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

Apellidos, Nombre: _____

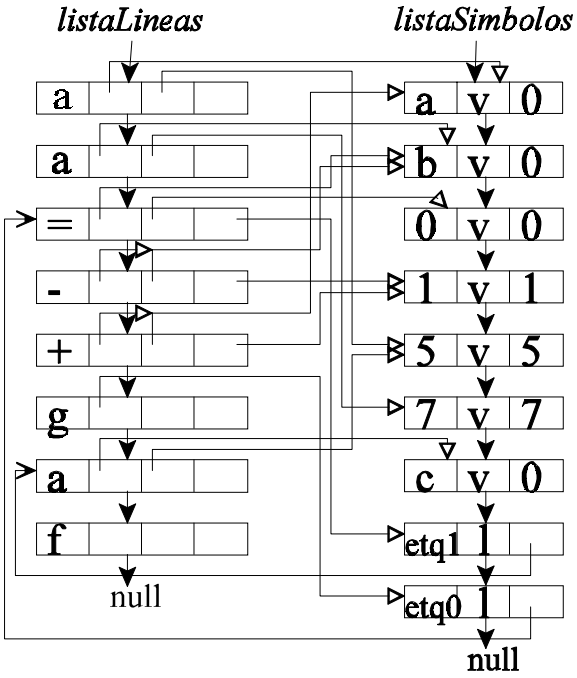
Calificación: _____

PRÁCTICA

Para comprobar el correcto funcionamiento del código de tercetos generado por los ejercicios realizados en clase y otros propuestos, se pretende realizar un intérprete de código de tercetos. El objetivo consiste en tomar como entrada una secuencia de tercetos, y generar una lista de *líneas*, donde cada *línea* es una estructura formada por cuatro elementos: una operación, el operando que recibe el resultado, y los operandos que intervienen en la operación. Por ejemplo, para la entrada siguiente:

```
        a = 5
        b = 7
label etq0:
        if b = 0 goto etq1
        b = b - 1
        a = a + 1
        goto etq0
label etq1:
        c = 5
```

se generaría la estructura que aparece en la figura adjunta.



Los distintos operandos de una *línea* son punteros a *símbolos* de la tabla de símbolos, de manera que en la tabla de símbolos se incluyen tanto las variables (de usuario y temporales), como los números, e incluso las etiquetas de salto. La estructura de una línea es:

```
typedef struct _linea{
        char operacion; // +, -, *, /, >, <, =, (a)signacion
simple, (g)oto, (f)in
        union{
                simbolo * lValor;
                simbolo * simbLabel;
        } uni;
        simbolo * rValor1;
        simbolo * rValor2;
        struct _linea * ptrSig;
} linea;
```

donde *operacion* denota el tipo de terceto de que se trata:

- (+, -, *, /) si es una asignación de una expresión.
- (a) si es una asignación simple
- (>, <, =) si es un if. Sólo se pueden utilizar los operadores relacionales >, < e =.
- (g) si es un goto.
- (f) reservado para denotar el final de la lista de *lineas*.

Como puede verse en el dibujo, no todos los tercetos hacen uso de los tres operandos. Por ejemplo, la (a)signación simple sólo utiliza dos operandos.

La estructura de un símbolo es la siguiente:

```
typedef struct _simb{
        char nombre[26];
        short int tipo; // l-label, v-variable
        union {
```

```
            int valor;
            struct _linea * label;
        } uni;
        struct _simb * ptrSig;
    } simbolo;
```

Un símbolo puede denotar una etiqueta de salto, o bien una variable (consideramos que los números enteros constantes también son variables, para facilitar el trabajo). Si el símbolo es una variable se almacenará su valor, mientras que si es una etiqueta, contendrá un puntero a la línea que sigue a dicha etiqueta en el código de tercetos fuente.

Una vez construidas las estructuras de la figura anterior, la función *ejecutar()* se encarga de ir recorriendo la *listaLineas*, e ir interpretando cada *línea*, como si se tratase de un contador de programa, esto es, efectuando las operaciones, las asignaciones, y evaluando las comparaciones, realizando los saltos si las comparaciones son satisfactorias, y continuando la ejecución secuencial en caso contrario.

Al final del todo, la operación *visualizar()* se encargará de emitir por pantalla el contenido de todas las variables contenidas en la tabla de símbolos.

Se pide:

- Construir los programas Lex y Yacc que permitan reconocer como entrada un bloque de tercetos y obtenga una estructura en memoria como la de la figura adjunta, haciendo uso de las funciones de tabla de símbolos que se proporcionan.
- Construir una función en C llamada *ejecutar()* que, a partir de la estructura en memoria previamente generada, realice la ejecución de las *lineas*, y que, por último, llame a la función *visualizar()* de la tabla de símbolos que se encarga de visualizar todo el contenido de la tabla de símbolos.

La gramática es la siguiente:

```
programa    :   g
            |   programa sent CR
            |   programa error CR
            ;
sent        :   ID '=' ID
            |   ID '=' ID op ID
            |   IF ID opRel ID GOTO ID
            |   GOTO ID
            |   LABEL ID ':'
            ;
op          :   '+'     |   '-'     |   '*'     |   '/'
            ;
opRel       :   '>'     |   '<'     |   '='
            ;
```

y la tabla de símbolos de que se dispone es como sigue:

```
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>

struct _linea;

typedef struct _simb{
    char nombre[26];
    short int tipo; // l-label, v-variable
    union {
        int valor;
        struct _linea * label;
    } uni;
    struct _simb * ptrSig;
```

```
} simbolo;

simbolo * listaSimbolos = NULL;

void insertar(simbolo * nuevo){
	nuevo->ptrSig = listaSimbolos;
	listaSimbolos = nuevo;
}

simbolo * buscar(char * nombre){
	simbolo * aux = listaSimbolos;
	while ((aux != NULL) && (strcmp(aux->nombre, nombre)))
		aux = aux->ptrSig;
	return aux;
}

typedef struct _linea{
	char operacion; // +, -, *, /, >, <, =, (a)signacion simple, (g)oto, (f)in
	union{
		simbolo * lValor;
		simbolo * simbLabel;
	} uni;
	simbolo * rValor1;
	simbolo * rValor2;
	struct _linea * ptrSig;
} linea;

linea * listaLineas = NULL;
linea * actualLinea = NULL;

linea * insertarLinea(){
	linea * nuevo = (linea *)malloc(sizeof(linea));
	nuevo->ptrSig = NULL;
	if (actualLinea == NULL)
		listaLineas = nuevo;
	else
		actualLinea->ptrSig = nuevo;
	actualLinea = nuevo;
	return nuevo;
}

void visualizar(){
	simbolo * aux = listaSimbolos;
	while (aux != NULL){
		if (aux->tipo != 'l') printf("%s: %d\n", aux->nombre, aux->uni.valor);
		aux = aux->ptrSig;
	}
}
```