Arquitectura de sistemas

Abelardo Pardo

University of Sydney
School of Electrical and Information Engineering
NSW, 2006, Australia
Autor principal del curso de 2009 a 2012

Iria Estévez Ayres

Damaris Fuentes Lorenzo

Pablo Basanta Val

Pedro J. Muñoz Merino

Hugo A. Parada

Derick Leony

Universidad Carlos III de Madrid
Departamento de Ingeniería Telemática
Avenida Universidad 30, E28911 Leganés (Madrid), España

Capítulo 3. Declaración de variables

Tabla de contenidos

En Java, el acceso a campos y métodos de un objeto puede ser controlado mediante tres ámbitos: público, privado y protegido. El lenguaje C dispone de reglas más simples para controlar lo que se conoce como el "ámbito de validez" de una variable.

3.1. Ámbito de una variable

El "ámbito de validez" de una variable está compuesto por las porciones de código desde donde se puede acceder y manipular su contenido. Estas porciones suelen corresponder con diferentes bloques de código escritos entre llaves ("{}"). Por ejemplo, cuando una función llama a otra, se abre un nuevo ámbito (el de la función llamada) dentro de otro (el de la función que llama) que desaparece cuando termina la función. Hay ciertas variables con ámbitos de validez intuitivos, como por ejemplo las que se definen al comienzo de una función. Pero C permite modificar estos ámbitos mediante el uso de prefijos en la declaración.

3.1.1. Variables globales

Toda variable declarada fuera de las funciones tiene ámbito global, es decir, puede ser accedida desde cualquier parte del programa. El siguiente código muestra un ejemplo de esta situación.

```
1  int number;
2  int function()
3  {
4      number++;
5      return number;
6  }
7
8  int main(int argc, char *argv[])
9  {
10     number = 0;
11     function();
12     return 0;
13 }
```

La variable `number` se declara en la línea 1, fuera de una función, y por tanto es global. A continuación se accede en las líneas 4 y 10.

Pero para acceder correctamente a una variable global hay que cumplir dos requisitos más derivados de la forma en cómo el compilador procesa los ficheros. Si la variable está declarada en el mismo fichero, esta declaración debe preceder su uso (el compilador sólo lee el fichero en un único paso). Si la variable está declarada en otro fichero, se debe incluir en el fichero exactamente la misma declaración pero con el prefijo "`extern`" (el compilador sólo recuerda información del fichero que está procesando).

Fichero 1

```
1  int number = 10;
2  int main(int argc, char *argv[])
3  {
4      return 1;
5  }
```

Fichero 2

```
1  extern int number;
2  int function()
3  {
4      number++;
5      return number;
6  }
```

La variable `number` se define como global en la línea 1 del fichero 1. Para poder acceder a ella en la línea 1 del fichero 2 se replica su declaración (sin inicializar) añadiendo el prefijo "`extern`". La variable global se accede en la línea 4 del fichero 2. Si se omite la línea 1 de este fichero, el compilador emite el error de que la variable `number` no ha sido declarada. Si en el fichero 2 se incluye la definición sin el prefijo "`extern`", el compilador notificará el error de que se ha definido una variable múltiples veces.

Sugerencia

Copia y pega el contenido de los dos ficheros del ejemplo en los ficheros `fichero1.c` y `fichero2.c`. Compila con el comando **gcc -Wall -o programa fichero1.c fichero2.c**. Comprueba que se genera el ejecutable con nombre `programa` y ejecútalo con el comando **./programa**. No debe imprimir nada por pantalla. Haz cambios en las declaraciones para comprobar las reglas que impone el compilador.

3.1.2. Variables estáticas

Cualquier declaración de una variable puede tener el prefijo "`static`". Las variables estáticas en C tienen las siguientes dos propiedades:

1. No pueden ser accedidas desde otro fichero. Por tanto, los prefijos "`extern`" y "`static`" no pueden ser utilizados en la misma declaración.
2. Mantienen su valor a lo largo de toda la ejecución del programa independientemente del ámbito en el que estén definidas.

Como consecuencia de estas dos propiedades se derivan los siguientes casos:

1. Si una variable estática está declarada fuera de las funciones, será accessible únicamente por el código que le siga en el mismo fichero de su declaración.
2. Si una variable estática está declarada en una función, sólo será accesible desde esa función y mantendrá su valor entre ejecuciones de la función.

   Este comportamiento es contra intuitivo porque estas variables se declaran en el mismo lugar que el resto de variables en una función, pero mientras que estas adquieren valores nuevos con cada ejecución, las estáticas conservan estos valores entre ejecuciones.

El siguiente programa muestra un ejemplo del comportamiento de una variable estática definida en una función.

```
1  #include <stdio.h>
2  int funcion()
3  {
4      static int número = 10; /* Variable estática */
5      número++;               /* Mantiene el valor de la ejecución anterior */
6      return número;
7  }
8  int main()
9  {
10     /* Imprime el resultado de dos invocaciones a la función */
11     printf("%d\n", funcion()); /* Imprime el número 11 */
```

```
12      printf("%d\n", funcion()); /* ¡Imprime el numero 12! */
13      return 0;
14  }
```

La línea 4 declara la variable estática `número` y la inicializa con el valor 10. La primera vez que se ejecuta esta función, por tanto, la variable tiene este valor al comienzo y se incrementa. Por este motivo, la línea 11 imprime el valor 11 por pantalla. Pero al ser invocada por segunda vez la función, la variable `número` mantiene su valor y por tanto se incrementa a 12, que es lo que se imprime en la línea 12.

Sugerencia

Copia y pega el programa del ejemplo anterior en un fichero de texto en tu entorno de desarrollo. Compila con el comando **gcc -o programa fichero.c** reemplazando **fichero.c** por el nombre del fichero que hayas creado. Ejecuta el programa con el comando **./programa** y comprueba que el resultado coincide con la explicación.

El siguiente ejemplo muestra el comportamiento de una variable estática definida fuera de una función.

Fichero 1

```
/* Variable global */
int numero = 10;
int main(int argc, char *argv[])
{
numero++;
return 0;
}
/* Accesible en este fichero a partir de este punto */
static int coeficiente = 20;
void funcion2()
{
numero++;coeficiente++;
}
```

Fichero 2

```
extern int numero;
/* Variable coeficiente no
es accesible desde este fichero */
int funcion()
{
numero++;
return numero;
}
```

La variable `numero` se declara global y puede ser accedida por otro fichero siempre y cuando se incluya su declaración con el prefijo "`extern`". Sin embargo, la variable `coeficiente` sólo es accesible en el fichero en el que se declara y a partir de esa declaración (no está visible en la función `main`.

3.1.3. Ensombrecimiento de variables

El problema de "ensombrecimiento" se produce cuando en un ámbito de validez se define una variable con nombre idéntico a otra válida en un ámbito superior. El siguiente ejemplo ilustra esta situación:

```
1   #include <stdio.h>
2   int numero = 10;
3   void funcion()
4   {
5       int numero;  /* Colisiona con variable global */
6       numero = 20; /* ¿A qué variable se le asigna el valor? */
7   }
8   int main(int argc, char *argv[])
9   {
10      funcion();
11      /* Imprime el valor de numero */
12      printf("%d\n", numero); /* ¿Qué valor imprime? */
13      return 0;
14  }
```

Las declaraciones de las líneas 2 y 5 son idénticas pero están hechas en ámbitos diferentes (global frente a local a la función). El compilador permite esta declaración. Pero cuando se ejecuta la función se produce el ensombrecimiento: la variable global `numero` no puede ser accedida, pues ese nombre se refiere a la variable local a la función. Al terminar la función, la variable global ya no está "ensombrecida" y vuelve a ser accesible y se imprime su valor (10) en la línea 12.

Sugerencia

Copia y pega este programa en un fichero de texto en tu entorno de desarrollo. Añade más declaraciones, compila y ejecuta el programa para verificar la política de ensombrecimiento del compilador.

3.2. Definición de sinónimos de tipos con `typedef`

C permite definir sinónimos para los tipos de datos mediante el operador `typedef` y la siguiente sintaxis:

```
typedef tipo_de_datos_ya_definido sinónimo
```

Por ejemplo, la siguiente línea define `entero` como tipo sinónimo de `int`.

```
typedef int entero
```

Este operador se usa con frecuencia para abreviar los nombres de las estructuras de datos. El nombre del tipo de datos de una estructura es `struct` seguido de su nombre. Con `typedef` se puede definir un sinónimo más compacto tal y como se muestra en el siguiente ejemplo.

```
1  #define SIZE_FIRST 100
2  #define SIZE_LAST 200
3  #define NUM_CONTACTS 100
4
5  /* Definición de la estructura */
6  struct contact_information
7  {
8      char firstname[SIZE_FIRST];
9      char lastname[SIZE_LAST];
10     unsigned int homephone;
11     unsigned int mobilephone;
12 };
13 /* Declaración de variable */
14 struct contact_information person2;
15 /* Definición del sinónimo */
16 typedef struct contact_information contact_info;
17 /* Declaración utilizando el sinónimo */
18 contact_info person1, contacts[NUM_CONTACTS];
```

La línea 16 define `contact_info` como sinónimo de la estructura. La linea 18 utiliza este sinónimo para declarar dos variables más de este tipo. La definición de una estructura y un sinónimo se pueden combinar en un único bloque tal y como se muestra en el siguiente ejemplo, aunque no es aconsejable, por legibilidad:

```
/* Definición de la estructura y su sinónimo */
typedef struct contact_information
{
    char firstname[SIZE_FIRST];
    char lastname[SIZE_LAST];
    unsigned int homephone;
    unsigned int mobilephone;
```

```
} contact_info;
```

3.3. Bibliografía de apoyo

- ***Declaración de variables (teoría)***: ""Programación en C: Metodología, algoritmos y estructura de datos" ", páginas 258-263.
- ***Problemas resueltos***: "Problemas resueltos de Programación en Lenguaje C", ejemplo 5.4 (página 124).

3.4. Ejercicios

1. Considera las siguientes declaraciones de variables (el código que no se muestra no modifica el valor de ninguna de las variables).

Fichero 1	Fichero 2	Fichero 3

`int n1 = 1;`
`static int n2 = 2;`
`int n3 = 3;`
 	
`void function1()`
`{`
`int n1 = 10;`
`....`
`}`
`void function2()`
`{`
`....`
`}`
	`extern int n1;`
	`static int n2 = 20;`
	
	`void function3()`
	`{`
	`static int n1 = 100;`
	
	`n1++;`
	`....`
	`}`
	`void function4()`
	`{`
	`....`
	`}`
		`int n4 = 4;`
		`static int n2 = 200;`
		
		`void function5()`
		`{`
		`static int n2 = 2000;`
		
		`n2++;`
		`....`
		`}`
		`void function6()`
		`{`
		`....`
		`}`

Responder a la siguientes preguntas.

Pregunta	Respuesta
¿Qué valor tiene `n1` en `function1`?	
¿Qué valor tiene `n1` en `function2`?	
¿Qué ámbito de validez tiene la variable `n4`?	
¿Qué sucede si se accede a `n4` en `function1` o `function2`?	
¿Qué valor tiene al final de la función `function3` la variable `n1` la primera vez que se ejecuta? ¿Y la siguiente?	
¿Qué valor tiene al final de la función `function4` la variable `n1` la primera vez que se ejecuta? ¿Y la siguiente?	
¿Cuántos casos de ensombrecimiento de variable hay en este programa?	
¿Que valor tiene la variable `n2` la segunda vez que se ejecuta `function6`?	
¿Qué valor tiene al final de la función `function5` la variable `n2` la primera vez que se ejecuta? ¿Y la siguiente?	

2. Un programador con un estilo de programación "mejorable" ha definido la siguiente estructura de datos en una aplicación:

```
struct la_estructura_de_datos_mas_importante_de_la_aplicacion
{
    int a;
    int b;
};
```

Necesitas declarar numerosas variables de ese tipo en varios lugares de tu código. No puedes modificar el código escrito por el otro programador. ¿Qué propones para facilitar estas

declaraciones?

3. En un programa que tiene un número muy elevado de ficheros se quiere guardar el nombre y apellidos del autor de cada fichero (que no es el mismo para todos) en una variable. El nombre de esta variable debe ser igual en cada fichero. ¿Cómo declararías estas variables en cada fichero? ¿De qué tipo?

4. ¿Puedes pensar una funcionalidad en un programa para la que se necesite una variable estática en una función?

5. ¿Cómo guardarías el número de veces que se ha llamado a una función de un programa pero sin definir variables globales?