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 2002
3er Curso de I.T. Informática de Sistemas.

Apellidos, Nombre: _____

Calificación: _____

PRÁCTICA

Se desea construir un **intérprete** mediante el cual es usuario pueda trabajar con matrices de **dos** dimensiones. El intérprete reconocerá una zona de declaraciones en la que se declaran las matrices, de la forma *numCols*numFilas ID, ID, ...;*. La zona de declaraciones comenzará con la palabra reservada *DECLARE*. Una vez finalizadas todas las declaraciones, comienza la zona de sentencias.

El sistema admite tres tipos de sentencias:

- Sentencias de inicialización. Permiten asignar valores constantes a cada componente de una matriz dada. Para ello, se dispone de dos tipos de asignaciones:
  - Por componente. Es de la forma *ID[col, fila]=NUM;* que asigna la constante *NUM* al elemento de *ID* situado en la posición *[col, fila]*.
  - Por fila. Es de la forma *ID[fila]={NUM, NUM, ..., NUM};* y asigna a una fila completa de la matriz *ID*. La longitud de la lista de números debe coincidir obligatoriamente con el número de columnas de *ID*.
- Sentencias de manipulación. Permite asignar a una matriz un cálculo entre matrices. Las operaciones que pueden hacerse entre matrices son la suma y el producto, aunque por razones de simplicidad, en este ejercicio **sólo** se pide la **suma**. Para poder sumar dos matrices, éstas han de poseer las mismas dimensiones.
- Sentencia de visualización. Permite visualizar una matriz por pantalla. En cada columna debe aparecer una fila completa de la matriz.

Un ejemplo de entrada válida, y su salida correspondiente sería (nótese que la línea horizontal **no** forma parte de la salida; se ha colocado ahí sólo para distinguir cuando acaba de imprimirse una matriz y empieza otra):

Entrada:	**Salida:**
`DECLARE`	
`  2*3 a, b, c;`	
`  3*2 k, l;`	
`  2*2 v;`	
`  3*3 u;`	
`BEGIN`	
`  a[1]={4, 5};`	
`  a[2]={1, 2};`	
`  a[3]={3, 8};`	
`  a[2,3]=3;`	
`  PRINT a;`	`4 5`
`  b[1]={4, 5};`	`1 2`
`  b[2]={1, 2};`	`3 3`
`  b[3]={3, 3};`	
`  c=a+b;`	
`  PRINT c;`	`8 10`
`  k[1]={1, 2, 3};`	`2 4`
`  k[2]={3, 2, 1};`	`6 6`
`  PRINT k;`	`1 2 3`
`  u=a*k;`	`3 2 1`
`  PRINT u;`	`19 18 17`
`  v=k*a;`	`7 6 5`
	`12 12 12`
`  PRINT v;`	`15 18`

```
v[1]={1, 2};                    17 22
v[2]={3, 3};
v=v+k*a;
PRINT v;                        16 20
                                20 25
```

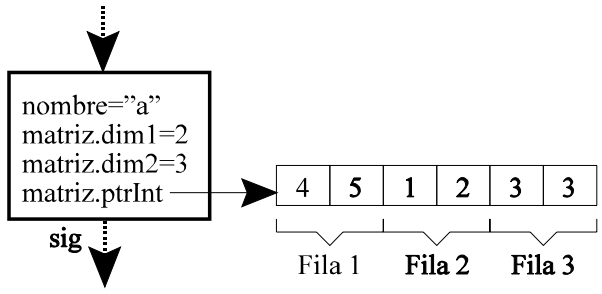
Se desea controlar los siguientes errores de tipo:
- Ninguna dimensión puede ser 0.
- Ninguna dimensión puede ser mayor de 10.
- No se pueden redeclarar variables.
- No se pueden utilizar matrices no declaradas.
- Las asignaciones deben respetar las dimensiones de la matriz l-valor, esto es:
    - No es posible asignar a una matriz ninguna otra con dimensiones diferentes.
    - Una asignación por componente no puede referirse a una columna o fila fuera de rango.
    - Una asignación por componente no puede referirse a una fila fuera de rango.
    - En una asignación por componente, la longitud de la lista de constantes debe ser igual al número de columnas de la matriz l-valor.
- Sólo se pueden sumar matrices de idénticas dimensiones.
- Controlar que si hay algún error de tipos al hacer una suma, **no** se haga la asignación final de dicho resultado erróneo (ver gramática).

Para abordar este problema se va autilizar una tabla de símbolos cuya estructura principal es:

```
typedef struct{
      int dim1, dim2;
      int * ptrInt;
} tipoMatriz;

typedef struct _simbolo
    {
      struct _simbolo * sig;
      char nombre[20];
      tipoMatriz matriz;
    } simbolo;
```

Como puede observarse, los valores de los diferentes elementos de una matriz se almacenan en una tabla de enteros, accesible a través del campo `ptrInt` del tipo `tipoMatriz`. Así, siguiendo el ejemplo anterior, el símbolo asociado a la matriz **a** tendría la siguiente estructura:



Nota: Se recuerda a los alumnos que, en C, un puntero y un *array* son los mismo. Por tanto, en el presente ejemplo **matriz.ptrInt[3]** devolvería el valor **2**.

El tamaño del bloque de memoria apuntado por `matriz.ptrInt` debe ser el justo para almacenar todos los elementos. No debe desperdiciarse memoria.

Se pide:
- Realizar los programas Lex/Yacc que realicen las tareas propuestas. Para ello se da la siguiente tabla de símbolos y el siguiente esqueleto Yacc.
- Indicar qué reglas de error serían necesarias para evitar que, al encontrarse con un error sintáctico, Yacc se pare.

Pista: Nótese que los cálculos intermedios deben almacenarse en una matriz transitoria que cuya memoria debe liberarse una vez utilizada.

Fichero: TabSimb.c

```
#include <stdlib.h>
#include <stdio.h>
/* En lo siguiente, dim1 es el número de columnas y dim2 es el de filas */
typedef struct{
      int dim1, dim2;
      int * ptrInt;
} tipoMatriz;

typedef struct _simbolo
    {
      struct _simbolo * sig;
      char nombre[20];
      tipoMatriz matriz;
    } simbolo;

simbolo * crear()
{
  return NULL;
};

simbolo * insertar(simbolo * *p_t, char nombre[20], int dim1, int dim2)

{
      simbolo * s;
      s = (simbolo *)malloc(sizeof(simbolo));
      strcpy(s->nombre, nombre);
      s->matriz.dim1=dim1;
      s->matriz.dim2=dim2;
      s->matriz.ptrInt=NULL;
      s->sig = (*p_t);
      (*p_t) = s;
      return s;
}

simbolo * buscar(simbolo *t, char nombre[20])

{
  while( (t != NULL) && (strcmp(nombre, t->nombre)) )
      t = t->sig;
  return (t);
}
/* En lo siguiente, pos1 es la columna y pos2 es la fila.
 *    Los errores deben haberse controlado antes
 */
/* asignar asigna un valor a la posición (pos1, pos2) de la matriz del símbolo
apuntado por s */
void asignar(simbolo * s, int pos1, int pos2, int valor){
      (* ((s->matriz.ptrInt)+(pos2-1)*(s->matriz.dim1)+(pos1-1)) ) = valor;
}
/* obtener devuelve el valor de la posicion (pos1, pos2) de la matriz tm */
int obtener(tipoMatriz tm, int pos1, int pos2){
      return (* ((tm.ptrInt)+(pos2-1)*(tm.dim1)+(pos1-1)) );
}
/* copiar copia la matriz matriz directamente en el array de enteros apuntado
por ptr */
void copiar(int * ptr, tipoMatriz matriz){
      int i;
      for (i=0; i < (matriz.dim1)*(matriz.dim2); i++)
            (* (ptr + i) ) = (* (matriz.ptrInt + i) );
}
/* sumar suma las matrices s1 y s2, y devuelve el resultado en el array ptr
(previamente ubicado) */
void sumar(int * ptr, tipoMatriz s1, tipoMatriz s2){
      int i, j;
      for (j=1; j<=s1.dim2; j++)
            for (i=1; i<=s1.dim1; i++)
                  (* (ptr+(j-1)*s1.dim1+(i-1)) ) = obtener(s1, i, j) +
obtener(s2, i, j);
}
```

Fichero ExaJu02l.lex:

```
%%
```

Fichero ExaJu02y.yac:

```
%{
#include "TabSimb.c"
simbolo * t; // Esta será la tabla de símbolos.
/* pruebaInsertar inserta la matriz identificada por nombre en la tabla de simbolos. Previamente hace los controles necesarios */
void pruebaInsertar(int dim1, int dim2, char * nombre){

}

	/* Vars. globales. No puede declararse ninguna variable global más. */
	int pos1, pos2;
	simbolo * actual;
%}

%union{
	char nombre[20];
	int numero;
	struct{
		int dim1, dim2;
	}dosNumeros;
	tipoMatriz matriz;
}

%%
prog	: DECLARE l_decl BEGINN l_sent END
	;
l_decl: /* Epsilon */
	| l_decl decl ';'
	;
```

```
decl    : NUM '*' NUM ID    {
                            }
        | decl ',' ID       {
                            }
        ;
l_sent: /* Epsilon */
        | l_sent sent ';'
        ;
sent    : PRINT mtrz        {
                            }
        | ID '=' mtrz       {
                            }
        | ID '[' NUM ',' NUM ']' '=' NUM        {
                                                }
        | ID '[' NUM ']' '=' '{'                {
                                                }
        ;
l_num   : NUM               {
```

```
                        }
        | l_num ',' NUM  {

                        }
        ;
mtrz    : ID            {

                        }
        | mtrz '+' mtrz     {

                        }
        | '(' mtrz ')'      {

                        }
        ;
%%
#include "ExaJu02l.c"
void main(){
    t = crear();
    yyparse();
}void yyerror(char * s){
    printf("%s\n", s);
}
```