Conceptos básicos

- **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 (tag) o los miembros de: estructuras, uniones y enumeraciones
  - "Alias" de tipos de datos (typedef)
  - Una etiqueta (para saltos)
  - Una macro o sus parámetros
  - Un atributo (C23)

Tipos

Referencia adicional: https://en.cppreference.com/w/c/language/type

- El significado de un valor guardado en un objeto o devuelto por una función está determinado por el tipo de expresión usado para accederlo.
- La expresión más simple es un identificador, y su tipo fue especificado al declararlo.
- Los tipos se dividen en **tipos de objetos** (variables) y **tipos de funciones** (que describen su firma)
- Los tipos de objetos pueden, en un determinado punto de la unidad de traducción, estar incompletos (y ser completados luego). Cuando esto ocurre no se puede especificar el tamaño necesario para guardarlo, y por lo tanto no se puede declarar variables de un tipo incompleto (pero si punteros). Casos: arreglos sin dimensión, estructuras o uniones sin sus miembros.

Tipos de Objetos (Básicos)

- void es un tipo especial que tiene un conjunto vacío de valores, se lo considera incompleto y no puede completarse.
- _Bool (bool en C23) es considerado de tipo entero sin signo con la capacidad necesaria para almacenar un 0 y un 1 (típicamente un byte). Si se le asigna el valor de una expresión entera su valor será cero si la expresión era igual a cero y 1 en cualquier otro caso.
- Tipos carácter: char, unsigned char, signed char. Se los considera 3 tipos diferentes, char termina siendo signed o unsigned dependiendo de la implementación. Tiene espacio necesario para guardar un caracter del juego básico (típicamente un byte).
- Tipos enumerados: cada vez que defino un enum tag se lo considera un tipo enumerado.

Tipos de Objetos (Básicos)

- Tipos enteros estándar: int y char con y sin signo. Incluye la variantes de int como short, long y long long y los tipos de enumeración.
- Tipos enteros: los enteros estándar más _Bool (bool)
- Tipos reales flotantes: float, double y long double.
- Tipos complejos: su implementación es optativa. float _Complex, double _Complex y long double _Complex.
- Tipos flotantes: la unión de reales flotantes y complejos.
- Tipos reales: la unión de reales flotantes más los tipos enteros.
- Tipos aritméticos: la unión de flotantes más los tipos enteros.
- Tipos básicos: la unión de tipos de carácter con tipos enteros y tipos flotantes. Son tipos completos. (todo lo nombrado hasta aquí excepto void)

Tipos Derivados

- A partir de los tipos de objetos y de funciones se pueden derivar otros (y se puede derivar recursivamente):
- Tipo arreglo, define una área contigua de memoria donde se ubican una cantidad no vacía de miembros que son objetos de un determinado tipo. Si llamamos T al tipo de sus elementos, entonces decimos que es un "arreglo de T". Un arreglo se caracteriza por el tipo de sus elementos y la cantidad de los mismos.
- Tipo estructura, aloja una secuencia no vacía de miembros objetos que pueden ser de diferentes tipos.
- Tipo unión, similar a la estructura pero memoria asignada a los miembros se solapa (todas comienzan al principio de la unión)

Tipos Derivados

- Tipo Función, las funciones se caracterizan por el tipo que devuelven y la cantidad y tipo de sus parámetros. Dado un tipo T la derivación en función es una función que devuelve T.
- Tipo puntero, pueden derivar de un tipo objeto o un tipo función al que genéricamente llamaremos tipo referenciado.
- Las reglas para derivar tipos se pueden aplicar recursivamente y así tener, por ejemplo “arreglo de estructuras” o “arreglos de arreglos de punteros a funciones”

Tipos, otras consideraciones

- Tipos agregados: la unión de tipo estructura y tipo arreglo.
- Tipos escalares: la unión de tipo aritmético y tipo puntero.
- ValorL: es una expresión que permite designar (ubicar) un objeto y recuperar su valor. El caso más simple es el identificador de una variable, por ejemplo de tipo básico. Otros ejemplos: un elemento de un arreglo o lo apuntado por un puntero.
- Un valorL es modificable si no fue calificado como constante, ni es una estructura o unión con un miembro calificado como constante. Tampoco son valorL modificables los arreglos (no hablo de sus elementos) y los tipos incompletos.
- ValorR: es el valor en si mismo es una expresión

Tipos de Datos, equivalencias

- 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, chequeos

- 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)

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/Alcance (Scope) - General

- 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.

Ámbito (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.

Ámbito y Vinculación - C

- 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)

Definiciones tentativas - Predefinido

- En C se admite una “mala práctica” por cuestiones históricas. Se puede definir varias veces una variable y tomar una sola como definición, el resto son “tentativas” que terminan actuando como declaración
  - En distintas unidades de traducción
  - En la misma pero a nivel archivo
- Identificador predefinido: en cada función está disponible sin necesidad de definirlo el identificador `__func__` que es arreglo de caracteres con el nombre de la función

Identificadores - Ejemplos

```
extern int var; //declaro var, definida en otro fuente
int varex; //ámbito el archivo, vinculación externa
static int varin; //ámbito el archivo, vinculación interna

int f1(void) //por defecto las funciones tiene vinculación externa
{
    int a; //defino a
    ...
}
static int f2(int arg) //vinculación interna
{
    int a; //diferente a la de f1, otro ámbito, sin vinculación
    static int b; //ámbito en f2 pero "vive" durante toda la
                  //ejecución
    int c = 1;
    int d, e = 2, *pi, arr[3]; //mal estilo en general, pero
                               //permitido
        ...
 }
```

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 (6 a partir de C23) espacios de nombre.
- Podemos repetir el mismo nombre en distintos espacios, pero no repetir uno dentro de un mismo espacio.
- 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 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 atributos (separados en estándar con sus prefijos y los que siguen a un prefijo)
- 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  }
```

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

Licencia