Identificadores

Alcance, Espacios de Nombre, Duración, y Enlace

Esp. Ing. José María Sola, profesor

Revisión 1.2.0
2021-12-13

Tabla de contenidos

Lista de ejemplos

1

Temas

- Identificadores.
- Máquinas de estado.
- Meta lenguajes.
- Visibilidad y Espacio de Nombres: Alcance léxico.
- Duración: Alcance dinámico.
- Vinculación: Asociación de identificadores de la misma entidad.

2

Introducción

Un *lenguaje de programación* puede ser definido y analizado en diferentes niveles: el nivel *léxico*, el nivel *sintáctico*, el nivel *semántico*, y el nivel *pragmático*. Dentro de las *restricciones* y *reglas semánticas* las que más se destacan son las asociadas a *tipos de datos* e *identificadores*. Los identificadores, o simplemente *nombres*, son *tokens* o *elementos léxicos* que se utilizan para referenciar entidades como variables, funciones, y tipos.

En este texto vamos a entrar en detalle en las definiciones, restricciones y reglas semánticas asociadas a identificadores.

El lenguaje de programación C es el LP (*lenguaje de programación*) principal que utilizo como línea conductora para la presentación de los temas y ejemplos, aunque también hay algunos ejemplos y ejercicios de investigación en otros lenguajes.

3

Los Identificadores en el Nivel Léxico

Los identificadores son una de las categorías en las que se pueden clasificar los tokens. La clasificación de los tokens tiene un grado de subjetividad y varía entre LP aunque hay muchas similitudes Para C, una categorización acepta de tokens es:

- palabra clave
- identificador
- constante
- cadena literal
- *punctuator* (símbolo de puntuación)

Léxicamente, los identificadores son una secuencia de *no-dígitos* y *dígitos*, donde el primer carácter debe ser no-dígito. La secuencia tiene una longitud mínima de un caracter y no hay límite específico para la longitud máxima. Los *dígitos* son: `0 1 2 3 4 5 6 7 8 9`. Los *no-dígitos* son las letras minúsculas y mayúsculas del alfabeto inglés, el guión bajo (*underscore*: _) y *otros caracteres definidos por la implementación*. [1] [2]

[1] La implementación es el compilador que hace realidad la especificación del lenguaje C.
[2] Esto permite caracteres propios de cada nación o cultura.

Ejemplo 3.1.

Sí son identificadores:

```
a
_
a1
abc3
_abc4
_1_2_3_
ñandú
emojis_#_son_válidos
total_general
TotalGeneral
```

No son identificadores:

```
1     // no pueden comenzar con dígito
1abc
1_abc
abc+3 // son tres lexemas
      // la palabra vacía no es un identificador
```

Las minúsculas y las mayúsculas son caracteres diferentes, significa que C es *case-sensitive*.

Ejemplo 3.2.

Los identificadores `TotalGeneral` y `totalgeneral` son distintos.

3.1. Lenguajes Formales e Identificadores

Del punto de vista de los *lenguajes formales*, los identificadores forman un *lenguaje infinito* (cardinalidad), siempre es posible generar un nuevo identificador porque su longitud no está acotada. Cada identificador, como toda palabra de un lenguaje, tiene una longitud (módulo) finita. El alfabeto está formado por los dígitos decimales, las minúsculas y mayúsculas inglés, el guión bajo y *otros caracteres definidos por la implementación*.

Se lo clasifica como un *Lenguaje Regular* o *Tipo 3* según la *Jerarquía de Chomsky*, porque se puede definir:

- una *expresión* regular que lo *represente*,
- un *autómata finito* que lo *reconozca*, y
- una *gramática regular* que lo *genere*.

En el libro *El Lenguaje de Programación C* por B. Kernighan & D. Ritchie [K&R1988], que por años sirvió como manual de referencia y estándar de facto de C, los autores especifican la sintaxis de los identificadores con reglas escritas en *lenguaje natural* (LN), pero con una alta precisión. Mientras que las keywords, al ser un LF finito, las definen por extensión.

3.1.1. Ejercicios

Para el LF identificadores:

1. Escriba una *expresión regular* que lo represente
2. Defina *formalmente* un *autómata finito* que lo reconozca.
3. Defina *formalmente* una *gramática regular* que lo genere.
4. Diseñe un *diagrama de sintaxis* ó *railroad diagram* que lo represente gráficamente.
5. Escriba un *BNF* (*Backus-Naur Form*) que lo describa.
   Idenfique los metasímbolos.
6. Escriba las reglas según la variante de *BNF* que utiliza el libro [K&R1988].
   Idenfique los metasímbolos.
7. Escriba una *regex* (*regular expression*) que permita *matchear* identificadores en un texto.
8. Busque la definición en LN de los identificadores en el libro [K&R1988] y determine si presenta ambigüedades.
9. Busque diferencias y simulitudes en la forma de los identificadores de: C, C ++, JavaScript, Go, Lisp, y SmallTalk.

3.2. Implementación del Lenguaje Formal Identificadores

En nuestro contexto, implementar implica una de dos alternativas:

1. Construir un mecanismo que dada un *string* (i.e., *cadena de caracteres*) indique si es un identificador o no según las reglas léxicas.
2. Construir un mecanismo que dado un *stream* (i.e., *flujo* ó *corriente de datos*) obtenga el lexema del siguiente identificador.

En C, vamos a encapsular cada mecanismo en funciones, propongo los siguientes prototipos para cada alternativa:

1. `bool IsId(const char *s);`
   Retorna `true` si en la cadena `s` hay un identificador, si no, `false`.
2. `char *GetNextId(FILE *in, char *s);`
   Lee de `in` el siguiente identificador y lo almacena en `s` como una cadena. Asume que `s` apunta a un buffer con suficiente espacio. Si pudo leer un identificador, retorna `s`, si no, retorna `NULL` y no se especifica qué queda en `s`.

Para ambas alternativas, el núcleo de la implementación es una *máquina de estados*. A su vez, la máquina de estado se puede implementar de múltiples formas. A continuación propongo diferentes implementaciones que utilizan distintas características del lenguaje C para implementar estados y transiciones.

Implementación #1

- Estados como variable `enum`.
- Transiciones como selección estructurada y actualización de variable.

Implementación #2

- Estados como *etiquetas*.
- Transiciones como selección estructurada y saltos incondicionales con `goto` (sí, el infame *goto*).

Implementación #3

- Estados como funciones recursivas.
- Transiciones como selección estructurada e invocaciones recursivas.

Implementación #4

- Estado como variable entera.
- Transiciones codificadas en tabla implementada como arreglo de arreglos.

Implementación #5

- Estados como funciones y variable puntero a función actual.
- Transiciones como selección estructurada, actualización de variable e invocaciones.

Implementación #6

- Estados como variable `enum`.
- Transiciones con función `GetNextState(s,c)`.

Se pueden diseñar muchas más implementaciones, que sean combinaciones de las anteriores o que apliquen otros conceptos como *orientación a objetos*.

3.2.1. Generadores de Scanners: Lex y Flex

Como vimos antes, los identificadores son un tipo de token, por lo tanto son reconocidos y tradados por *analizadores léxicos* o *scanners*, que son la parte de los compiladores que tratan el nivel léxico. La *generación automática* analizadores léxicos es posible con programas especiales llamados generadores de analizadores léxicos, entre los cuales se destacan *lex* y su contrapartida open source *flex*. Estos programas esperan en su entrada reglas, y a la salida producen un analizador léxico en C. Las reglas se especifican con *regex* (*regular expressions*) para la condición y con C para la acción.

3.2.2. Ejercicios

1. El objetivo es implementar la función `bool IsIdentificador(const char*);` que dado una cadena retorna si es un identificador.
   Las tareas son las siguientes:

   a. Escribir el contrato en `IsId.h`, indique con comentarios cualquier precondición.
   b. Escribir el programa `IsId-Test.c` que pruebe con `assert`.
   c. Escribir las siguientes implementaciones:

      i. `IsId-0-free.c`, sin restricciones.
      ii. `IsId-1-enum-sel-update.c`, sigue el modelo de implementación #1.
      iii. `IsId-2-label-sel-goto.c`, sigue el modelo de implementación #2.

iv. `IsId-3-fun-sel-rec.c`, sigue el modelo de implementación #3.

v. `IsId-4-int-table.c`, sigue el modelo de implementación #4.

vi. `IsId-5-funptr-sel.c`, sigue el modelo de implementación #5.

vii. `IsId-6-enum-getnextstate.c`, sigue el modelo de implementación #6.

viii `IsId-x.c`, proponer un modelo de implementación X e implementarlo.

d. Escribir un `Makefile` que pruebe todas las implementaciones.

2. El objetivo es escribir un programa que envía a `stdout` todos los identificadores, uno por línea, que lleguen por `stdin`.
   El programa debe recorrer `stdin` con llamadas a `GetNextId`. Las tareas son las siguientes:

   a. Escribir el contrato en `GetNextId.h` que declare `char *GetNextId(FILE *in, char *s);`.
      Indique con comentarios cualquier precondición.

   b. Escribir el programa `IdFilter.c` que su `main` invoca a `GetNextId`.

   c. Escribir las siguientes implementaciones:

      i. `GetNextId-0-free.c`, sin restricciones.

      ii. `GetNextId-1-enum-sel-update.c`, sigue el modelo de implementación #1.

      iii. `GetNextId-2-label-sel-goto.c`, sigue el modelo de implementación #2.

      iv. `GetNextId-3-fun-sel-rec.c`, sigue el modelo de implementación #3.

      v. `GetNextId-4-int-table.c`, sigue el modelo de implementación #4.

      vi. `GetNextId-5-funptr-sel.c`, sigue el modelo de implementación #5.

      vii. `GetNextId-6-enum-getnextstate.c`, sigue el modelo de implementación #6.

      viii `GetNextId-x.c`, proponer un modelo de implementación X e implementarlo.

   d. Escribir un `Makefile` que pruebe todas las implementaciones.

La prueba debe hacerse con un archivo de entrada, y la comparación de un archivo de salida esperado con la salida obtenida.

3. Rehacer el anterior ejercicio pero implementado como un programa *lex*: `IdFilter.l`.

4

Los Identificadores y la Especificación de los Lenguajes de Programación

Los identificadores se contrastan y relacionan con otros tokens de los LP.

4.1. Keywords

Como la gran mayoría de LP, C tiene identificadores que son reservados para dar un significado especial según el LP y actúan como *keywords* (*palabras clave*). Estas palabras son un sublenguaje finito de los identificadores. En el caso de C son 44:

```
auto        extern      short       while
break       float       signed      _Alignas
case        for         sizeof      _Alignof
char        goto        static      _Atomic
const       if          struct      _Bool
continue    inline      switch      _Complex
default     int         typedef     _Generic
do          long        union       _Imaginary
double      register    unsigned    _Noreturn
else        restrict    void        _Static_assert
enum        return      volatile    _Thread_local
```

Por ejemplo, aunque `if`, `for`, y `return` siguen las reglas de los identificadores son keywords que no pueden usarse como identificadores.

4.2. Identificadores Reservados

Son identificadores que se reservan para futuras evoluciones del LP o también, en el caso de C, para la biblioteca estándar.

La biblioteca estándar de C declara cientos de identificadores, por ejemplo: `printf`, `FILE`, `EOF`, `strlen`. Esos identificadores están reservados por la biblioteca estándar y resulta en un comportamiento indefinido si los declaramos en nuestros programas para otro uso.

En le caso de C, se reserva también otras conjuntos de identificadores, definidos por compresión: identificadores que comienzan con guión bajo.

Por último, especificación de la biblioteca estándar de C, anuncia que en futuras versiones podrían reservarse otros conjuntoos de identificadores. Por ejemplo, identificadores que comiencen con `str` o `mem`.

4.3. Identificadores Predefinidos

No son keywords y son similares a los identificadores normales pero el lenguaje, y no la biblioteca, los considera declarados como si los hubiésemos declarados nosotros.

El lenguaje C tiene solo un identificaador predefinido: `__func__`.
Contiene el nombre de la función que actualmente se está ejecutando. Se aplica durante el para el proceso de desarrollo y depuración para mejores diagnósticos.

La declaración es equivalente a que hayamos escrito al principio del bloque de cada función la siguiente definición:

`static const char __func__[] = "` *function-name* `";`

Donde *function-name* es el nombre de la función.

4.4. Límites

Aunque los identificadores no tienen una longitud máxima, la especificación de C establece máximos mínimos:

- para identificadores con enlace interno: 63 caracteres iniciales significantes.

- para identificadores con enlace externo: 31 caracteres iniciales significantes.

5

Los Identificadores en el Nivel Semántico

5.1. Categorías de Entidades Identificables

Los identificadores denotan diferentes categoría de entidades. En C son, estas son las categorías de las entidades identificables:

- Objetos (variables).
- Funciones.
- *Tags* (nombres) de estructuras, uniones o enumeraciones.
- Miembros de estructuras, uniones o enumeraciones (*constantes de enumeración*).
- Nombres de `typedef`.
- Etiquetas.

Este es una función que usa todos las categorías anteriores:

```
// Tipos de Identificador
void CadaTipoDeIdentificador(void){
 void NombreDeFunción(int nombreDeParámetro);
 typedef int nombreDeTipo;
 struct nombreDeStruct{ nombreDeTipo nombreDeCampo; };
 union nombreDeUnion{ int nombreDeCampo; double d; };
 enum nombreDeEnum{ nombreDeEnumerado };
 enum nombreDeEnum nombreDeVariable = nombreDeEnumerado;

 goto nombreDeEtiqueta;
```

```
nombreDeEtiqueta:
 return;
}
```

5.2. Atributos de los Identificadores

Los identificadores tienen los siguientes atributos:

Visibilidad o Alcance léxico (*scope*)
: Determina la región del texto del programa donde el identificador se conoce y se puede usar.

Espacio de nombres (*namespace*)
: Determina conjuntos disjuntos de identificadores. Esto permite que un mismo identificador en un mismo alcance denote entidades distintas, si y solo si está en diferente espacio de nombres.

Vinculación (*linkage*)
: Determina si dos identificadores iguales refieren a la misma entidad o determina si dos identificadores en diferentes unidades de traducción refieren a la misma entidad.

Duración de almacenamiento de datos (*storage duration*)
: Para los identificadores de objetos se establece el tiempo de vida (*lifetime*) de su almacenamiento.

5.3. Scope: Alcance Léxico

Por cada identificador se define un alcance; que es la región del texto del programa donde ese identificador es *visible*, es decir, es accesible para su uso. La región del texto se indica como un intervalo entre posiciones dentro del texto. Si dos entidades diferentes están designadas por un mismo indicador, entonces ocurre que:

- los identificadores tienen diferente alcance, ó que
- los identificadores pertenecen a diferente espacio de nombres.

Ejercicio

Escriba un ejemplos para cada situación, es decir, dos ejemplos, cada uno con dos entidades con mismmo identificador.

Existen cuatro clases de alcances:

- Archivo
- Prototipo
- Función
- Bloque

5.3.1. Alcance de Función

Los *nombres de etiqueta* son la única categora de identificador que tienen *alcance de función*. Se lo declara *implícitamente* con la sintaxis *identificador* : *sentencia*. El alcance del resto de los identificadores depende de la ubicación de su declaración. Se lo usa con una sentencia `goto`, en cualquier parte de la función, para llevar el flujo de ejecución a la sentencia etiquetada.

Ejercicio

Indique la cetegoría del identificador `OtraVez` y su alcance.

```
void f(void){
 int veces = 10;
OtraVez:
 putchar('a');
 veces--;
 if(veces) goto OtraVez;
 return;
}

void g(void){
 int veces = -10;
 {
OtraVez:
  putchar('a');
  veces++;
 }
 if(veces) goto OtraVez;
 return;
```

```
}
```

5.4. Namespaces: Espacio de nombres

Los identificadores se agrupan en diferentes espacios de nombres que no interfieren entre sí. Por eso, un mismo identificador puede ser utilizado para nombrar diferentes entidades, si los usos tienen diferentes propósitos, inclusive dentro de un mismo alcance.

Los espacios de nombres permiten que existan identificadores idénticos que compartan alcance pero que refieran a entidades diferentes, sin posiblidad de colisión.

Hay espacios de nombres para las diferentes categorías de identificadores:

- Etiquetas.
- Nombres (*tags*) de estructuras, uniones y enumeraciones.
- Miembros de estructuras o uniones; cada estructura o unión tiene su espacio de nombre independiente para sus miembros.
- Identificadores ordinarios.

Si en un determinado alcance existe más de un declarador para un mismo identificador y cada identificador refiere a diferentes entidades, el contexto sintáctico elimina la ambigüedad en el uso. Esto implica que dado un identificador, se puede determinar sintácticamente por su uso si refiere a una etiqueta, tag, miembro de una estructura o unión, o a otra entidad.

5.5. Linkage: Vinculación

La vinculación es un mecanismo que permite que un identificador declarado en diferentes alcances o en un mismo alcance más de una vez, refiera al mismo objeto o función; no pueden vincularse diferentes identificadores.

Hay tres tipos de vinculación: ninguna, externa y interna.

Hay entidades para los cuales no tiene sentido la vinculación.

Para los objetos y las funciones, la vinculación interna permite compartirlos dentro de una misma unidad de traducción, mientras que la vinculación externa permite compartirlos entre todas las unidades de traducción que forman el programa.

5.6. Binding

Binding es un concepto más general que vinculación, es la sociación entre dos entidades, mientras que la vinculación se particulariza entre entidades y objetos o funciones.

5.7. Lifetime: Duración de Almacenamiento de Objetos

En el contexto de C, un *objeto* es una región de almacenamiento de datos disponible en el ambiente de ejecución, su contenido representa valores. Esa región de almacenamiento es la *memoria principal*. El ejemplo más común de objeto es la variable, pero hay objetos que no son variables.

Ejercicio

¿Cuáles objetos no son variables?

Un objeto tiene un *tipo de dato* y una *clase de almacenamiento*. La clase de almacenamiento determina el *tiempo de vida* o *vigencia* del objeto. El tipo de dato determina la *semántica del valor* y las *operaciones aplicables* a él.

La duración del almacenamiento de un objeto determina su *tiempo de vida*, que es la porción de la ejecución del programa durante la cual se garantiza que el almacenamiento para el objeto está reservado.

Mientas un objeto vive, su dirección es constante y retiene su último valor almacenado a lo largo de su vida. El comportamiento queda indefinido cuando se trata de acceder a un objeto cuya vida finalizó. El valor de un puntero se vuelve indeterminado cuando el objeto al que apunta lleva al fin de su tiempo de vida.

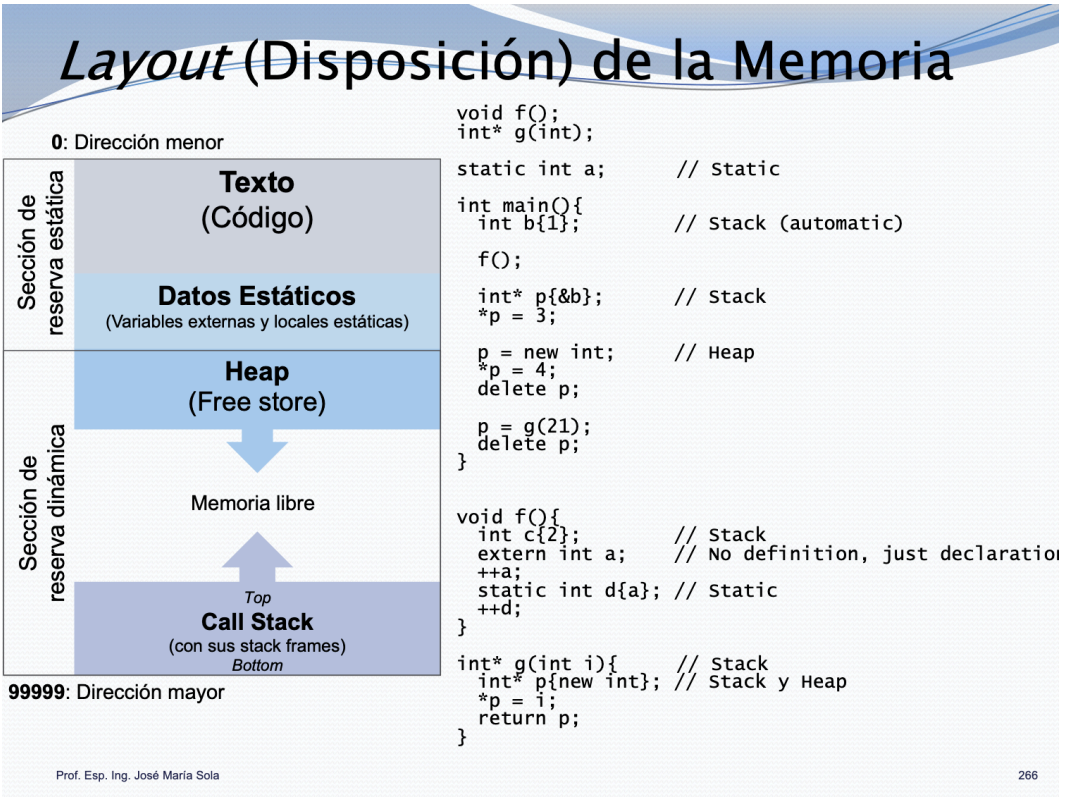
Hay cuatro tipo de duraciones del almacenamiento de un objeto:

- Estática
- Thread (hilo)
- Automática
- Alocada (de allocated, i.e. reservada)

Del punto de vista del lenguaje, hay tres clases: estática, thread y automática. La clase de almacenamiento se especifica con palabras reservadas y con el

contexto de la declaración del objeto. _La cuarta duración, alocada, se aplica con invocaciones funciones de la biblioteca estándar: `malloc`, `calloc`, `realloc` y `free`.

5.8. Stack, Heap & Static Data



6

Preguntas de Final

- Indique la categoría a la que pertenece este identificador.
- Indique el alcance de este identificador.
- Indique el espacio de nombre de este identificador.
- Indique si este código con un mismo identificador que se usa con diferentes objetivos es semánticamente válido.
- Indique la duración de este identificador.
- Indique cuantas entidades distintas con el mismo identificador existen a la vez en el instante tal.
- Indique la vinculación de este identificador.
- Explique el proceso por el cual dos variables en diferente UT refieren al mismo objeto.

7

Bibliografía

- ISO/IEC 9899:2-17, INTERNATIONAL STANDARD, Programming languages — C (2017)

8

Changelog

1.2.0+2021-12-13

- ADDED: Alcance de función.
- ADDED: Introducción sobre binding.
- ADDED: Más ejercicios de final.
- ADDED: Bibliografía.
- FIXED: Ancho de imagen de memoria.

1.1.0+2021-12-11

- ADDED: Explicación para todos los conceptos semánticos.
- FIX: Cantidad de keywords.

1.0.0+2021-12-09

- Primera versión.