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

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

PRÁCTICA

Se pretende generar código intermedio correspondiente a llamadas a funciones, asignaciones simples, y expresiones aritméticas que sólo contienen la suma.

Tan sólo vamos a hacer un prototipo, en el sentido de que las funciones es necesario declararlas, pero ***su cuerpo no es necesario especificarlo*** en esta primera versión del prototipo.

Cuando nos encontramos con una llamada a una función de la forma $F(a_1, a_2, ..., a_n)$, el código intermedio asociado es sencillamente:

a_1
a_2
...
a_n
tmp_x = call F(n)

Se supone que los a_i se colocan sobre una pila y que luego son recogidos por la llamada *call F(n)* donde *n* es el número de parámetros sobre la pila con los que *F* debe trabajar. El resultado de la función se almacena en una variable tmp_x.

Un programa estará compuesto por una zona de declaraciones y otra de instrucciones. La zona de declaraciones permite declarar funciones (**FUNCTION**) y variables (**VARIABLE**), de manera que toda variable o función que se utilice posteriormente en la zona de instrucciones ha debido ser previamente declarada. Es incorrecto declarar más de una vez una variable o función.

Cuando se declara una función, su nombre se precede de la palabra reservada **FUNCTION**, a continuación el nombre de la función, seguido por el número de parámetros que acepta. No es necesario indicar el tipo de los parámetros dado que se supone que sólo trabajamos con números.

Hay una palabra reservada **FUNCTION** por cada función declarada.

La palabra reservada **VARIABLE** se utiliza para declarar variables, que se indicarán detrás de dicha palabra separadas por comas.

La sección de instrucciones comienza por la palabra **BEGIN** y acaba con **END** seguida de un punto (**.**). Enmedio sólo puede ponerse instrucciones de asignación simple. Como expresiones puede colocarse cualquier identificador, número, suma, o llamada a función. A su vez, las llamadas a funciones admiten como parámetros cualesquiera de estas expresiones.En otras palabras, es posible realizar composición de funciones a cualquier nivel.

Por último, en cualquier lugar del código es posible colocar comentario del tipo “hasta final de línea”, que comenzarán con el símbolo de final de admiración (**!**).

A continuación se dan unos cuantos ejemplos de las salida que se deben obtener en función de las entradas.

Ejemplo1:

Código fuente	Código generado
DECLARE FUNCTION F(2); VARIABLE X; BEGIN X = F(8, 23); END.	tmp1 = 8 tmp1 tmp2 = 23 tmp2 tmp3 = call F(2) X = tmp3

Ejemplo2:

Código fuente	Código generado
DECLARE FUNCTION F(2); VARIABLE X, Y; BEGIN Y = 53; X = F(X, F(Y, Y)); END.	tmp1 = 53 Y = tmp1 X Y Y tmp2 = call F(2) tmp2 tmp3 = call F(2) X = tmp3

Ejemplo3:

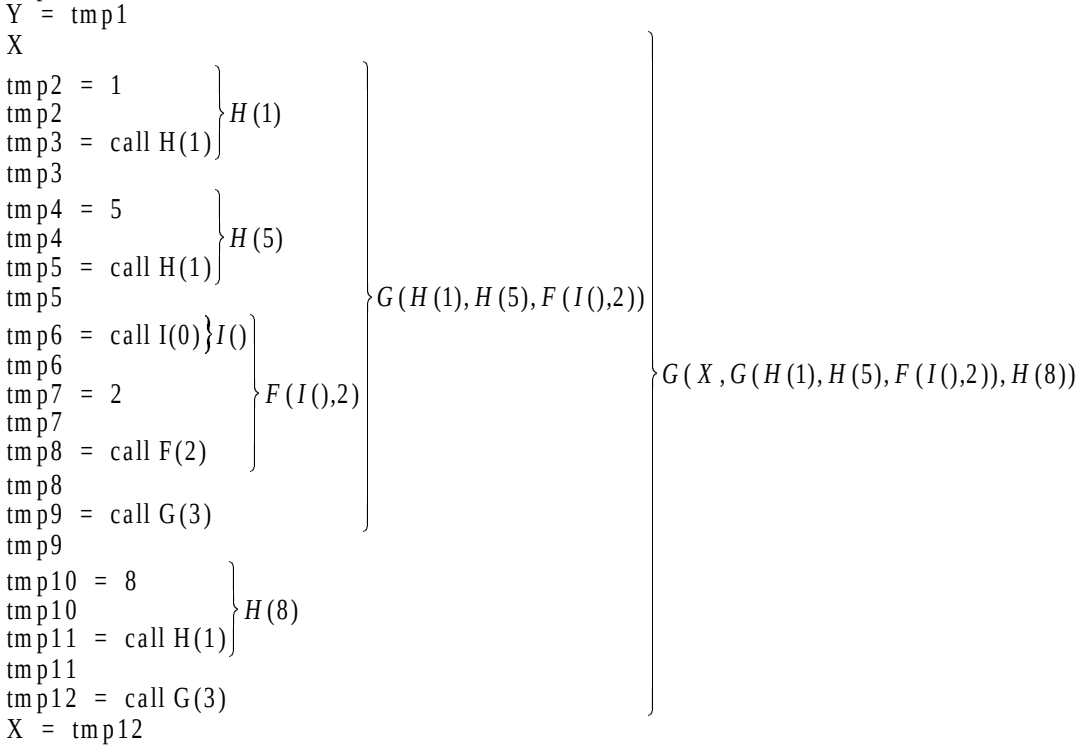
Código fuente	Código generado
DECLARE FUNCTION F(2); VARIABLE X, Y; BEGIN Y = 53+8; X = F(F(12, 45), X); END.	tmp1 = 53 tmp2 = 8 tmp3 = tmp1 + tmp2 Y = tmp3 tmp4 = 12 tmp4 tmp5 = 45 tmp5 tmp6 = call F(2) tmp6 X tmp7 = call F(2) X = tmp7

Ejemplo4:

Código fuente	Código generado

DECLARE FUNCTION F(2); FUNCTION G(3); FUNCTION H(1); FUNCTION I(0); VARIABLE X, Y; BEGIN Y = 53; X = G(X, G(H(1), H(5), F(I(), 2)), H(8)); END.	tmp1 = 53 Y = tmp1 X tmp2 = 1 tmp2 tmp3 = call H(1) tmp3 tmp4 = 5 tmp4 tmp5 = call H(1) tmp5 tmp6 = call I(0) tmp6 tmp7 = 2 tmp7 tmp8 = call F(2) tmp8 tmp9 = call G(3) tmp9 tmp10 = 8 tmp10 tmp11 = call H(1) tmp11 tmp12 = call G(3) X = tmp12

Para aclarar las cosas, el código generado anterior tiene la siguiente correspondencia con el fuente:



Ejemplo5:

Código fuente	Código generado
DECLARE FUNCTION F(2); FUNCTION F(2); FUNCTION F(5); VARIABLE X, Y; VARIABLE Y; BEGIN Y = T(); X = Y(); X = F(X); X = F(W); Z = 4; END.	F ya está declarada. F ya está declarada. Y ya está declarada. T no es una función. Y = T Y no es una función. X = Y X F tiene 2, y no 1 parametros X = F W no es una variable. W F tiene 2, y no 1 parametros X = F tmp1 = 4 Z no es una variable.

Se pide:

- Rellenar los siguientes esqueletos de Lex/Yacc que produzcan los resultados pedidos, y que sean congruentes con los ejemplos anteriores. Para ello, se tiene la siguiente tabla de símbolos:

TABSIM00.C

```
#include <stdlib.h>
#include <stdio.h>
typedef struct _simbolo
   {
    struct _simbolo * sig;
    char nombre[50];
    int tipo;
    int parametros;
   } simbolo;

simbolo * ts_crear()
{
 return NULL;
};

void ts_insertar(p_t, id, tipo, parametros)
simbolo * * p_t;
char * id;
int tipo;
int parametros;
{
simbolo * s;
```

```
  s = (simbolo *) malloc(sizeof(simbolo));
  strcpy(s->nombre, id);
  s->tipo = tipo;
  s->parametros = parametros;
  s->sig = (*p_t);
  (*p_t) = s;
}

int ts_buscarTipo(t, nombre)
simbolo * t;
char nombre[50];
{
  while( (t != NULL) && (strcmp(nombre, t->nombre)) )
      t = t->sig;
  return (t)?t->tipo:0;
}

int ts_buscarParametros(t, nombre)
simbolo * t;
char nombre[50];
{
  while( (t != NULL) && (strcmp(nombre, t->nombre)) )
      t = t->sig;
  return (t)?t->parametros:0;
}
```

EXDIC00L.LEX

```
%%
```

EXDIC00Y.YAC

```
%{
#include "tabsim00.c"
```



```
%}

%union{
        char nombre[50];
        int numero;
}
```



```
%%
prog    :       DECLARE lista_decl BEGIN lista_asig END '.'
                ;
lista_decl      :       /* Epsilon */
                |       lista_decl decl ';'
                |       lista_decl error ';' {                  }
                ;
decl    :       FUNCTION ID '(' NUMERO ')'
                        {
```



```
                        }
                |       VARIABLE lista_id
                ;
lista_id        :       ID      {
```

```
                        }
            |           lista_id ',' ID    {

                                            }
                        ;
lista_asig      :       /*Epsilon */
                |       lista_asig asig ';'
                |       lista_asig error ';' {                          }
                ;
asig    :       ID      '=' expr   {

                        }
        ;
expr    :       expr '+' expr      {

                        }
        |       ID '(' lista_expr ')'      {

                                }
        |       NUMERO              {

                                }
        |       ID                  {

                                }
                ;
lista_expr      :       /* Epsilon */           {}
                |       lista_no_vacia_expr     { }
                ;
lista_no_vacia_expr    :       expr            {
```

```
                                        }
                    |       lista_no_vacia_expr ',' expr    {

                                                            }
                    ;
%%
```