



**Universidad  
Europea de Madrid**

**LAUREATE** INTERNATIONAL UNIVERSITIES

**ANÁLISIS SEMÁNTICO**

**LA TABLA DE SÍMBOLOS**

© Todos los derechos de propiedad intelectual de esta obra pertenecen en exclusiva a la Universidad Europea de Madrid, S.L.U. Queda terminantemente prohibida la reproducción, puesta a disposición del público y en general cualquier otra forma de explotación de toda o parte de la misma.

La utilización no autorizada de esta obra, así como los perjuicios ocasionados en los derechos de propiedad intelectual e industrial de la Universidad Europea de Madrid, S.L.U., darán lugar al ejercicio de las acciones que legalmente le correspondan y, en su caso, a las responsabilidades que de dicho ejercicio se deriven.

## Índice

Presentación	4
Introducción	5
¿Cuál es la misión de la TS y qué operaciones podemos necesitar para utilizarla?	5
¿Por qué es importante la tabla de símbolos?	5
Implementaciones típicas I	7
Listas lineales	7
Árboles de búsqueda	7
Tabla de dispersión (también denominada tabla hash)	7
Implementaciones típicas II	8
¿Qué tamaño escoger para el array y cómo?	8
Ejemplo de función de dispersión	8
Contenido de la tabla de símbolos	9
¿Qué valores de tipo se pueden almacenar?	9
Declaraciones	10
Reglas de ámbito I	12
¿Cómo implementamos los bloques anidados?	12
Reglas de ámbito II	14
Tipos de ámbitos	14
Interacciones de declaraciones del mismo nivel	14
Operaciones y requisitos	16
¿Cuáles son las operaciones más comunes de la tabla de símbolos?	16
¿Qué requisitos debe cumplir una tabla de símbolos?	16
Resumen	17

## Presentación

El objetivo de este tema es entender la necesidad de esta estructura de datos denominada **tabla de símbolos** y por qué tiene cierta complejidad su gestión.

Los objetivos a conseguir en este tema son:

- Entender la misión de la tabla de símbolos, las operaciones necesarias, así como su importancia.
- Conocer cuáles son las implementaciones típicas y que características tienen.
- Comprender el contenido de la tabla de símbolos.
- Entender qué se hace con cada tipo de declaración.
- Conocer las reglas de ámbito y sus implementaciones.
- Identificar las operaciones necesarias y sus requisitos.



## Introducción

La tabla de símbolos (TS) es la estructura utilizada por el compilador para almacenar los atributos asociados a los símbolos que se utilizan en un lenguaje de programación. Los atributos que esta estructura almacena para cada símbolo pueden ser:

- **Tipo:** entero, real, char, boolean.
- **Valor:** 25, 13, 4, cadena, 0.
- **Dirección de memoria.**
- **Número de línea:** este atributo puede ser interesante en entornos integrados de desarrollo (IDE).
- **Ámbito:** es donde aplica la declaración de una variable, como por ejemplo una función, el programa principal o un método.

### ¿Cuál es la misión de la TS y qué operaciones podemos necesitar para utilizarla?

La misión de la tabla de símbolos es colaborar en las comprobaciones semánticas y facilitar la generación de código.

Las operaciones que podemos necesitar para utilizarla son la de **inserción, búsqueda o consulta, actualización y eliminación**. La eficiencia de estas operaciones depende de la estructura de datos utilizada, y puede hacer que el compilador consuma mucho tiempo en los accesos a la misma.

El contenido principal de esta tabla son los tipos disponibles en el lenguaje y la **información** que guardamos está en función de la estructura y el propósito de las declaraciones.

### ¿Por qué es importante la tabla de símbolos?

Porque la utilizan el analizador léxico, el sintáctico, el semántico para introducir información, y el **generador de código intermedio**, la fase de optimización y la de generación de código las utilizan para **generar el código necesario**.

En resumen, todas las fases del compilador, traductor o intérprete se apoyan en ella para escribir o para obtener información, las de análisis para insertar y actualizar y las de síntesis para obtener la información con la que generar el código.

### Información

Esta información que la tabla almacena está disponible en tiempo de compilación, puesto que se utiliza para construir el compilador.

Hay casos en los que puede estar disponible en tiempo de ejecución y esto es cuando el compilador incorpora un depurador y por tanto necesita ver los valores de los atributos de las distintas variables.

## Implementaciones típicas I

Hay varias **implementaciones típicas de la tabla de símbolos**:

### Listas lineales

Buena estructura de datos básica y proporciona implementaciones directas y fáciles de las tres operaciones básicas.

- Inserción en tiempo constante (al insertar al inicio o al final).
- Operaciones de búsqueda y eliminación de tiempo lineal.

### Árboles de búsqueda

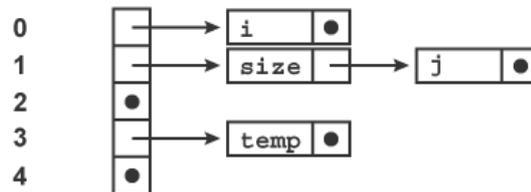
Poco útiles por no ser eficientes, pero también por la complejidad de la operación de eliminación.

### Tabla de dispersión (también denominada tabla hash)

- Es la mejor solución.
- Realiza las tres operaciones en tiempo constante y se utiliza muy frecuentemente.

Puesto que la tabla hash es la mejor solución entraremos en más detalle en su funcionamiento:

**Tabla hash:** es un array de entradas, indexado mediante un intervalo entero, generalmente desde 0 hasta el tamaño de la tabla menos 1. Utiliza una función de dispersión para su funcionamiento (Louden, 2004).



## Implementaciones típicas II

**Función de dispersión:** convierte la clave de búsqueda (en este caso el nombre del identificador) en un valor entero de dispersión en el intervalo del índice, y el elemento correspondiente a la clave de búsqueda se almacena en el array. Hay que tener cuidado con las **colisiones y su resolución**.

### ¿Qué tamaño escoger para el array y cómo?

Los tamaños típicos abarcan desde algunos cientos hasta algo más de un millar. Se recomienda escoger un número primo (Ej.: 200 → n. primo 211). Pero... ¿cómo?

- Se convierte cada carácter a un entero no negativo.
- Se combinan estos enteros para formar un entero simple.
- El entero resultante se escala al intervalo 0... tamaño -1.

Las **características** que son **deseables** son la **eficiencia** (debe ser rápido y fácil de calcular) y **eficacia** (debe producir listas colisiones pequeñas).

### Ejemplo de función de dispersión

```
1. | #define SIZE ...
2. | #define SHIFT 4
3. | int hash (char *key)
4. | { int temp = 0;
5. |   int i = 0;
6. |   while (key[i] != '\0')
7. |     { temp = (( temp << SHIFT) + key[i]) % SIZE;
8. |       ++i;
9. |     }
10. |   return temp;
```

## Contenido de la tabla de símbolos

Los contenidos están unidos a los identificadores del programa fuente, concretamente al nombre del identificador.

La tabla de símbolos también puede iniciarse con cierta información considerada de interés por el diseñador, como pueden ser constantes, funciones de librería, y si se considerase también palabras reservadas. De acuerdo con la estrategia que se decida, los identificadores los insertará en la TS el analizador léxico o el analizador sintáctico, validando antes que no existen ya en la TS.

### ¿Qué valores de tipo se pueden almacenar?

- **Tipos simples:**
  - **Entero** (int, long,...).
  - **Real** (float, double,...).
  - **Carácter** (char).
- **Tipos estructurados:**
  - **Arrays** (incluyendo dimensiones).
  - **Struct o Records** (incluyendo cada uno de sus miembros y tipos).
- **Tipos archivo o fichero:** la representación interna de los archivos varía de unos lenguajes a otros.
- **Tipos puntero.**
- **Funciones y procedimientos:** con sus parámetros y tipos. En el caso de las funciones el tipo que devuelven.
- **Clases:**
  - Con sus atributos y tipos.
  - Con sus métodos incorporando tipos y parámetros.
  - Relaciones de herencia, agregación y composición.



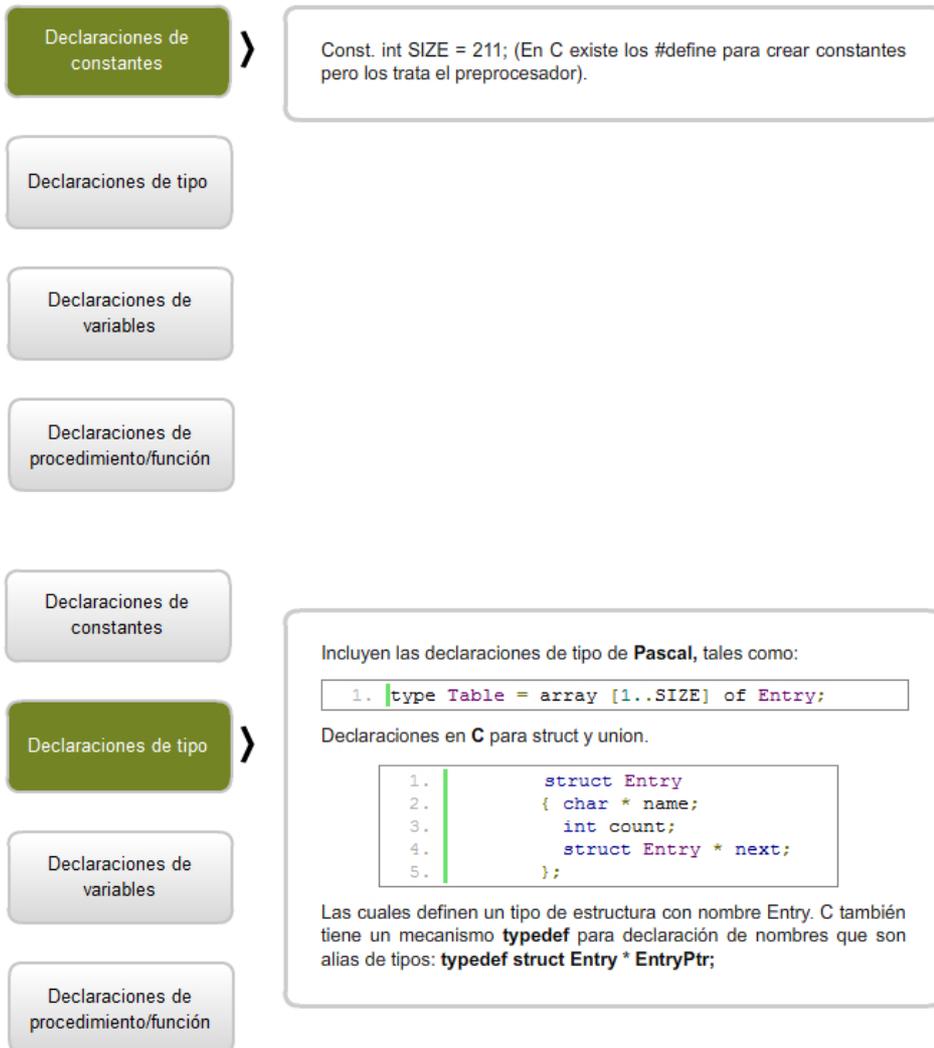
#### Identificador

**Son atributos del identificador:** tipo, valor, dirección de memoria, número de línea, ámbito o cualquier otro que el diseñador considere.

## Declaraciones

El comportamiento de una tabla de símbolos depende mucho de las propiedades de las declaraciones del lenguaje que se está traduciendo.

Hay cuatro **clases básicas de declaraciones**:



Declaraciones de constantes

Declaraciones de tipo

Declaraciones de variables

Declaraciones de procedimiento/función

Declaraciones de constantes

Declaraciones de tipo

Declaraciones de variables

Declaraciones de procedimiento/función

Las declaraciones de variables son la forma mas común de declaración e incluyen declaraciones en Fortran tales como:

```
1. integer a,b(100) → (Decl. explícita )
```

En Fortran variables comienzan I...N son enteras (Decl. Implícita).  
Y declaraciones en C como:

```
1. int a,b[100];
```

Ejemplo:

```
1. int hash (char * key)
```

## Reglas de ámbito I

**Las reglas de ámbito** en los lenguajes de programación varían mucho, aunque existen varias reglas que son comunes a muchos lenguajes (Louden, 2004).

**Declaración antes de uso**, es una regla común utilizada en **C** y **Pascal**, que requiere que se declare un nombre en el texto de un programa antes de referenciarlo.

- Permite construir la tabla de símbolos a medida que continúa el análisis sintáctico y que las búsquedas se realicen tan pronto como se encuentra una referencia de nombre en el código. Si falla la búsqueda es que no se ha declarado la variable → **Facilita la compilación de una sola pasada.**
- **Modula-2** no requiere de la declaración antes del uso → paso por separado para la construcción de la tabla de símbolos. **No es posible la compilación de una sola pasada.**

**Estructura de bloques**, es una propiedad común de los lenguajes modernos, pero... **¿Qué es un bloque?** Es cualquier construcción que pueda contener declaraciones. **¿Cuándo un lenguaje está estructurado en bloques?** Si permite la anidación de bloques dentro de otros bloques → **Regla de anidación más próxima.**



### Ejemplos

Ejemplo

### ¿Cómo implementamos los bloques anidados?

Inserción	<b>No debe sobrescribir declaraciones anteriores</b> , sino que las debe ocultar temporalmente, para que la operación de búsqueda solo encuentre la declaración para un nombre que se haya insertado más recientemente.
Eliminación	No debe <b>eliminar</b> todas las declaraciones correspondientes a un nombre, sino <b>solo la más reciente.</b>
Solución	Construir una nueva tabla de símbolos para cada ámbito y vincular las tablas desde ámbitos internos a ámbitos externos.

**Ejemplo**

**Ejemplos**

- **Pascal:** los bloques son el programa principal y las declaraciones de procedimiento/función.
- **C:** son las unidades de compilación (archivos de código), las declaraciones de procedimiento/función y las sentencias compuestas (sentencias entre llaves, {...}).
- **Lenguajes orientados a objetos:** las declaraciones de clase.

## Reglas de ámbito II

### Tipos de ámbitos

- **Ámbito léxico o estático:** la tabla de símbolos se construye de manera estática siguiendo el orden de escritura del programa, tal y como hemos visto hasta ahora. En los lenguajes de bloques, como Pascal o C, el ámbito es el procedimiento, la función o el registro.
- **Ámbito dinámico (LISP):** cuando la tabla de símbolos se construye siguiendo el orden de ejecución del programa.
  - Esto implica que se realicen operaciones de inserción y eliminación a medida que los ámbitos se introducen y extraen en tiempo de ejecución.
  - La tabla debe ser parte del ambiente de ejecución.
  - Se compromete la legibilidad de los programas porque las referencias no locales no pueden resolverse sin simular la ejecución del programa.
  - Es incompatible con la verificación de tipos estático, ya que los tipos de los datos de las variables deben mantenerse por medio de la tabla de símbolos. (Ej. Si en el main  $i = \text{double}$ ).

### Interacciones de declaraciones del mismo nivel

Caso 1	Caso 2	Caso 3
En C, Pascal o Ada no se puede volver a utilizar el mismo nombre en declaraciones del mismo nivel.	Declaración <b>secuencial</b> vs declaración <b>colateral</b> .	Declaración <b>recursiva</b> . Las declaraciones pueden hacer referencia a si mismas o entre sí. Esto es necesario en particular para declaraciones de proc/función.
 <a href="#">Ejemplo caso 1</a> Ejemplo	 <a href="#">Ejemplo caso 2</a> Ejemplo	 <a href="#">Ejemplo caso 3</a> Ejemplo

#### Ejemplo

##### Ejemplo caso 1

```
1. | typedef int i;
2. | int i; → Error de compilación
```

¿Cómo lo verificamos? Realizar **búsqueda** antes de cada inserción y determinar mediante algún mecanismo **el nivel de anidación**.

### Ejemplo

#### Ejemplo caso 2

```
1. | int i = 1
2. | void f(void)
3. | { int i = 2, j = i + 1;
4. |     ....
5. | }
```

¿Cuánto vale j? Depende de si usamos la declaración local.

**Secuencial (C)** → j = 3;

**Colateral (ML): No tendría en cuenta la declaración local** → j = 2;

(Todas las declaraciones se procesan "simultáneamente").

### Ejemplo

#### Ejemplo caso 3

```
1. | int mcd (int n, int m)
2. | { if (m == 0) return n;
3. |   else return mcd (m, n % m);
4. | }
```

Para que sea compilado correctamente, el compilador debe agregar el nombre de la función (mcd) a la tabla de símbolos **antes** de procesar el cuerpo de la función.

## Operaciones y requisitos

### ¿Cuáles son las operaciones más comunes de la tabla de símbolos?

Inserción	Al reconocer un identificador el compilador debe insertar un identificador.
Consulta	Previo a cada inserción hay que consultar en la tabla, si para ese ámbito, existe o no el identificador. Si ya existe, no debe permitirse la inserción. Nos interesa que estas consultas, puesto que es la operación que más se realiza, tarden siempre lo mismo.
Actualización	Conforme van avanzando a lo largo del árbol de análisis sintáctico los valores de los atributos van cambiando y esto requiere actualizarlos en la tabla de símbolos.
Eliminación	Se utiliza para eliminar la información de una declaración cuando esta ya no es necesaria.  La forma de eliminar la información es marcándola como inactiva, antes que eliminarla físicamente.

### ¿Qué requisitos debe cumplir una tabla de símbolos?

Rapidez	Se debe tener en cuenta que las consultas son las operaciones más frecuentes, por eso son las que hay que optimizar.
Eficiente en la gestión del espacio	Almacena gran variedad de información y de muchos tipos.
Flexible	Debe ser capaz de manejar muchos tipos de datos diferentes.
Fácil de mantener	Tanto las inserciones como las actualizaciones o eliminaciones deben realizarse de una forma sencilla.



## Resumen

En este tema hemos comprendido qué hace una tabla de símbolos y el porqué de su complejidad.

Es importante también el tener claras sus misiones:

- Colaborar en las comprobaciones semánticas.
- Facilitar la generación de código.

También se ha visto las implementaciones típicas: listas lineales, árboles de búsqueda y la tabla de dispersión también denominada tabla hash. Esta última es la mejor solución pero conviene elegir adecuadamente la función de dispersión.

Es importante conocer los atributos que se van a almacenar del identificador: tipo, valor, dirección de memoria, número de línea, ámbito o cualquier otro que el diseñador considere, teniendo en cuenta que pueden ser tipos simples (enteros), tipos estructurados (arrays), tipos puntero, funciones y procedimientos o clases.

Además es necesario saber que el comportamiento de una tabla de símbolos depende mucho de las propiedades de las declaraciones del lenguaje que se está traduciendo. Hay cuatro clases básicas de declaraciones: de constantes, de tipo, de variables y de procedimiento/función.

Otro aspecto a tener en cuenta es el ámbito, que determina cuando un símbolo determinado va a estar accesible y contempla dos posibilidades: declaraciones antes de uso o las estructuras de bloque.

Por otro lado, hay dos tipos de ámbito dependiendo de cuando se construye la tabla de símbolos: ámbito léxico o estático, donde la tabla de símbolos se construye de manera estática siguiendo el orden de escritura del programa, y el ámbito dinámico en el que la tabla de símbolos se construye siguiendo el orden de ejecución del programa.

Para finalizar se ha visto las operaciones más comunes, como es la consulta, y los requisitos que debe cumplir la tabla de símbolos.