Universidad de Málaga
Departamento de Lenguaje
Ciencias de la Computació
Campus de Teatinos, 29071 MÁL

Examen de Traductores, Intérpretes y Compiladores.
Convocatoria ordinaria de Junio de 2001
3er Curso de I.T. Informática de Sistemas.

Apellidos, Nombre: _____

Calificación: _____

PRÁCTICA

Con el presente problema, se pretende construir un **intérprete casi completo** de un lenguaje de programación **recursivo** que posee un juego de instrucciones muy reducido. En este lenguaje sólo se permite la creación de **funciones con al menos un parámetro** y posteriores **llamadas a dichas funciones**. Todos los parámetros y valores constantes son **enteros**, por lo que **no es necesario un control de tipos**.

La estructura básica de la definición de una función es:

```
FUNCION nombre(param1, param2, ..., paramn)
        CASO condición1 -> expr1
        CASO condición2 -> expr2
              :         :
        CASO condiciónn -> exprn
        OTRO CASO exprn+1
FIN FUNCION
```

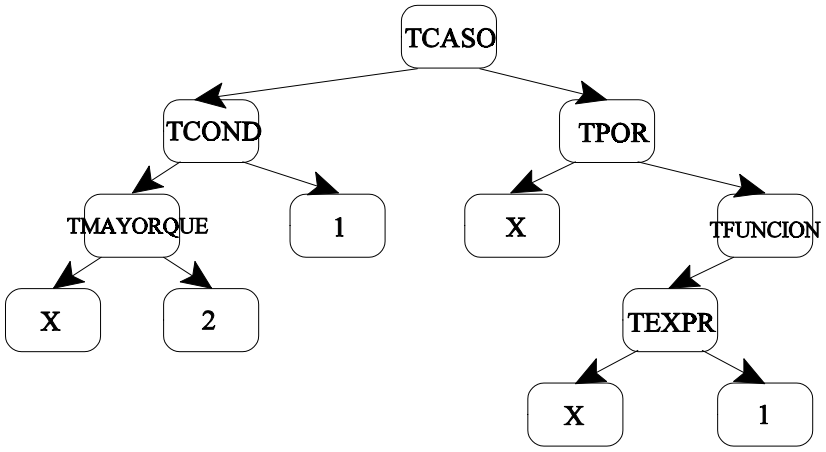
1. Función genérica

de manera que cuando se la invoca, esta función devuelve la $expr_1$ si se cumple la $condición_1$, o bien devuelve la $expr_2$ si se cumple la $condición_2$, y así sucesivamente. Si no se cumple ninguna de las condiciones entonces se devuelve la $expr_{n+1}$. El siguiente ejemplo ilustra la función *Factorial*:

```
FUNCION FACTORIAL(X)
        CASO X<2 -> 1
        OTRO CASO X*FACTORIAL(X-1)
FIN FUNCION;
```

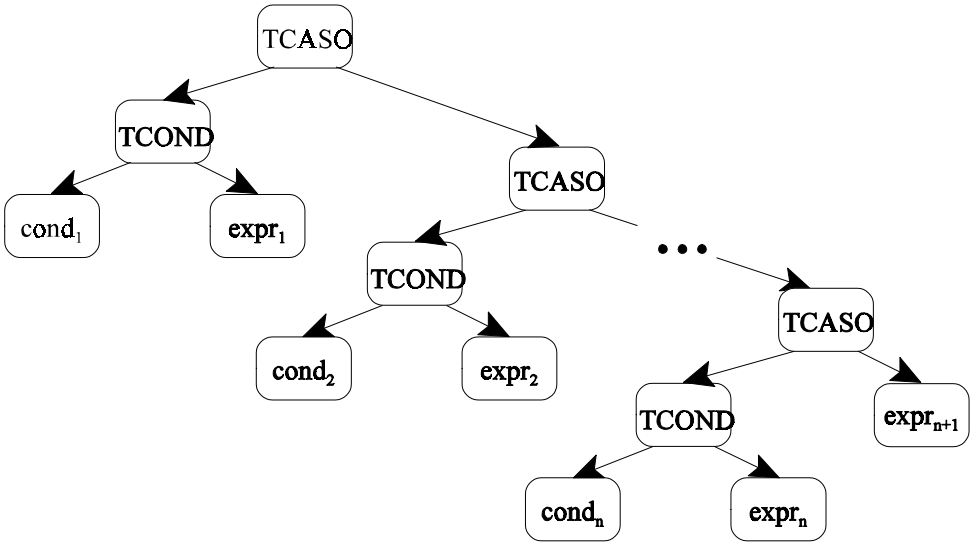
2. Función *Factorial*

Una vez definida una función, su código se almacena en una estructura en forma de árbol binario, de forma análoga a como se almacenan las expresiones aritméticas en un árbol binario, cuyo recorrido en inorden produce la expresión aritmética en notación infija. En concreto, el código asociado a la función *Factorial* tiene la siguiente estructura (**algunas cosas se refinarán más adelante**):

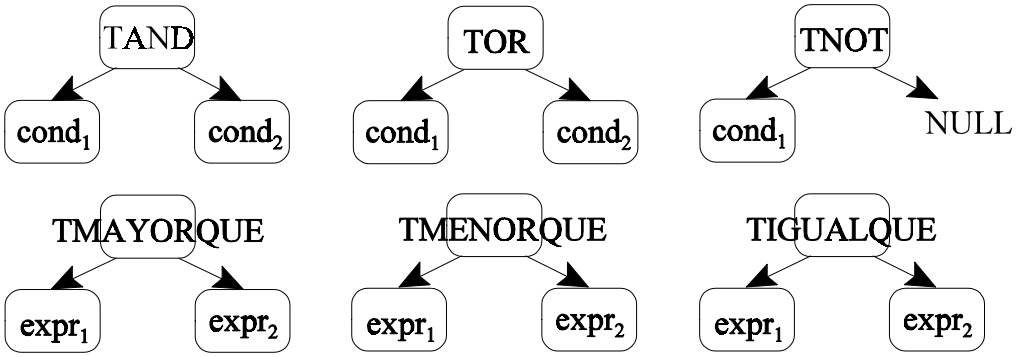


Open Course Ware
Gálvez Rojas, S.
(2011). Traductores, Compiladores e Intérpretes
OCW- Universidad de Málaga http://ocw.uma.es
Bajo licencia Creative Commons Attribution-Non-Comercial-ShareAlike
BY NC SA

En general, la estructura de una función como la de la figura 1, tiene la siguiente estructura completa:



donde las condiciones, a su vez, se representan como una de las siguientes posibilidades, según sea la condición:



En las expresiones se permite la aparición de constantes numéricas y de los parámetros de la función en la que se declaran. El formato de las expresiones es el que se ha comentado anteriormente para las expresiones infijas.

En concreto, para poder expresar bien las expresiones, nótese que los nodos hoja representan tanto identificadores como constantes numéricas. Las constantes se representarán con nodos etiquetado como TNUM que poseen además un campo con el valor de la constante. Sin embargo, los identificadores se representan mediante un nodo etiquetado TID que tiene además un campo con la **posición que dicho identificador ocupa en la lista de parámetros declarada en la cabecera de la función**.

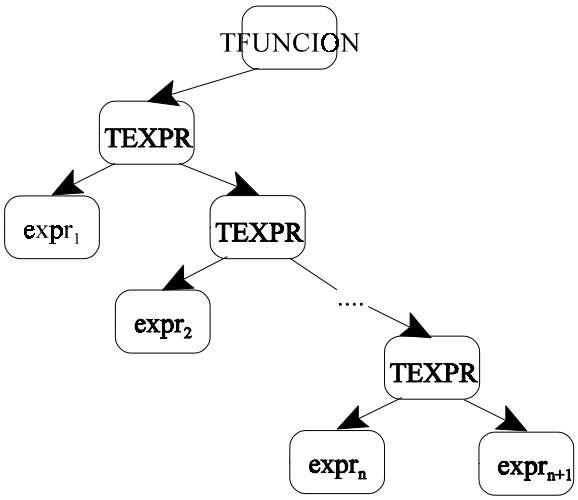
Sin embargo, dentro de una expresión puede aparecer una llamada a otra función, o bien una llamada recursiva a sí misma. En cualquiera de los dos casos, para representar una expresión que es una llamada a función, de la forma:

nombreFunción($expr_1$, $expr_2$, ..., $expr_n$, $expr_{n+1}$)

se utiliza un árbol como el de la figura.

Además, los nodos de tipo TFUNCION poseen un campo que apunta a la función que se pretende llamar.

Bajo licencia Creative Commons Attribution-Non-Comercial-ShareAlike



nos interesa saber su nombre, su número de parámetros, y un puntero al árbol que define su cuerpo. Así, el siguiente código de entrada define la estructura que aparece a continuación en la siguiente página:

```
FUNCION SUMATORIO(X, Y)
        CASO X>Y -> 0
        OTRO CASO SUMATORIO(X+1, Y)+X
FIN FUNCION;
FUNCION FACTORIAL(X)
        CASO X<2 -> 1
        OTRO CASO X*FACTORIAL(X-1)
FIN FUNCION;
FUNCION FIBONACCI(X)
        CASO X<2 -> 1
        OTRO CASO FIBONACCI(X-1)+FIBONACCI(X-2)
FIN FUNCION;
FUNCION GRAN_FUNCION(X)
        CASO 1=2 -> 0
        CASO 2=3 -> 0
        OTRO CASO FACTORIAL(X)+FIBONACCI(X)+SUMATORIO(1, X)
FIN FUNCION;
```

Para ello se suministran las siguientes estructuras:

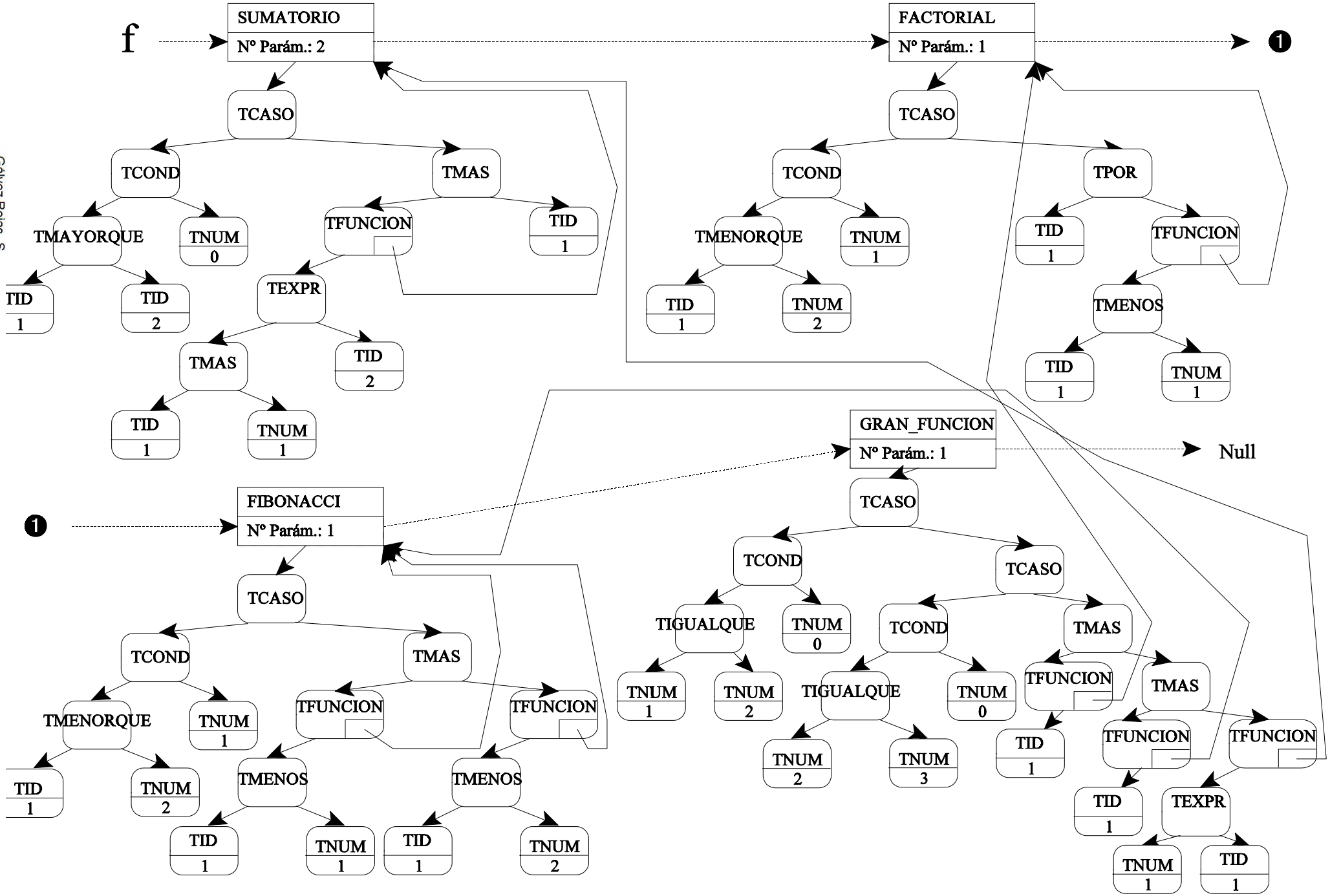
```
typedef struct _funcion {
        struct _funcion * sig;
        char nombre[20];
        struct _nodo * cuerpo;
        unsigned short numeroParametros;
} Funcion;

typedef struct _nodo {
        struct _nodo * izq;
        struct _nodo * dch;
        unsigned short tipo;
        int valor;
        Funcion * ptrFuncion;
} Nodo;

typedef struct _id {
        struct _id * sig;
        char nombre[20];
        short posicion;
} Id;
```

```
typedef struct _pila {
        struct _pila * sig;
        int valor;
} NodoPila;
```


Bajo licencia Creative Commons Attribution-Non-Comercial-ShareAlike



Open Course Ware
Gálvez Rojas, S. (2011). Traductores, Compiladores e Intérpretes
OCW- Universidad de Málaga http://ocw.uma.es
Bajo licencia Creative Commons Attribution-Non-Comercial-ShareAlike

Opción A.

Partiendo de los ejemplos de estructuras dadas, y de los registros de C indicados, así como de las funciones que se suministran en el fichero **auxiliar.c** del anexo, se pide

- Indicar cómo quedaría el árbol que define el cuerpo de la siguiente función (1 punto):

```
FUNCION NUEVA(X, Y)
        CASO X=Y -> X
        CASO X>Y -> NUEVA(X-Y, Y)
        OTRO CASO NUEVA(X, Y-X)
FIN FUNCION;
```

- Crear el árbol sintáctico que reconoce la función anterior a partir de la gramática dada en el esqueleto que se adjunta (0,5) puntos.
- Hacer los programas Lex/Yacc que, a partir de una función dada, genere el árbol que representa su cuerpo, e introduzca a tal función y su cuerpo asociado en la tabla de funciones (4,5 puntos). Para ello, se debe partir del esqueleto que se adjunta. Para ello se deben controlar los siguientes errores semánticos:

- Evitar redeclaraciones de funciones.
- Evitar nombres de parámetros repetidos (para una misma función).
- Controlar que cuando se llama a una función, ésta esté ya declarada. Hay que tener en cuenta que una función puede llamarse a sí misma, antes de que su cuerpo se haya compilado del todo.
- Toda función debe invocarse con el número adecuado de parámetros, ni más ni menos.
- Los únicos identificadores que pueden usarse en el cuerpo de una función son sus parámetros.

Nota: Nótese que los parámetros se introducen en una tabla local en la declaración de la función; tabla que debe destruirse una vez que finaliza la definición de dicha función.

Nota: Nótese que **no** se permiten declaraciones anidadas de funciones.

Nota importante: Nótese que en este ejercicio no se propone **nada acerca de la ejecución** de los árboles que definen el cuerpo de las funciones. Asimismo, la gramática que se adjunta **tampoco posee construcciones que permitan invocar a una función**. Por tanto, **tales aspectos son irrelevantes** desde el punto de vista del presente ejercicio.

Nota importante: El tipo *NodoPila* no se usa para nada si se elige esta opción del problema.

Bajo licencia Creative Commons Attribution-Non-Comercial-ShareAlike

Opción B.

En este apartado se pide implementar una función *evaluarCuerpo()* que, a partir de un cuerpo de función y de una pila de parámetros produzca el resultado final de la función y lo visualice por pantalla.

Para ello, **se suministra la gramática que permite realizar llamadas a funciones, pasando como parámetros sólo número naturales**.

Cuando se produce una llamada a una función, se colocan en la pila tantos números como parámetros tenga dicha función, y se colocan en orden, esto es, el primer parámetro en la cima de la pila, el segundo parámetro debajo, y así sucesivamente. En cuanto la función retorna, sus parámetros se quitan de la cima de la pila.

El árbol se debe evaluar siguiendo su estructura lógica, esto es, se retornará como resultado de la evaluación el valor de la primera expresión cuya condición asociada sea **True**. La propia función *evaluarCuerpo()* también se encarga de evaluar condiciones, devolviendo **1** si la condición se cumple y **0** en caso contrario.

Partiendo de los ejemplos de estructuras dadas, y de los registros de C indicados, así como de las funciones que se suministran en el fichero **auxiliar.c** del anexo, se pide

- Indicar cómo quedaría el árbol que define el cuerpo de la siguiente función (1 punto):

```
FUNCION NUEVA(X, Y)
        CASO X=Y -> X
        CASO X>Y -> NUEVA(X-Y, Y)
        OTRO CASO NUEVA(X, Y-X)
FIN FUNCION;
```

- Crear el árbol sintáctico que reconoce la función anterior a partir de la gramática dada en el esqueleto que se adjunta (0,5) puntos.
- Rellenar las acciones emánticas asociadas al trozo de gramática que permite invocar a una función, especificando además qué atributos poseen los terminales y no terminales implicados. (1,5 puntos). Para ello, se debe partir del esqueleto que se adjunta. Además se deben controlar los siguientes errores semánticos:
  - Evitar que la función no exista
  - Controlar que cuando se invoque a una función, el numéro de parámetros reales coincida con el de parámetros formales.
- Especificar en pseudocódigo la función *evaluarCuerpo()*. cuyo esqueleto se suministra más adelante.

Nota: En esta opción no se generan los cuerpos de las funciones, sino que **sólo** se evalúan.

Bajo licencia Creative Commons Attribution-Non-Comercial-ShareAlike

Anexo. Fichero **auxiliar.c**

```
typedef struct _nodo;

typedef struct _funcion {
        struct _funcion * sig;
        char nombre[20];
        struct _nodo * cuerpo;
        unsigned short numeroParametros;
} Funcion;

typedef struct _nodo {
        struct _nodo * izq;
        struct _nodo * dch;
        unsigned short tipo;
        int valor;
        Funcion * ptrFuncion;
} Nodo;

typedef struct _id {
        struct _id * sig;
        char nombre[20];
        short posicion;
} Id;

typedef struct _pila {
        struct _pila * sig;
        int valor;
} NodoPila;

/* Constantes para guardar en el campo tipo de un Nodo */
#define TINDEFINIDO     0
#define TCASO           1
#define TCOND           2
#define TAND            3
#define TOR             4
#define TNOT            5
#define TMAYORQUE       6
#define TMENORQUE       7
#define TIGUALQUE       8
#define TMAS            9
#define TMENOS          10
#define TPOR            11
#define TDIVIDIDO       12
#define TID             13
#define TNUM            14
#define TFUNCION        15
#define TEXPR           16

/* Crea un nodo de tipo Nodo */
Nodo * crearNodo(){
        Nodo * x;
        x = (Nodo *)malloc(sizeof(Nodo));
        x->izq = x-> dch = NULL;
        x->ptrFuncion = NULL;
        return x;
}
/* Crea un nodo de tipo NodoPila */
NodoPila * crearNodoPila(){
        NodoPila * x;
        x = (NodoPila *)malloc(sizeof(NodoPila));
        x->sig = NULL;
        return x;
}
/* Crea un nodo de tipo Id (nombre de parámetro) */
Id * crearId(){
        Id * x;
```


Bajo licencia Creative Commons Attribution-Non-Comercial-ShareAlike

```
        x = (Id *)malloc(sizeof(Id));
        x->sig = NULL;
        x->posicion = 0;
        return x;
}
/* Crea un nodo de tipo Funcion */
Funcion * crearFuncion(){
        Funcion * x;
        x = (Funcion *)malloc(sizeof(Funcion));
        x->sig = NULL;
        x->cuerpo = NULL;
        x->numeroParametros = 0;
        return x;
}
/* Inserta una función en la lista de funciones */
void insertarFuncion(Funcion ** f, Funcion * ptr)
/* Inserta un NodoPila en la pila f*/
void insertarPila(NodoPila ** f, NodoPila * ptr)
/* Inserta un parámetro en la tabla */
void insertarId(Id ** t, Id * ptr)
/* Crea la tabla de parámetros */
void crearTablaId(Id ** t)
/* Destruye la tabla de parámetros, que es local a la función */
void destruirTablaId(Id ** t)
/* Libera la memoria ocupada por la pila dada*/
void borrarPila(NodoPila ** p)
/* Busca un parámetro por su nombre */
Id * buscarId(Id * t, char * nombre)
/* Busca una función por su nombre */
Funcion * buscarFuncion(Funcion * t, char * nombre)
```


Bajo licencia Creative Commons Attribution-Non-Comercial-ShareAlike

Solución Opción A:

Programa Lex:

Programa Yacc:

```
%{
#include "auxiliar.c"
Id * t = NULL;
Funcion * f = NULL;
%}
%union {
	Funcion * ptrFuncion;
	Nodo * ptrNodo;
	Id * ptrId;
	char nombre[20];
	int numero;
	int cantidad;
	struct {
		Nodo * ptrNodo;
		short numeroExpresiones;
	} llamadaAFuncion;
}
```


Bajo licencia Creative Commons Attribution-Non-Comercial-ShareAlike

```
%%
prog    :       /* Epsilon */
                |       prog deffun ';'
                |       prog error ';' {                 }
                ;
deffun:         FUNCION ID '(' {


                        lispar  {


                                }
                        ')' casos FIN FUNCION       {


                                                        }
                ;
lispar:         id              {
                                        $$ = 1;
                                        $1->posicion = $$;
                                }
                | lispar ',' id {
                                        $$ = $1 + 1;
                                        $3->posicion = $$;
                                }
                ;
id              :       ID              {


                                        }
                ;
casos   :       uncaso OTRO CASO expr    {
```


Bajo licencia Creative Commons Attribution-Non-Comercial-ShareAlike

```
                                        }
                | uncaso casos   {

                                        }
                ;
uncaso:         CASO cond THEN expr     {

                                        }
                ;
cond    :       cond AND cond           {

                                        }
                | cond OR cond          {
                  Similar al anterior
                                        }
                | NOT cond              {

                                        }
                | '(' cond ')'          {
                                          $$ = $2;
                                        }
                | expr '>' expr         {

                                        }
                | expr '<' expr         {
                  Similar  al anterior
                                        }
                | expr '=' expr         {
                  Similar  al anterior
                                        }
                ;
expr    :       ID '(' lisexp ')'  {

```


Bajo licencia Creative Commons Attribution-Non-Comercial-ShareAlike

```
                        }
        |   expr '+' expr    {
                        }
        |   expr '*' expr    {
            Similar al anterior
                        }
        |   expr '-' expr    {
            Similar al anterior
                        }
        |   expr '/' expr    {
            Similar al anterior
                        }
        |   '(' expr ')'     {
                            $$ = $2;
                        }
        |   NUM              {
                            $$ = crearNodo();
                            $$->tipo = TNUM;
                            $$->valor = $1;
                        }
        |   ID               {
                        }
        ;
lisexp: expr                {
                        }
        |   expr ',' lisexp  {
```


Bajo licencia Creative Commons Attribution-Non-Comercial-ShareAlike

```
                                                    }
        ;
```

Solución Opción B.

Indicar los atributos de ID, lisnum y NUM, y rellenar los huecos siguientes. Para seleccionar los atributos, examinar el *%union* que se da en la opción A.

```
usefun:     ID '('          {

                            }
            lisnum ')'      {

                            }
        ;
lisnum:     NUM             {

                            }
        |       NUM ',' lisnum          {

                                        }
        ;
%%
/* Área de funciones */

```

```
/* Aqui viene la funcion principal, que se encarga de evaluar un cuerpo */
int evaluarCuerpo(Nodo * x, NodoPila ** p){
#define pila ((*p))
    switch (x->tipo) {
        case (TINDEFINIDO) : {          }
        case (TNUM)        : {          }
        case (TID)         : {

```


Bajo licencia Creative Commons Attribution-Non-Comercial-ShareAlike

```
                    }
        case (TFUNCION):    {

/* Evaluar parametros reales y meterlos en una pila auxiliar*/

/* Volcar resultados en la pila principal */

/* Invocar la funcion */

/* Desapilar */

                    }
        case (TCASO)    :   {

                    }
        case (TMAS)     :   {

                    }
        case (TPOR)     :   {

                    }
        case (TMENOS)   :   {

                    }
        case (TDIVIDIDO):   {

                    }
```


Bajo licencia Creative Commons Attribution-Non-Comercial-ShareAlike

```
                                        }
                case (TOR)          :   {

                                        }
                case (TNOT)         :   {

                                        }
                case (TMAYORQUE)        :       {

                                        }
                case (TMENORQUE)        :       {

                                        }
                case (TIGUALQUE)    :   {

                                        }
                default             :   {

                                        }
        }
#undef pila
}
```


Bajo licencia Creative Commons Attribution-Non-Comercial-ShareAlike