados.
  - Un tipo de valor retornado.
  - Una parte de declaración de variables locales.
  - Un cuerpo que sólo permite asignaciones, operaciones de E/S, llamadas a otras funciones con parámetros reales compatibles con los formales, y una sentencia de retorno de la función.
- Se permite la sobrecarga de funciones, esto es, funciones con el mismo nombre, pero con distintos parámetros.
- Dos funciones son diferentes si tienen distinto nombre o si el orden y/o tipo de sus parámetros es distinto.

La parte semántica de nuestro compilador debe:

- Controlar que en las operaciones aritméticas sólo intervienen expresiones enteras. **0,5p**
- Controlar que las concatenaciones de cadenas de caracteres son válidas. **0,5p**
- Controlar que las variables que se usen hayan sido previamente declaradas correctamente, o bien sean parámetros formales. **0,375p**
- Controlar que no hay dos funciones con el mismo nombre y los mismos parámetros en el mismo orden. **1p**
- Controlar que una función no tiene más de 20 parámetros. **1p**
- Controlar que una misma función no tiene parámetros formales ni variables locales con el mismo nombre. **0,375**
- Controlar que toda función es invocada con parámetros reales del mismo tipo, número y orden que los formales. **2p**
- Controlar que toda función tiene en su cuerpo, al menos, una sentencia **return**. **0,5p**
- Controlar que todas las sentencias **return** van seguidas de una expresión del mismo tipo que la función a que pertenece. **0,5p**

- PCLex debe eliminar comentarios. Estos comienzan por // y se extienden hasta el final de línea. **0,25p**

A continuación se muestra un ejemplo de programa fuente en Protolen así como los mensajes de error que se deben de emitir y las líneas donde residen los errores:

```

1 INTEGER FUNCTION Uno(STRING a, STRING b)
2 VAR
3     c, d : INTEGER;
4     e : STRING;
5 BEGIN
6     PRINT a;
7     RETURN 7;
8 END.
9
10 STRING FUNCTION Dos(INTEGER cont, STRING j)
11 VAR
12 BEGIN
13     PRINT cont;
14     PRINT j;
15     cont := 7;
16     j := "Literal de letras";
17     cont := Uno(j, j);
18     RETURN "casa";
19 END.
20
21 STRING FUNCTION Dos(STRING g)
22 VAR
23 BEGIN
24     RETURN "Sobrecarga";
25 END.
26
27 STRING FUNCTION const()
28 VAR
29 BEGIN
30     RETURN "Prueba de " + 2 + "concatenaciones.";
31 END.
32
33 STRING FUNCTION errores(INTEGER d, STRING d)
34 VAR
35     a, b : STRING;
36     c, d : INTEGER;
37 BEGIN
38     PRINT "a" + 7;           // Funciona bien.
39     PRINT 7 + "a";         // No se puede concatenar en ese orden.
40     PRINT "" + 7 + "a";    // Funciona bien.
41     PRINT "a" + (7 + "a"); // Mezcla de expresiones erroneas.
42     PRINT a + c;          // Funciona bien.
43     PRINT c + a;         // No se puede concatenar en ese orden.
44     PRINT "" + c + a;    // Funciona bien.
45     PRINT a + (c + a);   // Mezcla de expresiones erroneas.
46     RETURN 7;
47     RETURN 7*"a";
48 END.
49
50 STRING FUNCTION prueba(INTEGER cont, STRING cadena)
51 VAR
52     alfa, beta : INTEGER;
53 BEGIN
54     PRINT Dos(7, "taras");
55     INPUT alfa;
56     PRINT Dos(7*8, "probando"+ Dos(7, "bulba"));
57     beta := Uno(7, "a");
58     delta := Uno("", "a");
59     delta := 8*gamma;
60 END.

```

```

61 STRING FUNCTION prueba(INTEGER cont, STRING cadena)
62 VAR
63     alfa, beta : INTEGER;
64 BEGIN
65     RETURN "stachka";
66 END.
67
68
69 STRING FUNCTION demasiado(STRING a01, STRING a02, STRING a03, STRING a04, STRING a05,
70     STRING a06, STRING a07, STRING a08, STRING a09, STRING a10,
71     STRING a11, STRING a12, STRING a13, STRING a14, STRING a15,
72     STRING a16, STRING a17, STRING a18, STRING a19, STRING a20,
73     STRING a21, STRING a22)
74 VAR
75 BEGIN
76     PRINT prueba("a01", "a02", "a03", "a04", "a05", "a06", "a07", "a08", "a09", "a10",
77     "a11", "a12", "a13", "a14", "a15", "a16", "a17", "a18", "a19", "a20",
78     "a21", "a22"+"otro");
79     RETURN "a";
80 END.

```

Línea 33: variable d redeclarada.  
Línea 36: variable d redeclarada.  
Línea 39: expresion incorrecta.  
Línea 41: expresion incorrecta.  
Línea 43: expresion incorrecta.  
Línea 45: expresion incorrecta.  
Línea 46: la expresion retornada es incompatible con el tipo de la funcion.  
Línea 47: expresion incorrecta.  
Línea 47: la expresion retornada es incompatible con el tipo de la funcion.  
Línea 57: funcion Uno(ec) no declarada.  
Línea 57: expresion incorrecta.  
Línea 58: variable delta no declarada.  
Línea 59: variable gamma no declarada.  
Línea 59: expresion incorrecta.  
Línea 59: variable delta no declarada.  
Línea 60: falta RETURN en la función.  
Línea 62: funcion prueba redeclarada.  
Línea 73: no se pueden declarar mas de 20 parametros.  
Línea 73: no se pueden declarar mas de 20 parametros.  
Línea 78: no se admiten mas de 20 parametros.  
Línea 78: no se admiten mas de 20 parametros.  
Línea 78: funcion prueba(cccccccccccccccccc) no declarada.  
Línea 78: expresion incorrecta.

Tabla de funciones

```

-----
demasiado(cccccccccccccccccc)
prueba(ec)
errores(ec)
const()
Dos(c)
Dos(ec)
Uno(cc)

```

Se pide, partiendo de las tablas de símbolos dadas y del esqueleto de Yacc que se suministra:

- construir los programas Lex/Yacc
- las funciones de apoyo indicadas en Yacc
- así como la operación de búsqueda de funciones en la tabla de funciones.

que consigan el efecto enunciado.

## Fichero tbsjun04.c

```
#include <stdlib.h>
#include <stdio.h>
typedef struct _variable
{
    struct _variable * sig;
    char nombre[20];
    char tipo;
} variable;

void insertarVariable(variable * * p_t, char nombre[20], char tipo){
    variable * s = (variable *) malloc(sizeof(variable));
    strcpy(s->nombre, nombre);
    s->tipo = tipo;
    s->sig = (*p_t);
    (*p_t) = s;
}

variable * buscarVariable(variable * t, char nombre[20]){
    while( (t != NULL) && (strcmp(nombre, t->nombre) )
        t = t->sig;
    return (t);
}

void liberarVariables(variable * * p_t){
    variable * aux = (* p_t);
    while(aux != NULL){
        (*p_t) = aux;
        aux = aux->sig;
        free( (*p_t) );
    }
    (* p_t) = NULL;
}

typedef struct _funcion
{
    struct _funcion * sig;
    char nombre[20];
    char retorno;
    int numParametros;
    char parametros[21];
} funcion;

// Operación buscarFuncion
```

0,5p

```
void insertarFuncion(funcion * * p_t, char nombre[20], char parametros[21], char
retorno){
    funcion * s = (funcion *) malloc(sizeof(funcion));
    strcpy(s->nombre, nombre);
    s->retorno = retorno;
    s->numParametros = strlen(parametros);
    strcpy(s->parametros, parametros);
    s->sig = (*p_t);
    (*p_t) = s;
}
```

```

void imprimirFunciones(funcion * t){
    printf("\nTabla de funciones\n-----\n");
    while(t != NULL){
        printf("%s(%s)\n", t->nombre, t->parametros);
        t = t->sig;
    }
}

void imprimirVariables(variable * t){
    while(t != NULL){
        printf("%s\n", t->nombre);
        t = t->sig;
    }
}

```

## Fichero exjun04l.lex

%%

0,5p

## Fichero exjun04y.yac

```

%{
#include "tbsjun04.c"

int yylineno;
char mensaje[256];
variable * tablaVariables = NULL;
funcion * tablaFunciones = NULL;

char tipoFuncionActual; // Controla el tipo del valor que debe retornar una función
short hayReturn;       // Controla si una función contiene o no algún RETURN.

// Función de error que saca siempre el número de línea actual.
void yyerror(char *s){
    printf("Linea %d: %s\n", yylineno, s);
}

// Inserta una variable si no esta ya en la tabla de variables.
void controlarInsercionVariable(char nombre[20], char tipo){

```

0,35p

```
}
```

```
// Retorna el tipo de una variable. Si no está retorna 'i'ndefinido.  
char controlarUsoVariable(char nombre[20]){
```

```
0,35p
```

```
}
```

```
// Inserta una funcion en la tabla de funciones.  
// Dos funciones son distintas si tienen nombres y/o parametros distintos.  
void controlarInsercionFuncion(char nombre[20], char parametros[21], char tipo){
```

```
0,35p
```

```
}
```

```
// Retorno el tipo del valor de retorno de una funcion. si no está retorna  
'i'ndefinido.
```

```
char controlarUsoFuncion(char nombre[20], char parametros[21]){
```

```
0,35p
```

```
}
```

```
%}
```

```
%union{  
    char tipo;          // 'e'ntero, 'c'adena, 'i'ndefinido  
    char nombre[100];  
    struct {  
        char param[21];  
        int numParametros;  
    } parametros;  
}
```

```
0,6p
```

```
%left '+' '-'  
%left '*' '/'
```

```

%%
prog      :      /* Epsilon */
           |      prog funcion '.'
           |      {
           |      }
           |      prog error '.'
           |      {
           |      }

funcion   ;
funcion   :      tipo FUNCTION ID '(' parametros ')'
           |      {
           |      }
           |      VAR declaraciones BEGINN cuerpo END
           |      {
           |      }

parametros ;
parametros :      /* Epsilon */
           |      {
           |      }
           |      paramNoVacios
           |      {
           |      }

paramNoVacios ;
paramNoVacios :      unParametro
           |      {
           |      }
           |      paramNoVacios ',' unParametro
           |      {
           |      }

unParametro ;
unParametro :      tipo ID
           |      {

```

```

}

;
declaraciones      :      /* Epsilon */
                    |      declaraciones declaracion
                    ;
declaracion        :      ID ':' tipo ';'
                    {
                    }
                    |      ID ',' declaracion
                    {
                    }

;
tipo               :      STRING
                    {
                    }
                    |      INTEGER
                    {
                    }

;
cuerpo            :      /* Epsilon */
                    |      cuerpo sent ';'
                    |      cuerpo error ';' { yyerrok; }
                    ;
sent              :      PRINT exprFinal
                    |      INPUT ID
                    {
                    }
                    |      ID ASIG exprFinal {
                    }
                    }
                    |      RETURN exprFinal
                    {
                    }
                    }

;
expr              :      expr '+' expr {
                    }
                    |      expr '*' expr
                    {
                    }
                    |      expr '/' expr
                    {
                    }
                    |      expr '-' expr
                    {
                    }
                    |      '(' expr ')'
                    {
                    }
                    |      ID '(' expresiones ')'
                    {
                    }
                    |      ID
                    {
                    }
                    |      NUM
                    {
                    }
                    |      CADENA
                    {
                    }

;
expresiones       :      /* Epsilon */

```

```

    {
        |   exprNoVacias   {
    }
}
;
exprNoVacias :   exprFinal   {
    }
|   exprNoVacias ',' exprFinal   {
}
;
exprFinal :   expr   {
}
;

```

```

%%
#include "exjun041.c"
void main(){
    yylineno = 1;
    yyparse();
    imprimirFunciones(tablaFunciones);
}

```