

# LENGUAJES DE PROGRAMACIÓN Y PROCESADORES DE LENGUAJES

Construcción de un compilador

MenosC

Parte-II: comprobaciones semánticas

## Material auxiliar de prácticas

- `Makefile`. Una nueva versión.
- `principal.c`. Una nueva versión en el directorio **src**.
- `libtds`. Librería con las operaciones para la manipulación de la [Tabla de Símbolos](#)
  - `libtds.h`, el fichero de cabecera, en el directorio **include**;
  - `libtds.a`, la librería, en el directorio **lib**.
- *Programas de prueba*.

- Todas las variables y funciones deben declararse antes de ser utilizadas.
- Debe haber una función, y solo una, con el nombre `main`.
- La información de los parámetros se situará en la TdS, en orden inverso a su declaración.
- El paso de parámetros se hace siempre por valor.
- Se admite la recursividad en las funciones.
- En el compilador solo se usan constantes enteras. Si el analizador léxico encuentra una constante real se debe devolver su valor entero truncado.
- La talla de los tipos *entero* y *lógico* se debe definir en `TALLA_TIPO_SIMPLE= 1`.
- El tipo lógico `bo1` 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 `bool`.
- Los índices de los vectores van de 0 a `cte-1`, siendo `cte` el número de elementos, que debe ser un entero positivo.
- No es necesario comprobar los índices de los vectores en tiempo de ejecución.
- Los operadores de incremento y decremento (infijos y postfijos) solo se pueden aplicar a variables (p.ej. `(i+j)++` es ilegal).
- En la instrucción `for` las *expresiones opcionales* deben ser asignaciones o expresiones (pueden no aparecer). La *expresión*, como en C, debe ser de tipo lógico y debe aparecer explícitamente.
- La expresión de la instrucción `if-else` debe ser de tipo lógico.
- Por defecto las restricciones semánticas serán las propias del lenguaje ANSI C.

## ➤ Estructura de la TDS

Constantes, variables globales y estructuras básicas (ver Sección 5.1 del Enunciado)

## ➤ Funciones de manipulación de la TDS

```
void cargaContexto (int n) ;  
/* Crea el contexto necesario para los objetos globales y para los objetos  
   locales a las funciones */  
void descargaContexto (int n) ;  
/* Libera en la TdB y la TdS el contexto asociado con la función. */  
int insTdS (char *nom, int cat, int tipo, int n, int desp, int ref) ;  
/* Inserta en la TdS toda la información asociada con una variable de nombre,  
   "nom"; categoría, "cat"; tipo, "tipo"; nivel del bloque, "n"; desplaza-  
   miento relativo, "desp"; y referencia, "ref", a posibles subtablas de  
   vectores o dominios, siendo (-1) si es de tipo simple. Si la variable ya  
   existe devuelve el valor "FALSE=0" ("TRUE=1" en caso contrario). */  
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 está declarado, devuelve "T_ERROR" en el campo "tipo". */
```

# TABLA DE SÍMBOLOS

---

```
int insTdA (int telem, int nelem) ;
/* Inserta en la Tabla de array la información de un array con elementos de
   tipo, "telem"; y número de elementos, "nelem". Devuelve su referencia en
   la Tabla de Arrays. */
DIM obtTdA (int ref) ;
/* Devuelve toda la información asociada con un array referenciado por "ref"
   en la Tabla de Arrays. En caso de error devuelve "T_ERROR" en el campo
   "telem". */
int insTdD (int refe, int tipo) ;
/* Para un dominio existente referenciado por "refe", inserta en la Tabla
   de Dominios la informacion del "tipo" del parámetro. Si "ref= -1" entonces
   crea una nueva entrada en la tabla de dominios para el tipo de este
   parámetro y devuelve su referencia. Si la funcion no tiene parametros,
   debe crearse un dominio vacio con: "refe = -1" y "tipo = T_VACIO". */
INF obtTdD (int refe) ;
/* Si "refe<0" entonces devuelve la informacion de la funcion actual, y si
   "refe>=0", devuelve la informacion de una función ya compilada con
   referencia "refe". La informacion es: el nombre y el tipo del rango de la
   función y la talla del segmento de parámetros. Si "refe" no se corresponde
   con una funcion ya compilada, devuelve "T_ERROR" en el campo "tipo". */
```

# TABLA DE SÍMBOLOS

---

```
int cmpDom (int refx, int refy) ;  
/* Si los dominios referenciados por "refx" y "refy" no coinciden devuelve  
   "FALSE=0" ("TRUE=1" si son iguales). */  
void mostrarTdS () ;  
/* Muestra toda la información de la TdS para objetos globales y locales.  
   Se recomienda hacerlo (si "verTdS = true") al finalizar la compilación  
   de la función, justo antes de ("descargarContexto"). */
```

### ➤ Ejemplo de comprobaciones de tipo en *declaraciones*

```
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, VARIABLE, T_ARRAY, niv, dvar, refe) )  
        yyerror ("Identificador repetido");  
      else dvar += numelem * TALLA_TIPO_SIMPLE;  
    }
```



### ➤ Ejemplo de comprobaciones de tipos en la *asignación*

```
instruccionAsignacion
    | ID_  ASIG_  expresion  PCOMA_

    { SIMB sim = obtTdS($1);

      if (sim.t == T_ERROR) yyerror("Objeto no declarado");
      else if (! ((sim.t == $3.t == T_ENTERO) ||
                  (sim.t == $3-t == T_LOGICO)))
          yyerror("Error en la 'instrucción de asignación'");
    }
```

---

† Advertid que este código se debería modificar para que solo proporcione 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.

$\text{exp} \rightarrow \text{fac} * \text{cte}$

CTE_	○
OPMULT_	—
<fac>	○
...	...

símbolos  $\longleftrightarrow$  atributos

$\text{exp} \rightarrow \text{fac} * \text{cte}$

CTE_	○
OPMULT_	—
<fac>	○
...	...

símbolos  $\longleftrightarrow$  atributos

## 1. Definir tipos de atributos

Equivale a definir los tipos de los elementos de la pila de atributos. La cima de la pila de atributos es la variable `yylval`

Para resolver la no-homogeneidad de los tipos se define, en el programa BISON, una estructura `%union`

```
%union {  
    char* ident; /* terminal "identificador"*/  
    int    cent; /* terminal "cte" entera */  
}
```

$\text{exp} \rightarrow \text{fac} * \text{cte}$

CTE_	○
OPMULT_	—
<fac>	○
...	...

símbolos  $\longleftrightarrow$  atributos

```
%union {  
    char* ident;  
    int    cent;  
}
```

## 2. Asociar tipos a símbolos

Asigna a cada símbolo de la gramática sus atributos correspondientes

```
%token<ident> ID_  
%token<cent>  CTE_  
%type<cent>   exp fac
```

$\text{exp} \rightarrow \text{fac} * \text{cte}$

CTE_	○
OPMULT_	—
<fac>	○
...	...

símbolos  $\longleftrightarrow$  atributos

```
%union {  
    char* ident;  
    int    cent;  
}
```

### 3. Calcular atributos de terminales

La información de la `%union` también se propaga en el fichero `asin.h` al AL.

`[0-9]+`

```
{yyval.cent=atoi(yytext); return(CTE_);}
```

$\text{exp} \rightarrow \text{fac} * \text{cte}$

CTE_	○
OPMULT_	—
<fac>	○
...	...

símbolos  $\longleftrightarrow$  atributos

```
%union {  
    char* ident;  
    int    cent;  
}
```

## 4. Acciones semánticas en las reglas

```
exp : fac OPMULT_ CTE_ { $$ = $1 * $3; }  
;
```

## 5. Acciones semánticas en mitad de las reglas

$$A \rightarrow B \{ \overset{\alpha}{\cdot \cdots \cdot} \} C \{ \overset{\beta}{\cdot \cdots \cdot} \}$$

## 5. Acciones semánticas en mitad de las reglas

$$A \rightarrow B \{ \overset{\alpha}{\dots} \} C \{ \overset{\beta}{\dots} \}$$

Equivale a:

$$\begin{aligned} A &\rightarrow B @1 C \{ \overset{\beta}{\dots} \} \\ @1 &\rightarrow \epsilon \{ \overset{\alpha}{\dots} \} \end{aligned}$$



## 5. Acciones semánticas en mitad de las reglas

$$A \rightarrow B \{ \overset{\alpha}{\dots} \} C \{ \overset{\beta}{\dots} \}$$

Equivale a:

$$\begin{aligned} A &\rightarrow B @1 C \{ \overset{\beta}{\dots} \} \\ @1 &\rightarrow \epsilon \{ \overset{\alpha}{\dots} \} \end{aligned}$$

Ejemplo:

$$A \rightarrow B \{ \$\$ = \$1 * 10; \} C \{ \$\$ = \$2 + \$3; \}$$

## 5. Acciones semánticas en mitad de las reglas

$$A \rightarrow B \{ \overset{\alpha}{\dots} \} C \{ \overset{\beta}{\dots} \}$$

Equivale a:

$$\begin{aligned} A &\rightarrow B @1 C \{ \overset{\beta}{\dots} \} \\ @1 &\rightarrow \epsilon \{ \overset{\alpha}{\dots} \} \end{aligned}$$

Ejemplo:

$$A \rightarrow B \{ \$ < cent > \$ = \$1 * 10; \} C \{ \$\$ = \$ < cent > 2 + \$3; \}$$