Parte II
Analizador Semántico

El objetivo de esta segunda parte del proyecto es la implementación, usando BISON[4], de las restricciones semánticas en general y las comprobaciones de tipos en particular para el lenguaje `MenosC` que se comenzó a desarrollar en la primera fase del proyecto. Además, en esta parte también se deberá realizar la manipulación de la información de los objetos del programa en la Tabla de Símbolos (TDS) y la gestión de memoria estática.

Para facilitar la tarea de codificación se proporciona el siguiente material auxiliar:

- `Makefile`. Una nueva versión que incluye la gestión de una nueva librería.
- `principal.c`, (en el directorio `src`) Una versión actualizada para permitir la opción de visualizar o no, la TDS.
- `libtds`. Librería con las operaciones para la manipulación de la TDS.

  En los directorios `include` y `lib` se sitúan respectivamente el fichero con la cabecera, `libtds.h`, y el objeto, `libtds.a`, de la librería.
- `Programas de prueba`. (En el directorio `tmp`) Un conjunto de programas de prueba, [$b\{0,1,2,3,4,5\}.c$], con y sin errores semánticos. El compilador deberá detectar todos los errores presentes en estos programas de prueba.

4. Especificación semántica

Las restricciones semánticas que se definen para `MenosC` son las siguientes:

- Todas las variables deben declararse antes de ser utilizadas.
- El compilador solo trabaja con constantes enteras. Si el analizador léxico encuentra una constante real en el programa debe devolver su valor entero truncado.
- El tipo lógico, `bol`, se representa numéricamente como un entero: con el valor 0, para el caso `falso`, y 1, para el caso `verdad`.
- No existe conversión de tipos entre `int` y `bol`.
- La talla de los tipos simples, *entero* y *lógico*, debe definirse, por medio de la constante TALLA_TIPO_SIMPLE= 1, en el fichero `header.h` del directorio `include`.

[4]Su manual puede encontrarse en `/asigDSIC/ETSINF/lppl/doc`

- Las variables de tipo simple se pueden inicializar en su declaración. En ese caso, el tipo de la declaración debe ser idéntico al tipo de la expresión constante.
- Los índices de los vectores van de `0` a `cte-1`, siendo `cte` el número de elementos definido en su declaración. El número de elementos de un vector debe ser un entero positivo.
- No es necesario comprobar los índices de los vectores en tiempo de ejecución.
- El operador "%" calcula el resto de una división entera, por tanto los dos argumentos deben ser enteros.
- La expresión de las instrucciones `if-else` y `while`, debe ser de tipo lógico.
- En cualquier otro caso, las restricciones semánticas por defecto serán las propias del lenguaje ANSI *C*.

5. Gestión de la TDS

En esta sección se presenta la estructura de la *Tabla de Símbolos* que se va a utilizar en la práctica junto con las funciones para su manipulación. Todo esto esta recogido en la librería `libtds` que describiremos a continuación.

5.1. Estructura de la Tabla de Símbolos (TDS)

En el fichero `libtds.h`, del directorio `include`, aparecen las definiciones de las constantes simbólicas, variables globales, estructuras usadas y cabeceras de funciones que serán de utilidad al implementar las acciones semánticas para manipular la TDS. A modo ilustrativo podemos destacar:

- **Constantes simbólicas**, definidas para representar los tipos de los objetos del lenguaje que se utilizan en la librería:

```
/************************* Constantes para los tipos en la Tabla de Símbolos */
#define T_VACIO       0
#define T_ENTERO      1
#define T_LOGICO      2
#define T_ARRAY       3
#define T_RECORD      4
#define T_ERROR       5
```

- **Variables globales**, de uso en todo el compilador:

```
int dvar;            /* Desplazamiento relativo en el Segmento de Variables  */
```

- **Estructura básica**, que contienen la información de la TDS para los objetos simples, vectores y campos de registros. A continuación veremos algunas funciones de consulta a la TDS que devuelven estas estructuras.

```
typedef struct simb /****************************** Estructura para la TDS */

  int   tipo;           /* Tipo del objeto                                  */
  int   desp;           /* Desplazamiento relativo en el segmento variables */
  int   ref;            /* Campo de referencia de usos múltiples            */
 SIMB;
typedef struct dim  /* Estructura para la información obtenida de la TDArray */

  int   telem;                                    /* Tipo de los elementos */
  int   nelem;                                    /* Número de elementos   */
 DIM;
typedef struct reg  /* Estructura para los campos de un registro            */

  int   tipo;                         /* Tipo del campo                    */
  int   desp;                         /* Desplazamiento relativo en memoria */
CAMP;
```

5.2. Funciones de manipulación de la TDS

En la Figura 2 se presenta el listado de las funciones que deben emplearse para acceder a la TDS.

6. Ejemplos ilustrativos

En esta sección se muestran dos ejemplos sencillos de comprobación de tipos: uno en la fase de declaración de variables (inferencia de tipos) y otro en el de las expresiones (comprobación de tipos).

6.1. Comprobación de tipos en *declaraciones*

Para la declaración de un objeto elemental de tipo array, un posible ejemplo de comprobación de tipos y de gestión estática de memoria podría ser:

```
declaracion | tipoSimple ID_ AC_ CTE_ CC_ PCOMA_

            { int numelem = $4;
              if ($4 <= 0)  {
                yyerror("Talla inapropiada del array");
                numelem = 0;
              }
              int refe = insTdA($1, numelem);
              if ( ! insTdS($2, T_ARRAY, dvar, refe) )
                yyerror ("Identificador repetido");
             else dvar += numelem * TALLA_TIPO_SIMPLE;
            }
```

```
int insTdS (char *nom, int tipo, int desp, int ref) ;
/* Inserta en la TDS toda la información asociada con una variable de nombre,
   "nom"; tipo, "tipo"; desplazamiento relativo en el segmento de variables,
   "desp"; y referencia a posibles subtablas, "ref", de vectores o registros
   (-1 si es de tipo simple). Si la variable ya existe devuelve el valor
   "FALSE=0" ("TRUE=1" en caso contrario).                                  */

int insTdA (int telem, int nelem) ;
/* Inserta en la Tabla de Arrays la información de un array con elementos de
   tipo, "telem"; y número de elementos, "nelem". Devuelve su referencia en
   la Tabla de Arrays.                                                      */

int insTdR (int refe, char *nom, int tipo, int desp) ;
/* Inserta en la Tabla de Registros, referenciada por "refe", la información
   de un determinado campo: nombre de campo, "nom"; tipo de campo, "tipo"; y
   desplazamiento relativo del campo en el registro, "desp". Si "ref = -1"
   entonces crea una nueva nueva entrada en la Tabla de Registros con este
   campo y devuelve su referencia. Comprueba además que el nombre del campo
   no este repetido en el registro, devolviendo "-1" en ese caso.           */

SIMB obtTdS (char *nom) ;
/* Obtiene toda la información asociada con un objeto de nombre, "nom", y la
   devuelve en una estructura de tipo "SIMB" (ver "libtds.h"). Si el objeto
   no esta declarado, devuelve "T_ERROR" en el campo "tipo".                */

DIM obtTdA (int ref) ;
/* Devuelve toda la información asociada con un array referenciado por "ref"
   en la Tabla de Arrays en una estructura de tipo "DIM" (ver "libtds.h").
   En caso de error devuelve  "T_ERROR" en el campo "telem".                */

CAMP obtTdR (int ref, char *nom) ;
/* Obtiene toda la informacion asociada con el campo de nombre, "nom", de un
   registro referenciado por su referencia, "ref", en la Tabla de Registros.
   y lo devuelve en una estructura de tipo "CAMP" (ver "libtds.h"). En caso
   de error devuelve "T_ERROR" en el campo "tipo".                          */

void verTdS () ;
/* Muestra toda la información de la TDS.                                   */
```

Figura 2: Perfil de las funciones de manipulación de la TDS.

6.2. Comprobación de tipos en las *expresiones*

En el caso de una expresión de asignación, donde se espera que los operandos sean de tipo simple, su comprobación de tipos podría ser:

```
expresion
       | ID_  operadorAsignacion  expresion

         { $$.tipo = T_ERROR;

           SIMB sim = obtTdS($1);
           if (sim.tipo == T_ERROR) yyerror("Objeto no declarado");
           else if (! ((sim.tipo == $3 == T_ENTERO) ||
                       (sim.tipo == $3 == T_LOGICO)))
             yyerror("Error de tipos en la 'instrucción de asignación'");
           else $$.tipo = sim.tipo;
         }
```

Advertid que para evitar una secuencia de errores redundantes debería modificarse este código para que solo se de un nuevo mensaje de error si el error se produce en esta regla, y no si proviene de errores anteriores a través de $1 o $3.

7. Recomendaciones finales de implementación

7.1. Atributos léxicos

Para trabajar con los atributos de los símbolos del lenguaje, en primer lugar, hay que definir el conjunto de posibles tipos de atributos. Para ello:

1. Especificar la colección completa de los (tipos de) atributos en una declaración `%union` en `Bison`; por ejemplo, para los atributos léxicos podríamos definir:

```
%union {
   char *ident;        /* Nombre del identificador */
   int cent;           /* Valor de la cte numérica entera */
}
```

2. Evaluar los atributos léxicos asociados con los *identificadores* y las *constantes enteras*. Los atributos de los terminales se asignan a la variable `yylval` en las reglas del `Flex` donde se define cada token. Por ejemplo:

```
{numero}        { yylval.cent  = atoi(yytext);   return(CTE_); }
{identificador} { yylval.ident = strdup(yytext); return(ID_);  }
```

   Donde `CTE_` y `ID_` son codificaciones arbitrarias para las *constantes enteras* y los *identificadores* de variables.

7.2. Fichero de cabeceras

Las constantes, estructuras y variables globales que se utilicen en todo el compilador, es conveniente definirlas en vuestro fichero de cabecera `header.h` (y situarlo en el directorio `include`). Algunas sugerencias para añadir a vuestro `header.h` de la Parte-1 podrían ser:

- **Constantes simbólicas**

```
/************************************ Tallas asociadas a los tipos simples */
#define TALLA_TIPO_SIMPLE     1
```

- **Variables Globales**

```
/*********************** Variables externas definidas en Programa Principal */
extern int verTDS;                  /* Flag para saber si mostrar la TDS     */

/***************************** Variables externas definidas en las librerías */
extern int dvar;              /* Desplazamiento en el Segmento de Variables  */
```