Identificadores - Conceptos relacionados

- **Objeto**: zona contigua de memoria que puede contener un valor de un tipo dado.
- **Tipo**: define un conjunto de valores posibles y un conjunto de operaciones.
- **Valor**: es un conjunto de bits en la memoria interpretados según un tipo.
- Un **identificador** sirve para designar
  - Objetos: variables
  - Funciones
  - El nombre o los miembros de: estructuras, uniones y enumeraciones
  - "Alias" de tipos de datos (typedef)
  - Una etiqueta (para saltos)
  - Una macro o sus parámetros

Tipos de Datos

- Tipos derivados: construídos en base a los tipos elementales.
- Equivalencias de tipos de datos: por ejemplo `int` es equivalente a `*&int`.
- Compatibilidad de tipos: cuando puedo usar un tipo de datos donde se esperaba otro, por ejemplo usar short donde se espera un long.
- Inferencia de tipos: dada un expresión, que tipo de dato tiene.

Tipos de Datos

- Chequeo estático: los tipos se definen en el fuente, en tiempo de compilación
- Chequeo dinámico: los tipos se definen en tiempo de ejecución
- Combinación: hay lenguajes que usan los dos tipos de chequeos.
- Fuerte/Débilmente tipados: no es un categoría precisa, es más bien de uso informal. Fuertemente tipado exige que el programador haga conversiones explícitas (o bien, no permite hacerlas)

Identificadores – Declaración y Definición

- Declaración: especifica los atributos y la interpretación del identificador.
  - Sirve también para que una unidad de traducción referencie a objetos o funciones definidas en otra unidad de traducción.
- Definición: es una declaración que además:
  - Si es un objeto causa que se reserve memoria para alojarlo.
  - Si es una función incluye el código de la misma.
- Para un objeto o función se puede tener varias declaraciones pero solo una definición

Ámbito y Vinculación

- Ámbito (Scope): partes de la unidad de traducción donde un identificador puede ser usado para referirse al objeto que designa.
  - Un mismo identificador puede designar diferentes objetos en distintos ámbitos, por ejemplo dos funciones pueden tener una variable que se llame igual.
- Vinculación (Linkage): Se refiere a que el mismo identificador pueda designar el mismo objeto (o no) en distintas unidades de traducción o solo dentro de una
  - Externa: Desde distintas unidades de traducción
  - Interna: Desde una unidad de traducción
  - Sin vinculación: solo en su bloque (variables locales y argumentos de la función)

Alcance (Scope)

- Dado un identificador, este hace referencia a un objeto, pero solo en ciertas partes del programa esta vinculación está activa.
- Hablamos de “donde es visible” o donde está vinculado o enlazado (el nombre con el objeto), en inglés: name binding.
- Léxico (estático) vs dinámico
  - Léxico: definido por el fuente.
  - Dinámico: definido por el orden de llamada entre las subrutina.

Alcance (Scope)

- Niveles
  - Expresión: en general lenguajes funcionales
  - Bloque: como en C/C++
  - Función: Similar al anterior
  - Archivo: Tomar C como ejemplo
  - Modulo: Conjunto de archivos.
  - Global: Todo el sistema.

Duración (tiempo de vida)

- Lapso en el cuál hay una zona de memoria reservada para el objeto.
- Deterministico: su creación/destrucción está determinada de antemano, como en C con las variables automáticas o estáticas.
- Dinámico: depende de la ejecución del programa: malloc()/free() - new/delete
- Variantes:
  - Constructor / destructor
  - Manejo de recursos
  - Recolector de basura

Espacios de nombres

- El objetivo básico es poder repetir identificadores sin que haya colisión en un mismo ámbito
- Algunos lenguajes permiten crear espacios propios como C++, o usan mecanismos similares (packages, ojo depende del lenguaje)
- Nombres calificados: es usar, normalmente un prefijo, para indicar de que espacio de nombres estoy tomando el identificador

Espacios de nombres (name spaces)

- Los espacios de nombres tienen por objetivo poder referirse a objetos distintos, con un mismo identificador en el mismo ámbito.
- En lenguaje C no hay un manejo de espacios de nombre por parte del programador, pero hay definidos 4 espacios de nombre.
- Podemos repetir el mismo nombre en distintos espacios, pero no repetir uno dentro de un mismo espacio.

Espacios de nombres en C

- Las etiquetas, reconocidas por su sintaxis (seguida de :)
- Los tags (el nombre) de estructuras, uniones y enumeraciones. Al estar en un mismo espacio de nombre no puedo usar, por ejemplo, para una estructura el mismo nombre que use para una enumeración.
- Los miembros de las estructuras y uniones, cada tag crea su propio espacio de nombres, por eso pueden repetirse de, por ejemplo, una estructura a otra.
- Los identificadores ordinarios (comunes): todos los demás, incluyendo variables, funciones, constantes de enumeración y alias de tipos introducidos con `typedef`.

Ejemplo

```
 5  int fun(void)
 6  {
 7          return 3;
 8  }
 9  int fun;
10  //main.c:9:5: error: ‘fun’ redeclared as different kind of symbol
11
12  int main(void)
13  {
14          int nombre; //ok
15          struct nombre {int campo;}; //ok
16          struct otra {int campo;}; //ok, campo se repite
17                                      //pero en otro espacio de nombre
18          enum otra {UNO, DOS, nombre}; // dos errores
19          //se repite el tag otra y se repite nombre en el espacio ordinario
20          //main.c:18:7: error: ‘otra’ defined as wrong kind of tag
21          //main.c:18:23: error: ‘nombre’ redeclared as different
22          //                          kind of symbol
23          int fun; //Distinto ámbito, oculta lo declarado a
24                   //nivel de archivo
25  nombre: //ok, las etiquetas tienen su propio espacio de nombres
26          return EXIT_SUCCESS;
27  }
```

Licencia