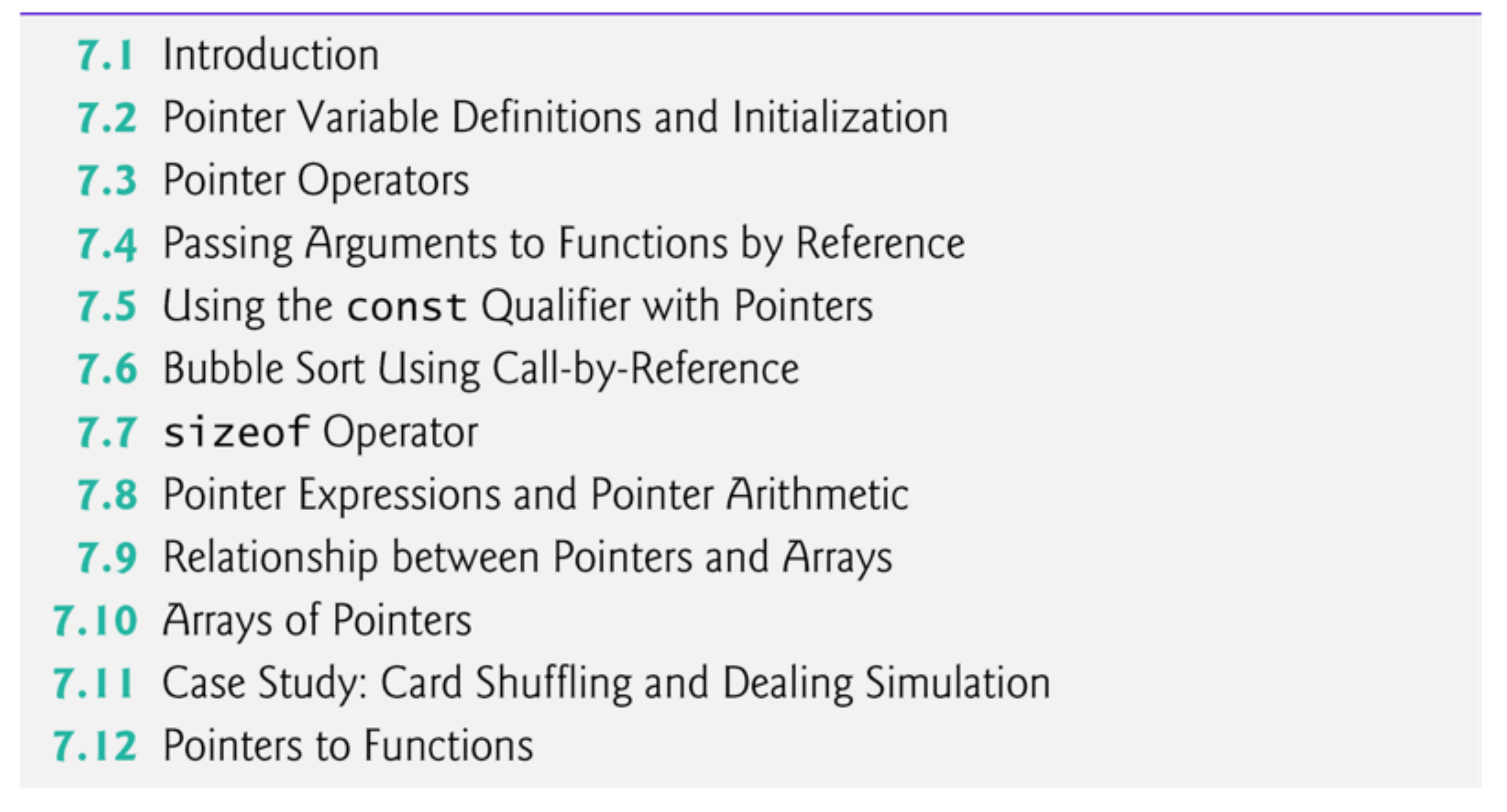
Computer Programming I

Ming-Feng Tsai (Victor Tsai)

Dept. of Computer Science
National Chengchi University

C Pointers



7.1 Introduction

7.2 Pointer Variable Definitions and Initialization

7.3 Pointer Operators

7.4 Passing Arguments to Functions by Reference

7.5 Using the `const` Qualifier with Pointers

7.6 Bubble Sort Using Call-by-Reference

7.7 `sizeof` Operator

7.8 Pointer Expressions and Pointer Arithmetic

7.9 Relationship between Pointers and Arrays

7.10 Arrays of Pointers

7.11 Case Study: Card Shuffling and Dealing Simulation

7.12 Pointers to Functions

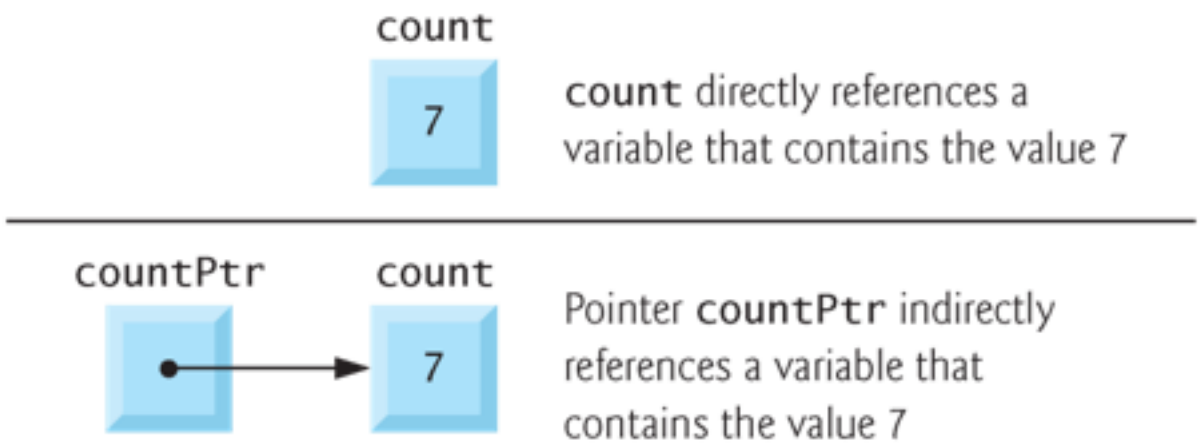
Introduction

- Pointers enable programs to simulate call-by-reference and to create and manipulate dynamic data structures, i.e., data structures that can grow and shrink at execution time, such as linked lists, queues, stacks and trees.
- Chapter 10 examines the use of pointers with structures.
- Chapter 12 introduces dynamic memory management techniques and presents examples of creating and using dynamic data structures.

Pointer Variable Definitions and Initialization

- Pointers are variables whose values are **memory addresses**.
- Normally, a variable directly contains a specific value.
- A pointer, on the other hand, contains an **address** of a variable that contains a specific value.
- Referencing a value through a pointer is called **indirection**.

Pointer Variable Definitions and Initialization (Cont.)



count directly references a variable that contains the value 7

Pointer countPtr indirectly references a variable that contains the value 7

Fig. 7.1 | Directly and indirectly referencing a variable.

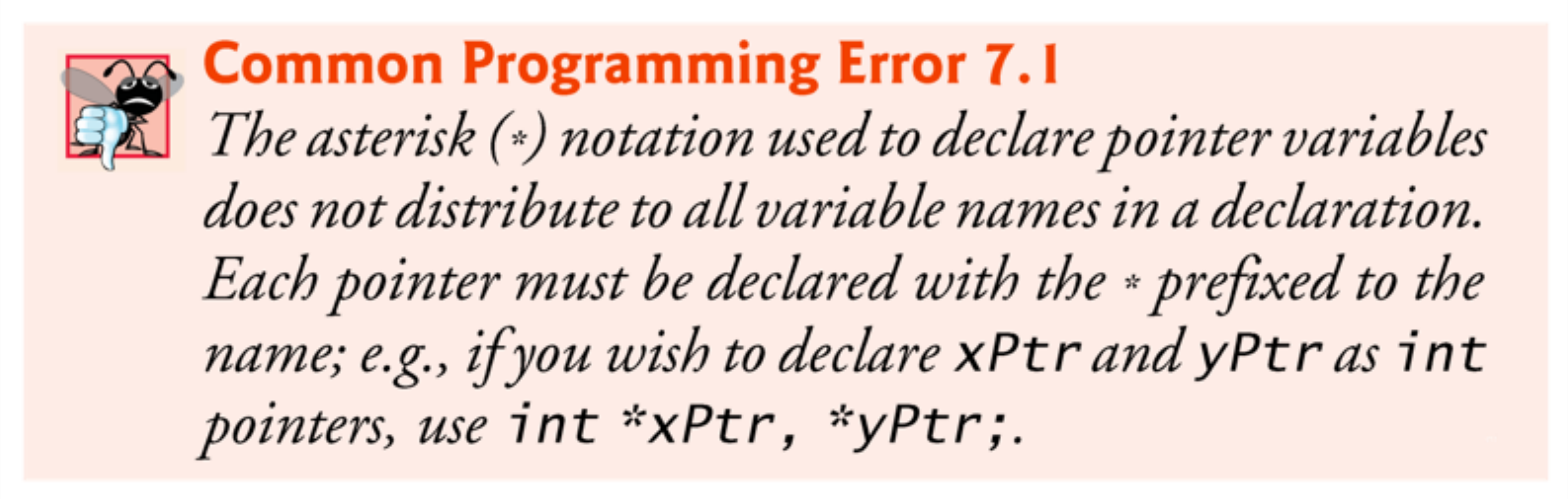
Pointer Variable Definitions and Initialization (Cont.)

- Pointer definition

```
int *countPtr;
```

  - **`countPtr`**: a pointer to int
  - `*`: indicates the variable being defined as a pointer
  - Pointers can be defined to point to objects of any type

Pointer Variable Definitions and Initialization (Cont.)



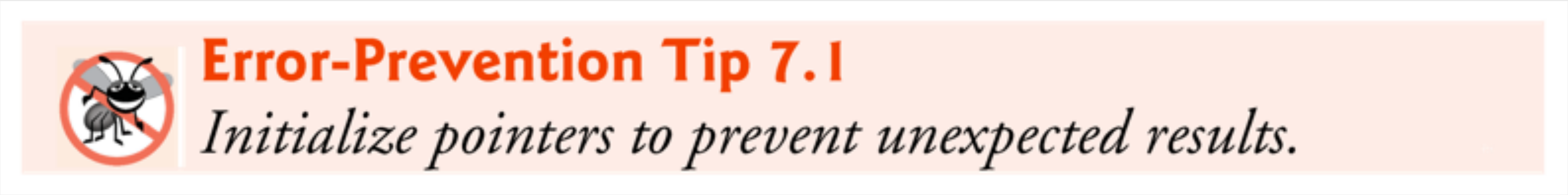
Common Programming Error 7.1

The asterisk (*) notation used to declare pointer variables does not distribute to all variable names in a declaration. Each pointer must be declared with the * prefixed to the name; e.g., if you wish to declare `xPtr` and `yPtr` as `int` pointers, use `int *xPtr, *yPtr;`.

Pointer Variable Definitions and Initialization (Cont.)

- A pointer may be initialized to NULL, 0 or an address.
- A pointer with the value NULL points to nothing.
  - Initializing a pointer to 0 is equivalent to initializing a pointer to NULL, but NULL is preferred.

Pointer Variable Definitions and Initialization (Cont.)



Error-Prevention Tip 7.1

Initialize pointers to prevent unexpected results.

Pointer Operators

- The **&**, or address operator, is a unary operator that returns the address of its operand.
- For example, assuming the definitions

```
int y = 5;
int *yPtr;

yPtr = &y;
```

  - Assigns the address of the variable **y** to pointer variable **yPtr**
  - Variable **yPtr** is then said to “point to” **y**.

Pointer Operators (Cont.)

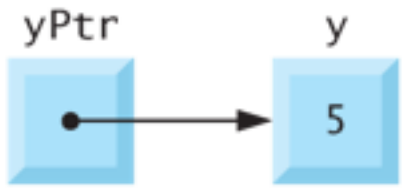


Fig. 7.2 | Graphical representation of a pointer pointing to an integer variable in memory.

Pointer Operators (Cont.)

- The operand of the address operator (**&**) must be a **variable**;
- The address operator (**&**) cannot be applied to constants, to expressions or to variables declared with the storage-class register.

Pointer Operators (Cont.)

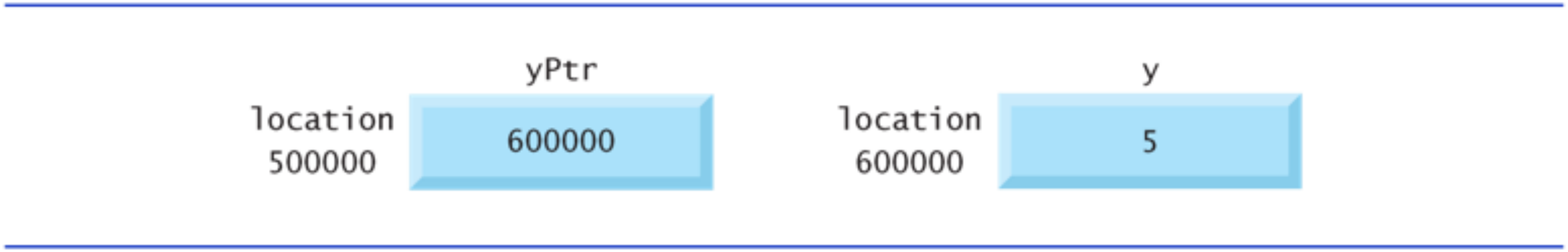


Fig. 7.3 | Representation of y and yPtr in memory.

Pointer Operators (Cont.)

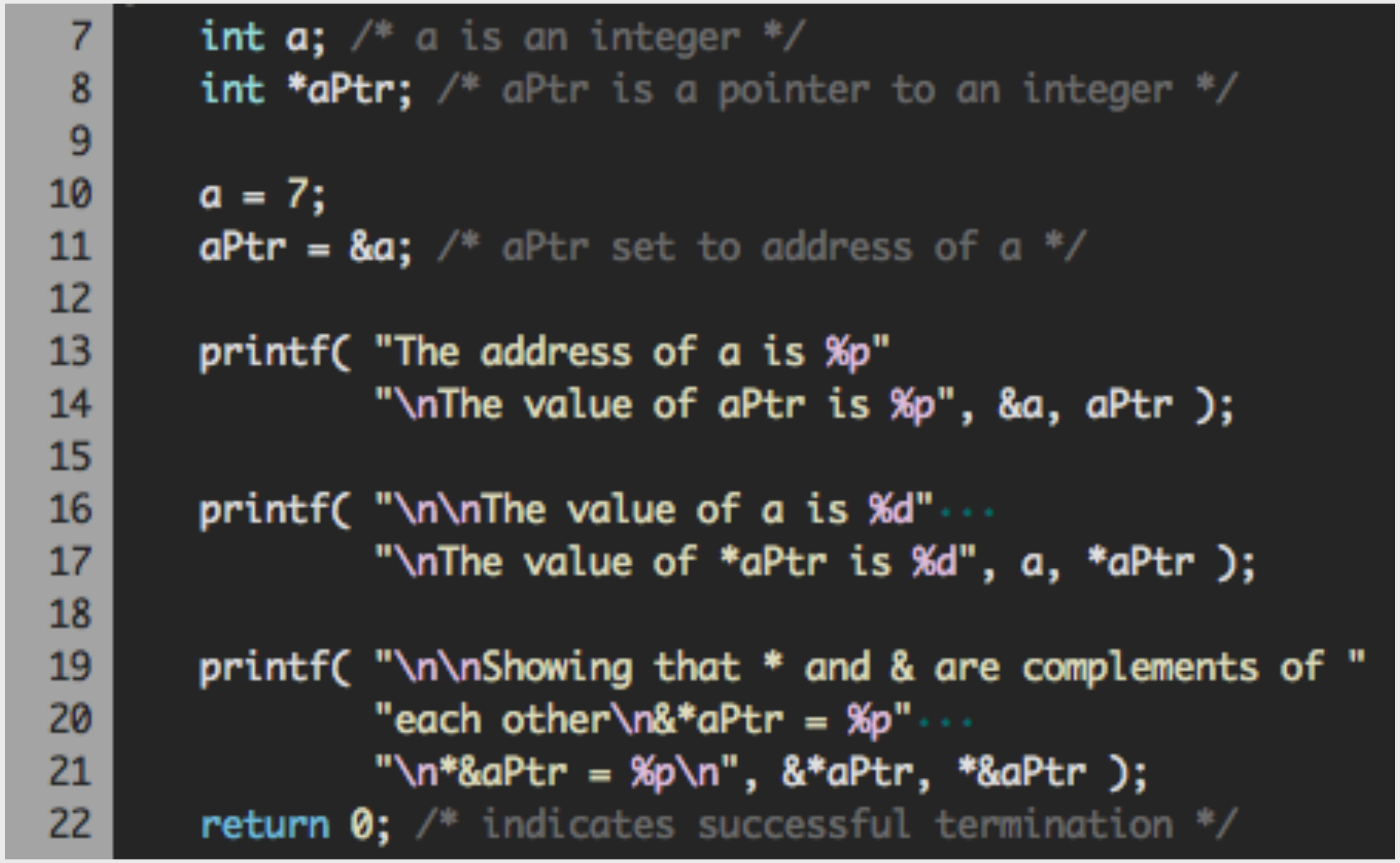
- The unary `*` operator, commonly referred to as the indirection operator or dereferencing operator, returns the value of the object to which its operand (i.e., a pointer) points.
- For example, the statement

```
printf("%d", *yPtr);
```

  - prints the value of variable `y`, namely 5.
- Using `*` in this manner is called dereferencing a pointer.

Pointer Operators (Cont.)

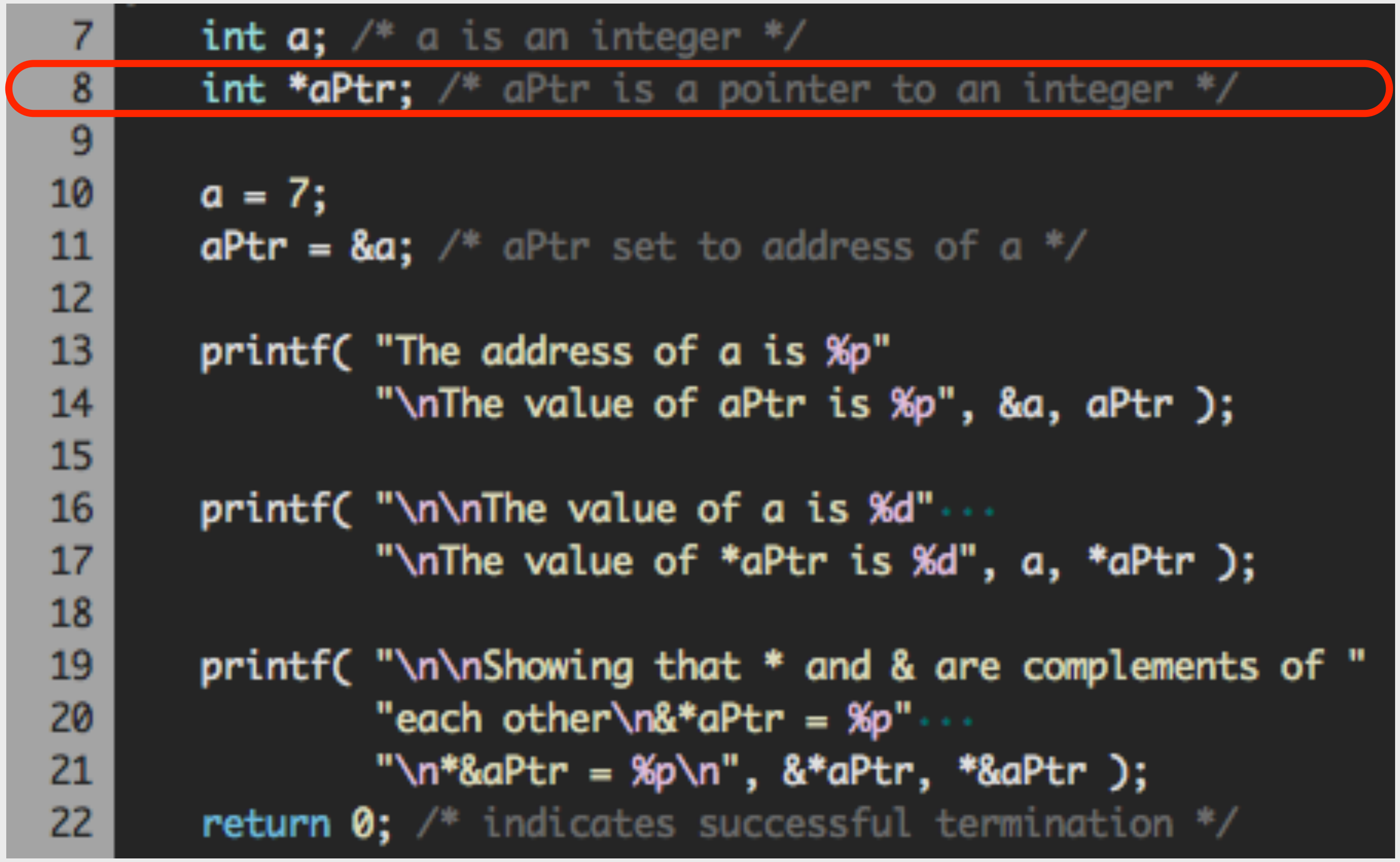
- Example: fig07_04.c



```
   int a; /* a is an integer */
   int *aPtr; /* aPtr is a pointer to an integer */

   a = 7;
   aPtr = &a; /* aPtr set to address of a */

   printf( "The address of a is %p"
           "\nThe value of aPtr is %p", &a, aPtr );

   printf( "\n\nThe value of a is %d"
           "\nThe value of *aPtr is %d", a, *aPtr );

   printf( "\n\nShowing that * and & are complements of "
           "each other\n&*aPtr = %p"
           "\n*&aPtr = %p\n", &*aPtr, *&aPtr );
   return 0; /* indicates successful termination */
```

Pointer Operators (Cont.)

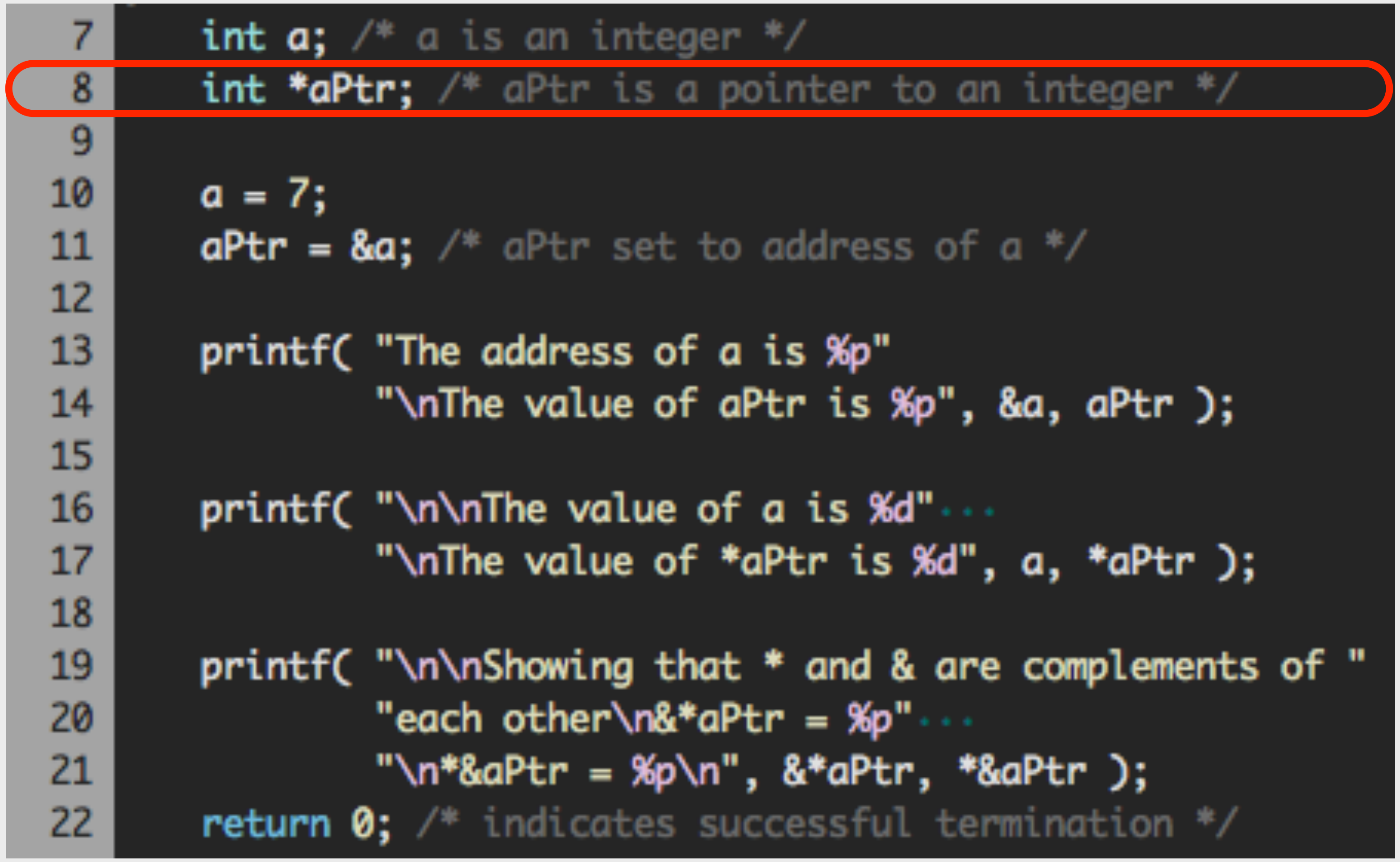
- Example: fig07_04.c



```
   int a; /* a is an integer */
   int *aPtr; /* aPtr is a pointer to an integer */

   a = 7;
   aPtr = &a; /* aPtr set to address of a */

   printf( "The address of a is %p"
           "\nThe value of aPtr is %p", &a, aPtr );

   printf( "\n\nThe value of a is %d"
           "\nThe value of *aPtr is %d", a, *aPtr );

   printf( "\n\nShowing that * and & are complements of "
           "each other\n&*aPtr = %p"
           "\n*&aPtr = %p\n", &*aPtr, *&aPtr );
   return 0; /* indicates successful termination */
```

Pointer Operators (Cont.)

- Example: fig07_04.c

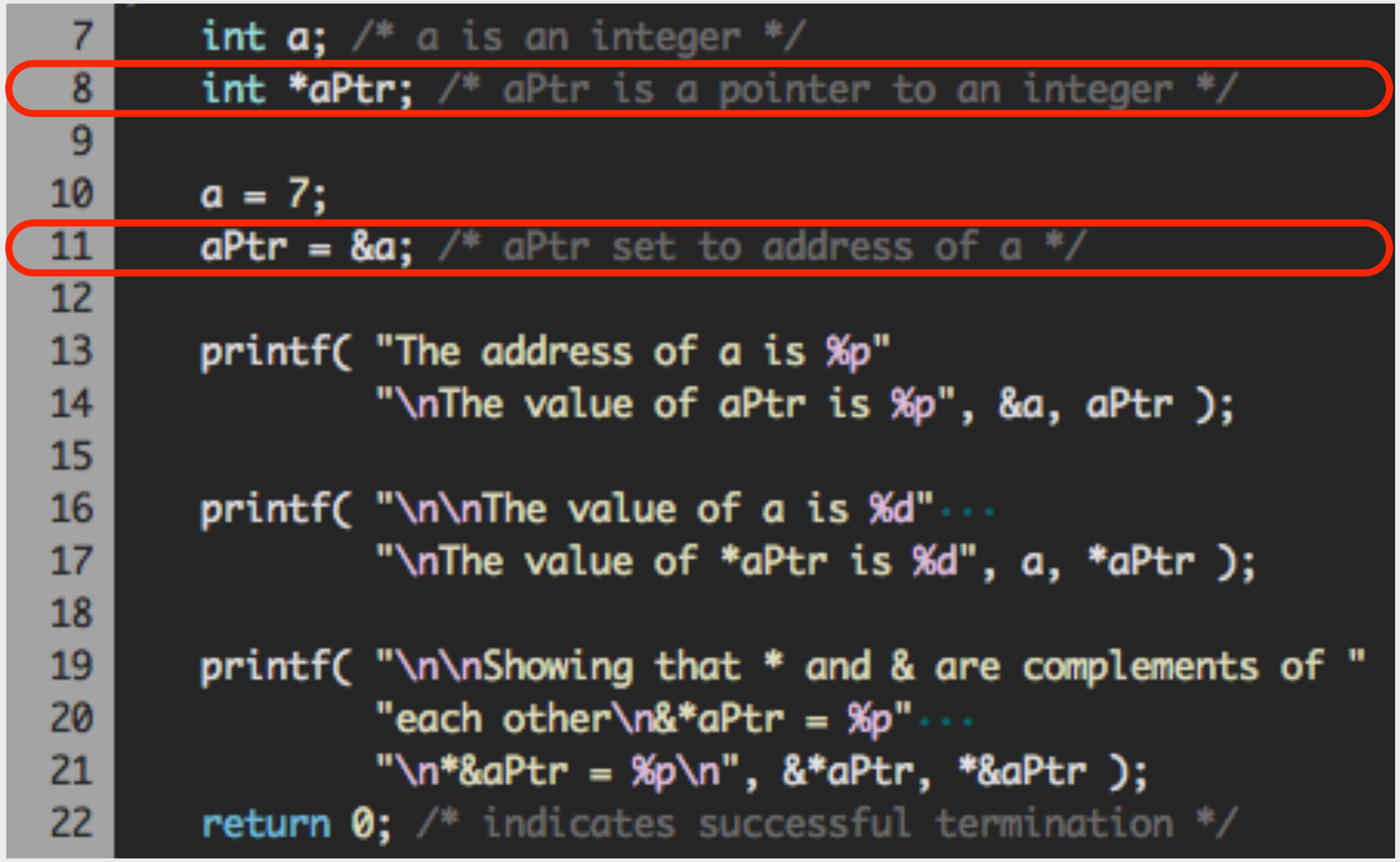


```
int a; /* a is an integer */
int *aPtr; /* aPtr is a pointer to an integer */

a = 7;
aPtr = &a; /* aPtr set to address of a */

printf( "The address of a is %p"
        "\nThe value of aPtr is %p", &a, aPtr );

printf( "\n\nThe value of a is %d"
        "\nThe value of *aPtr is %d", a, *aPtr );

printf( "\n\nShowing that * and & are complements of "
        "each other\n&*aPtr = %p"
        "\n*&aPtr = %p\n", &*aPtr, *&aPtr );
return 0; /* indicates successful termination */
```

declare a pointer to an integer

Pointer Operators (Cont.)

- Example: fig07_04.c

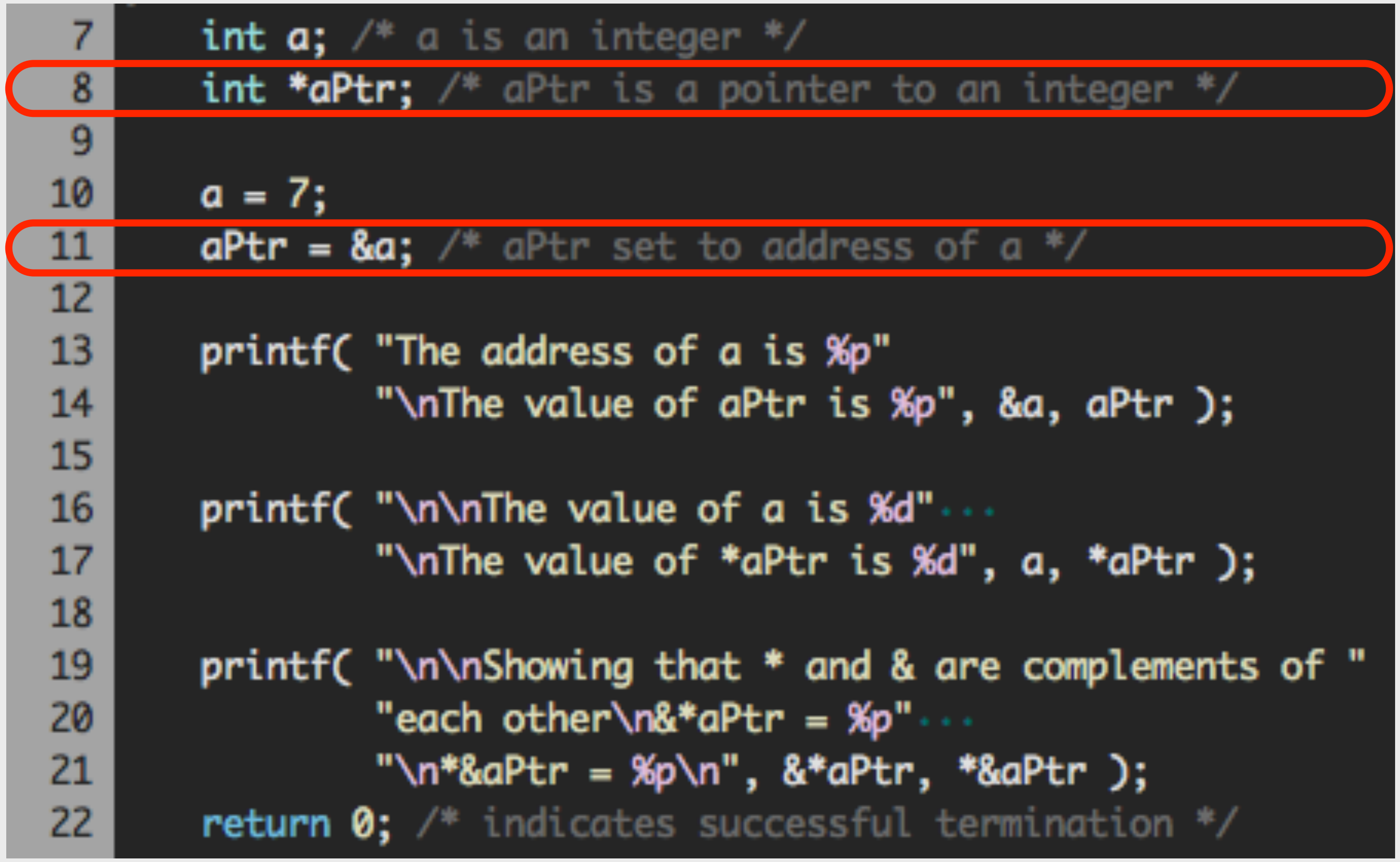


```c
int a; /* a is an integer */
int *aPtr; /* aPtr is a pointer to an integer */

a = 7;
aPtr = &a; /* aPtr set to address of a */

printf( "The address of a is %p"
        "\nThe value of aPtr is %p", &a, aPtr );

printf( "\n\nThe value of a is %d"
        "\nThe value of *aPtr is %d", a, *aPtr );

printf( "\n\nShowing that * and & are complements of "
        "each other\n&*aPtr = %p"
        "\n*&aPtr = %p\n", &*aPtr, *&aPtr );
return 0; /* indicates successful termination */
```

declare a pointer to an integer

Pointer Operators (Cont.)

- Example: fig07_04.c



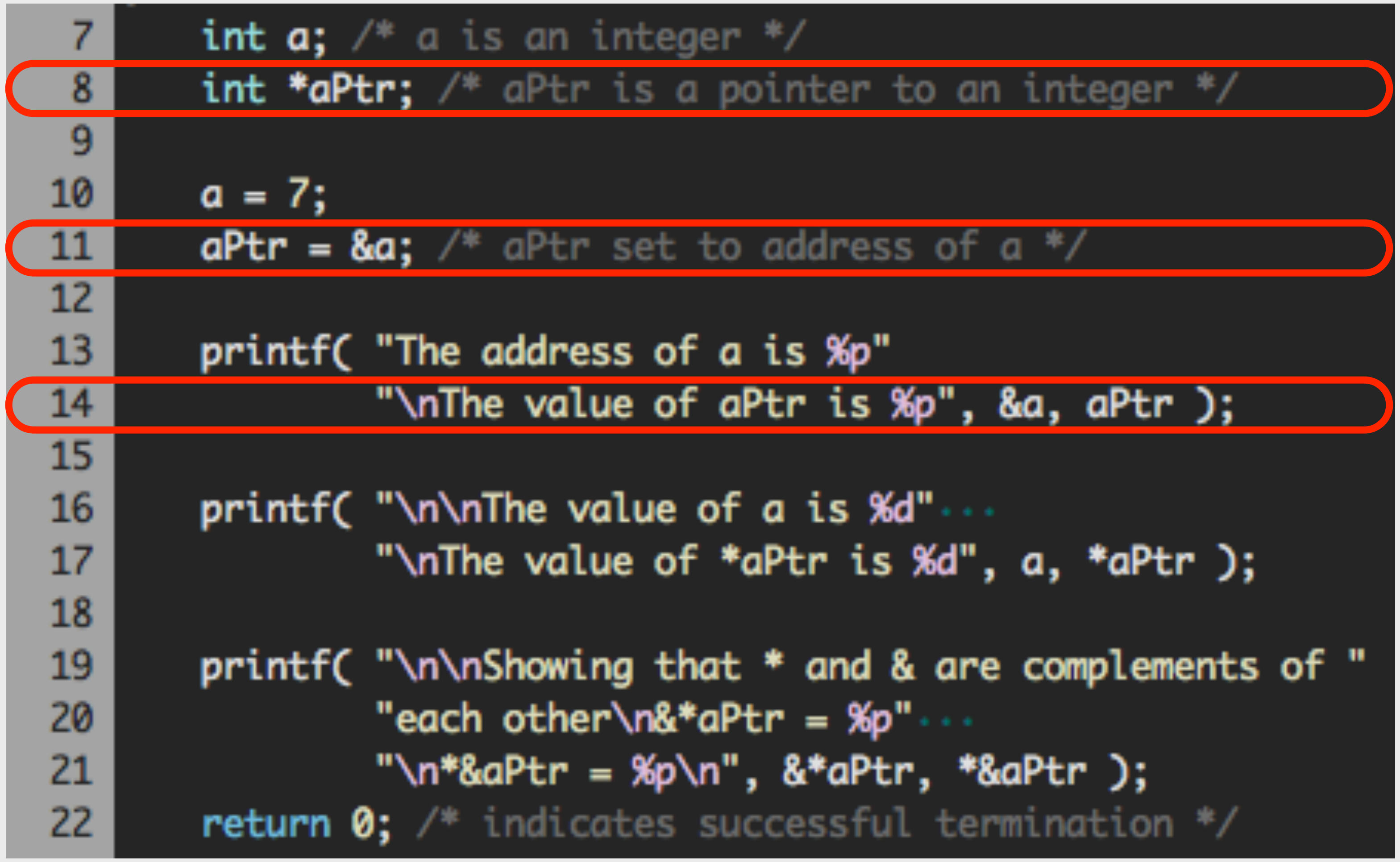
```
int a; /* a is an integer */
int *aPtr; /* aPtr is a pointer to an integer */

a = 7;
aPtr = &a; /* aPtr set to address of a */

printf( "The address of a is %p"
        "\nThe value of aPtr is %p", &a, aPtr );

printf( "\n\nThe value of a is %d"
        "\nThe value of *aPtr is %d", a, *aPtr );

printf( "\n\nShowing that * and & are complements of "
        "each other\n&*aPtr = %p"
        "\n*&aPtr = %p\n", &*aPtr, *&aPtr );
return 0; /* indicates successful termination */
```

declare a pointer to an integer

set the address of a to aPtr

Pointer Operators (Cont.)

- Example: fig07_04.c



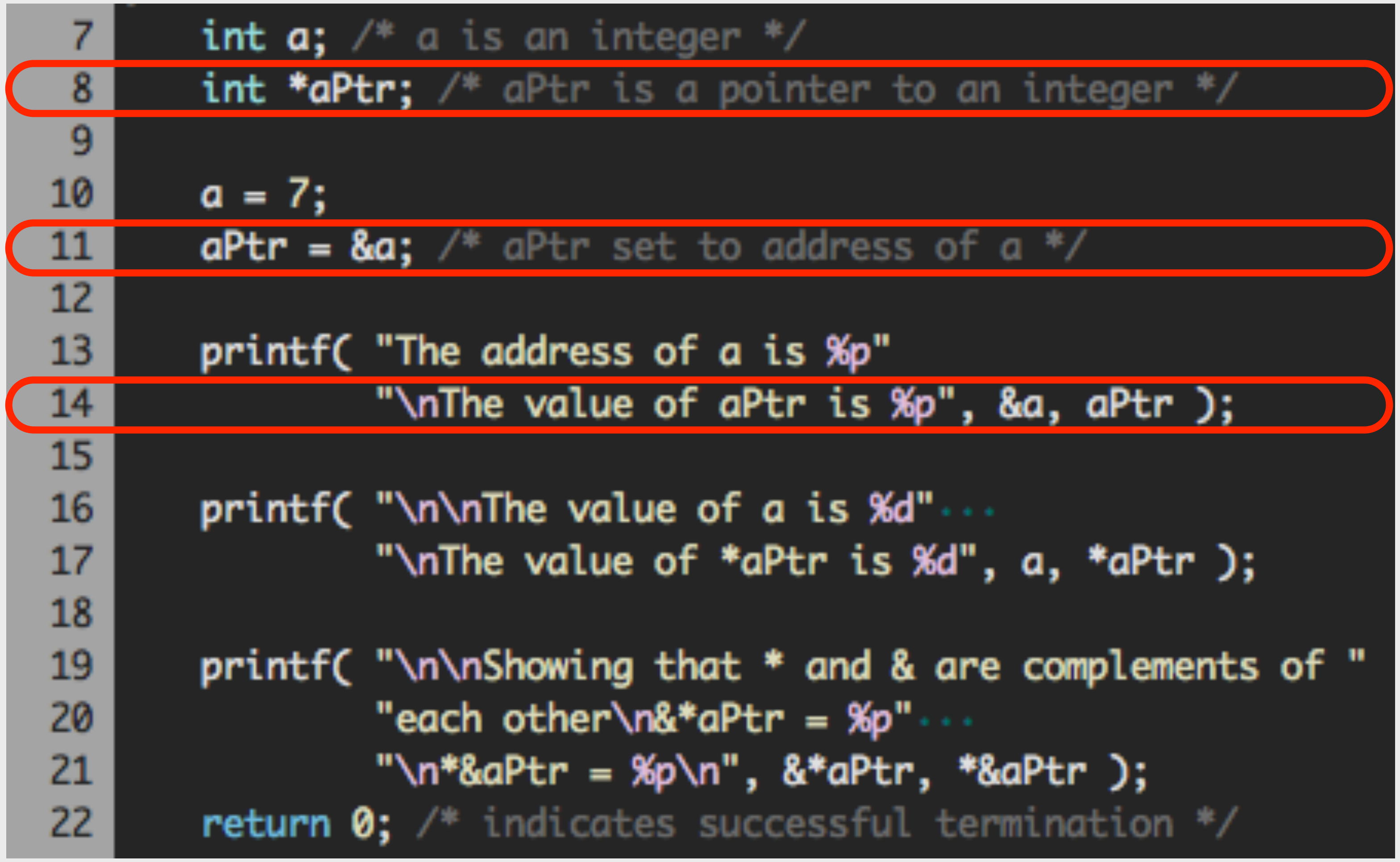
```c
int a; /* a is an integer */
int *aPtr; /* aPtr is a pointer to an integer */

a = 7;
aPtr = &a; /* aPtr set to address of a */

printf( "The address of a is %p"
        "\nThe value of aPtr is %p", &a, aPtr );

printf( "\n\nThe value of a is %d"
        "\nThe value of *aPtr is %d", a, *aPtr );

printf( "\n\nShowing that * and & are complements of "
        "each other\n&*aPtr = %p"
        "\n*&aPtr = %p\n", &*aPtr, *&aPtr );
return 0; /* indicates successful termination */
```

declare a pointer to an integer

set the address of a to aPtr

Pointer Operators (Cont.)

- Example: fig07_04.c



```
int a; /* a is an integer */
int *aPtr; /* aPtr is a pointer to an integer */

a = 7;
aPtr = &a; /* aPtr set to address of a */

printf( "The address of a is %p"
        "\nThe value of aPtr is %p", &a, aPtr );

printf( "\n\nThe value of a is %d"
        "\nThe value of *aPtr is %d", a, *aPtr );

printf( "\n\nShowing that * and & are complements of "
        "each other\n&*aPtr = %p"
        "\n*&aPtr = %p\n", &*aPtr, *&aPtr );
return 0; /* indicates successful termination */
```

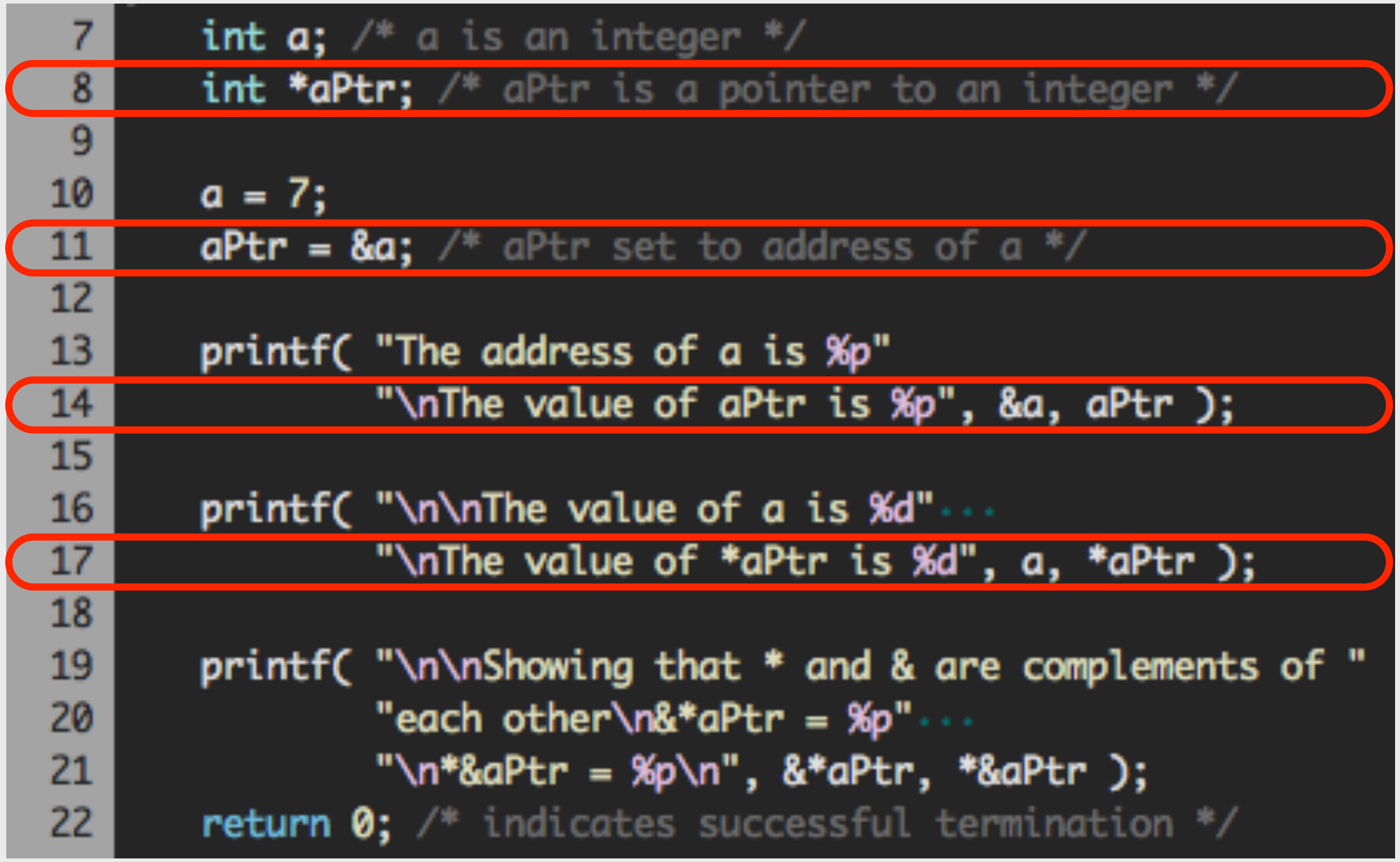
declare a pointer to an integer

set the address of a to aPtr

use **&** to get address

Pointer Operators (Cont.)

- Example: fig07_04.c



```
int a; /* a is an integer */
int *aPtr; /* aPtr is a pointer to an integer */

a = 7;
aPtr = &a; /* aPtr set to address of a */

printf( "The address of a is %p"
        "\nThe value of aPtr is %p", &a, aPtr );

printf( "\n\nThe value of a is %d"
        "\nThe value of *aPtr is %d", a, *aPtr );

printf( "\n\nShowing that * and & are complements of "
        "each other\n&*aPtr = %p"
        "\n*&aPtr = %p\n", &*aPtr, *&aPtr );
return 0; /* indicates successful termination */
```

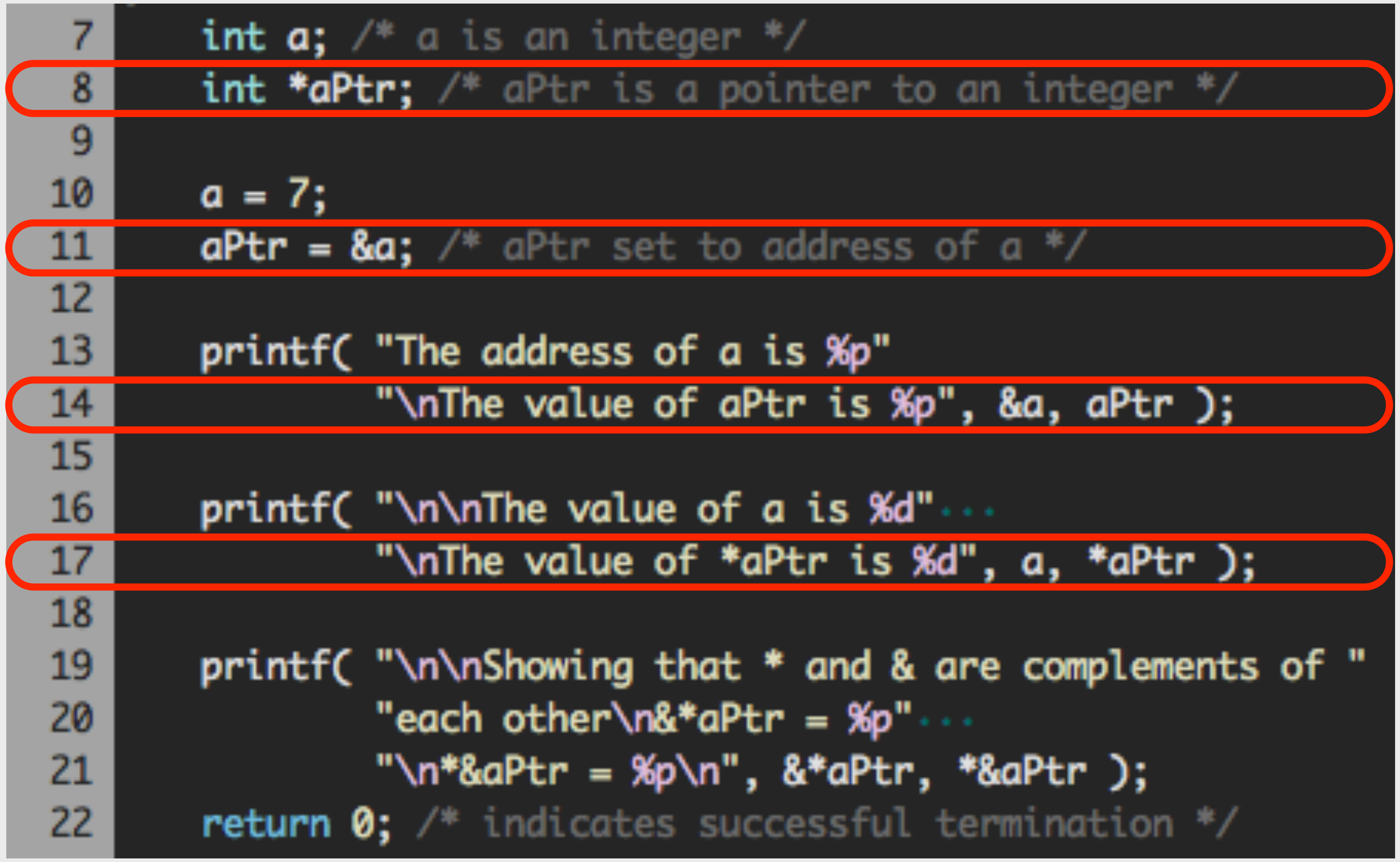
declare a pointer to an integer

set the address of a to aPtr

use **&** to get address

Pointer Operators (Cont.)

- Example: fig07_04.c



```c
int a; /* a is an integer */
int *aPtr; /* aPtr is a pointer to an integer */

a = 7;
aPtr = &a; /* aPtr set to address of a */

printf( "The address of a is %p"
        "\nThe value of aPtr is %p", &a, aPtr );

printf( "\n\nThe value of a is %d"···
        "\nThe value of *aPtr is %d", a, *aPtr );

printf( "\n\nShowing that * and & are complements of "
        "each other\n&*aPtr = %p"···
        "\n*&aPtr = %p\n", &*aPtr, *&aPtr );
return 0; /* indicates successful termination */
```

declare a pointer to an integer

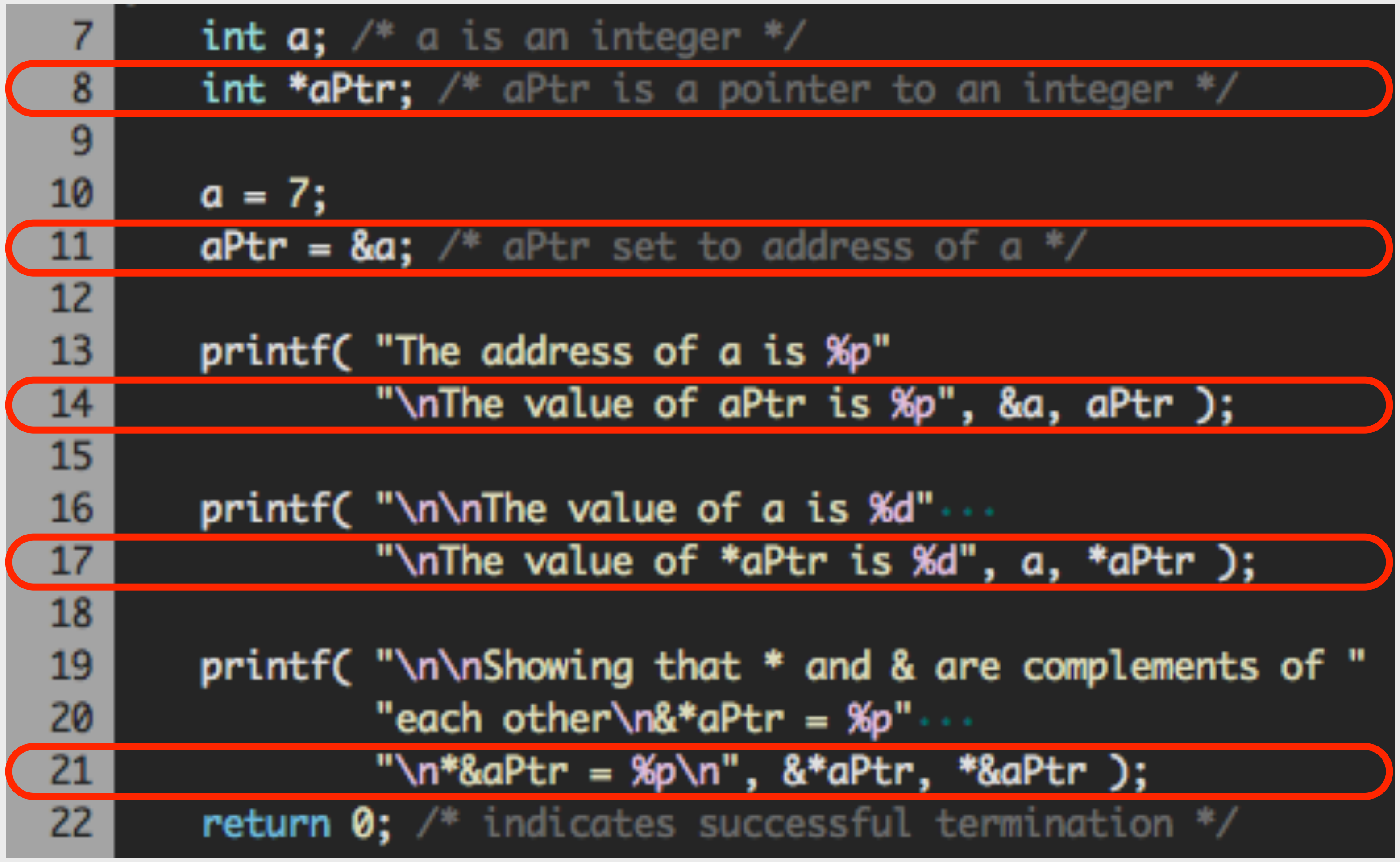
set the address of a to aPtr

use **&** to get address

use * to dereference a pointer

Pointer Operators (Cont.)

- Example: fig07_04.c



```
   int a; /* a is an integer */
   int *aPtr; /* aPtr is a pointer to an integer */

   a = 7;
   aPtr = &a; /* aPtr set to address of a */

   printf( "The address of a is %p"
           "\nThe value of aPtr is %p", &a, aPtr );

   printf( "\n\nThe value of a is %d"
           "\nThe value of *aPtr is %d", a, *aPtr );

   printf( "\n\nShowing that * and & are complements of "
           "each other\n&*aPtr = %p"
           "\n*&aPtr = %p\n", &*aPtr, *&aPtr );
   return 0; /* indicates successful termination */
```

declare a pointer to an integer

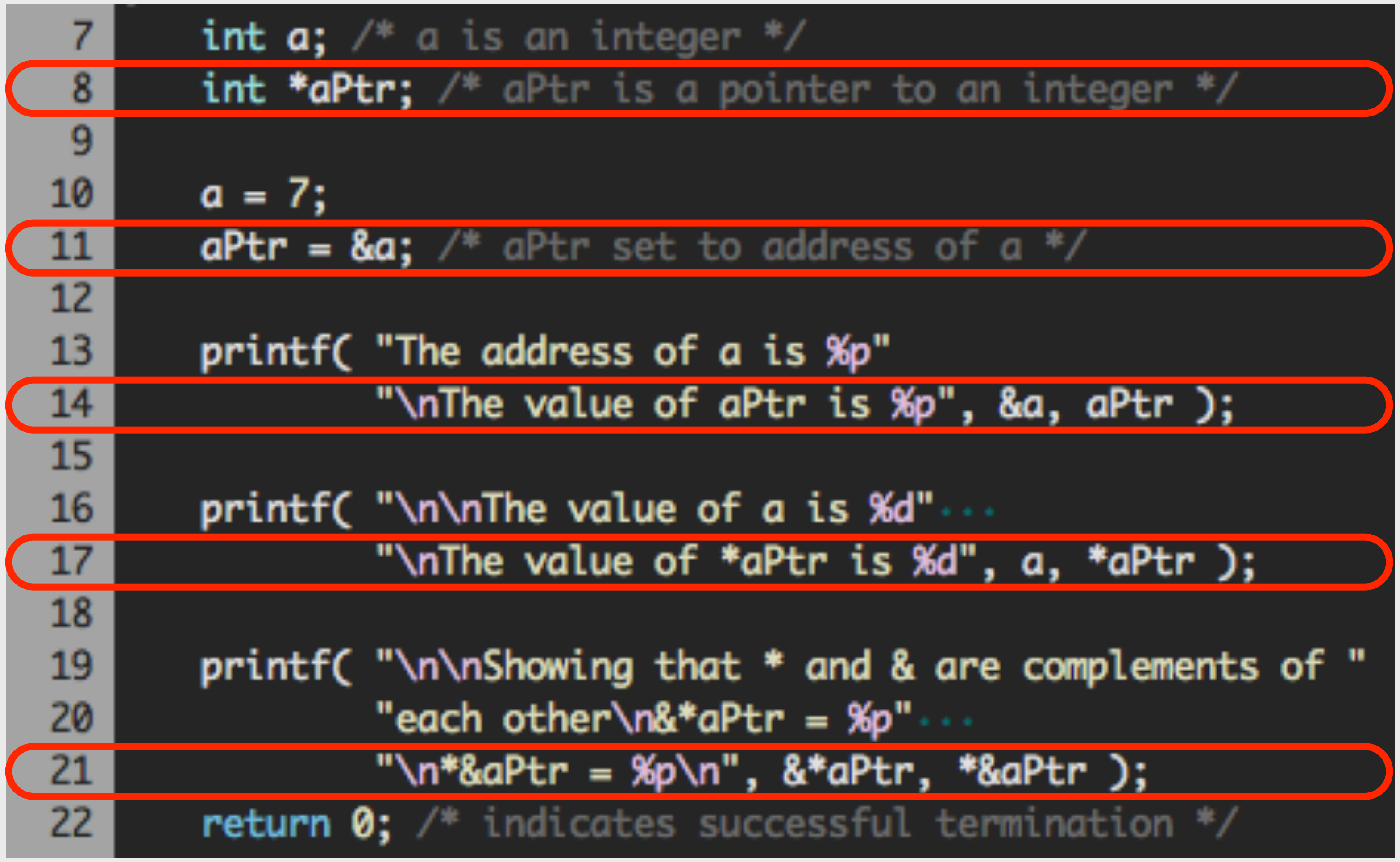
set the address of a to aPtr

use **&** to get address

use * to dereference a pointer

Pointer Operators (Cont.)

- Example: fig07_04.c



```c
    int a; /* a is an integer */
    int *aPtr; /* aPtr is a pointer to an integer */

    a = 7;
    aPtr = &a; /* aPtr set to address of a */

    printf( "The address of a is %p"
            "\nThe value of aPtr is %p", &a, aPtr );

    printf( "\n\nThe value of a is %d"
            "\nThe value of *aPtr is %d", a, *aPtr );

    printf( "\n\nShowing that * and & are complements of "
            "each other\n&*aPtr = %p"
            "\n*&aPtr = %p\n", &*aPtr, *&aPtr );
    return 0; /* indicates successful termination */
```

declare a pointer to an integer

set the address of a to aPtr

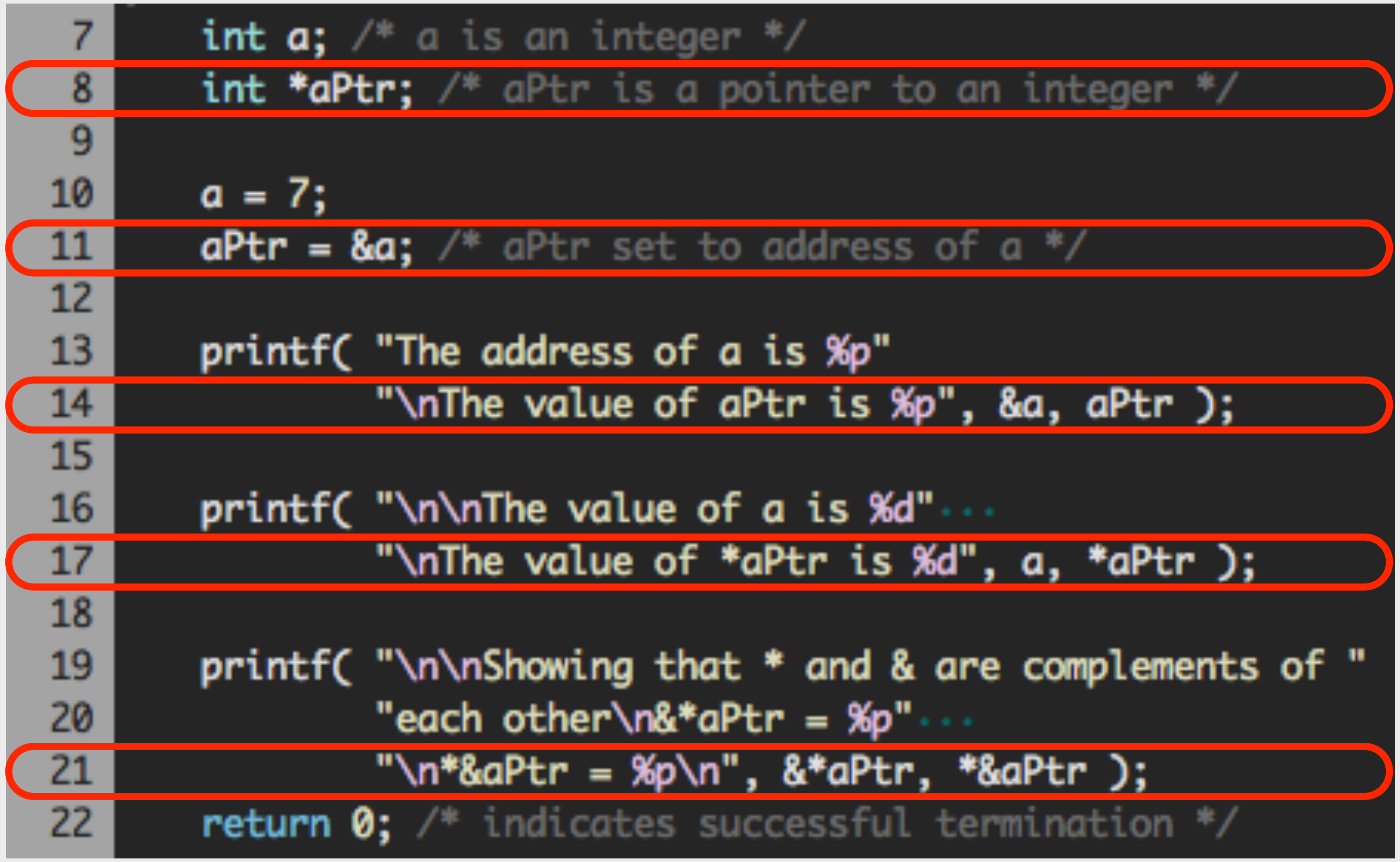
use **&** to get address

use ***** to dereference a pointer

***** and **&** are complements to each other

Pointer Operators (Cont.)

- Example: fig07_04.c



```c
int a; /* a is an integer */
int *aPtr; /* aPtr is a pointer to an integer */

a = 7;
aPtr = &a; /* aPtr set to address of a */

printf( "The address of a is %p"
        "\nThe value of aPtr is %p", &a, aPtr );

printf( "\n\nThe value of a is %d"
        "\nThe value of *aPtr is %d", a, *aPtr );

printf( "\n\nShowing that * and & are complements of "
        "each other\n&*aPtr = %p"
        "\n*&aPtr = %p\n", &*aPtr, *&aPtr );
return 0; /* indicates successful termination */
```

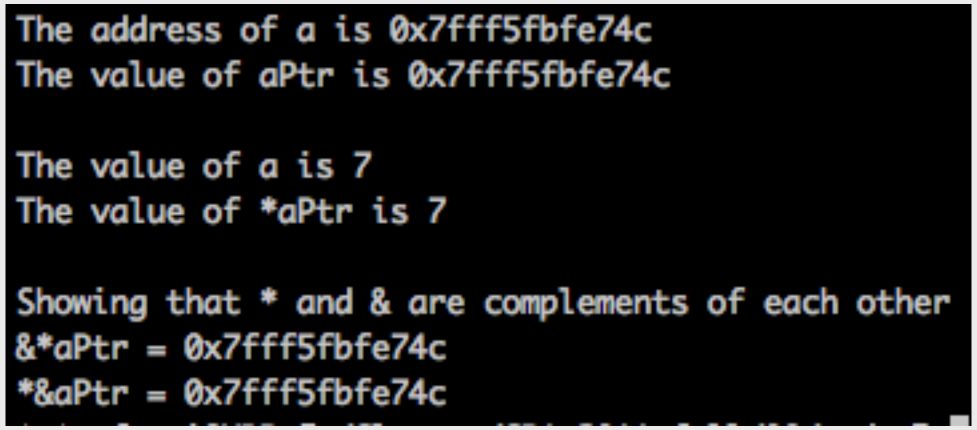
declare a pointer to an integer

set the address of a to aPtr

use **&** to get address

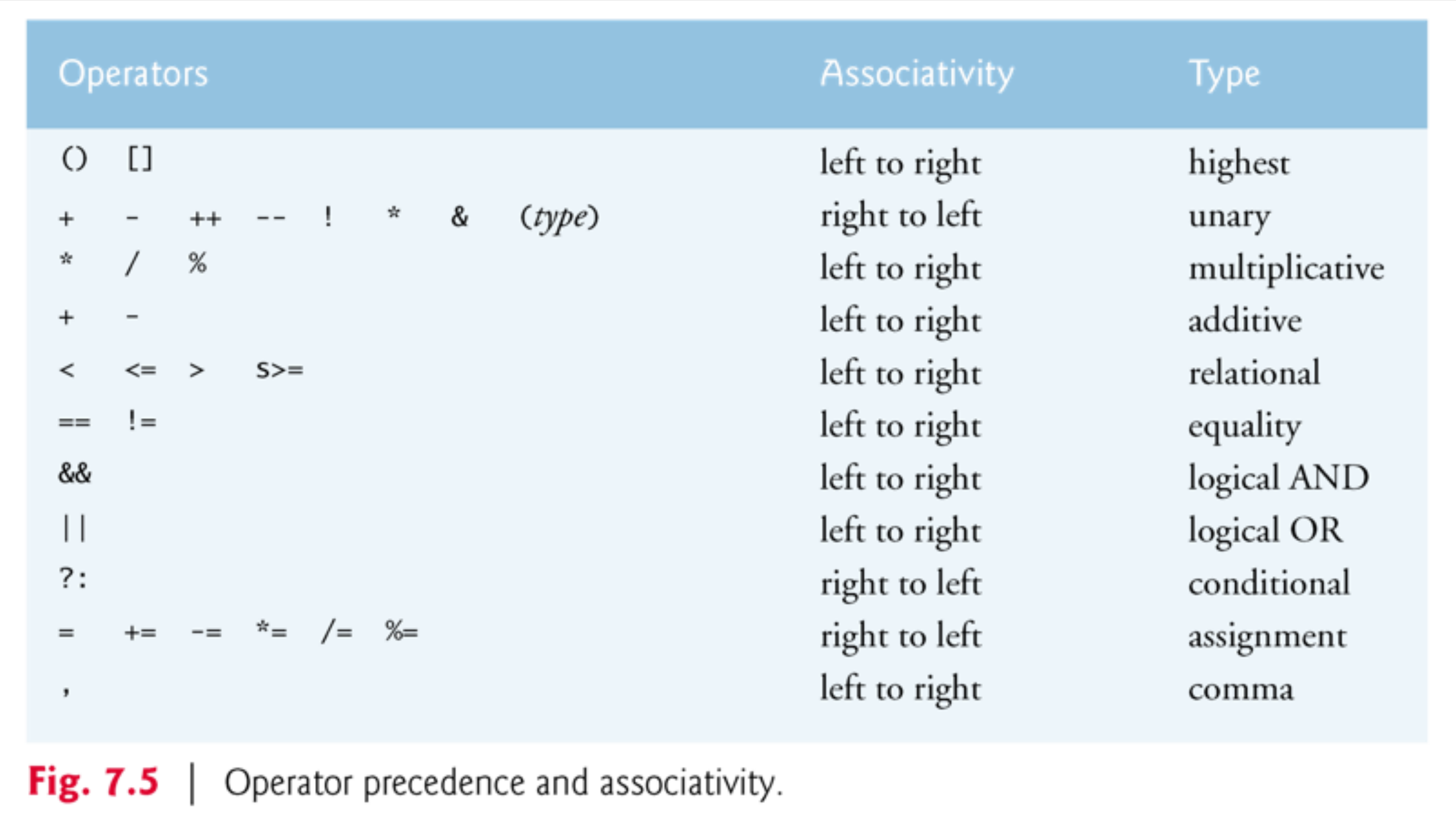
use ***** to dereference a pointer

***** and **&** are complements to each other



```
The address of a is 0x7fff5fbfe74c
The value of aPtr is 0x7fff5fbfe74c

The value of a is 7
The value of *aPtr is 7

Showing that * and & are complements of each other
&*aPtr = 0x7fff5fbfe74c
*&aPtr = 0x7fff5fbfe74c
```

Pointer Operators (Cont.)



Operators	Associativity	Type
() []	left to right	highest
+ - ++ -- ! * & (*type*)	right to left	unary
* / %	left to right	multiplicative
+ -	left to right	additive
< <= > s>=	left to right	relational
== !=	left to right	equality
&&	left to right	logical AND
||	left to right	logical OR
?:	right to left	conditional
= += -= *= /= %=	right to left	assignment
,	left to right	comma

Fig. 7.5 | Operator precedence and associativity.

Passing Arguments to Functions by Reference

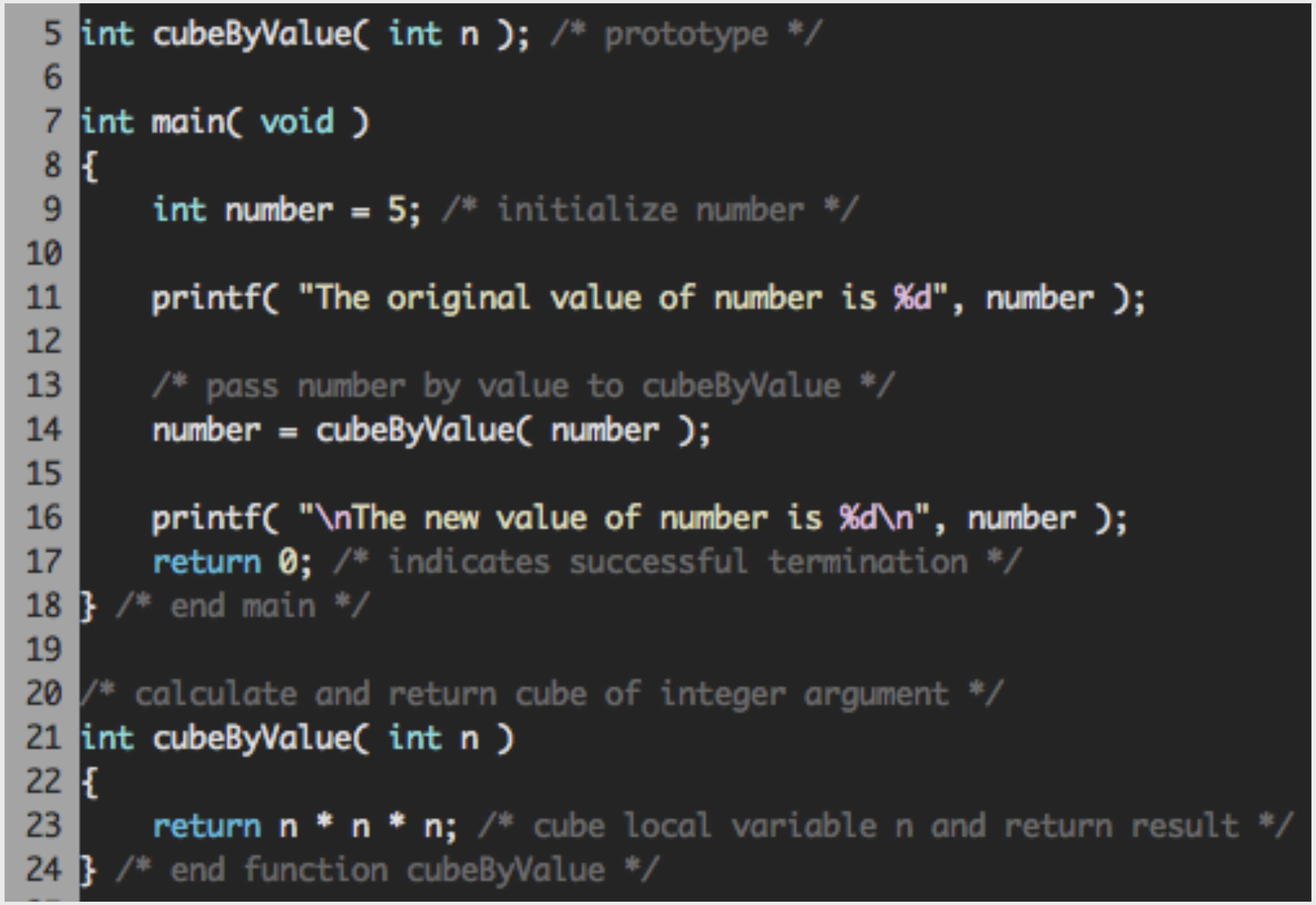
- There are two ways to pass arguments to a function—call-by-value and call-by-reference.
- All arguments in C are passed by value.
- C provides the capabilities for simulating call-by-reference.
- In C, you use pointers and the indirection operator to simulate call-by-reference.

Passing Arguments to Functions by Reference (Cont.)

- This is normally accomplished by applying the address operator (&) to the variable (in the caller) whose value will be modified.
- When the address of a variable is passed to a function, the indirection operator (*) may be used in the function to modify the value at that location in the caller's memory.

Passing Arguments to Functions by Reference (Cont.)

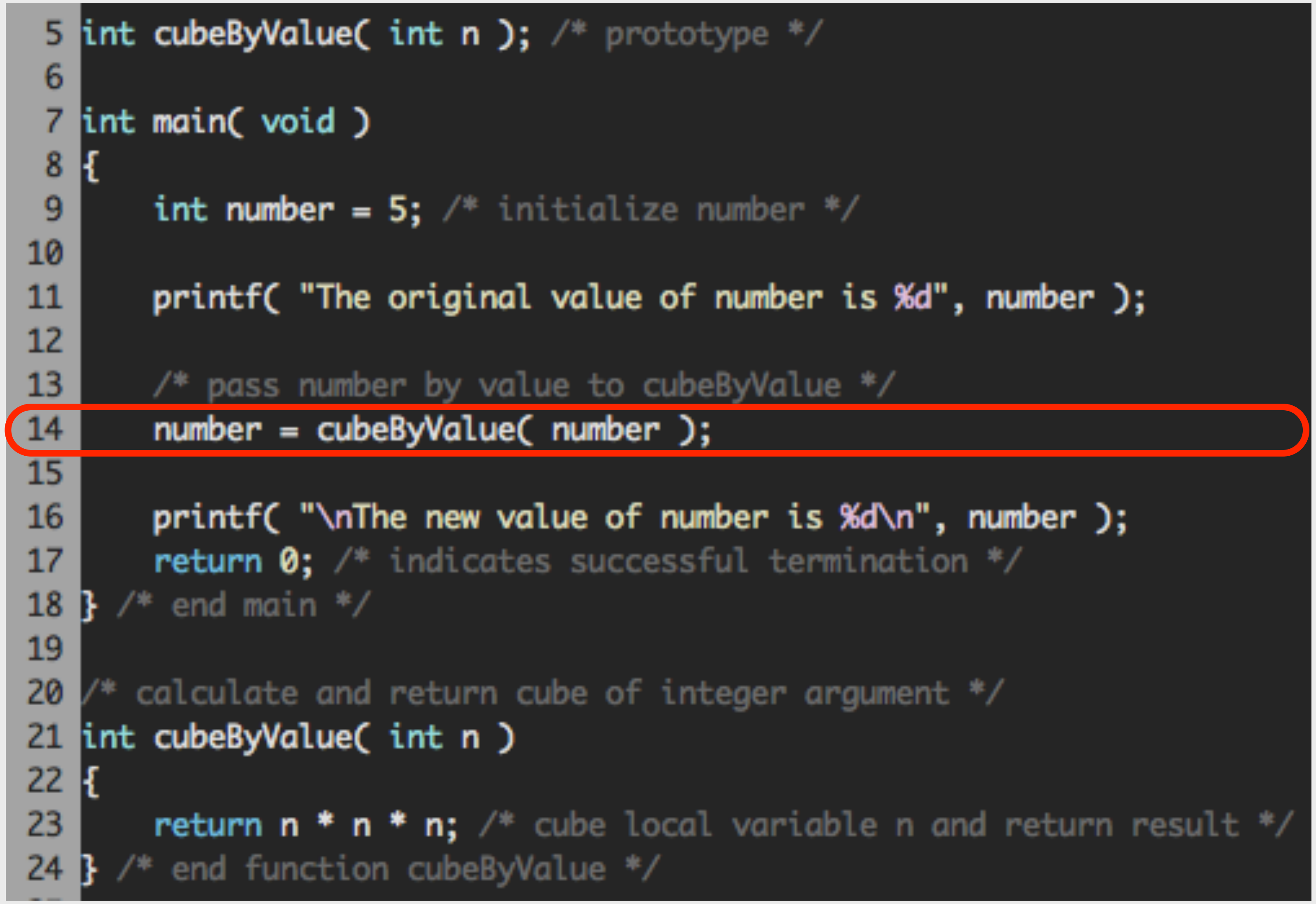
- Example: fig07_06.c



```c
int cubeByValue( int n ); /* prototype */

int main( void )
{
    int number = 5; /* initialize number */

    printf( "The original value of number is %d", number );

    /* pass number by value to cubeByValue */
    number = cubeByValue( number );

    printf( "\nThe new value of number is %d\n", number );
    return 0; /* indicates successful termination */
} /* end main */

/* calculate and return cube of integer argument */
int cubeByValue( int n )
{
    return n * n * n; /* cube local variable n and return result */
} /* end function cubeByValue */
```

Passing Arguments to Functions by Reference (Cont.)

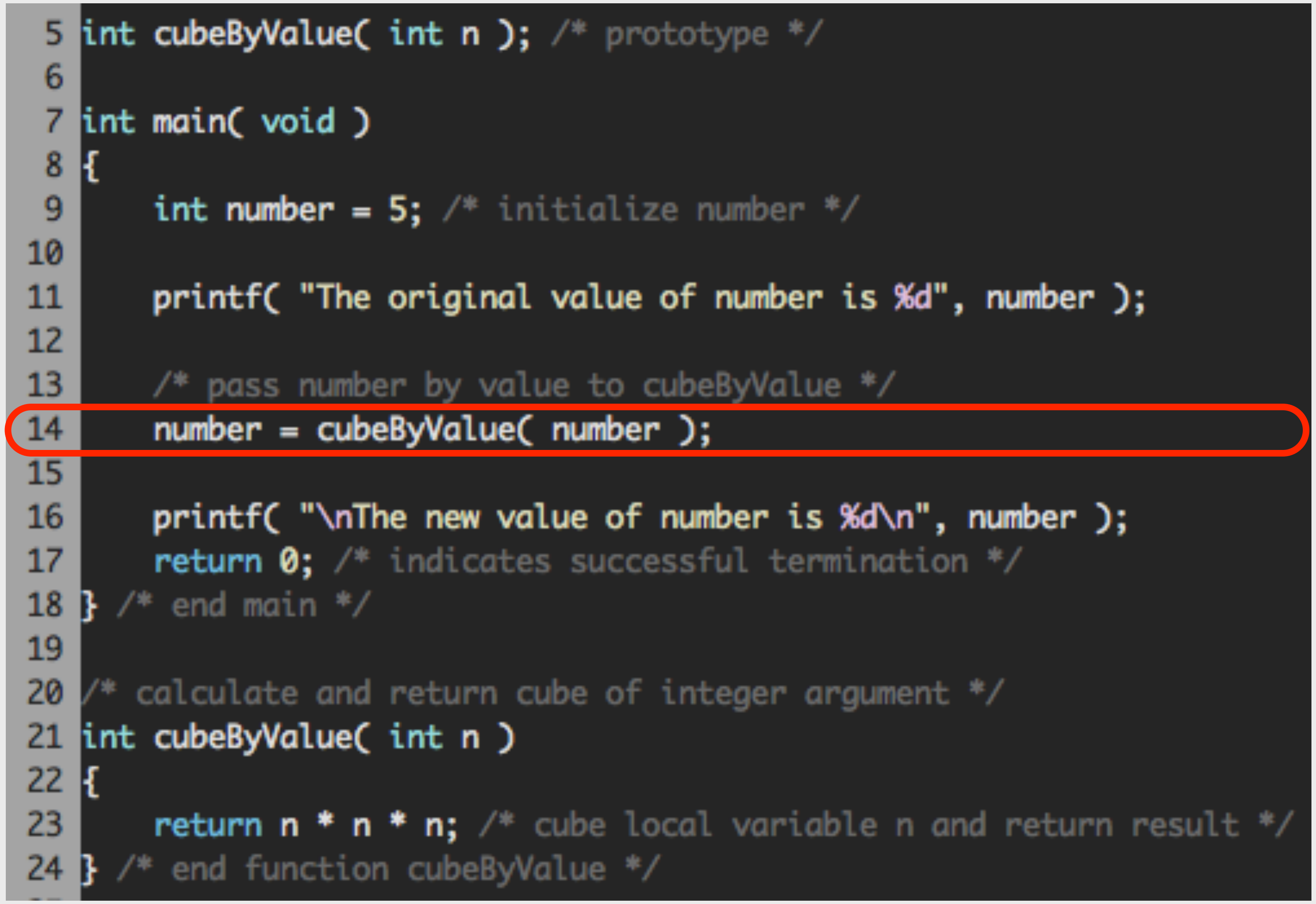
- Example: fig07_06.c



```c
int cubeByValue( int n ); /* prototype */

int main( void )
{
    int number = 5; /* initialize number */

    printf( "The original value of number is %d", number );

    /* pass number by value to cubeByValue */
    number = cubeByValue( number );

    printf( "\nThe new value of number is %d\n", number );
    return 0; /* indicates successful termination */
} /* end main */

/* calculate and return cube of integer argument */
int cubeByValue( int n )
{
    return n * n * n; /* cube local variable n and return result */
} /* end function cubeByValue */
```

Passing Arguments to Functions by Reference (Cont.)

- Example: fig07_06.c

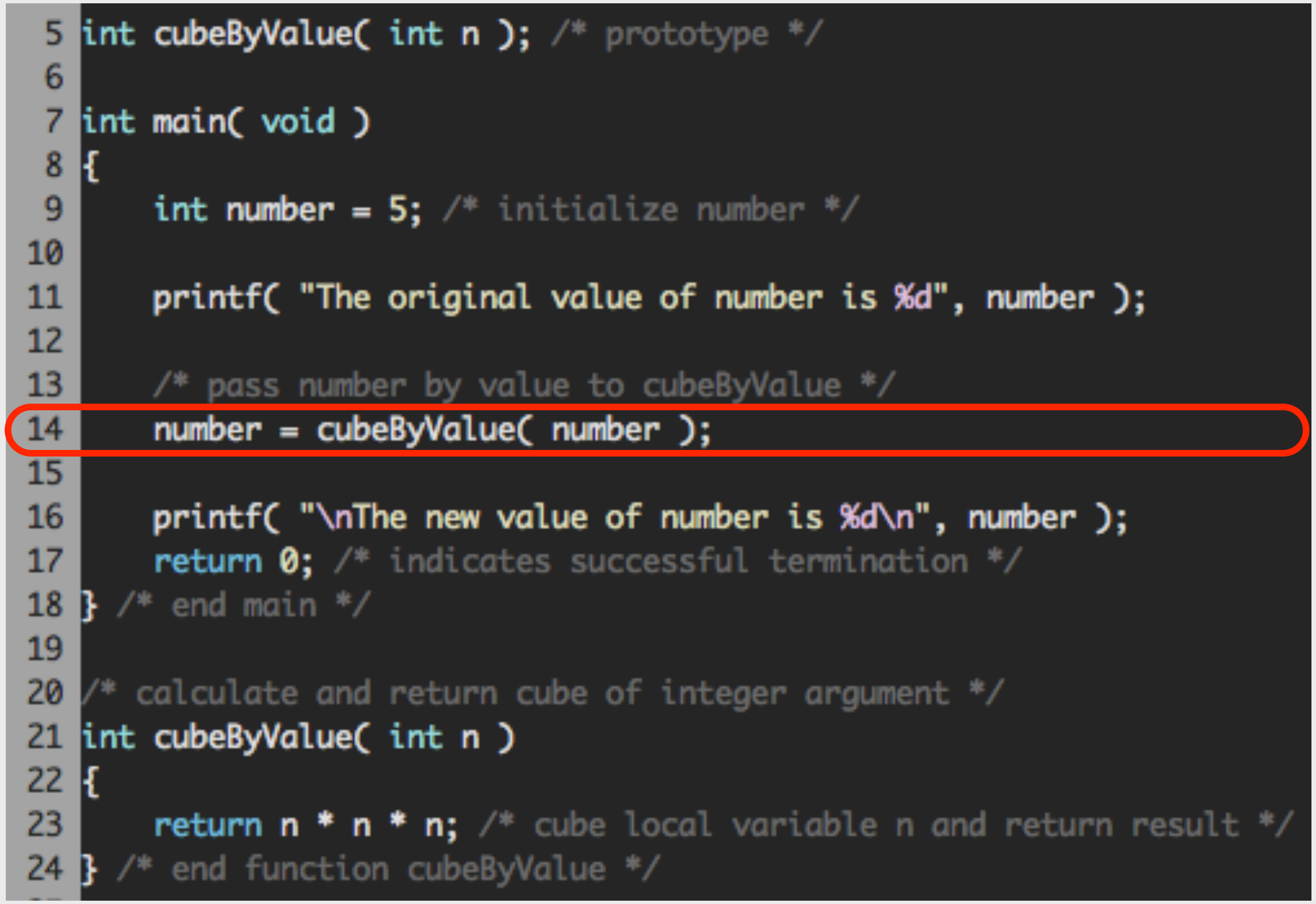


```c
int cubeByValue( int n ); /* prototype */

int main( void )
{
    int number = 5; /* initialize number */

    printf( "The original value of number is %d", number );

    /* pass number by value to cubeByValue */
    number = cubeByValue( number );

    printf( "\nThe new value of number is %d\n", number );
    return 0; /* indicates successful termination */
} /* end main */

/* calculate and return cube of integer argument */
int cubeByValue( int n )
{
    return n * n * n; /* cube local variable n and return result */
} /* end function cubeByValue */
```

call by value

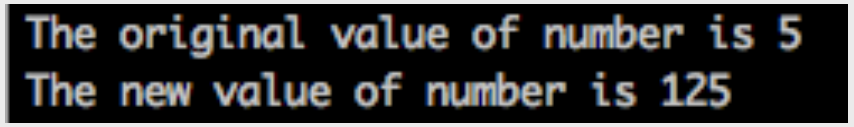
Passing Arguments to Functions by Reference (Cont.)

- Example: fig07_06.c



```
int cubeByValue( int n ); /* prototype */

int main( void )
{
    int number = 5; /* initialize number */

    printf( "The original value of number is %d", number );

    /* pass number by value to cubeByValue */
    number = cubeByValue( number );

    printf( "\nThe new value of number is %d\n", number );
    return 0; /* indicates successful termination */
} /* end main */

/* calculate and return cube of integer argument */
int cubeByValue( int n )
{
    return n * n * n; /* cube local variable n and return result */
} /* end function cubeByValue */
```

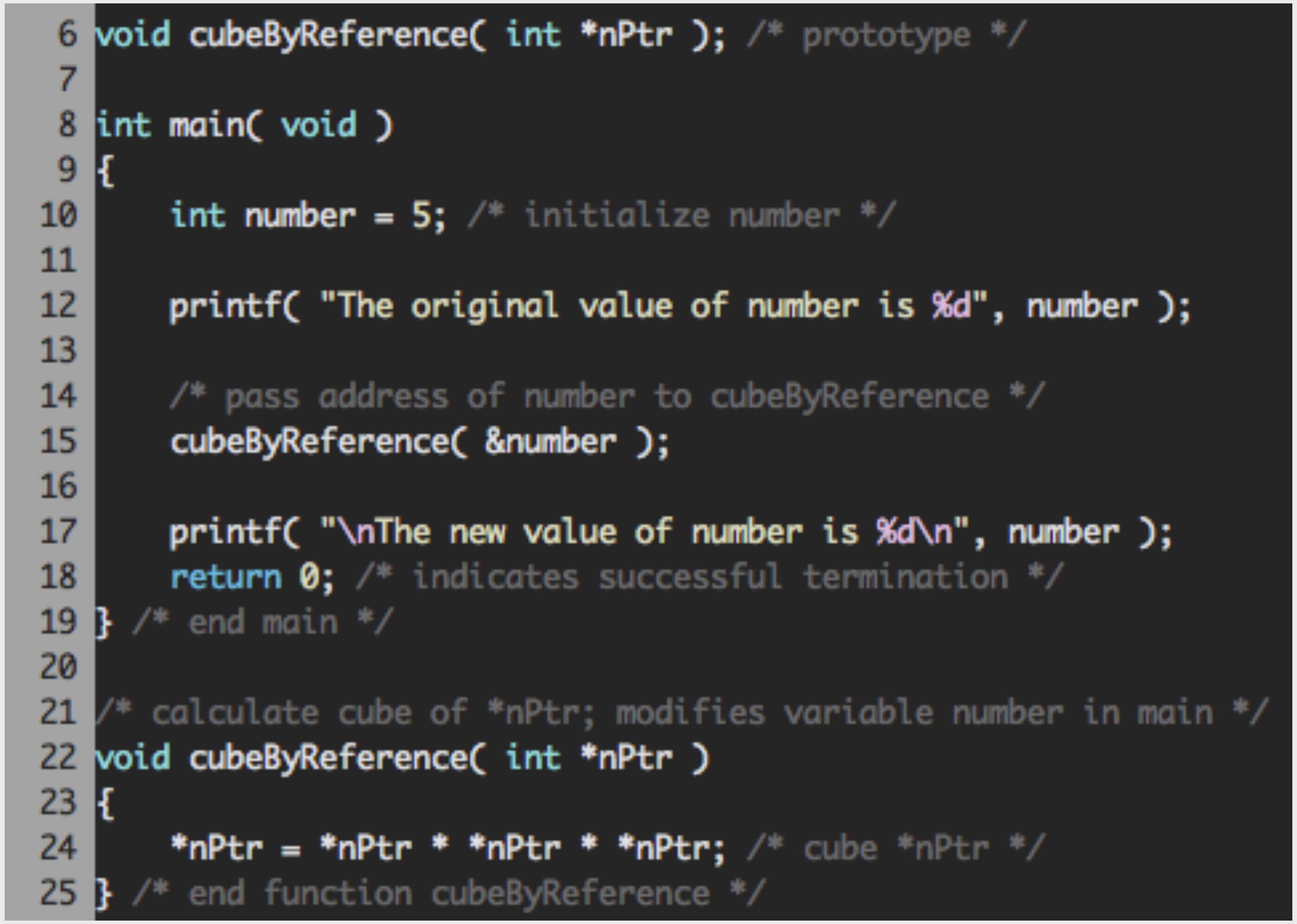
call by value



```
The original value of number is 5
The new value of number is 125
```

Passing Arguments to Functions by Reference (Cont.)

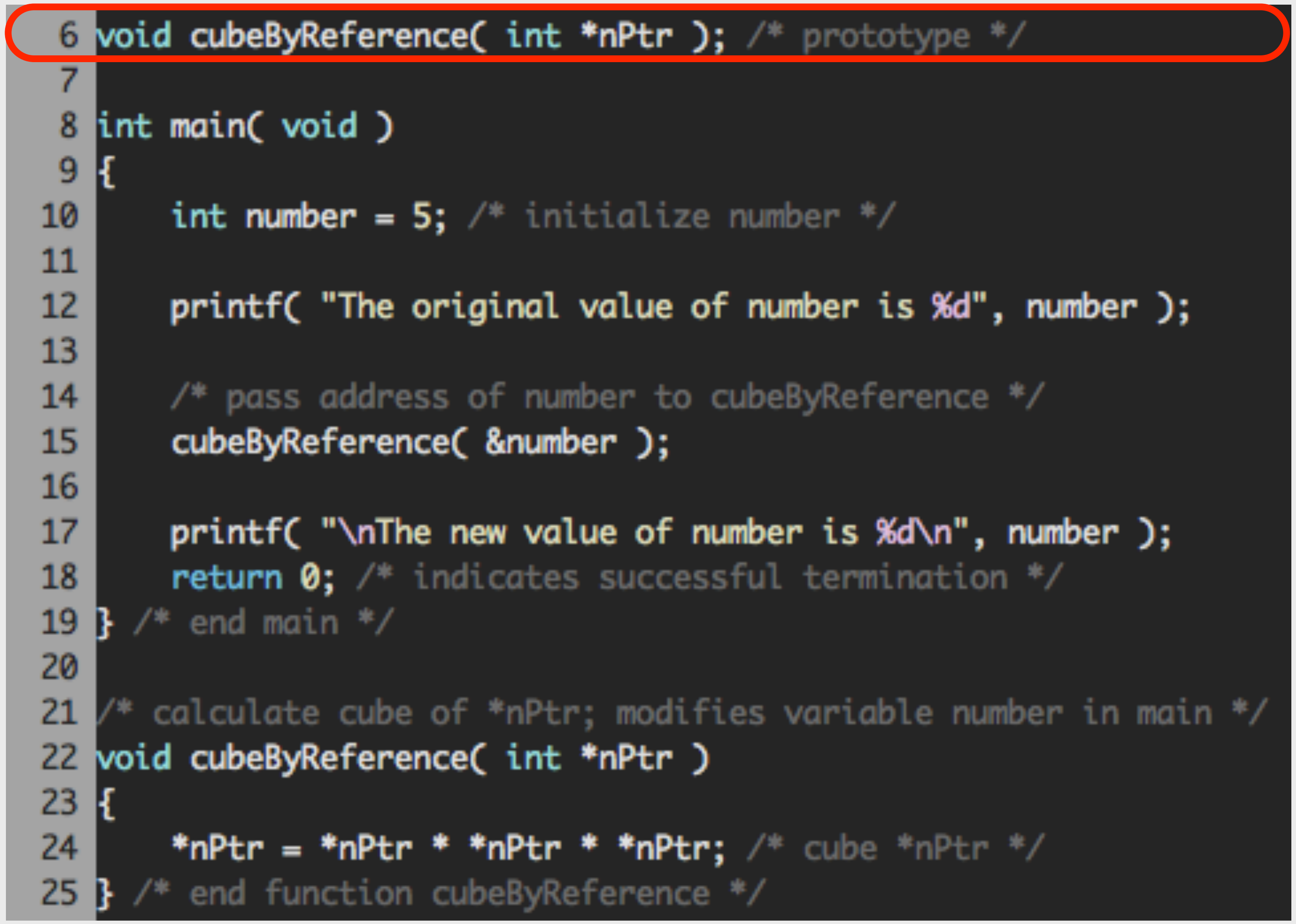
- Example: fig07_07.c



```c
void cubeByReference( int *nPtr ); /* prototype */

int main( void )
{
    int number = 5; /* initialize number */

    printf( "The original value of number is %d", number );

    /* pass address of number to cubeByReference */
    cubeByReference( &number );

    printf( "\nThe new value of number is %d\n", number );
    return 0; /* indicates successful termination */
} /* end main */

/* calculate cube of *nPtr; modifies variable number in main */
void cubeByReference( int *nPtr )
{
    *nPtr = *nPtr * *nPtr * *nPtr; /* cube *nPtr */
} /* end function cubeByReference */
```

Passing Arguments to Functions by Reference (Cont.)

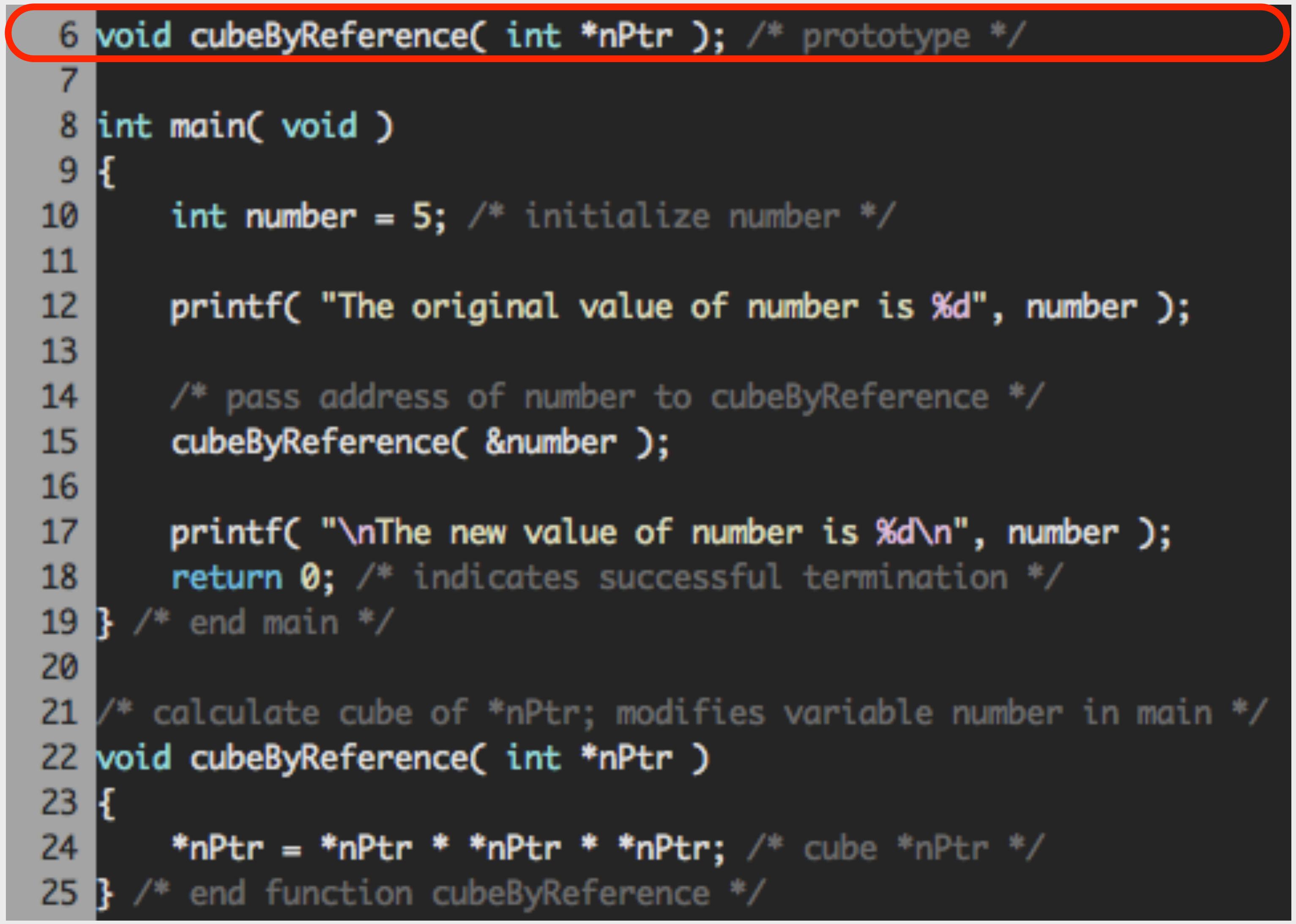
- Example: fig07_07.c



```c
void cubeByReference( int *nPtr ); /* prototype */

int main( void )
{
    int number = 5; /* initialize number */

    printf( "The original value of number is %d", number );

    /* pass address of number to cubeByReference */
    cubeByReference( &number );

    printf( "\nThe new value of number is %d\n", number );
    return 0; /* indicates successful termination */
} /* end main */

/* calculate cube of *nPtr; modifies variable number in main */
void cubeByReference( int *nPtr )
{
    *nPtr = *nPtr * *nPtr * *nPtr; /* cube *nPtr */
} /* end function cubeByReference */
```

Passing Arguments to Functions by Reference (Cont.)

- Example: fig07_07.c

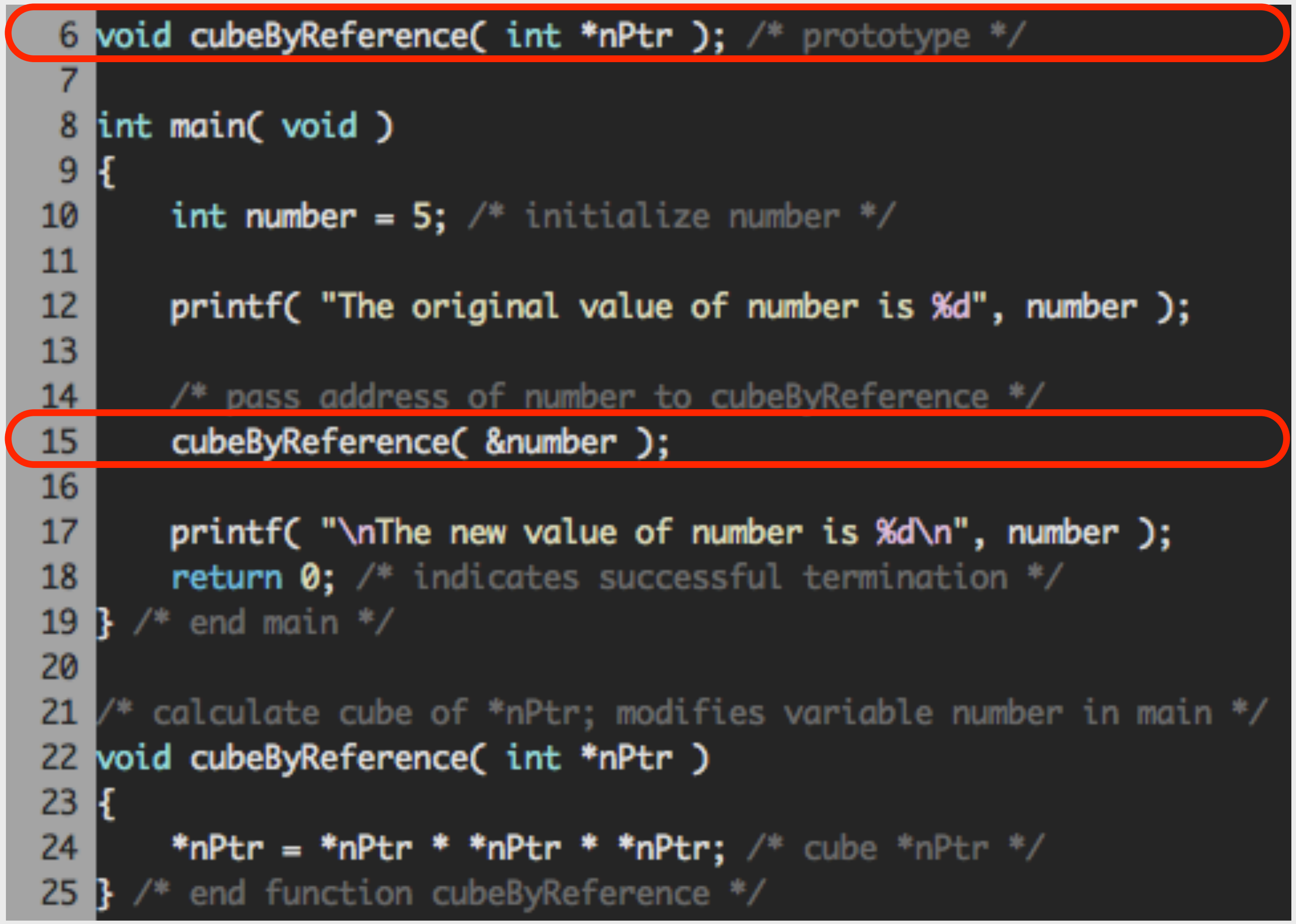


```c
void cubeByReference( int *nPtr ); /* prototype */

int main( void )
{
    int number = 5; /* initialize number */

    printf( "The original value of number is %d", number );

    /* pass address of number to cubeByReference */
    cubeByReference( &number );

    printf( "\nThe new value of number is %d\n", number );
    return 0; /* indicates successful termination */
} /* end main */

/* calculate cube of *nPtr; modifies variable number in main */
void cubeByReference( int *nPtr )
{
    *nPtr = *nPtr * *nPtr * *nPtr; /* cube *nPtr */
} /* end function cubeByReference */
```

declare a function that can receive pointers as arguments

Passing Arguments to Functions by Reference (Cont.)

- Example: fig07_07.c

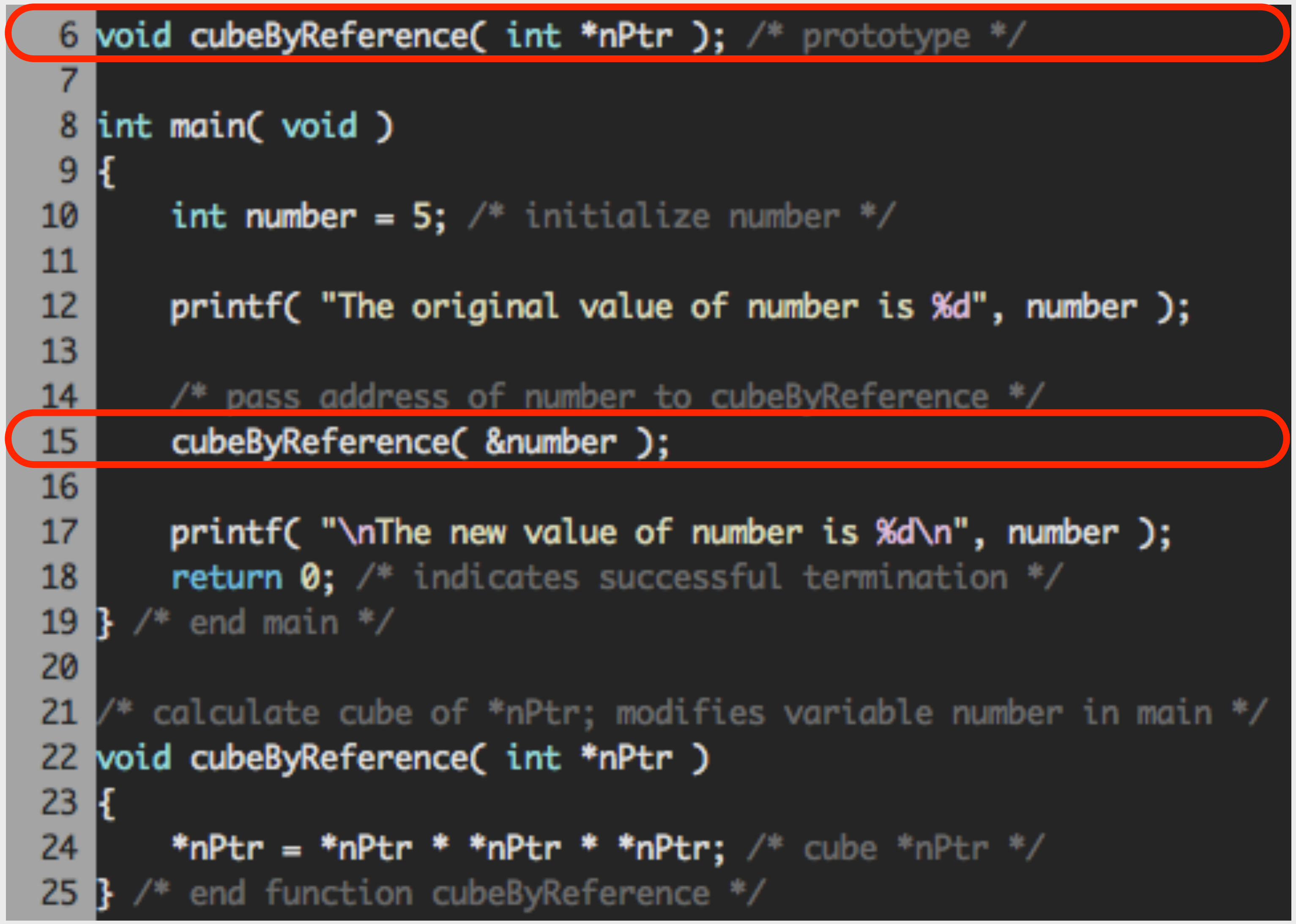


```c
void cubeByReference( int *nPtr ); /* prototype */

int main( void )
{
    int number = 5; /* initialize number */

    printf( "The original value of number is %d", number );

    /* pass address of number to cubeByReference */
    cubeByReference( &number );

    printf( "\nThe new value of number is %d\n", number );
    return 0; /* indicates successful termination */
} /* end main */

/* calculate cube of *nPtr; modifies variable number in main */
void cubeByReference( int *nPtr )
{
    *nPtr = *nPtr * *nPtr * *nPtr; /* cube *nPtr */
} /* end function cubeByReference */
```

declare a function that can receive pointers as arguments

Passing Arguments to Functions by Reference (Cont.)

- Example: fig07_07.c



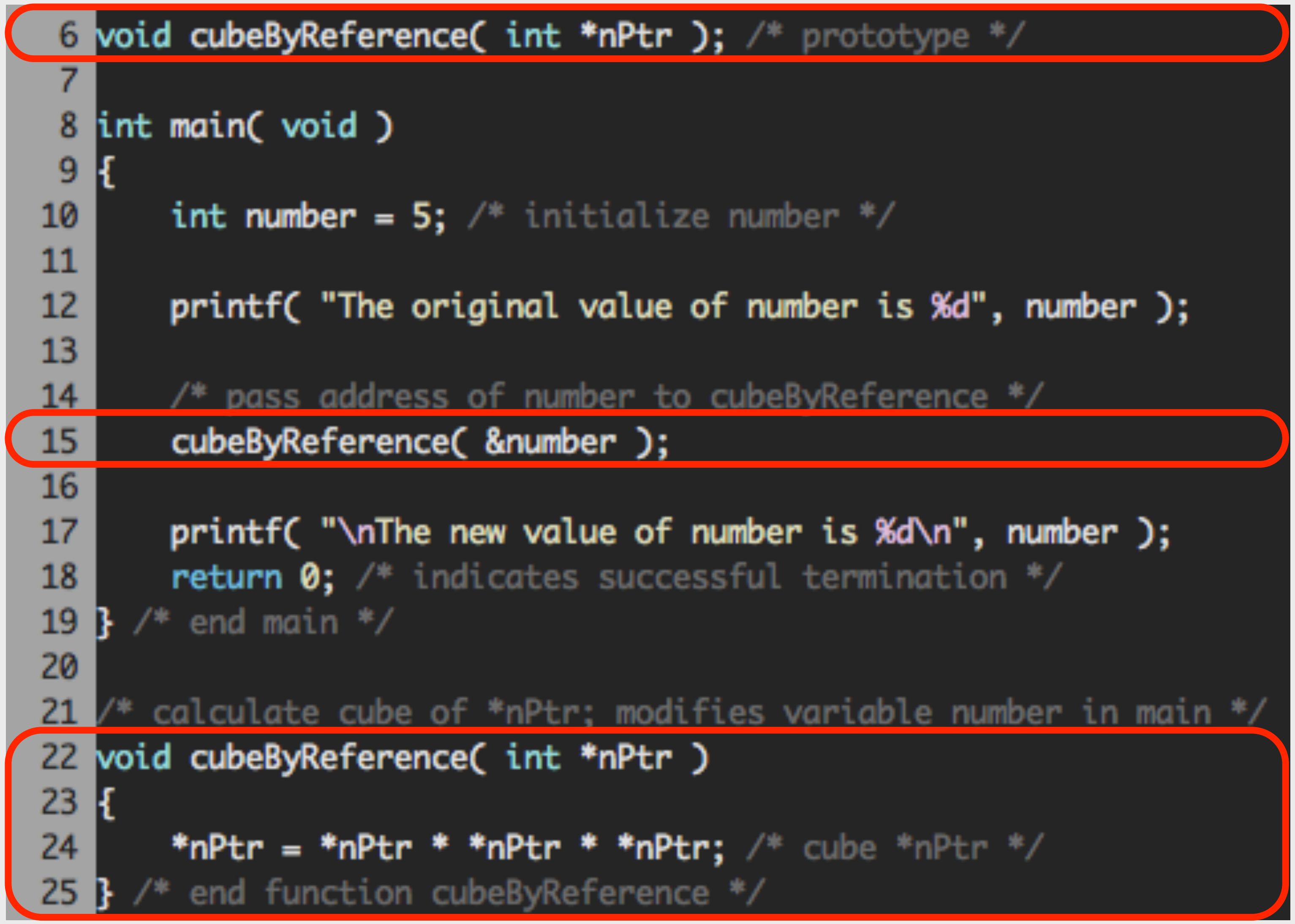
```c
void cubeByReference( int *nPtr ); /* prototype */

int main( void )
{
    int number = 5; /* initialize number */

    printf( "The original value of number is %d", number );

    /* pass address of number to cubeByReference */
    cubeByReference( &number );

    printf( "\nThe new value of number is %d\n", number );
    return 0; /* indicates successful termination */
} /* end main */

/* calculate cube of *nPtr; modifies variable number in main */
void cubeByReference( int *nPtr )
{
    *nPtr = *nPtr * *nPtr * *nPtr; /* cube *nPtr */
} /* end function cubeByReference */
```

declare a function that can receive pointers as arguments

use **&** to pass the address of number

Passing Arguments to Functions by Reference (Cont.)

- Example: fig07_07.c



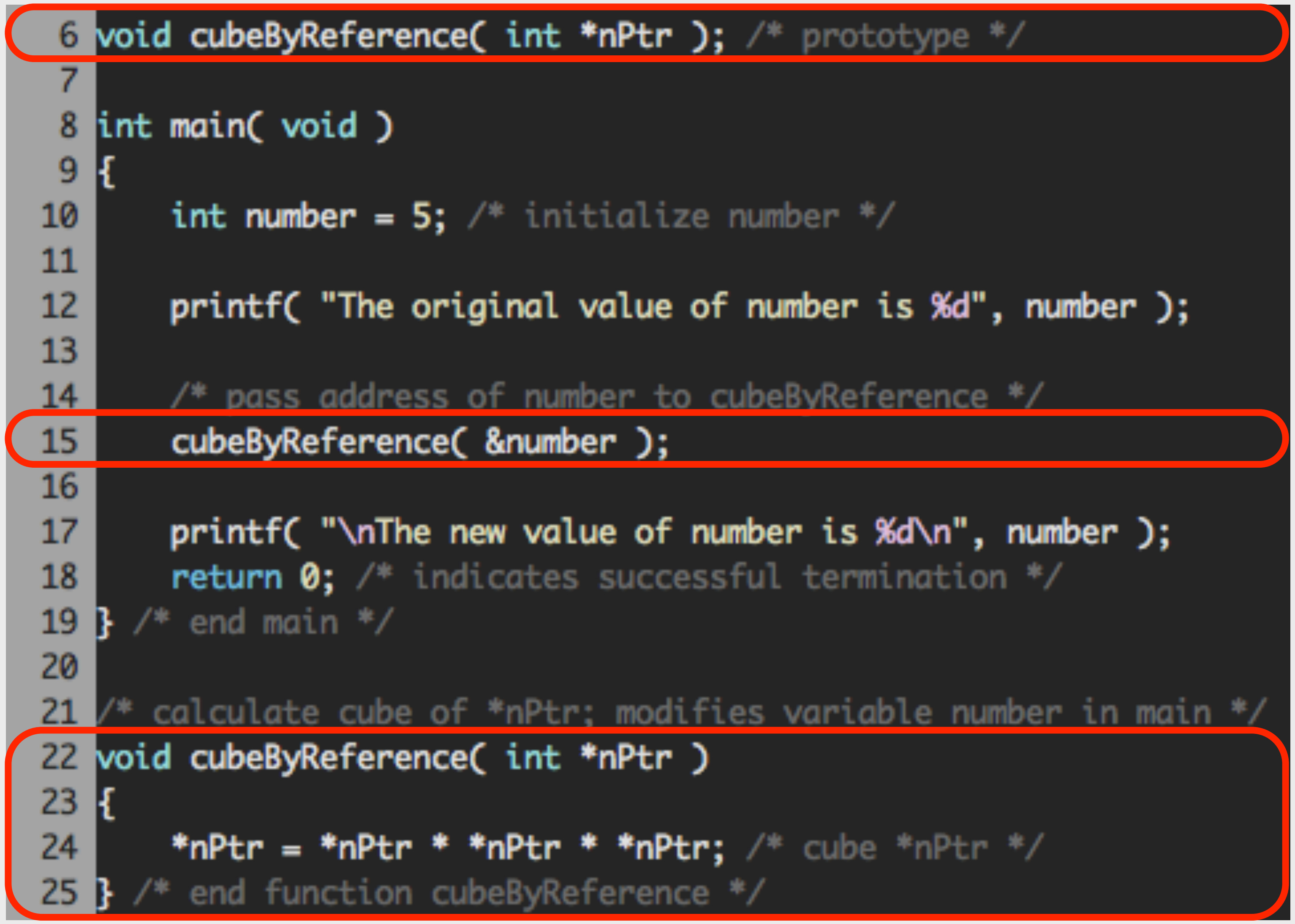
```
void cubeByReference( int *nPtr ); /* prototype */

int main( void )
{
    int number = 5; /* initialize number */

    printf( "The original value of number is %d", number );

    /* pass address of number to cubeByReference */
    cubeByReference( &number );

    printf( "\nThe new value of number is %d\n", number );
    return 0; /* indicates successful termination */
} /* end main */

/* calculate cube of *nPtr; modifies variable number in main */
void cubeByReference( int *nPtr )
{
    *nPtr = *nPtr * *nPtr * *nPtr; /* cube *nPtr */
} /* end function cubeByReference */
```

declare a function that can receive pointers as arguments

use **&** to pass the address of number

Passing Arguments to Functions by Reference (Cont.)

- Example: fig07_07.c



```c
void cubeByReference( int *nPtr ); /* prototype */

int main( void )
{
    int number = 5; /* initialize number */

    printf( "The original value of number is %d", number );

    /* pass address of number to cubeByReference */
    cubeByReference( &number );

    printf( "\nThe new value of number is %d\n", number );
    return 0; /* indicates successful termination */
} /* end main */

/* calculate cube of *nPtr; modifies variable number in main */
void cubeByReference( int *nPtr )
{
    *nPtr = *nPtr * *nPtr * *nPtr; /* cube *nPtr */
} /* end function cubeByReference */
```

declare a function that can receive pointers as arguments

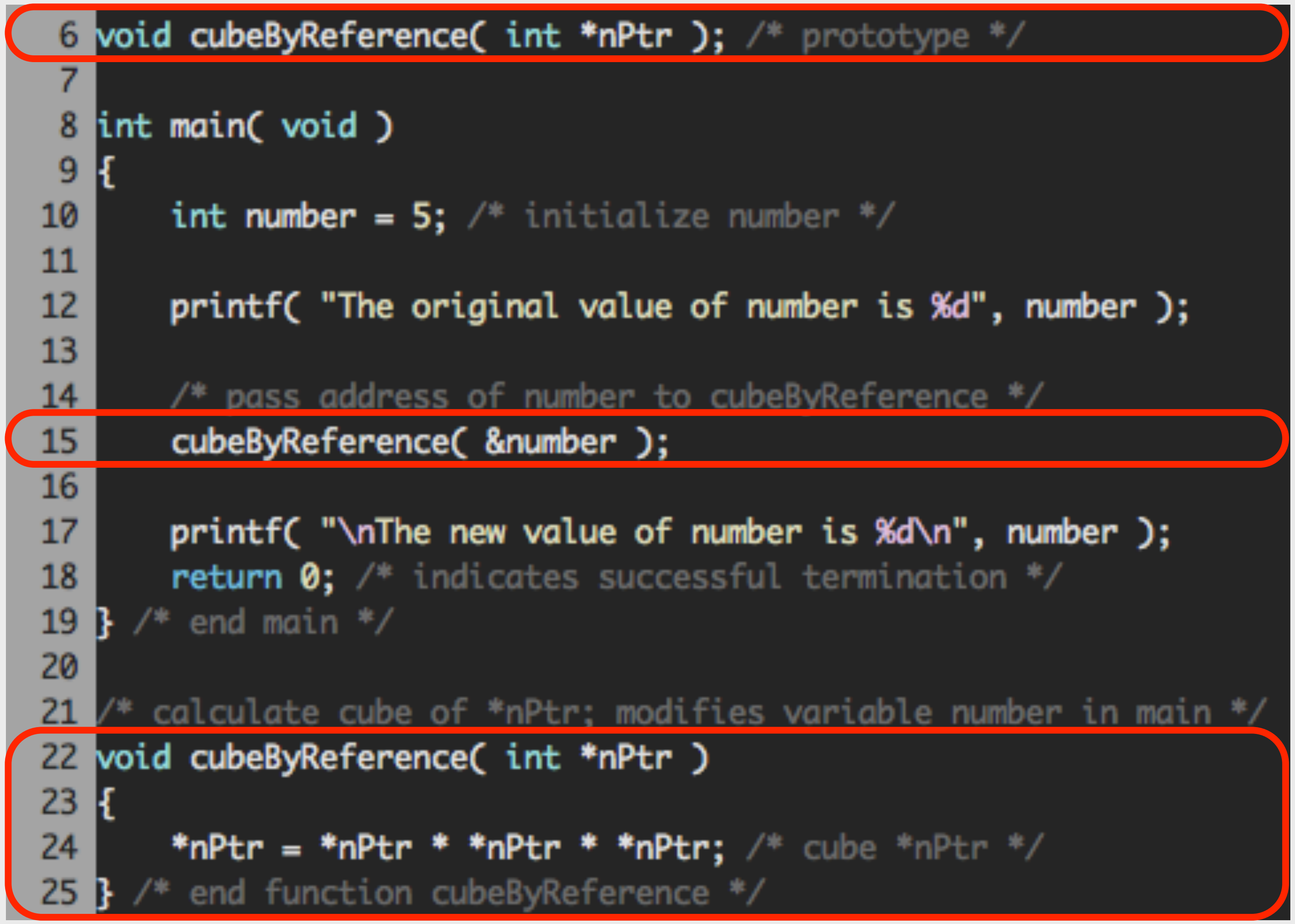
use **&** to pass the address of number

1. declare a pointer, nPtr, to receive the address

2. use * to dereference the pointer

Passing Arguments to Functions by Reference (Cont.)

- Example: fig07_07.c



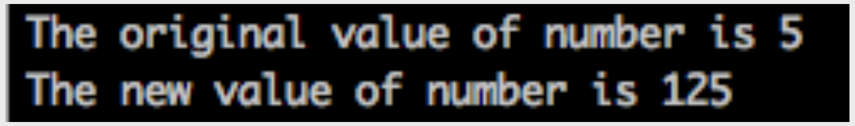
```c
void cubeByReference( int *nPtr ); /* prototype */

int main( void )
{
    int number = 5; /* initialize number */

    printf( "The original value of number is %d", number );

    /* pass address of number to cubeByReference */
    cubeByReference( &number );

    printf( "\nThe new value of number is %d\n", number );
    return 0; /* indicates successful termination */
} /* end main */

/* calculate cube of *nPtr; modifies variable number in main */
void cubeByReference( int *nPtr )
{
    *nPtr = *nPtr * *nPtr * *nPtr; /* cube *nPtr */
} /* end function cubeByReference */
```

declare a function that can receive pointers as arguments

use **&** to pass the address of number

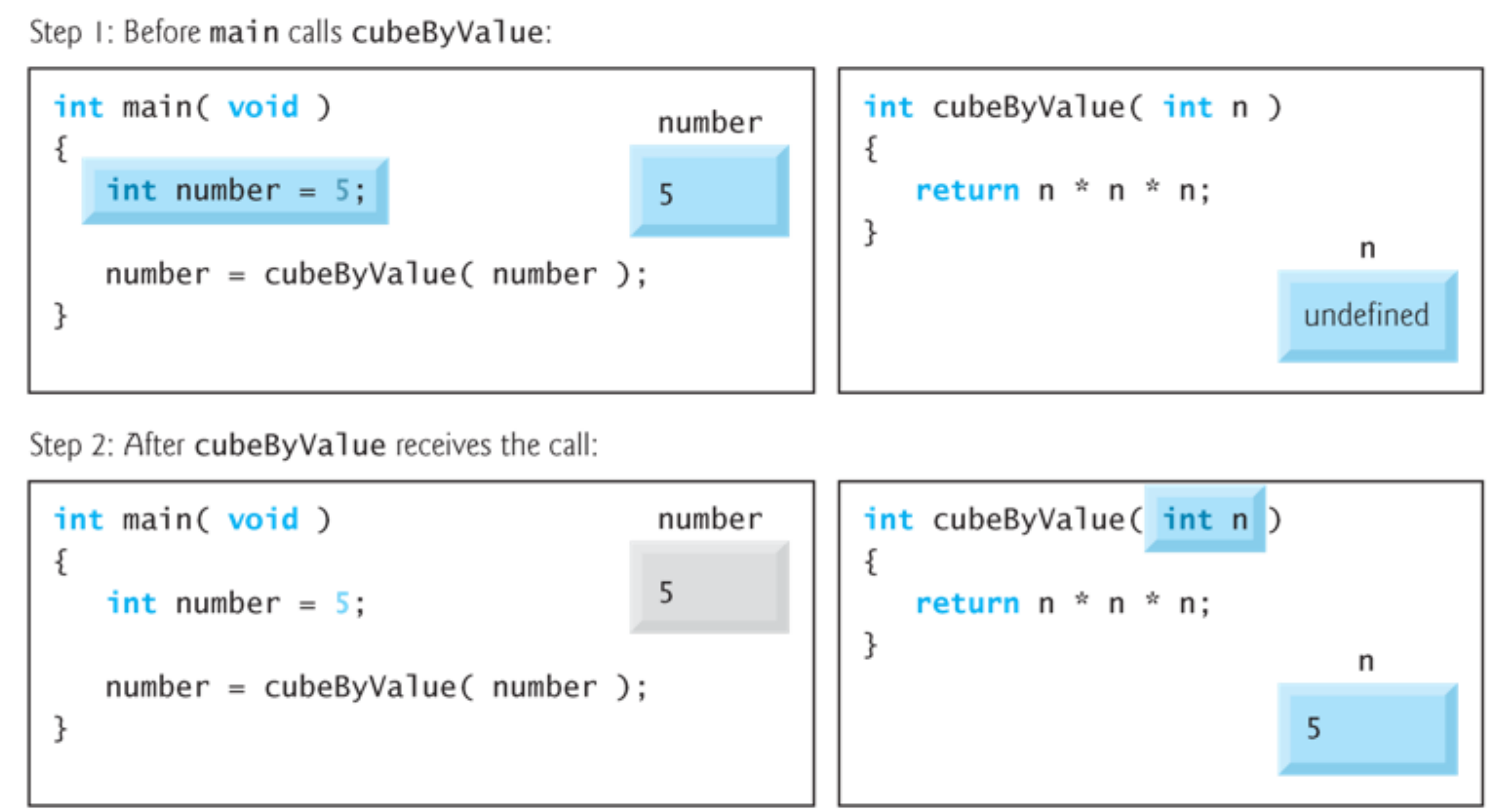
1. declare a pointer, nPtr, to receive the address

2. use * to dereference the pointer



```
The original value of number is 5
The new value of number is 125
```

Passing Arguments to Functions by Reference (Cont.)



Step 1: Before main calls cubeByValue:

```
int main( void )
{
   int number = 5;

   number = cubeByValue( number );
}
```

```
int cubeByValue( int n )
{
   return n * n * n;
}
```

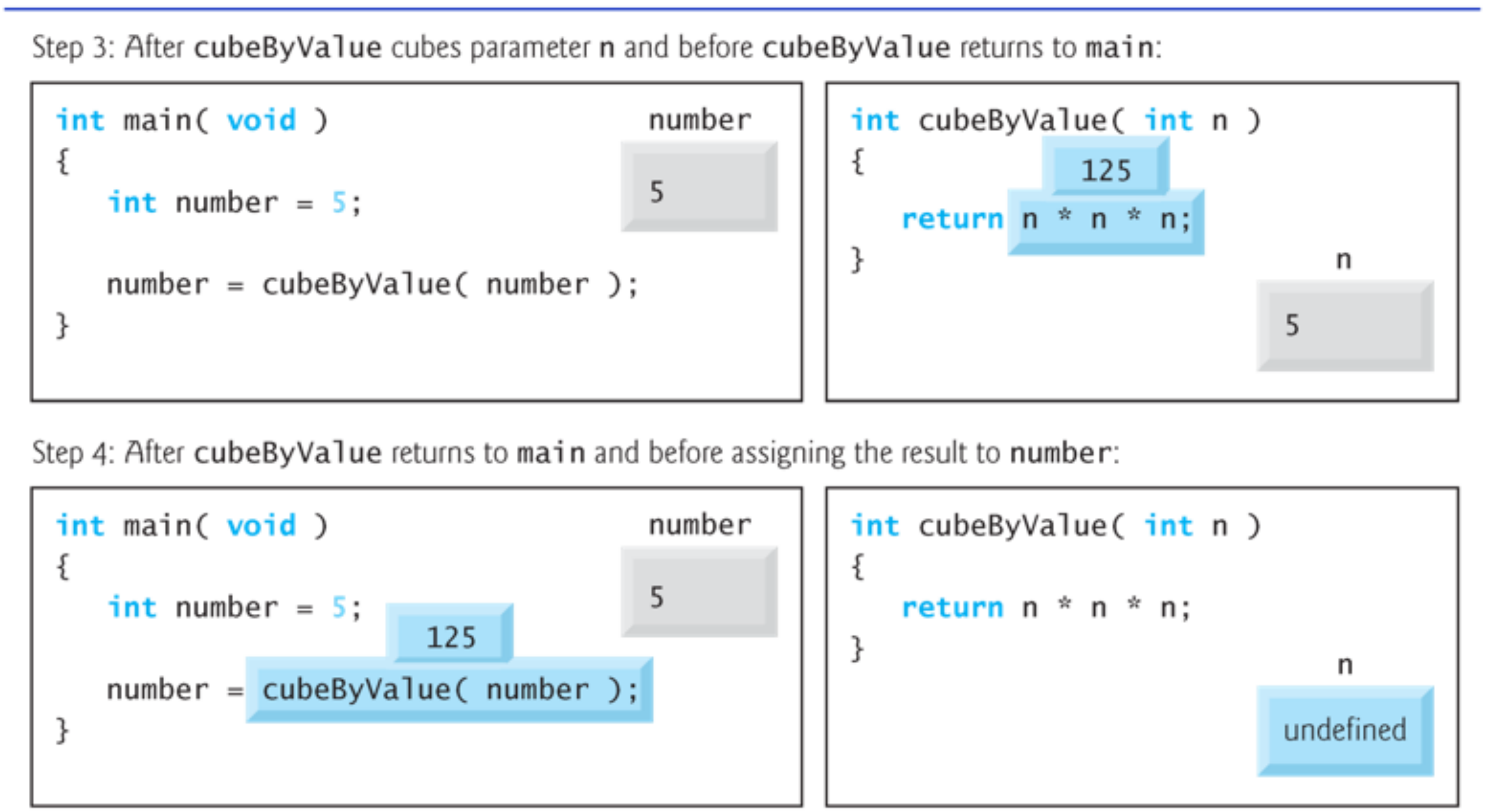
Step 2: After cubeByValue receives the call:

```
int main( void )
{
   int number = 5;

   number = cubeByValue( number );
}
```

```
int cubeByValue( int n )
{
   return n * n * n;
}
```

Fig. 7.8 | Analysis of a typical call-by-value. (Part 1 of 3.)

Passing Arguments to Functions by Reference (Cont.)



Step 3: After **cubeByValue** cubes parameter **n** and before **cubeByValue** returns to **main**:

```
int main( void )
{
   int number = 5;

   number = cubeByValue( number );
}
```

```
int cubeByValue( int n )
{
   return n * n * n;
}
```

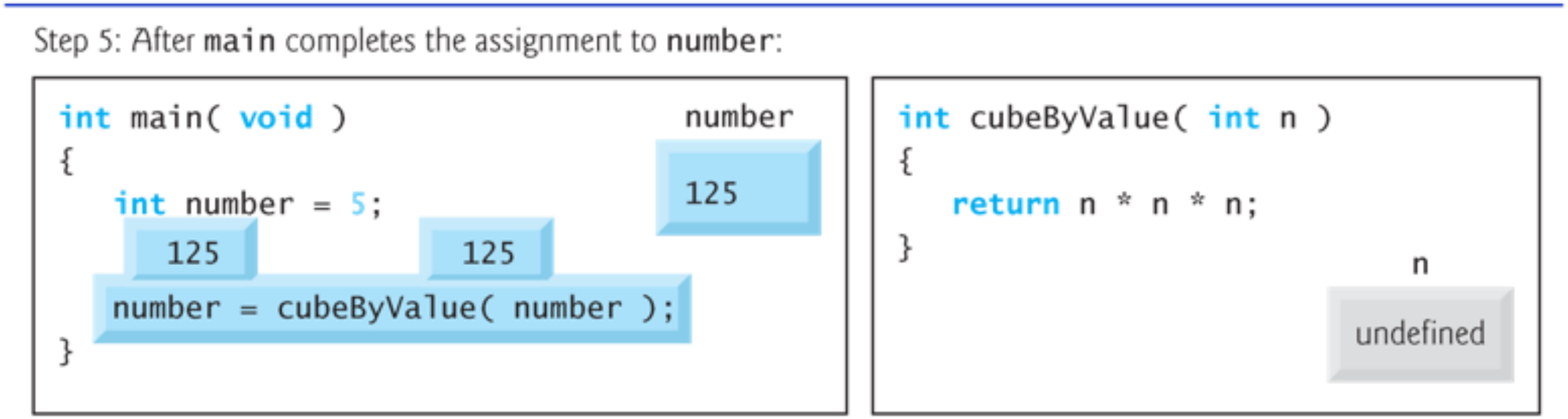
Step 4: After **cubeByValue** returns to **main** and before assigning the result to **number**:

```
int main( void )
{
   int number = 5;

   number = cubeByValue( number );
}
```

```
int cubeByValue( int n )
{
   return n * n * n;
}
```

Fig. 7.8 | Analysis of a typical call-by-value. (Part 2 of 3.)

Passing Arguments to Functions by Reference (Cont.)



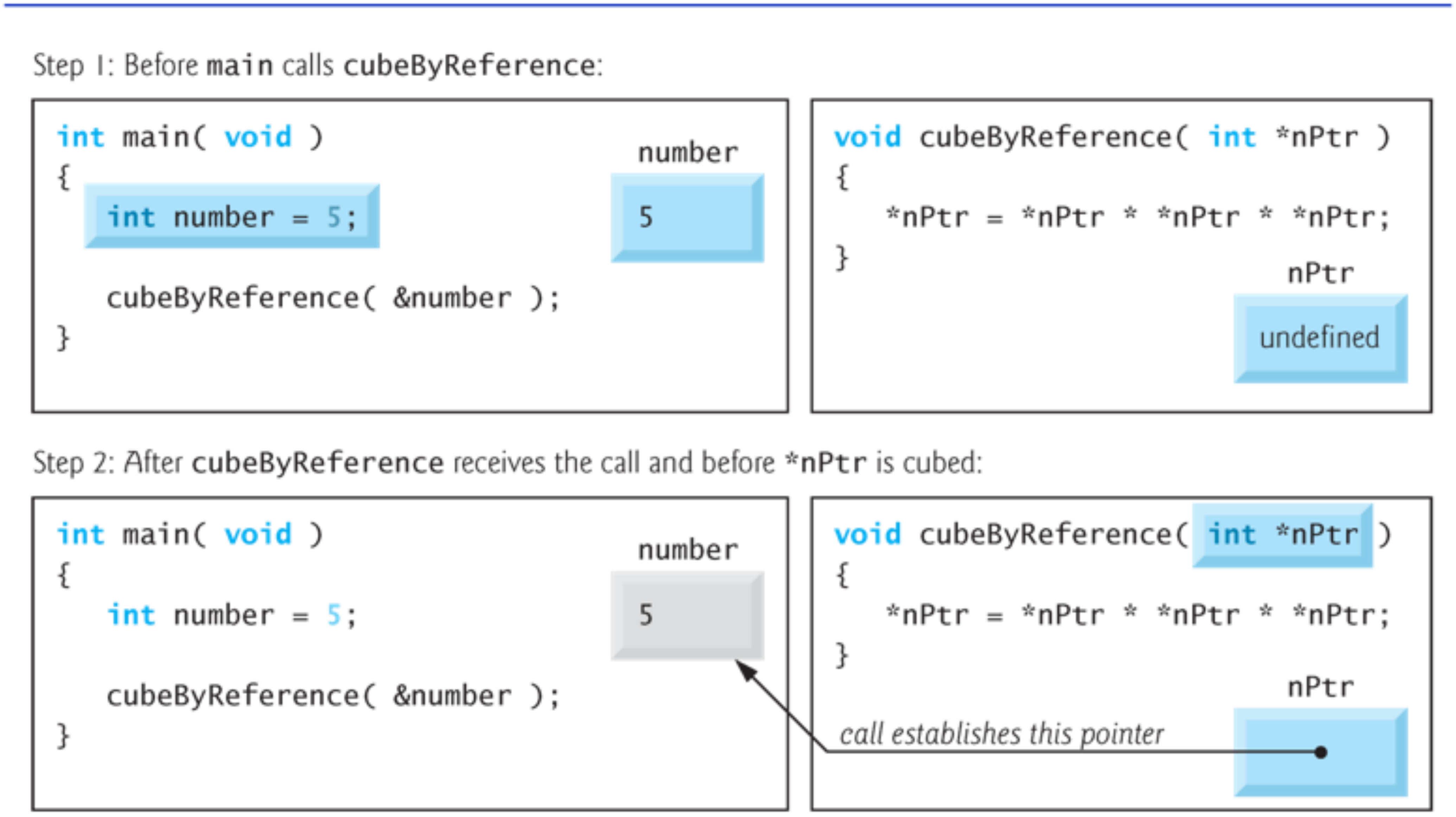
Step 5: After **main** completes the assignment to **number**:

```
int main( void )
{
   int number = 5;
   number = cubeByValue( number );
}
```

```
int cubeByValue( int n )
{
   return n * n * n;
}
```

Fig. 7.8 | Analysis of a typical call-by-value. (Part 3 of 3.)

Passing Arguments to Functions by Reference (Cont.)



Step 1: Before main calls cubeByReference:

```
int main( void )
{
   int number = 5;

   cubeByReference( &number );
}
```

```
void cubeByReference( int *nPtr )
{
   *nPtr = *nPtr * *nPtr * *nPtr;
}
```

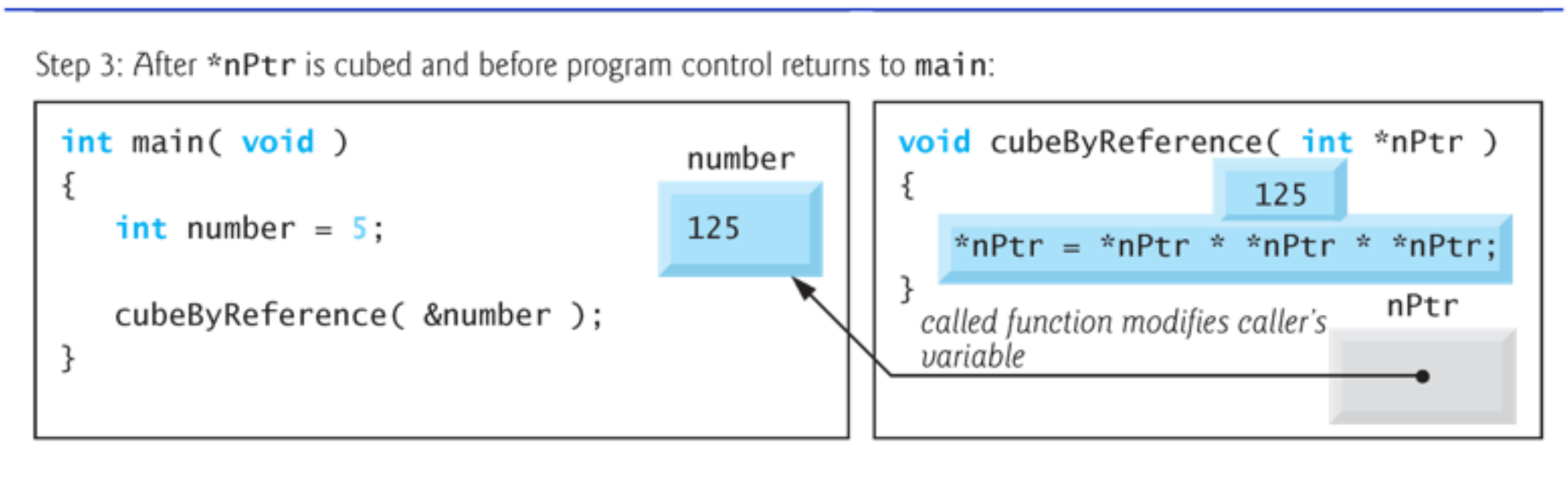
Step 2: After cubeByReference receives the call and before *nPtr is cubed:

```
int main( void )
{
   int number = 5;

   cubeByReference( &number );
}
```

```
void cubeByReference( int *nPtr )
{
   *nPtr = *nPtr * *nPtr * *nPtr;
}
```

Fig. 7.9 | Analysis of a typical call-by-reference with a pointer argument.

Passing Arguments to Functions by Reference (Cont.)



Step 3: After *nPtr is cubed and before program control returns to main:

```
int main( void )
{
   int number = 5;

   cubeByReference( &number );
}
```

```
void cubeByReference( int *nPtr )
{
   *nPtr = *nPtr * *nPtr * *nPtr;
}
```

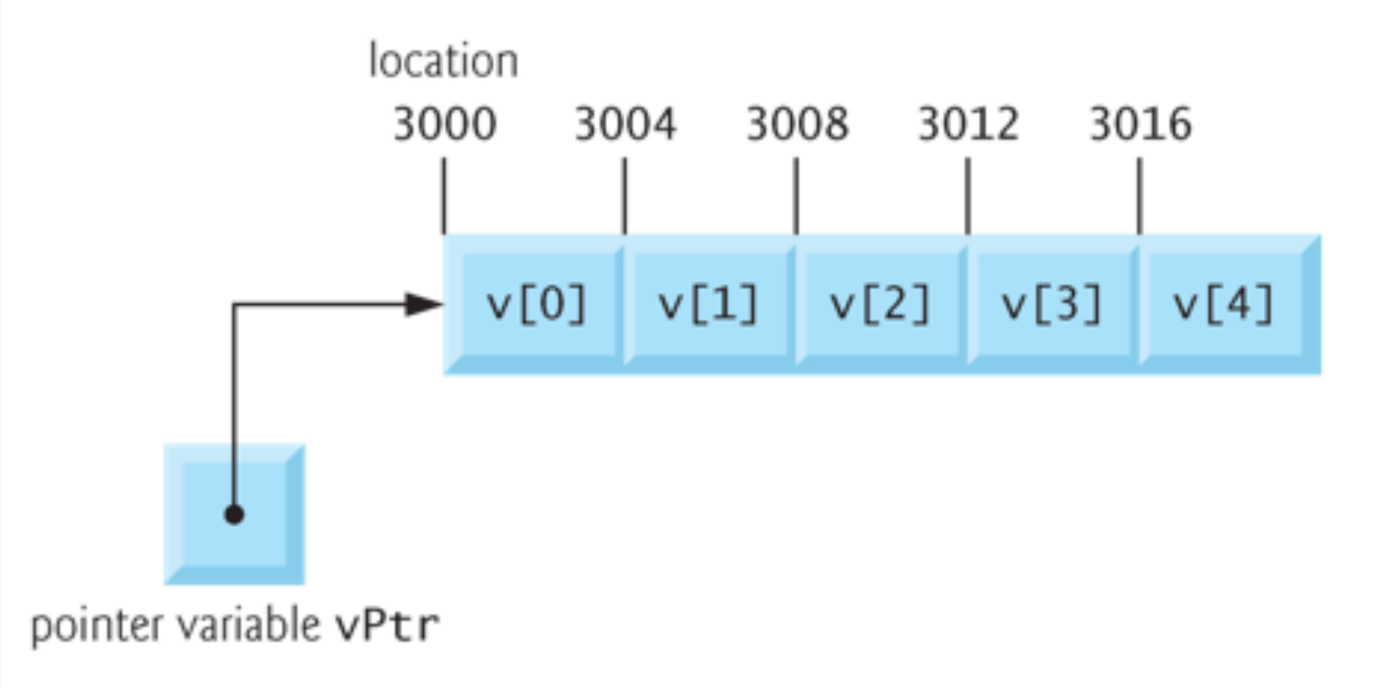
Fig. 7.9 | Analysis of a typical call-by-reference with a pointer argument.

Pointer Expressions and Pointer Arithmetic

- Pointers are valid operands in arithmetic expressions, assignment expressions and comparison expressions.
- This section describes the operators that can have pointers as operands, and how these operators are used.

Pointer Expressions and Pointer Arithmetic (Cont.)

- Assume that array **`int v[5]`** has been defined and its first element is at location 3000 in memory.
- Assume pointer **`vPtr`** has been initialized to point to **`v[0]`**—i.e., the value of **`vPtr`** is 3000.

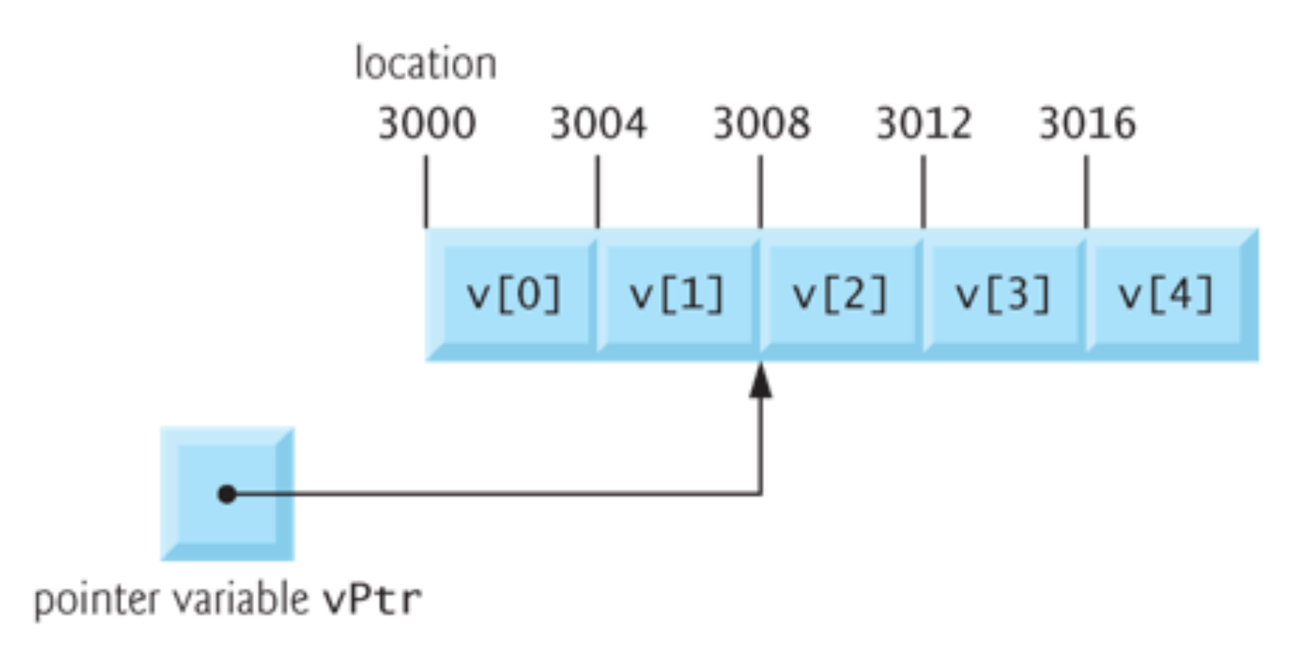


Pointer Expressions and Pointer Arithmetic (Cont.)

- The statement

```
vPtr += 2;
```

would produce 3008 = (3000 + 2 * 4), assuming an integer is stored in 4 bytes of memory.

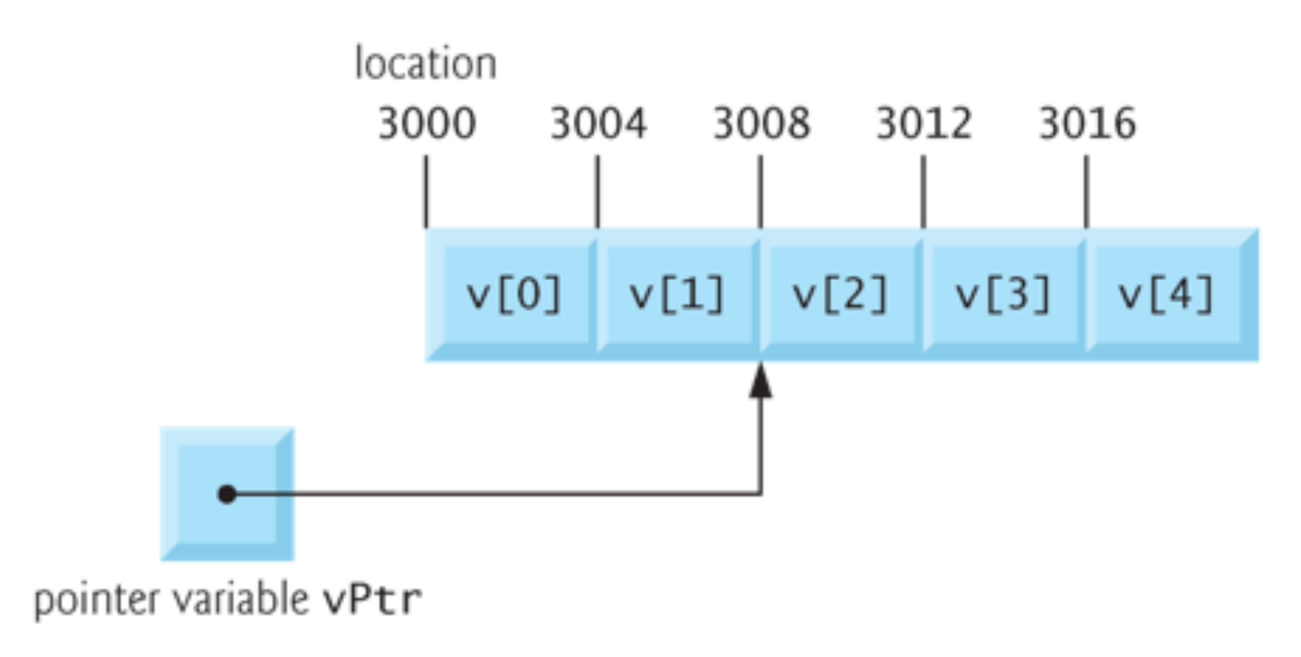


Pointer Expressions and Pointer Arithmetic (Cont.)

- If **`vPtr`** had been incremented to 3016, which points to **`v[4]`**, the statement

```
vPtr -= 4;
```

would set **`vPtr`** back to 3000—the beginning of the array.



Pointer Expressions and Pointer Arithmetic (Cont.)

- If a pointer is being incremented or decremented by one, the increment (++) and decrement (--) operators can be used.
- Either of the statements

```
++vPtr;
vPtr++;
```

  increments the pointer to point to the next location in the array.

- Either of the statements

```
--vPtr;
vPtr--;
```

  decrements the pointer to point to the previous element of the array.

Pointer Expressions and Pointer Arithmetic (Cont.)

- For example, if **`vPtr`** contains the location 3000, and **`v2Ptr`** contains the address 3008, the statement

  ```
  x = v2Ptr - vPtr;
  ```

  would assign to **`x`** **the number of array elements** from **`vPtr`** to **`v2Ptr`**, in this case **2** (not 8).

Pointer Expressions and Pointer Arithmetic (Cont.)

- Pointer to **`void`** (i.e., **`void *`**), which is a generic pointer that can represent any pointer type.
  - A pointer to **`void`** can be assigned a pointer of any type.
  - A pointer to void cannot be dereferenced.
- The compiler must know the data type to determine the number of bytes to be dereferenced for a particular pointer.

Relationship between Pointers and Arrays

- Arrays and pointers are intimately related in C and often may be used interchangeably.
- Assume that integer array **`b[5]`** and integer pointer variable **`bPtr`** have been defined.
- Since the array name (without a subscript) is a pointer to the first element of the array, we can set **`bPtr`** equal to the address of the first element in array **`b`** with the statement

```
bPtr = b;
```

Relationship between Pointers and Arrays (Cont.)

- The above statement is equivalent to taking the address of the array’s first element as follows:

```
bPtr = &b[0];
```

- Array element **`b[3]`** can alternatively be referenced with the pointer expression

```
*(bPtr + 3)
```

- The preceding notation is referred to as pointer/ offset notation.
- The parentheses are necessary because the precedence of * is higher than the precedence of **+**.

Relationship between Pointers and Arrays (Cont.)

- Just as the array element can be referenced with a pointer expression, the address

  ```
  &b[3]
  ```

  can be written with the pointer expression

  ```
  bPtr + 3
  ```

- The array itself can be treated as a pointer and used in pointer arithmetic.

Relationship between Pointers and Arrays (Cont.)

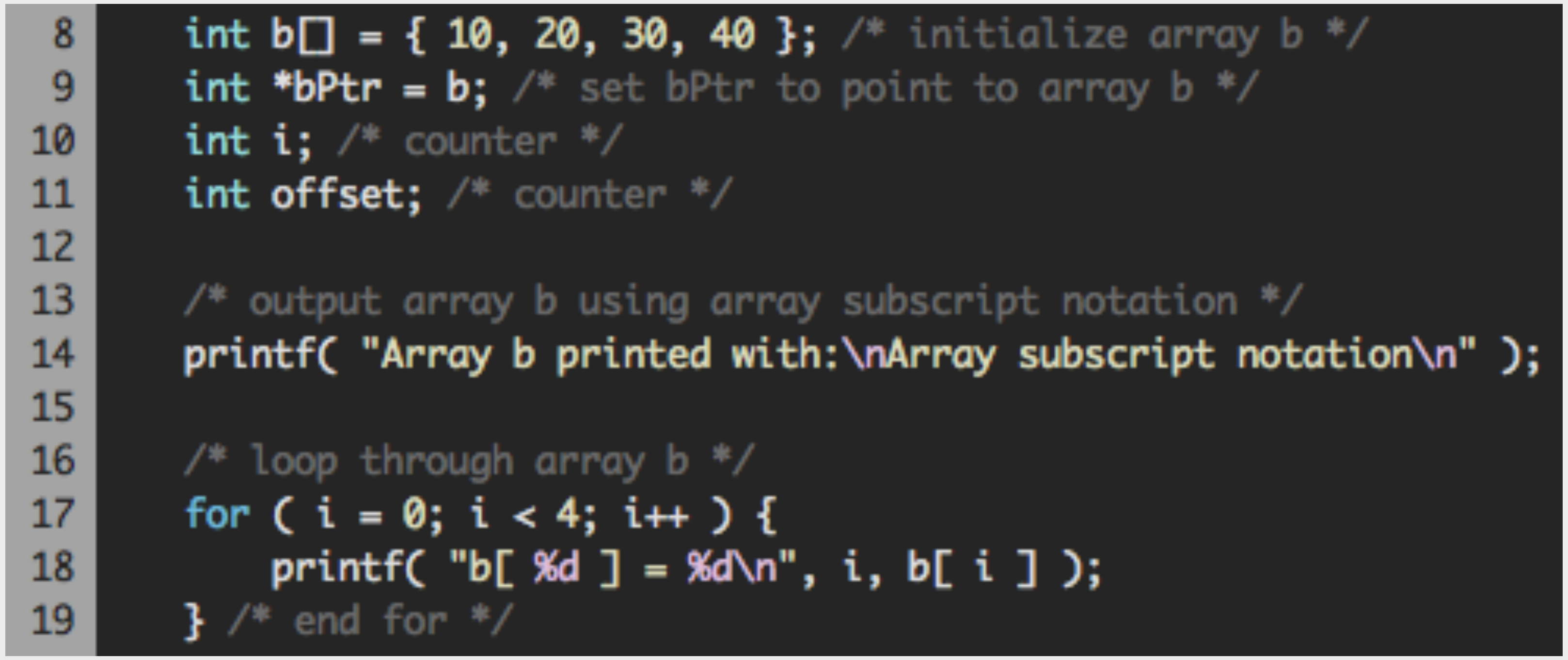
- For example, if **`bPtr`** has the value **`b`**, the expression

  ```
  bPtr[1]
  ```

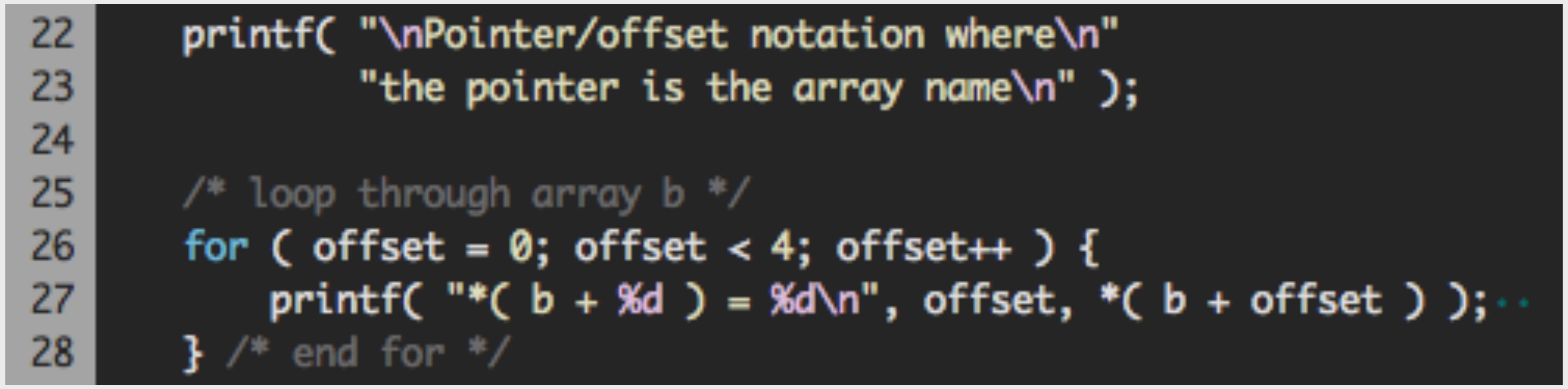
  refers to the array element **`b[1]`**.
- This is referred to as pointer/subscript notation.

Relationship between Pointers and Arrays (Cont.)

- Example: fig07_20.c



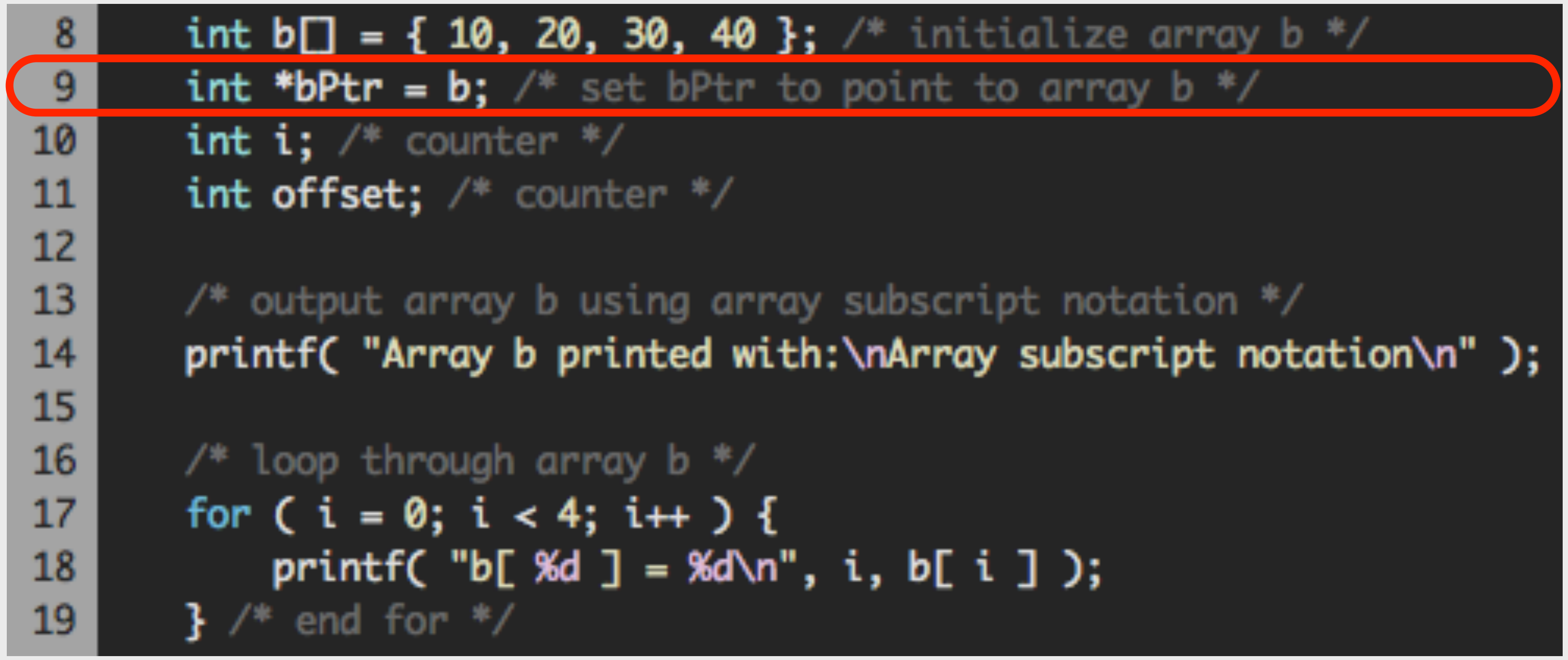
```c
    int b[] = { 10, 20, 30, 40 }; /* initialize array b */
    int *bPtr = b; /* set bPtr to point to array b */
    int i; /* counter */
    int offset; /* counter */

    /* output array b using array subscript notation */
    printf( "Array b printed with:\nArray subscript notation\n" );

    /* loop through array b */
    for ( i = 0; i < 4; i++ ) {
        printf( "b[ %d ] = %d\n", i, b[ i ] );
    } /* end for */
```



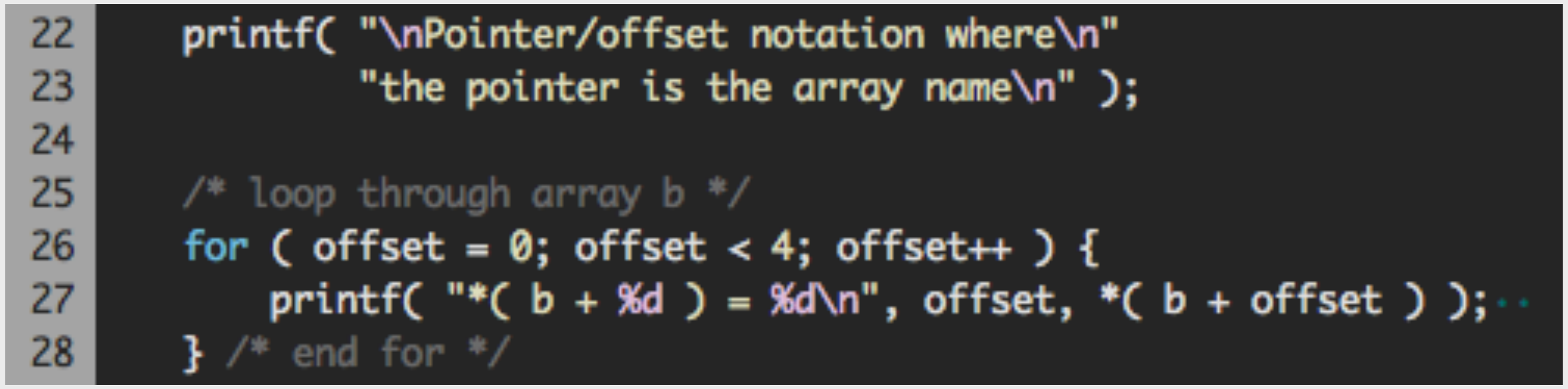
```c
    printf( "\nPointer/offset notation where\n"
            "the pointer is the array name\n" );

    /* loop through array b */
    for ( offset = 0; offset < 4; offset++ ) {
        printf( "*( b + %d ) = %d\n", offset, *( b + offset ) );
    } /* end for */
```

Relationship between Pointers and Arrays (Cont.)

- Example: fig07_20.c



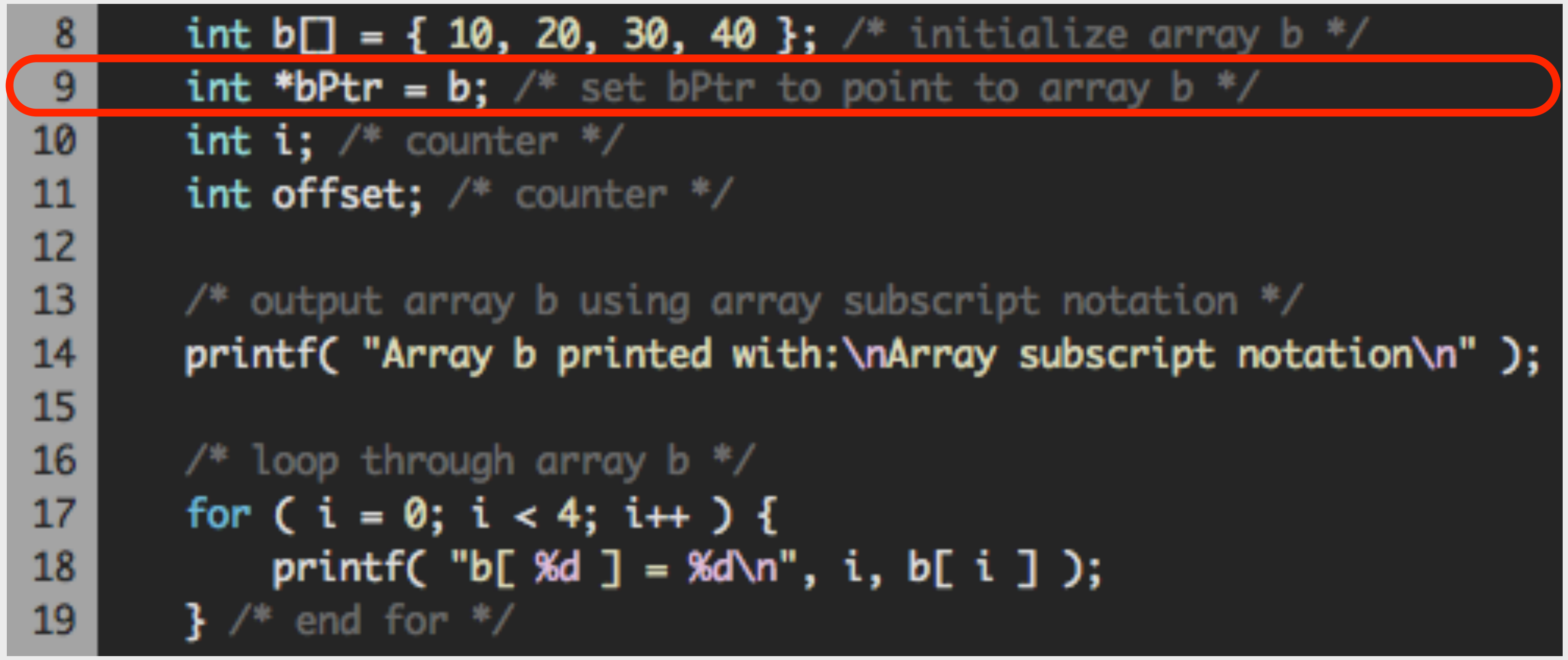
```c
    int b[] = { 10, 20, 30, 40 }; /* initialize array b */
    int *bPtr = b; /* set bPtr to point to array b */
    int i; /* counter */
    int offset; /* counter */

    /* output array b using array subscript notation */
    printf( "Array b printed with:\nArray subscript notation\n" );

    /* loop through array b */
    for ( i = 0; i < 4; i++ ) {
        printf( "b[ %d ] = %d\n", i, b[ i ] );
    } /* end for */
```



```c
    printf( "\nPointer/offset notation where\n"
            "the pointer is the array name\n" );

    /* loop through array b */
    for ( offset = 0; offset < 4; offset++ ) {
        printf( "*( b + %d ) = %d\n", offset, *( b + offset ) );
    } /* end for */
```

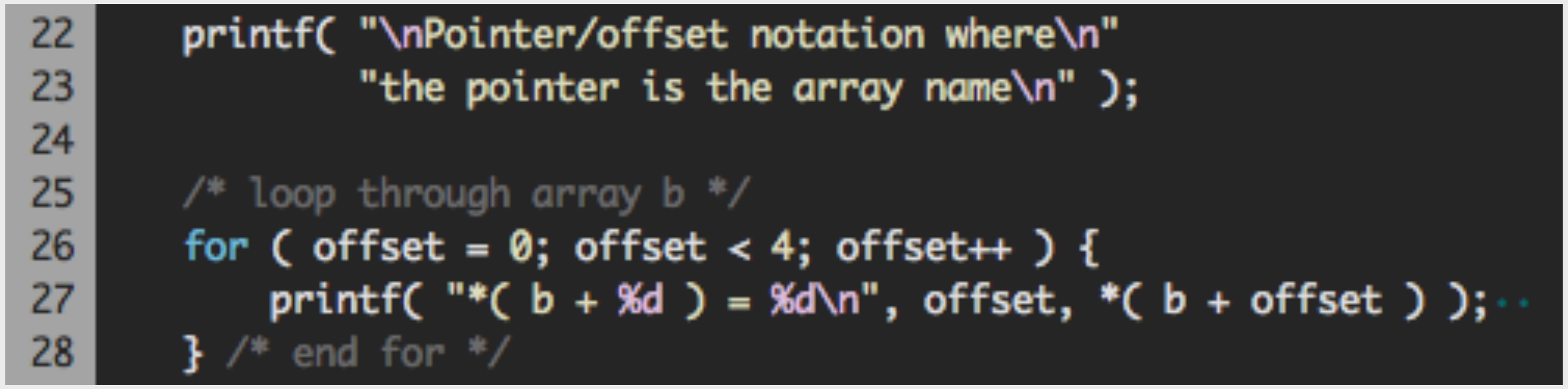
Relationship between Pointers and Arrays (Cont.)

- Example: fig07_20.c



```
int b[] = { 10, 20, 30, 40 }; /* initialize array b */
int *bPtr = b; /* set bPtr to point to array b */
int i; /* counter */
int offset; /* counter */

/* output array b using array subscript notation */
printf( "Array b printed with:\nArray subscript notation\n" );

/* loop through array b */
for ( i = 0; i < 4; i++ ) {
    printf( "b[ %d ] = %d\n", i, b[ i ] );
} /* end for */
```

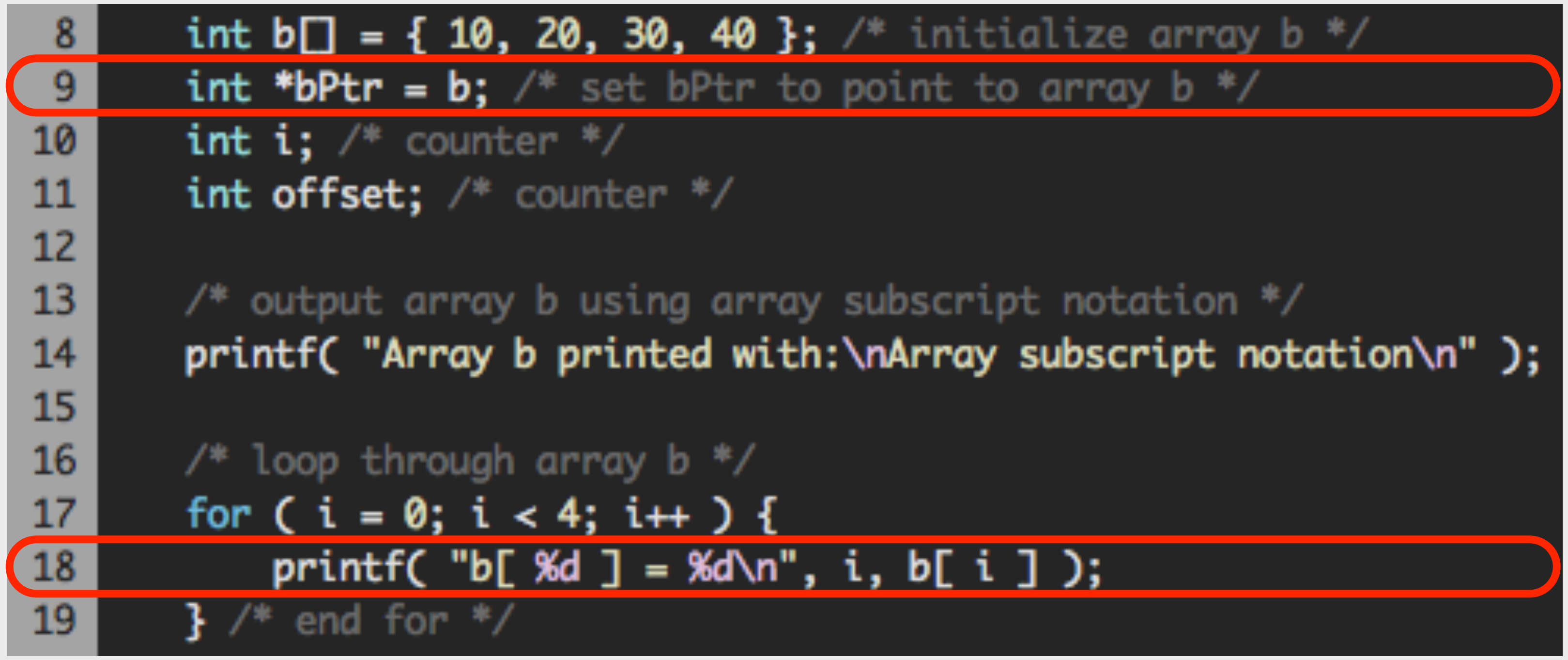
set bPtr to point to array b



```
printf( "\nPointer/offset notation where\n"
        "the pointer is the array name\n" );

/* loop through array b */
for ( offset = 0; offset < 4; offset++ ) {
    printf( "*( b + %d ) = %d\n", offset, *( b + offset ) );
} /* end for */
```

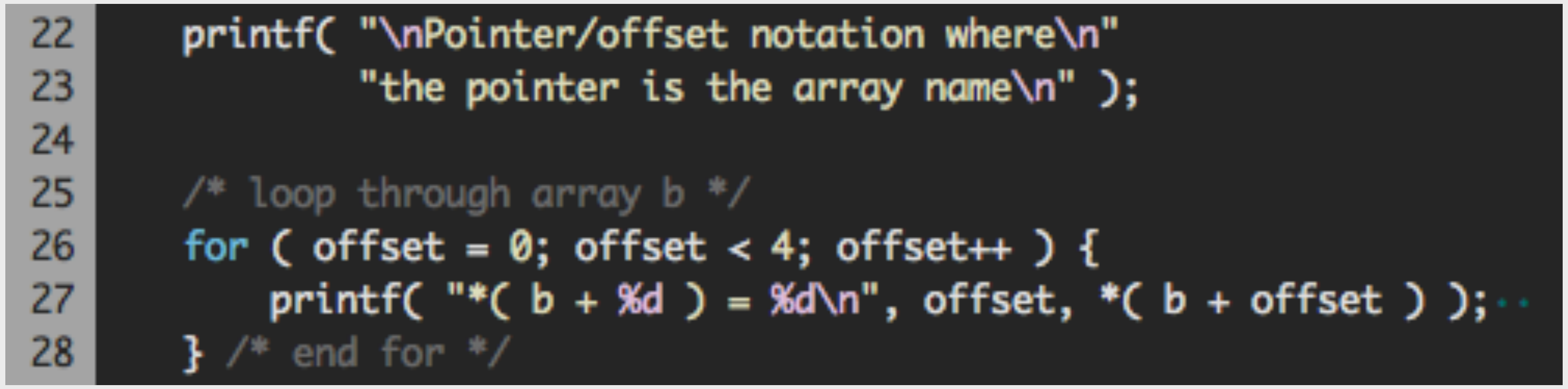
Relationship between Pointers and Arrays (Cont.)

- Example: fig07_20.c



```
   int b[] = { 10, 20, 30, 40 }; /* initialize array b */
   int *bPtr = b; /* set bPtr to point to array b */
   int i; /* counter */
   int offset; /* counter */

   /* output array b using array subscript notation */
   printf( "Array b printed with:\nArray subscript notation\n" );

   /* loop through array b */
   for ( i = 0; i < 4; i++ ) {
      printf( "b[ %d ] = %d\n", i, b[ i ] );
   } /* end for */
```

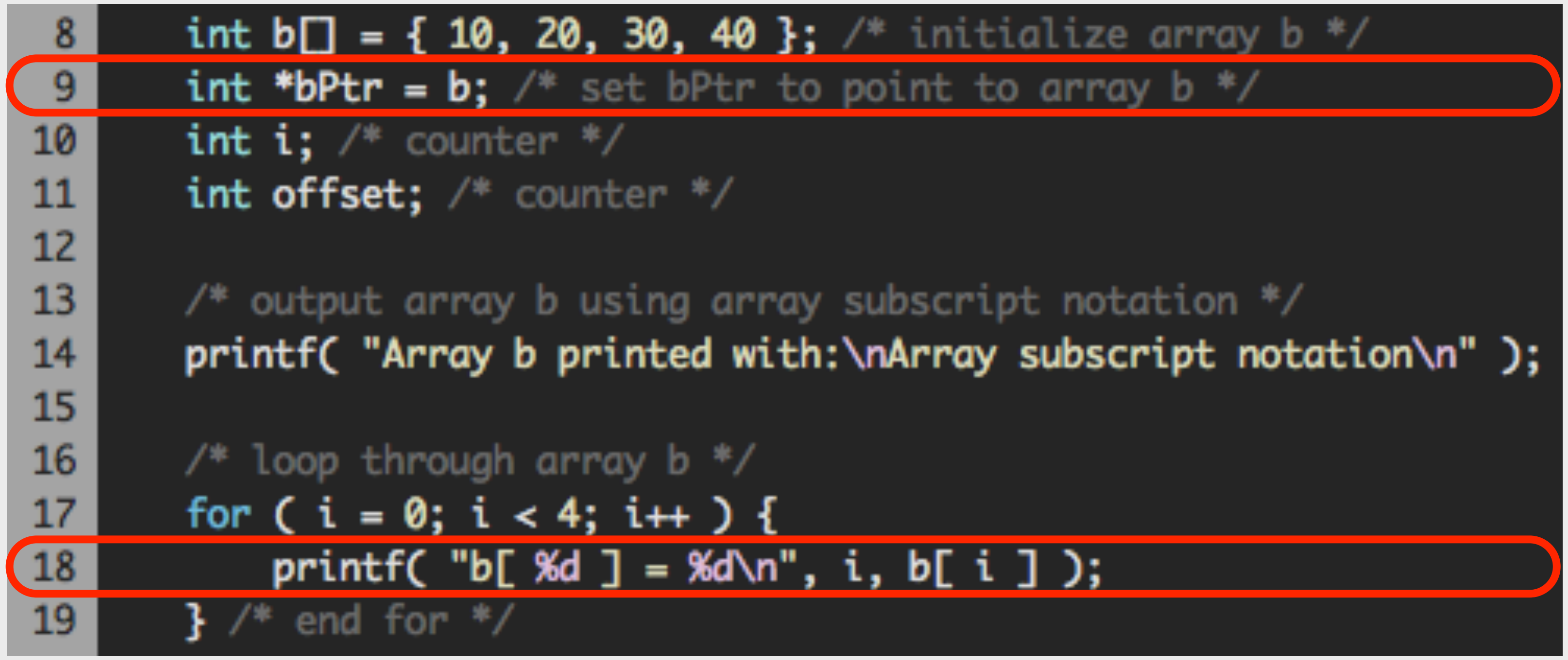
set bPtr to point to array b



```
   printf( "\nPointer/offset notation where\n"
           "the pointer is the array name\n" );

   /* loop through array b */
   for ( offset = 0; offset < 4; offset++ ) {
      printf( "*( b + %d ) = %d\n", offset, *( b + offset ) );
   } /* end for */
```

Relationship between Pointers and Arrays (Cont.)

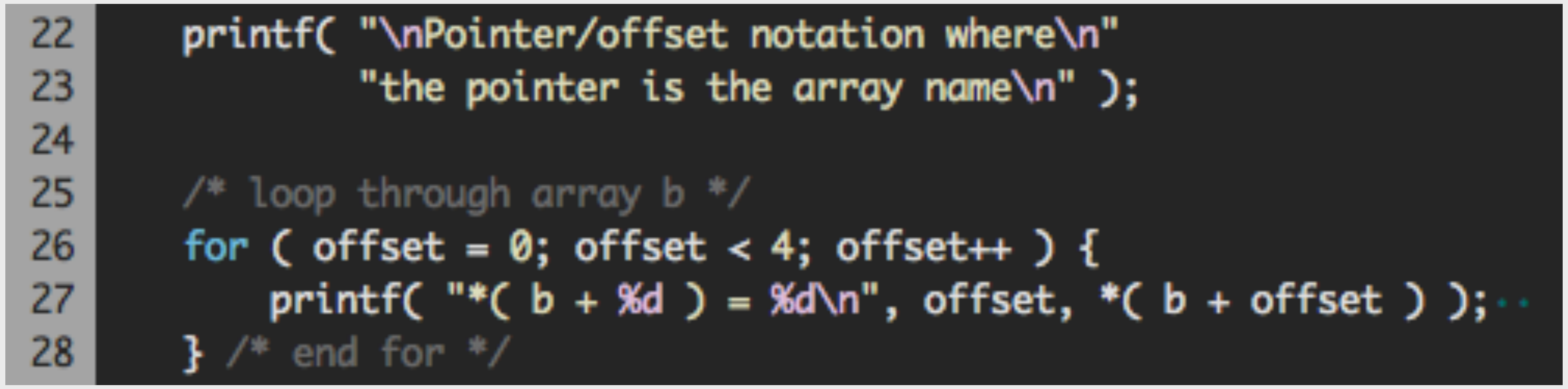
- Example: fig07_20.c



```c
   int b[] = { 10, 20, 30, 40 }; /* initialize array b */
   int *bPtr = b; /* set bPtr to point to array b */
   int i; /* counter */
   int offset; /* counter */

   /* output array b using array subscript notation */
   printf( "Array b printed with:\nArray subscript notation\n" );

   /* loop through array b */
   for ( i = 0; i < 4; i++ ) {
      printf( "b[ %d ] = %d\n", i, b[ i ] );
   } /* end for */
```

set bPtr to point to array b

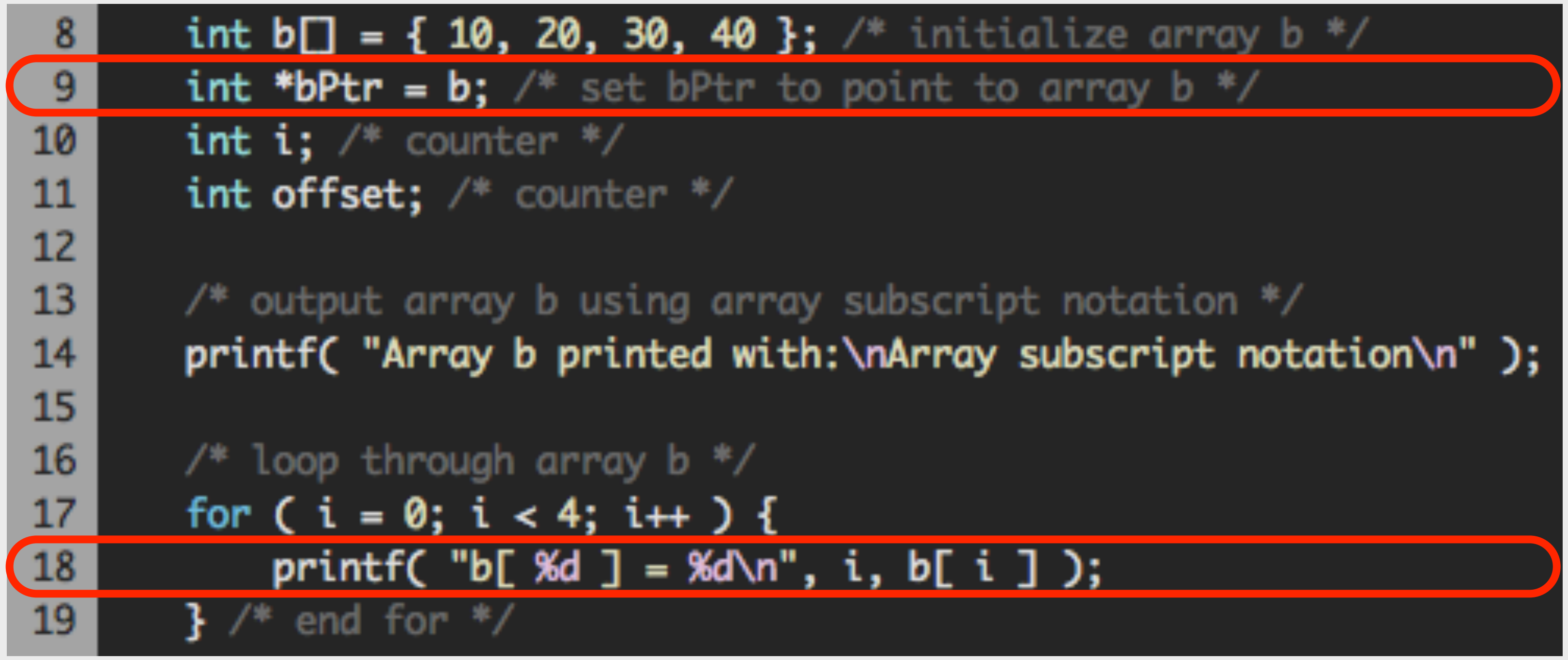
array subscript notation



```c
   printf( "\nPointer/offset notation where\n"
           "the pointer is the array name\n" );

   /* loop through array b */
   for ( offset = 0; offset < 4; offset++ ) {
      printf( "*( b + %d ) = %d\n", offset, *( b + offset ) );
   } /* end for */
```

Relationship between Pointers and Arrays (Cont.)

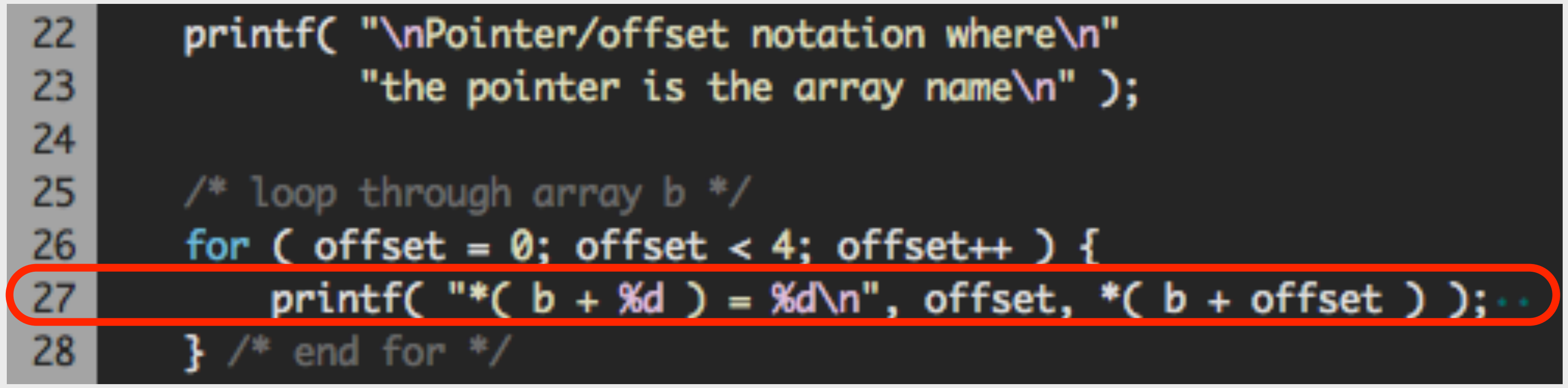
- Example: fig07_20.c



```c
    int b[] = { 10, 20, 30, 40 }; /* initialize array b */
    int *bPtr = b; /* set bPtr to point to array b */
    int i; /* counter */
    int offset; /* counter */

    /* output array b using array subscript notation */
    printf( "Array b printed with:\nArray subscript notation\n" );

    /* loop through array b */
    for ( i = 0; i < 4; i++ ) {
        printf( "b[ %d ] = %d\n", i, b[ i ] );
    } /* end for */
```

set bPtr to point to array b

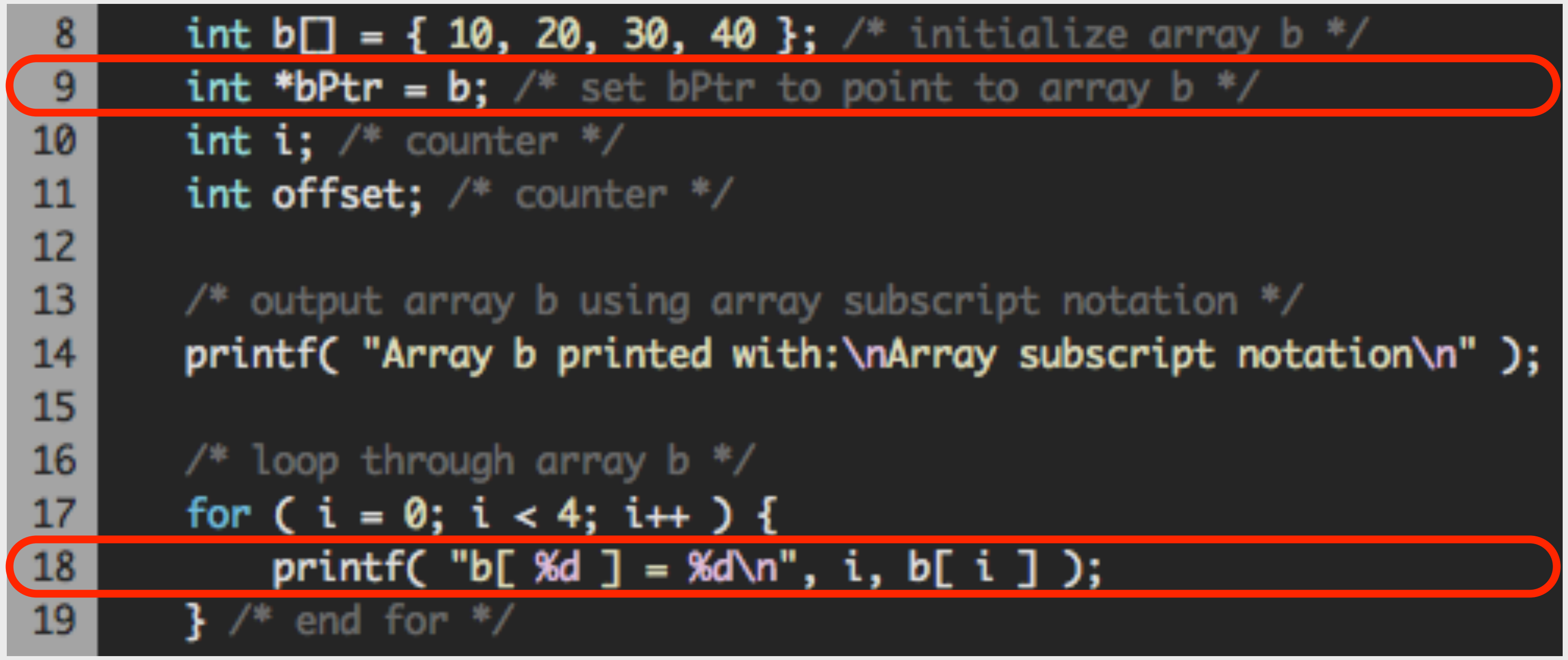
array subscript notation



```c
    printf( "\nPointer/offset notation where\n"
            "the pointer is the array name\n" );

    /* loop through array b */
    for ( offset = 0; offset < 4; offset++ ) {
        printf( "*( b + %d ) = %d\n", offset, *( b + offset ) );
    } /* end for */
```

Relationship between Pointers and Arrays (Cont.)

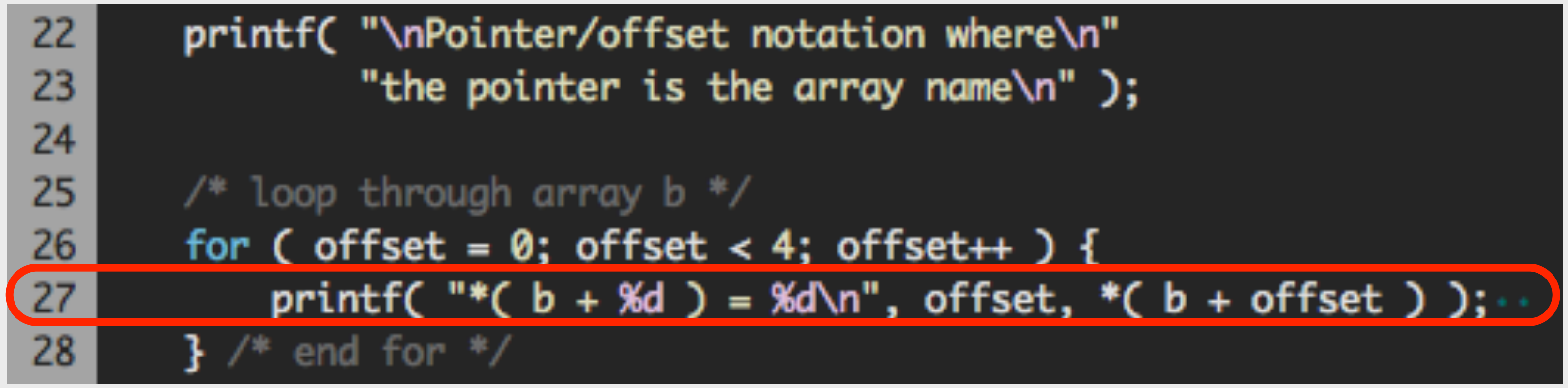
- Example: fig07_20.c



```c
    int b[] = { 10, 20, 30, 40 }; /* initialize array b */
    int *bPtr = b; /* set bPtr to point to array b */
    int i; /* counter */
    int offset; /* counter */

    /* output array b using array subscript notation */
    printf( "Array b printed with:\nArray subscript notation\n" );

    /* loop through array b */
    for ( i = 0; i < 4; i++ ) {
        printf( "b[ %d ] = %d\n", i, b[ i ] );
    } /* end for */
```

set bPtr to point to array b

array subscript notation

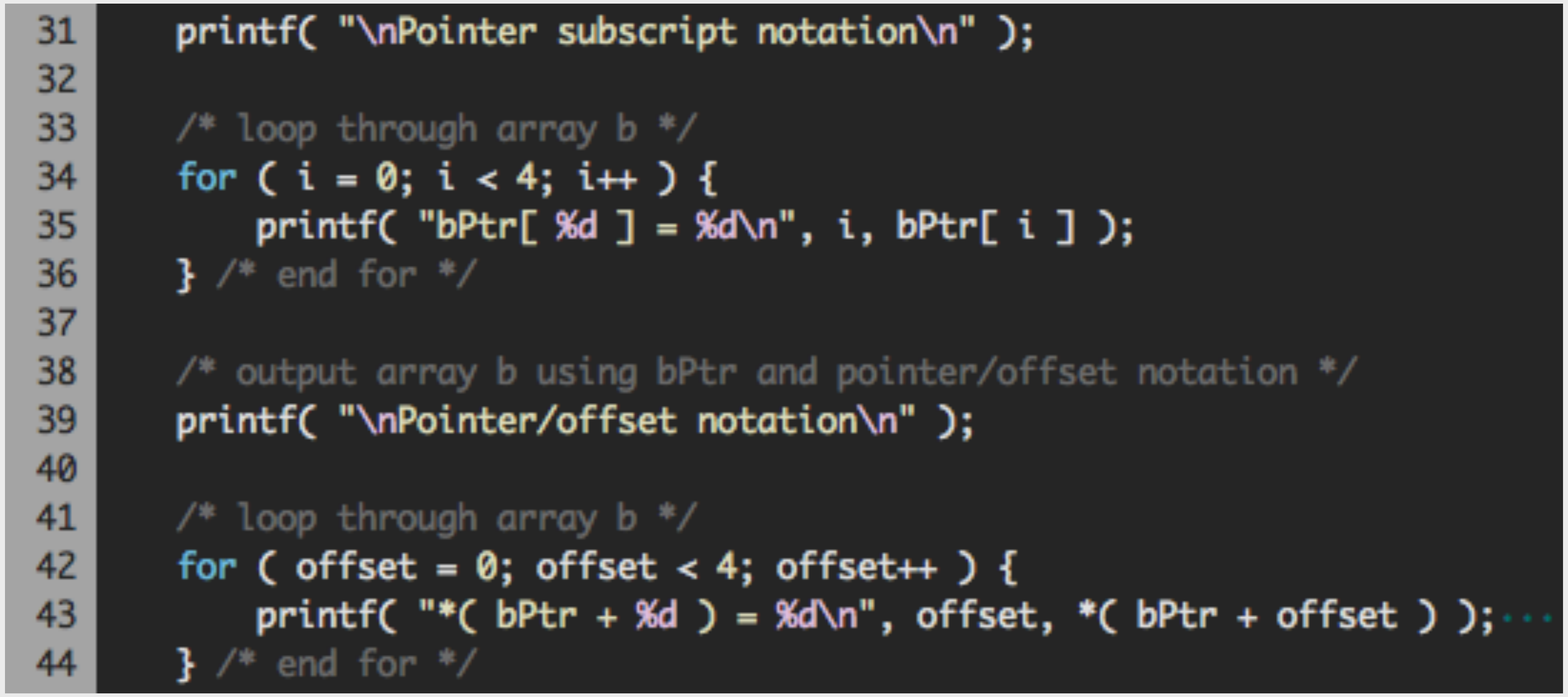


```c
    printf( "\nPointer/offset notation where\n"
            "the pointer is the array name\n" );

    /* loop through array b */
    for ( offset = 0; offset < 4; offset++ ) {
        printf( "*( b + %d ) = %d\n", offset, *( b + offset ) );
    } /* end for */
```

pointer/offset notation

Relationship between Pointers and Arrays (Cont.)

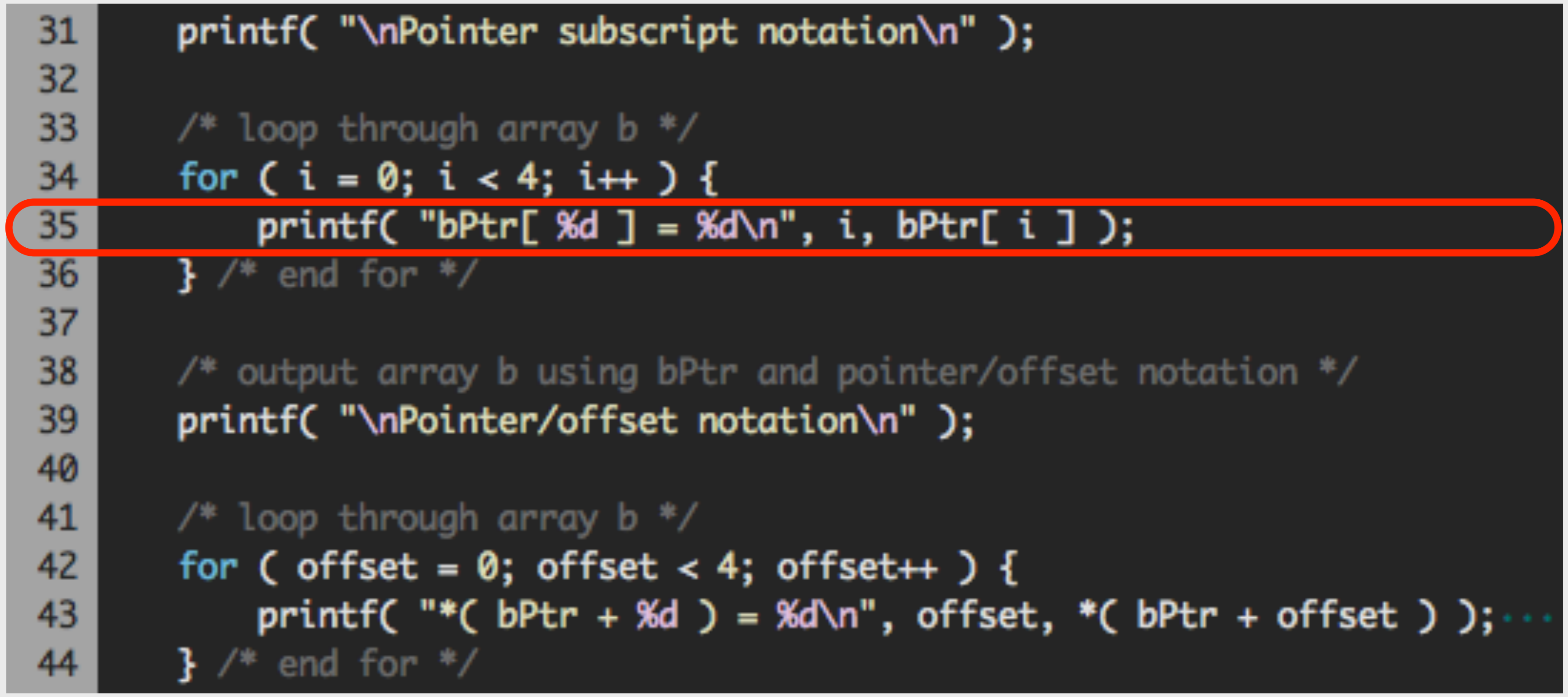
- Example: fig07_20.c



```c
    printf( "\nPointer subscript notation\n" );

    /* loop through array b */
    for ( i = 0; i < 4; i++ ) {
        printf( "bPtr[ %d ] = %d\n", i, bPtr[ i ] );
    } /* end for */

    /* output array b using bPtr and pointer/offset notation */
    printf( "\nPointer/offset notation\n" );

    /* loop through array b */
    for ( offset = 0; offset < 4; offset++ ) {
        printf( "*( bPtr + %d ) = %d\n", offset, *( bPtr + offset ) );
    } /* end for */
```

Relationship between Pointers and Arrays (Cont.)

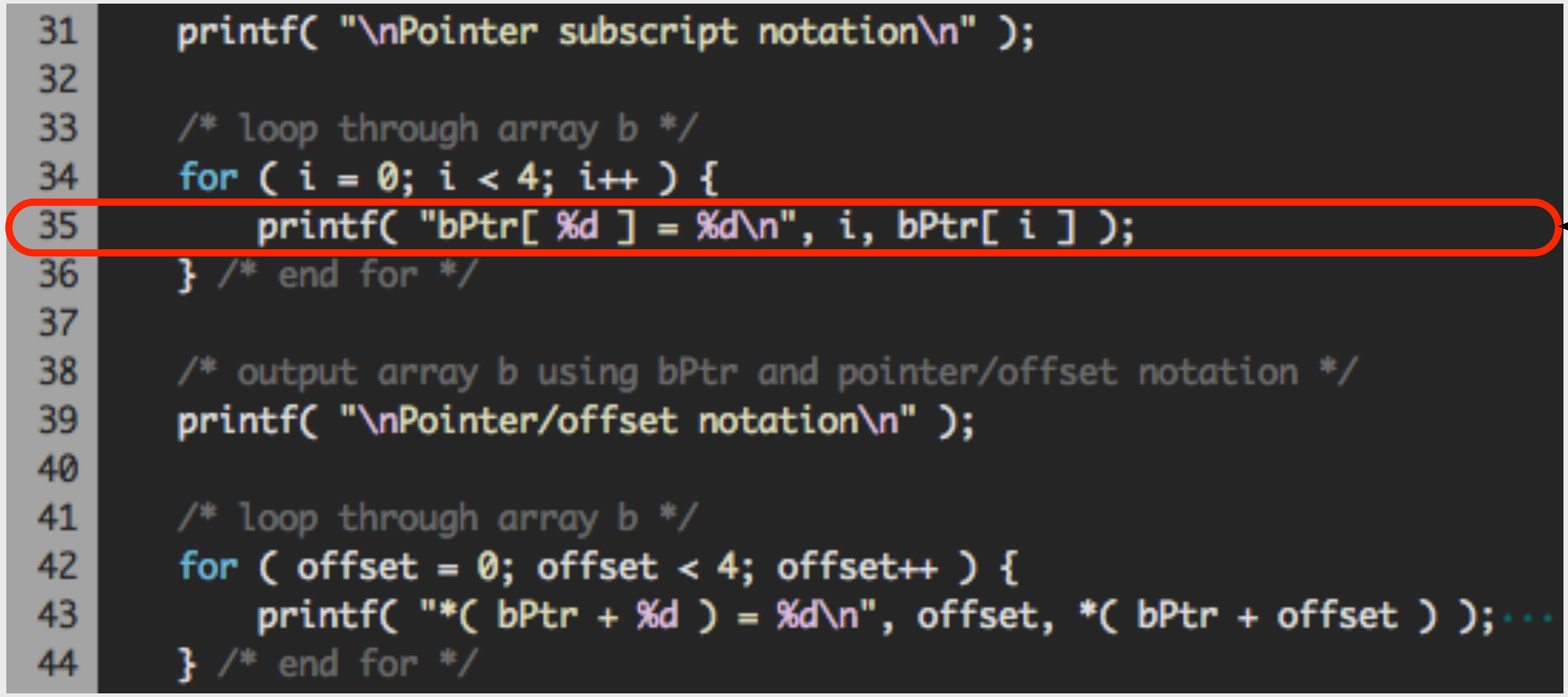
- Example: fig07_20.c



```c
   printf( "\nPointer subscript notation\n" );

   /* loop through array b */
   for ( i = 0; i < 4; i++ ) {
      printf( "bPtr[ %d ] = %d\n", i, bPtr[ i ] );
   } /* end for */

   /* output array b using bPtr and pointer/offset notation */
   printf( "\nPointer/offset notation\n" );

   /* loop through array b */
   for ( offset = 0; offset < 4; offset++ ) {
      printf( "*( bPtr + %d ) = %d\n", offset, *( bPtr + offset ) );
   } /* end for */
```

Relationship between Pointers and Arrays (Cont.)

- Example: fig07_20.c

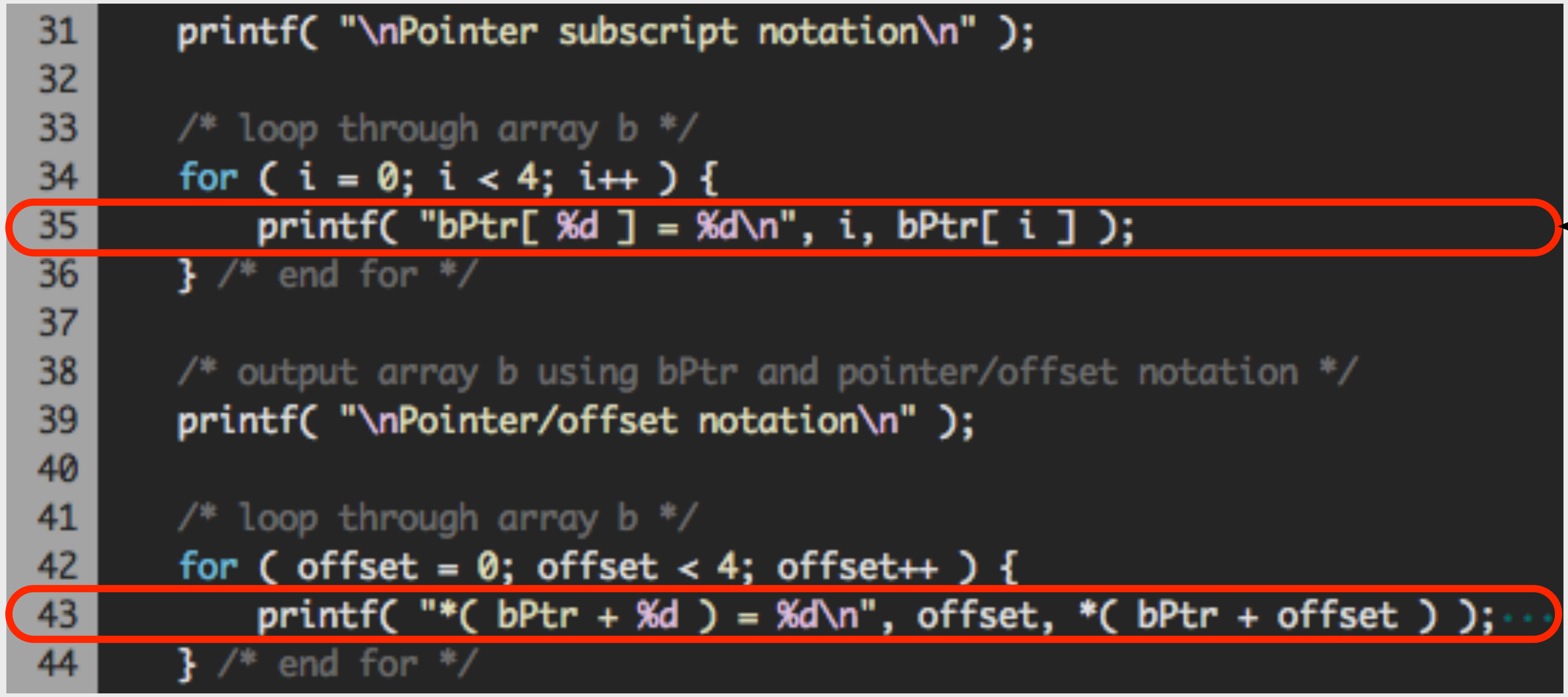


```c
    printf( "\nPointer subscript notation\n" );

    /* loop through array b */
    for ( i = 0; i < 4; i++ ) {
        printf( "bPtr[ %d ] = %d\n", i, bPtr[ i ] );
    } /* end for */

    /* output array b using bPtr and pointer/offset notation */
    printf( "\nPointer/offset notation\n" );

    /* loop through array b */
    for ( offset = 0; offset < 4; offset++ ) {
        printf( "*( bPtr + %d ) = %d\n", offset, *( bPtr + offset ) );
    } /* end for */
```

pointer subscript notation

Relationship between Pointers and Arrays (Cont.)

- Example: fig07_20.c

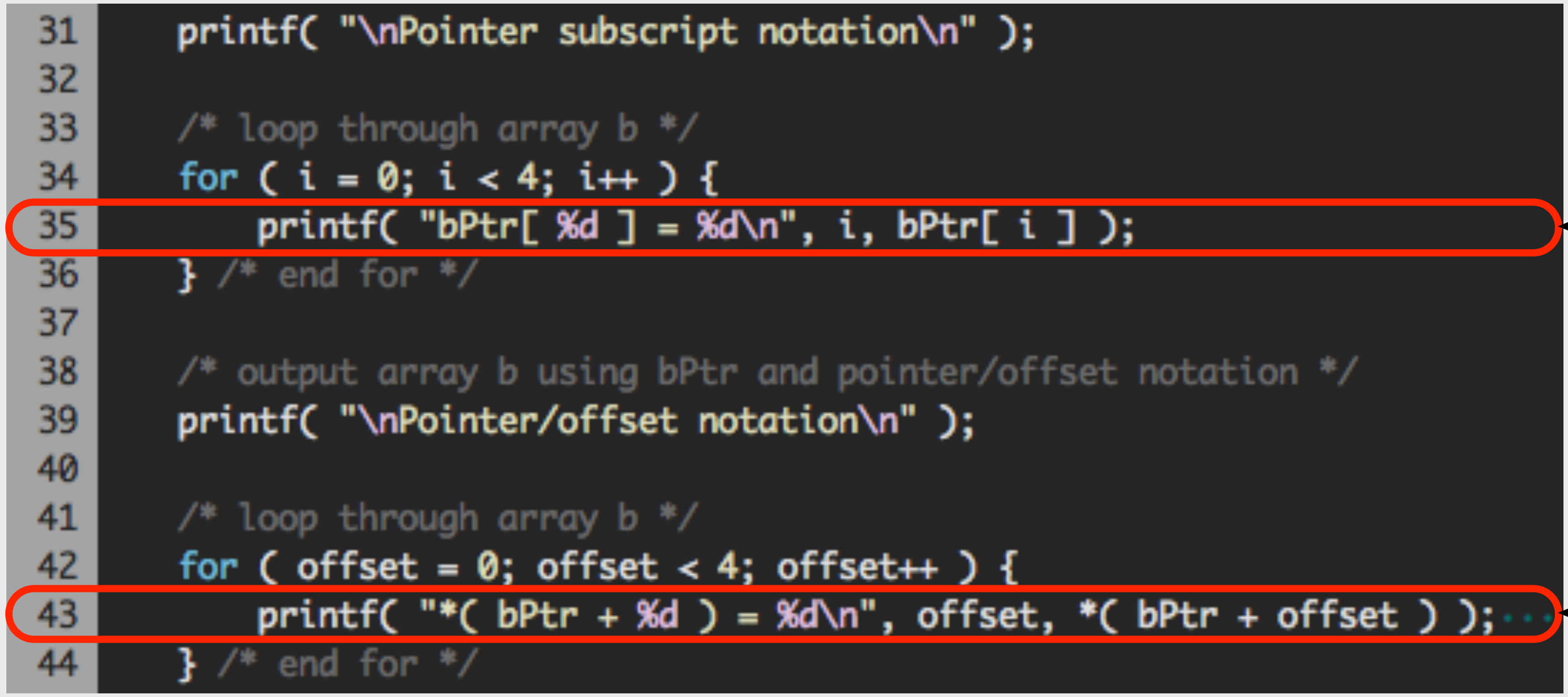


```c
    printf( "\nPointer subscript notation\n" );

    /* loop through array b */
    for ( i = 0; i < 4; i++ ) {
        printf( "bPtr[ %d ] = %d\n", i, bPtr[ i ] );
    } /* end for */

    /* output array b using bPtr and pointer/offset notation */
    printf( "\nPointer/offset notation\n" );

    /* loop through array b */
    for ( offset = 0; offset < 4; offset++ ) {
        printf( "*( bPtr + %d ) = %d\n", offset, *( bPtr + offset ) );
    } /* end for */
```

pointer subscript notation

Relationship between Pointers and Arrays (Cont.)

- Example: fig07_20.c



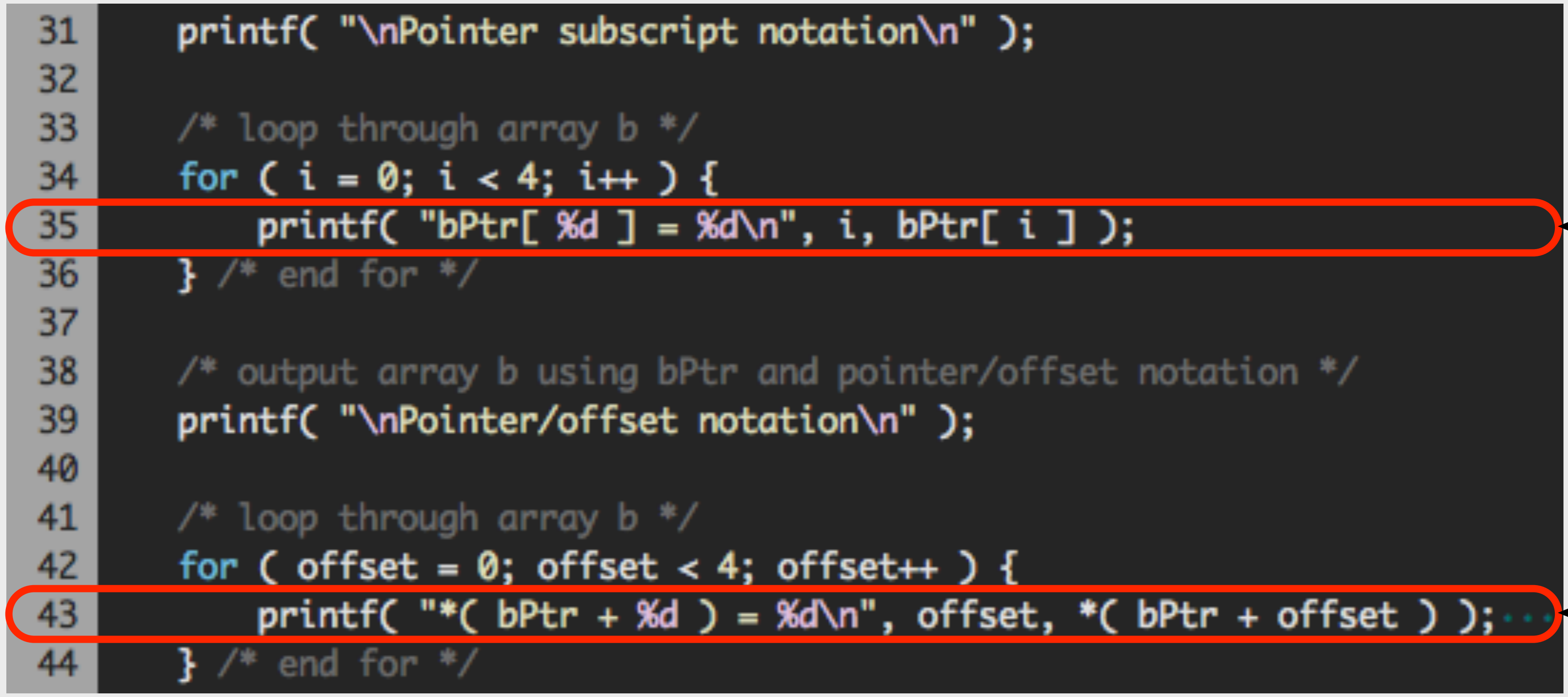
```
    printf( "\nPointer subscript notation\n" );

    /* loop through array b */
    for ( i = 0; i < 4; i++ ) {
        printf( "bPtr[ %d ] = %d\n", i, bPtr[ i ] );
    } /* end for */

    /* output array b using bPtr and pointer/offset notation */
    printf( "\nPointer/offset notation\n" );

    /* loop through array b */
    for ( offset = 0; offset < 4; offset++ ) {
        printf( "*( bPtr + %d ) = %d\n", offset, *( bPtr + offset ) );
    } /* end for */
```

pointer subscript notation

pointer/offset notation

Relationship between Pointers and Arrays (Cont.)

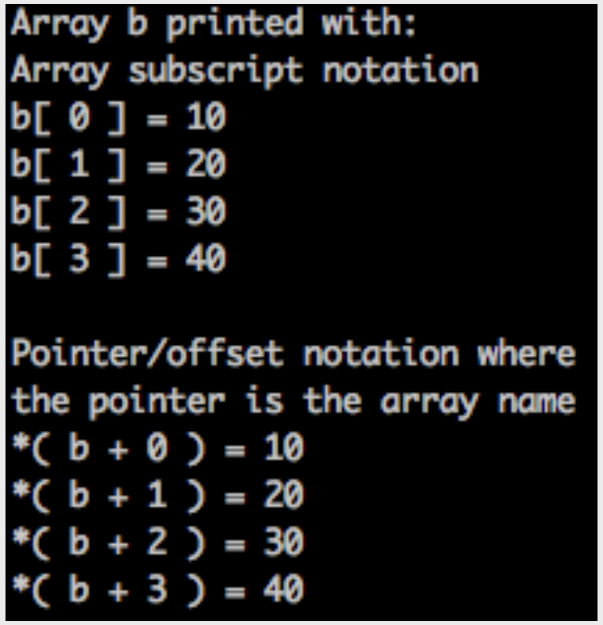
- Example: fig07_20.c



```
    printf( "\nPointer subscript notation\n" );

    /* loop through array b */
    for ( i = 0; i < 4; i++ ) {
        printf( "bPtr[ %d ] = %d\n", i, bPtr[ i ] );
    } /* end for */

    /* output array b using bPtr and pointer/offset notation */
    printf( "\nPointer/offset notation\n" );

    /* loop through array b */
    for ( offset = 0; offset < 4; offset++ ) {
        printf( "*( bPtr + %d ) = %d\n", offset, *( bPtr + offset ) );
    } /* end for */
```

pointer subscript notation

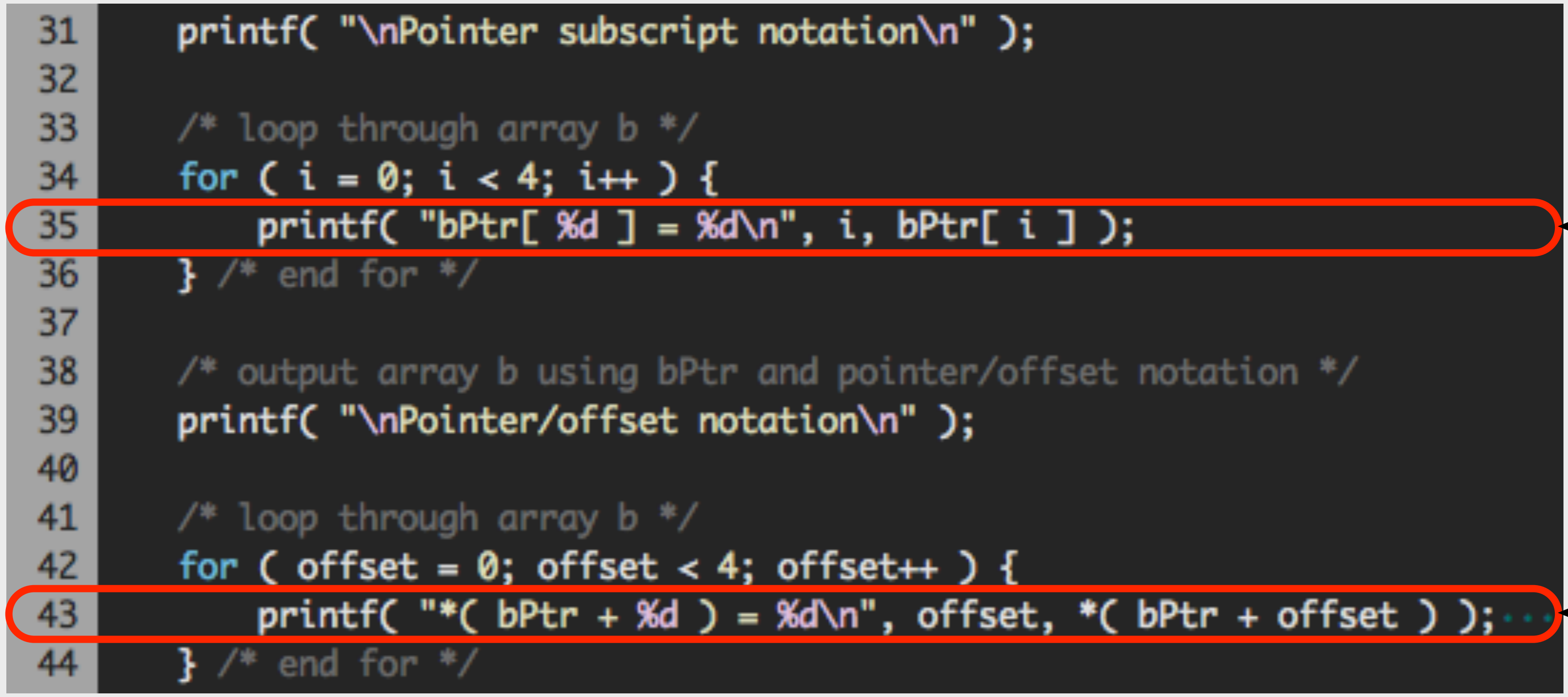
pointer/offset notation



```
Array b printed with:
Array subscript notation
b[ 0 ] = 10
b[ 1 ] = 20
b[ 2 ] = 30
b[ 3 ] = 40

Pointer/offset notation where
the pointer is the array name
*( b + 0 ) = 10
*( b + 1 ) = 20
*( b + 2 ) = 30
*( b + 3 ) = 40
```

Relationship between Pointers and Arrays (Cont.)

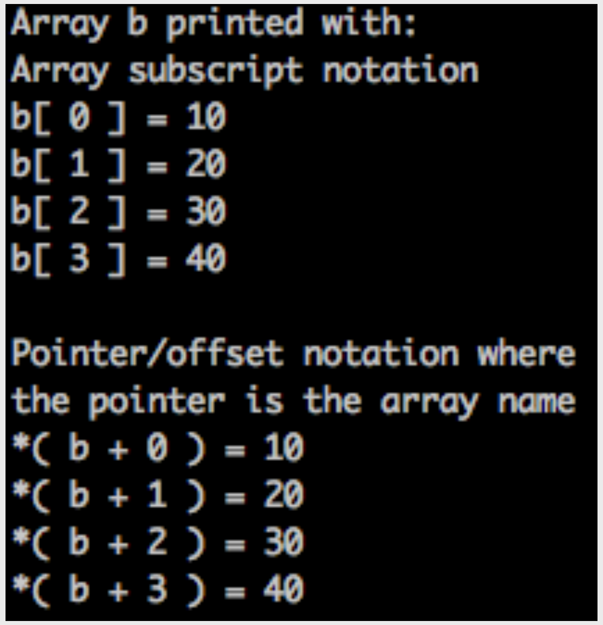
- Example: fig07_20.c



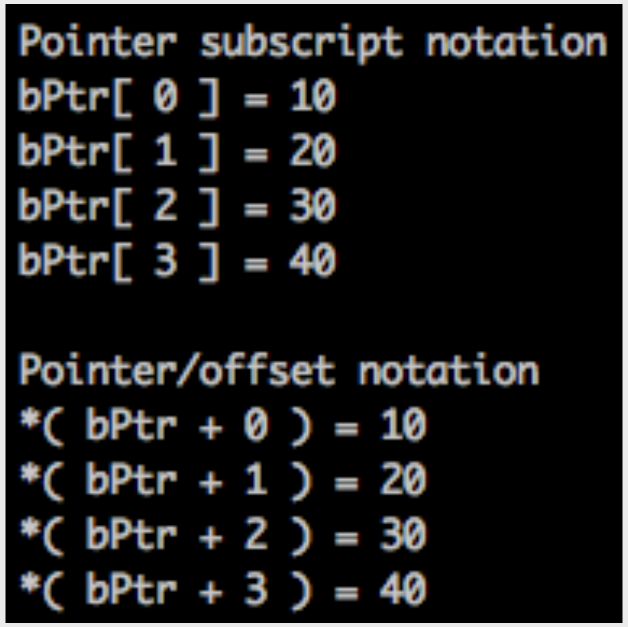
```c
   printf( "\nPointer subscript notation\n" );

   /* loop through array b */
   for ( i = 0; i < 4; i++ ) {
      printf( "bPtr[ %d ] = %d\n", i, bPtr[ i ] );
   } /* end for */

   /* output array b using bPtr and pointer/offset notation */
   printf( "\nPointer/offset notation\n" );

   /* loop through array b */
   for ( offset = 0; offset < 4; offset++ ) {
      printf( "*( bPtr + %d ) = %d\n", offset, *( bPtr + offset ) );
   } /* end for */
```

pointer subscript notation

pointer/offset notation



```
Array b printed with:
Array subscript notation
b[ 0 ] = 10
b[ 1 ] = 20
b[ 2 ] = 30
b[ 3 ] = 40

Pointer/offset notation where
the pointer is the array name
*( b + 0 ) = 10
*( b + 1 ) = 20
*( b + 2 ) = 30
*( b + 3 ) = 40
```



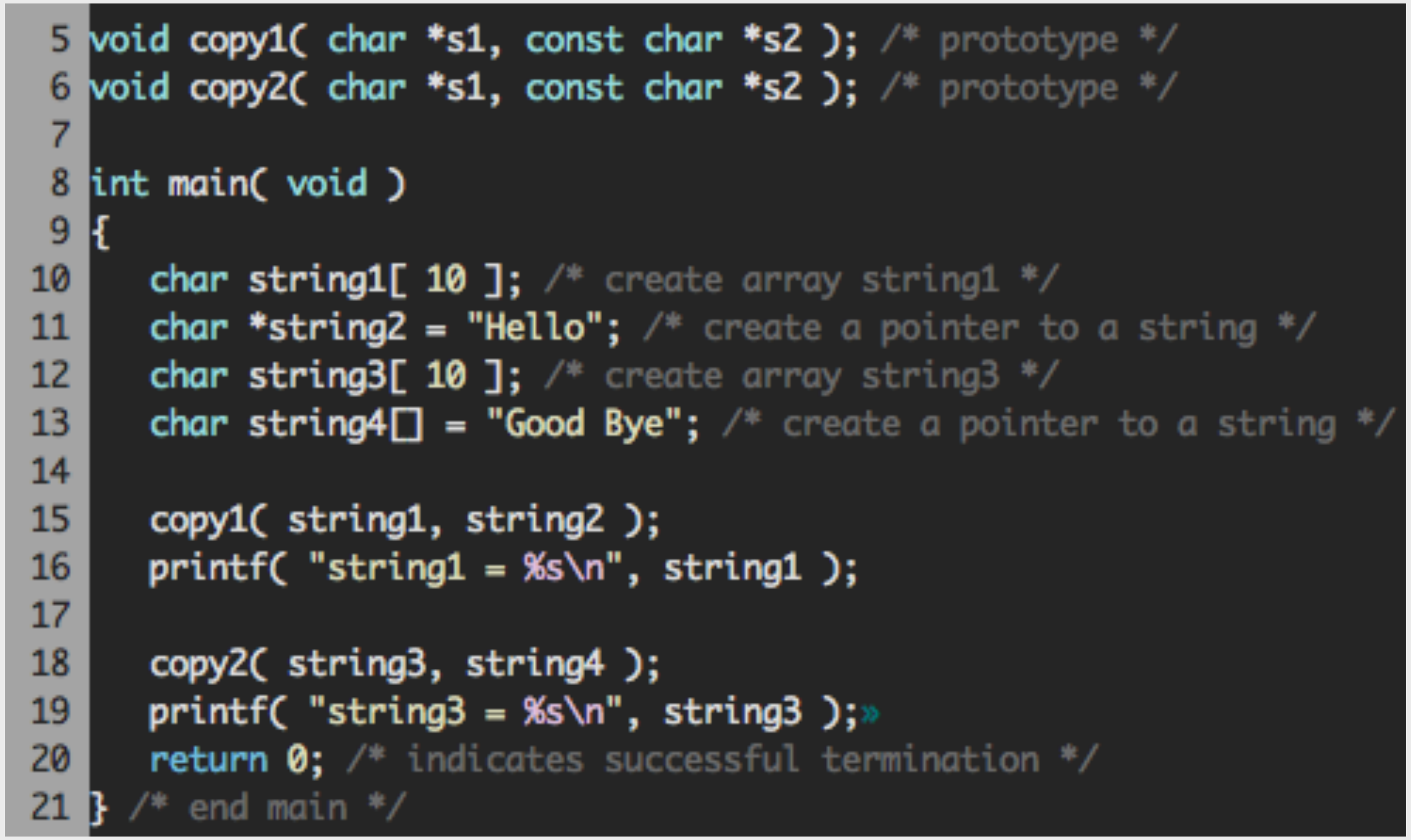
```
Pointer subscript notation
bPtr[ 0 ] = 10
bPtr[ 1 ] = 20
bPtr[ 2 ] = 30
bPtr[ 3 ] = 40

Pointer/offset notation
*( bPtr + 0 ) = 10
*( bPtr + 1 ) = 20
*( bPtr + 2 ) = 30
*( bPtr + 3 ) = 40
```

Relationship between Pointers and Arrays (Cont.)

- To further illustrate the interchangeability of arrays and pointers, let’s look at the two string-copying functions—**copy1** and **copy2** —in the program of Fig. 7.21.

Relationship between Pointers and Arrays (Cont.)

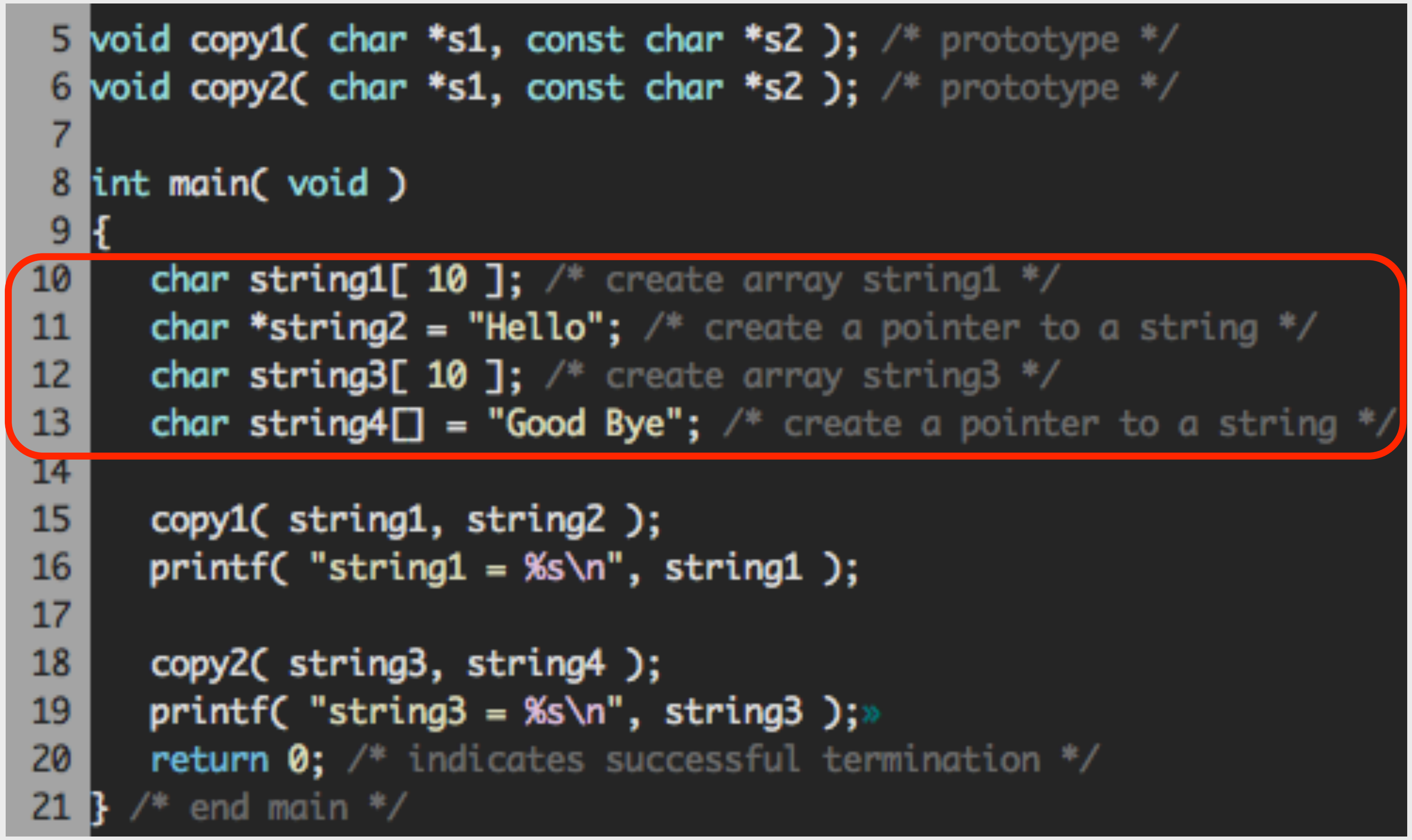
- Example: fig07_21.c



```c
void copy1( char *s1, const char *s2 ); /* prototype */
void copy2( char *s1, const char *s2 ); /* prototype */

int main( void )
{
   char string1[ 10 ]; /* create array string1 */
   char *string2 = "Hello"; /* create a pointer to a string */
   char string3[ 10 ]; /* create array string3 */
   char string4[] = "Good Bye"; /* create a pointer to a string */

   copy1( string1, string2 );
   printf( "string1 = %s\n", string1 );

   copy2( string3, string4 );
   printf( "string3 = %s\n", string3 );
   return 0; /* indicates successful termination */
} /* end main */
```

Relationship between Pointers and Arrays (Cont.)

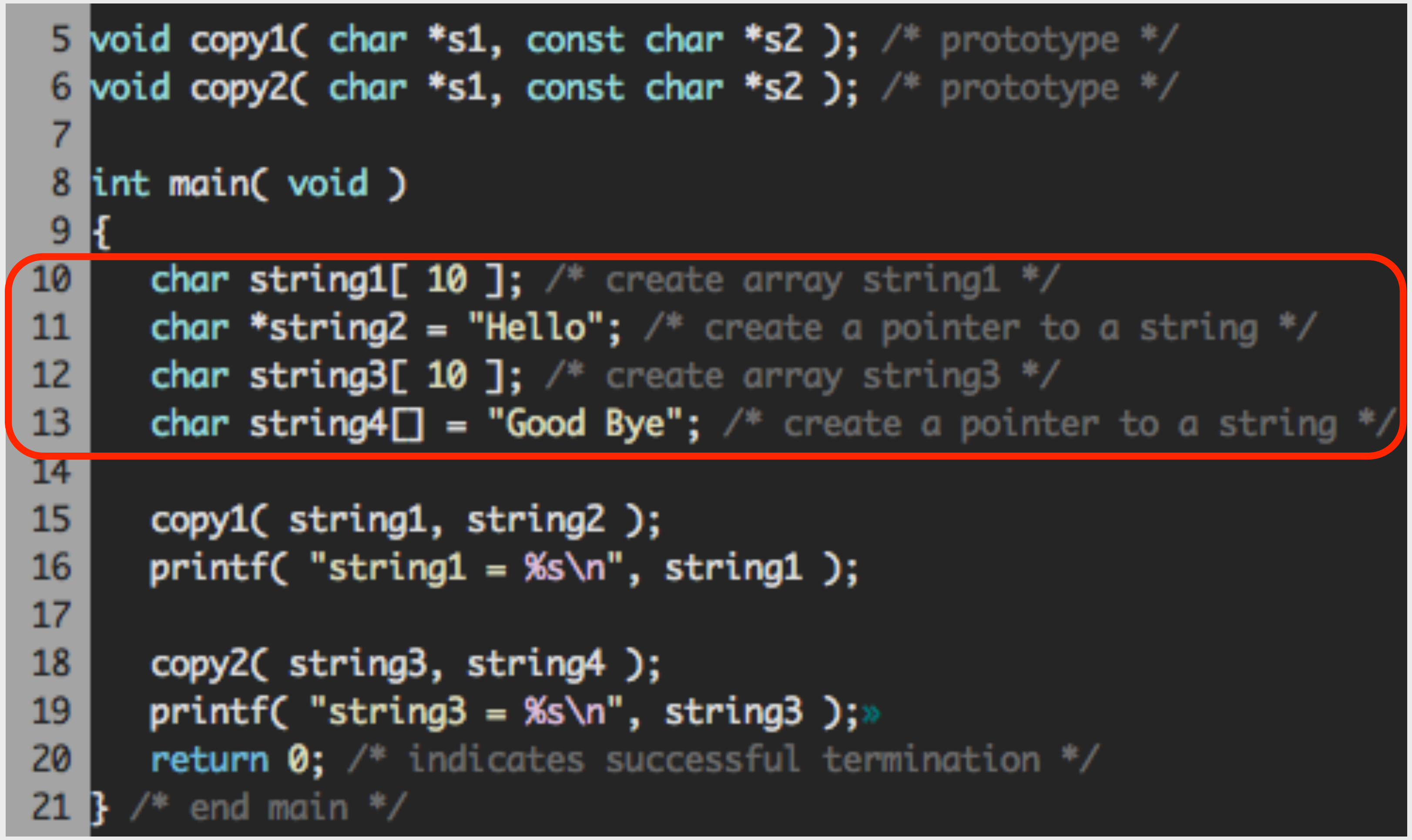
- Example: fig07_21.c



```c
void copy1( char *s1, const char *s2 ); /* prototype */
void copy2( char *s1, const char *s2 ); /* prototype */

int main( void )
{
   char string1[ 10 ]; /* create array string1 */
   char *string2 = "Hello"; /* create a pointer to a string */
   char string3[ 10 ]; /* create array string3 */
   char string4[] = "Good Bye"; /* create a pointer to a string */

   copy1( string1, string2 );
   printf( "string1 = %s\n", string1 );

   copy2( string3, string4 );
   printf( "string3 = %s\n", string3 );
   return 0; /* indicates successful termination */
} /* end main */
```

Relationship between Pointers and Arrays (Cont.)

- Example: fig07_21.c

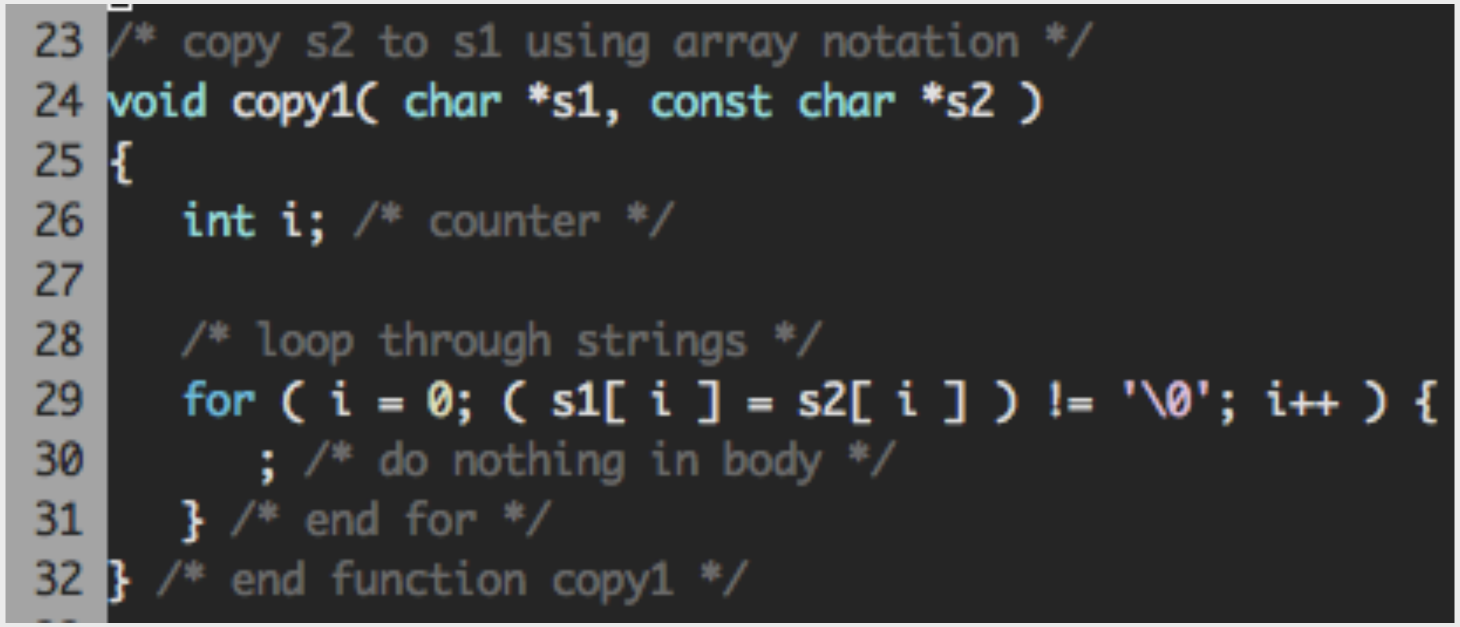


```
void copy1( char *s1, const char *s2 ); /* prototype */
void copy2( char *s1, const char *s2 ); /* prototype */

int main( void )
{
   char string1[ 10 ]; /* create array string1 */
   char *string2 = "Hello"; /* create a pointer to a string */
   char string3[ 10 ]; /* create array string3 */
   char string4[] = "Good Bye"; /* create a pointer to a string */

   copy1( string1, string2 );
   printf( "string1 = %s\n", string1 );

   copy2( string3, string4 );
   printf( "string3 = %s\n", string3 );
   return 0; /* indicates successful termination */
} /* end main */
```

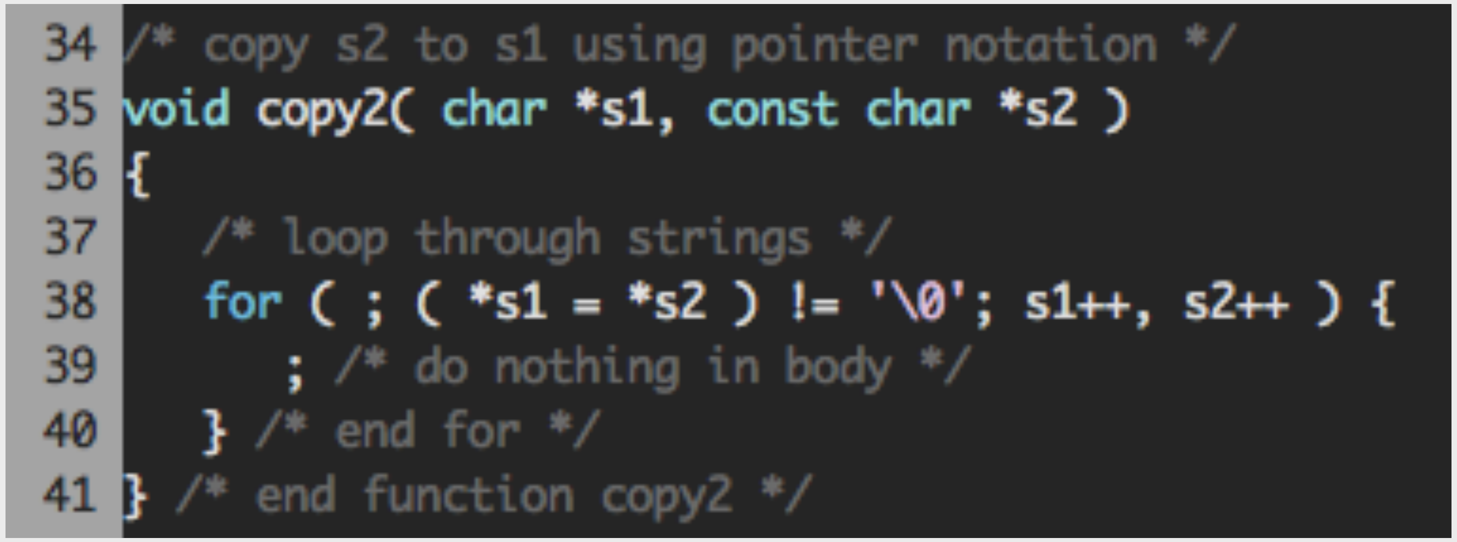
declare four strings

Relationship between Pointers and Arrays (Cont.)

- Example: fig07_21.c



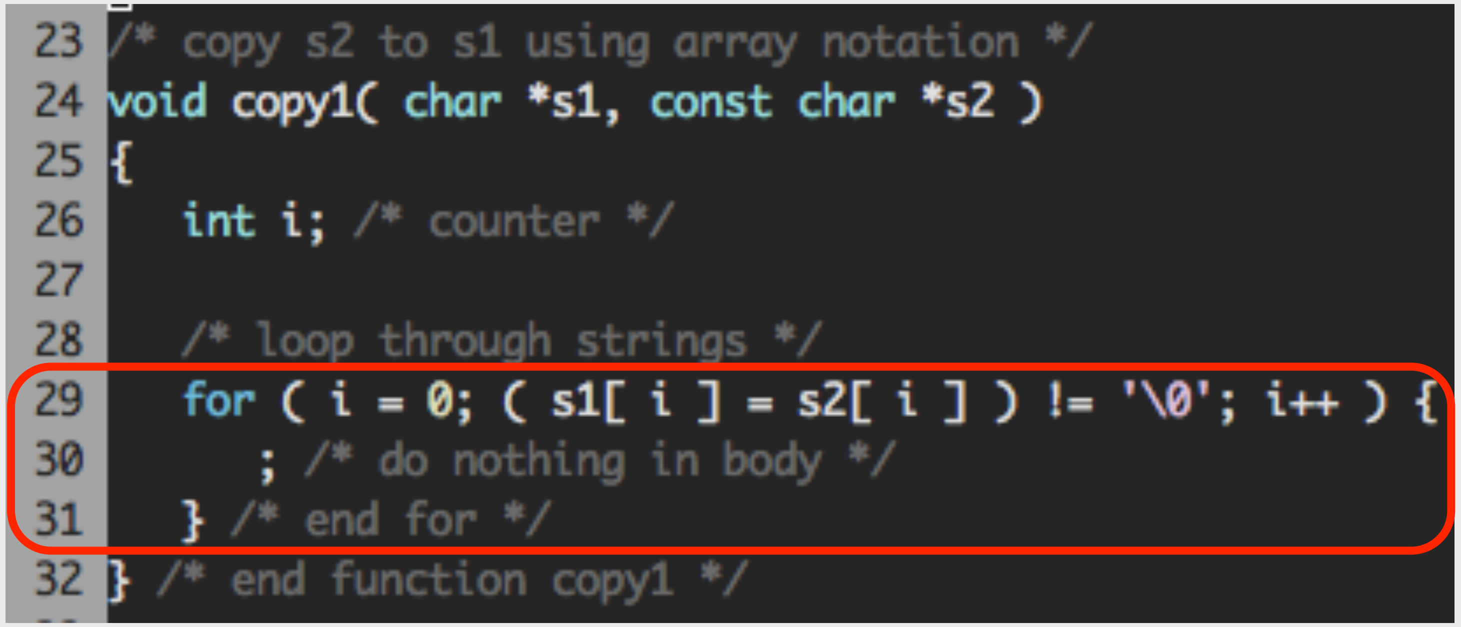
```c
/* copy s2 to s1 using array notation */
void copy1( char *s1, const char *s2 )
{
   int i; /* counter */

   /* loop through strings */
   for ( i = 0; ( s1[ i ] = s2[ i ] ) != '\0'; i++ ) {
      ; /* do nothing in body */
   } /* end for */
} /* end function copy1 */
```



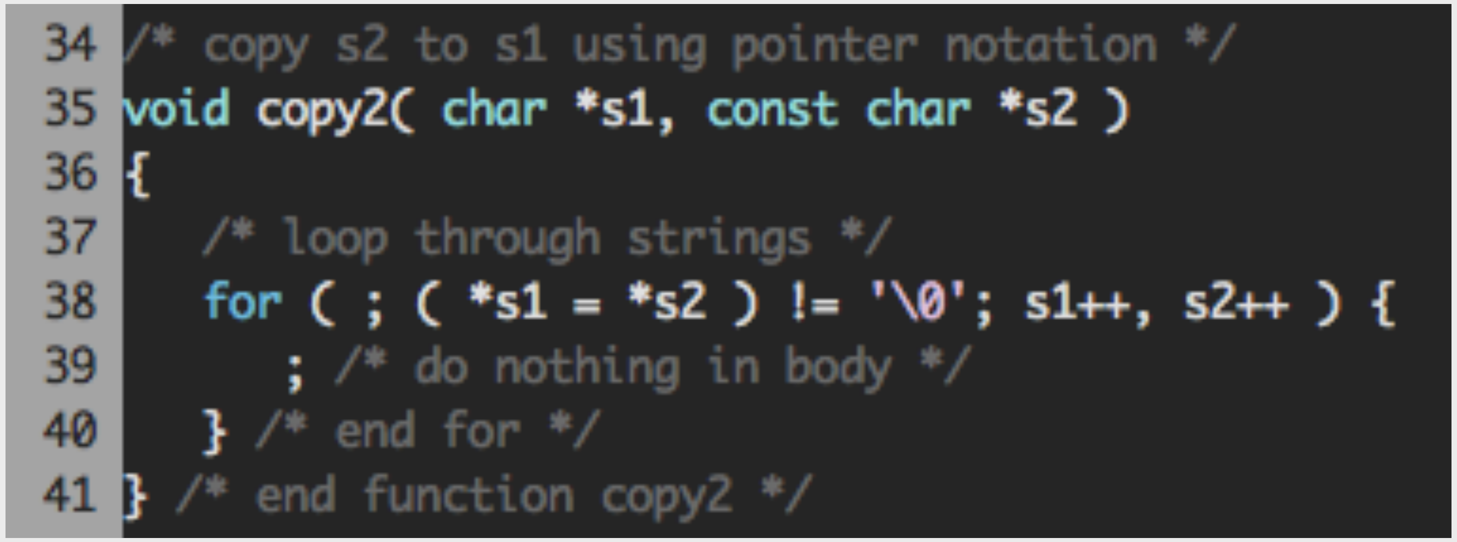
```c
/* copy s2 to s1 using pointer notation */
void copy2( char *s1, const char *s2 )
{
   /* loop through strings */
   for ( ; ( *s1 = *s2 ) != '\0'; s1++, s2++ ) {
      ; /* do nothing in body */
   } /* end for */
} /* end function copy2 */
```

Relationship between Pointers and Arrays (Cont.)

- Example: fig07_21.c



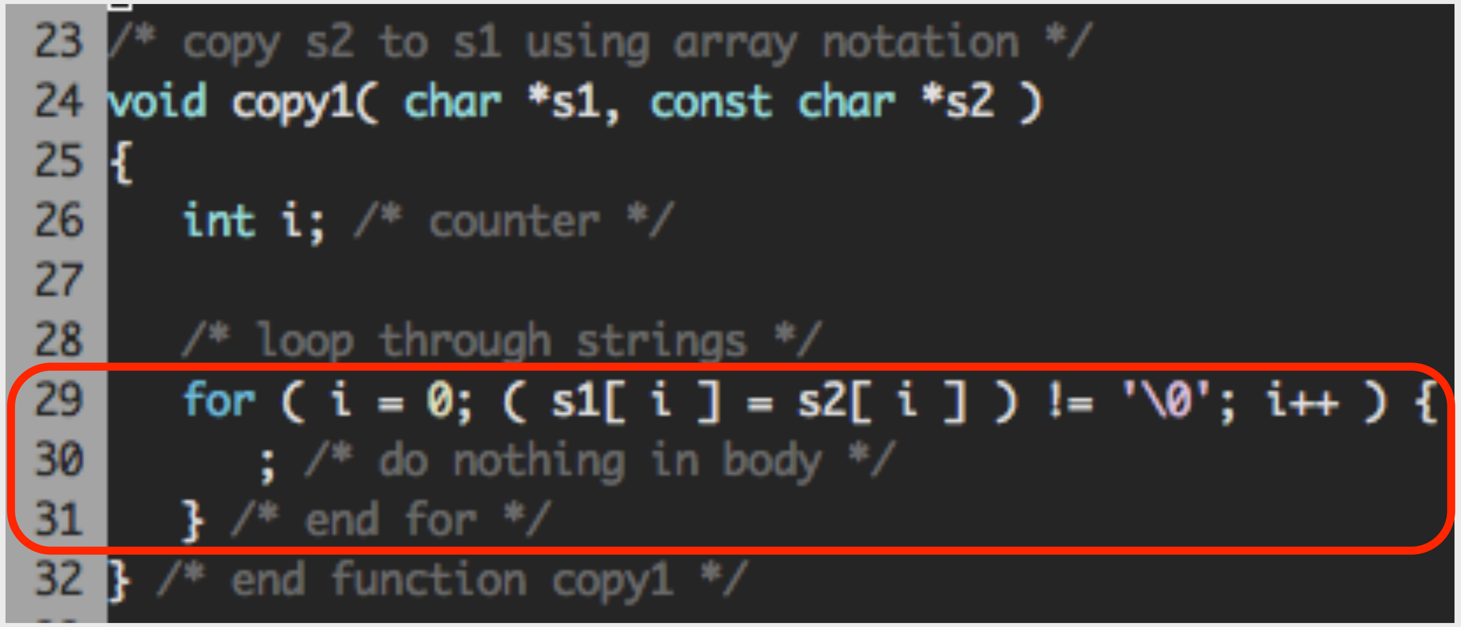
```
/* copy s2 to s1 using array notation */
void copy1( char *s1, const char *s2 )
{
   int i; /* counter */

   /* loop through strings */
   for ( i = 0; ( s1[ i ] = s2[ i ] ) != '\0'; i++ ) {
      ; /* do nothing in body */
   } /* end for */
} /* end function copy1 */
```



```
/* copy s2 to s1 using pointer notation */
void copy2( char *s1, const char *s2 )
{
   /* loop through strings */
   for ( ; ( *s1 = *s2 ) != '\0'; s1++, s2++ ) {
      ; /* do nothing in body */
   } /* end for */
} /* end function copy2 */
```

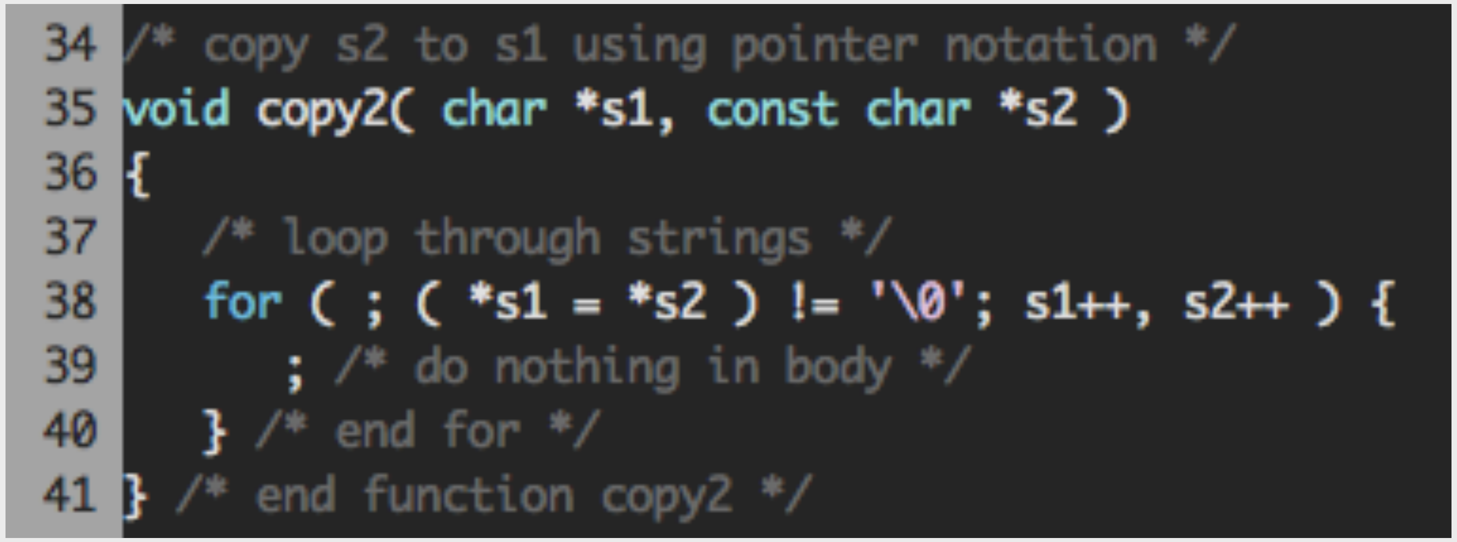
Relationship between Pointers and Arrays (Cont.)

- Example: fig07_21.c



```c
/* copy s2 to s1 using array notation */
void copy1( char *s1, const char *s2 )
{
   int i; /* counter */

   /* loop through strings */
   for ( i = 0; ( s1[ i ] = s2[ i ] ) != '\0'; i++ ) {
      ; /* do nothing in body */
   } /* end for */
} /* end function copy1 */
```

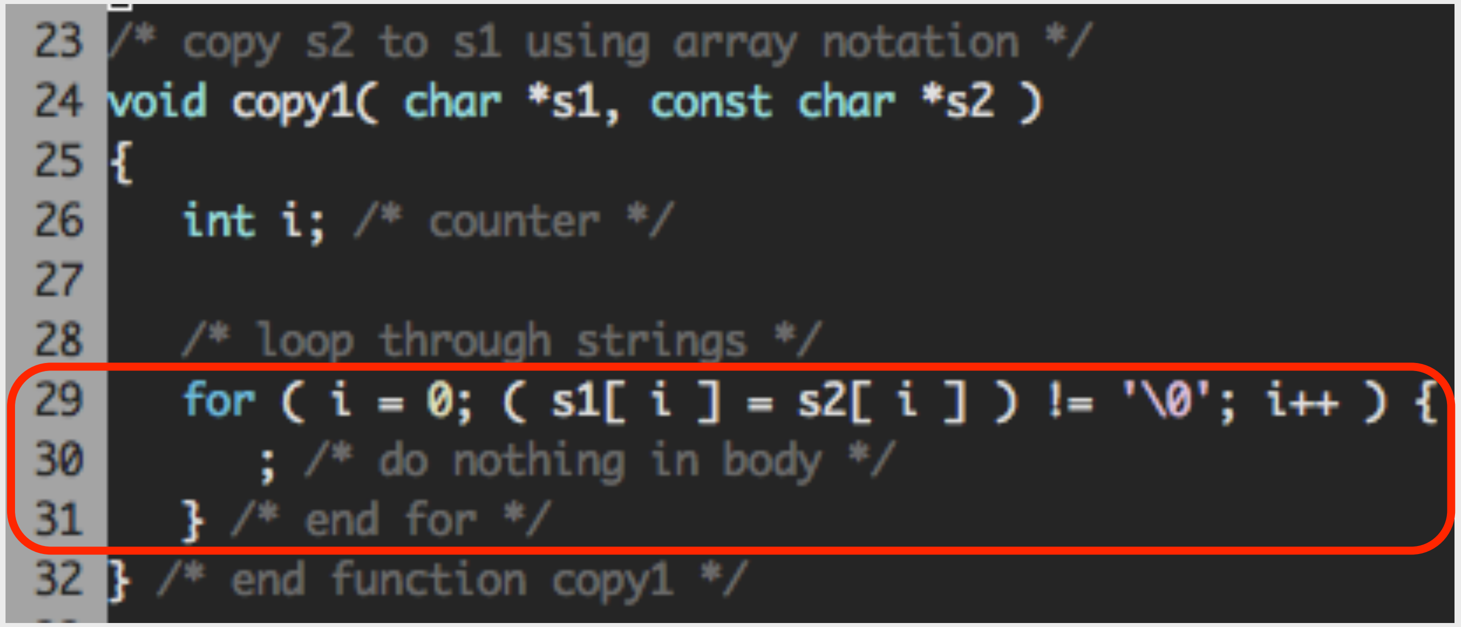
use array subscription to copy the string



```c
/* copy s2 to s1 using pointer notation */
void copy2( char *s1, const char *s2 )
{
   /* loop through strings */
   for ( ; ( *s1 = *s2 ) != '\0'; s1++, s2++ ) {
      ; /* do nothing in body */
   } /* end for */
} /* end function copy2 */
```

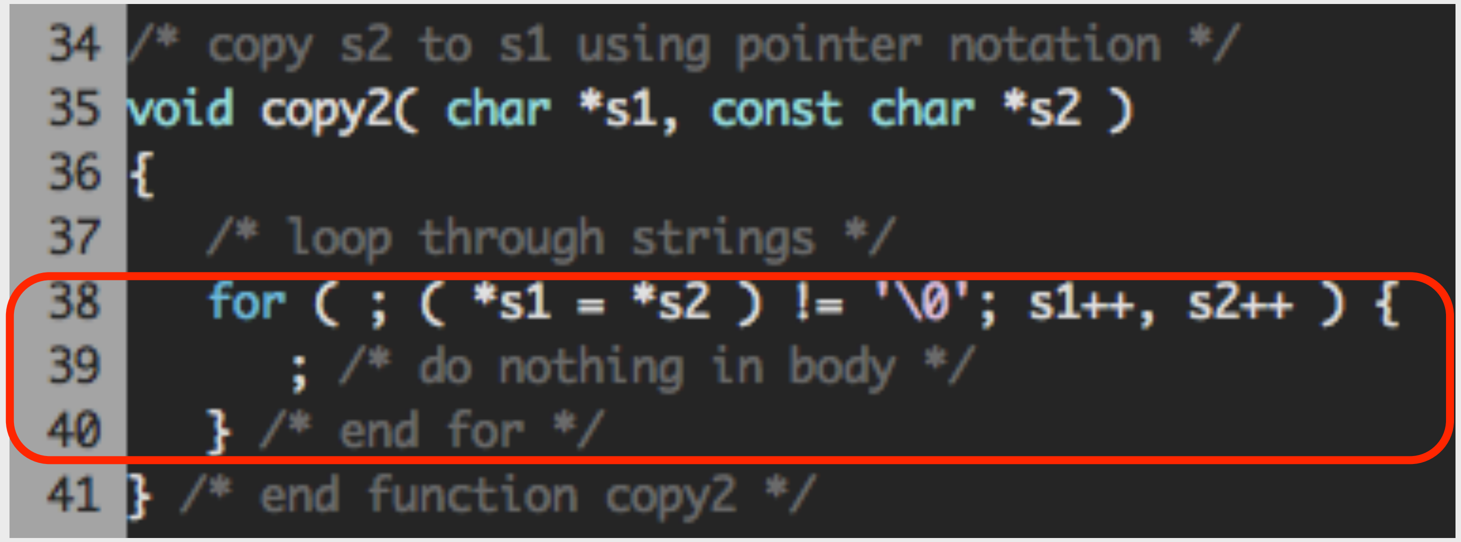
Relationship between Pointers and Arrays (Cont.)

- Example: fig07_21.c



```c
/* copy s2 to s1 using array notation */
void copy1( char *s1, const char *s2 )
{
   int i; /* counter */

   /* loop through strings */
   for ( i = 0; ( s1[ i ] = s2[ i ] ) != '\0'; i++ ) {
      ; /* do nothing in body */
   } /* end for */
} /* end function copy1 */
```

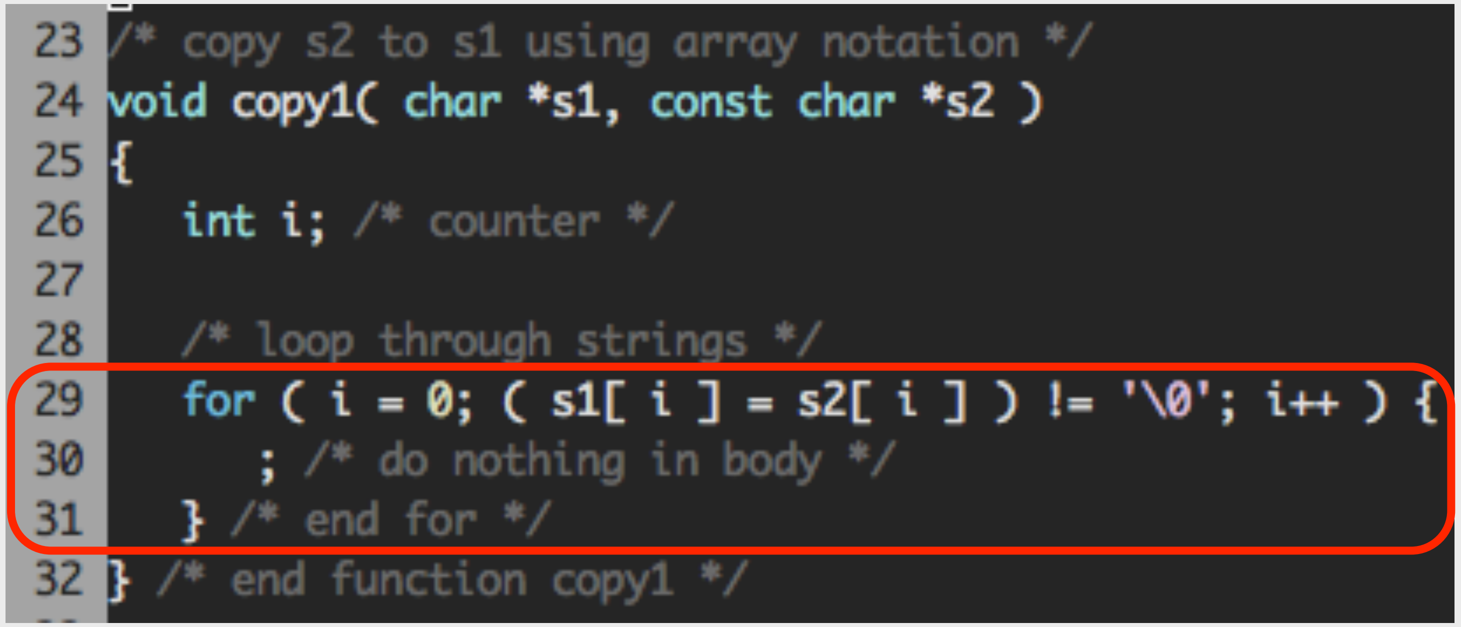
use array subscription to copy the string



```c
/* copy s2 to s1 using pointer notation */
void copy2( char *s1, const char *s2 )
{
   /* loop through strings */
   for ( ; ( *s1 = *s2 ) != '\0'; s1++, s2++ ) {
      ; /* do nothing in body */
   } /* end for */
} /* end function copy2 */
```

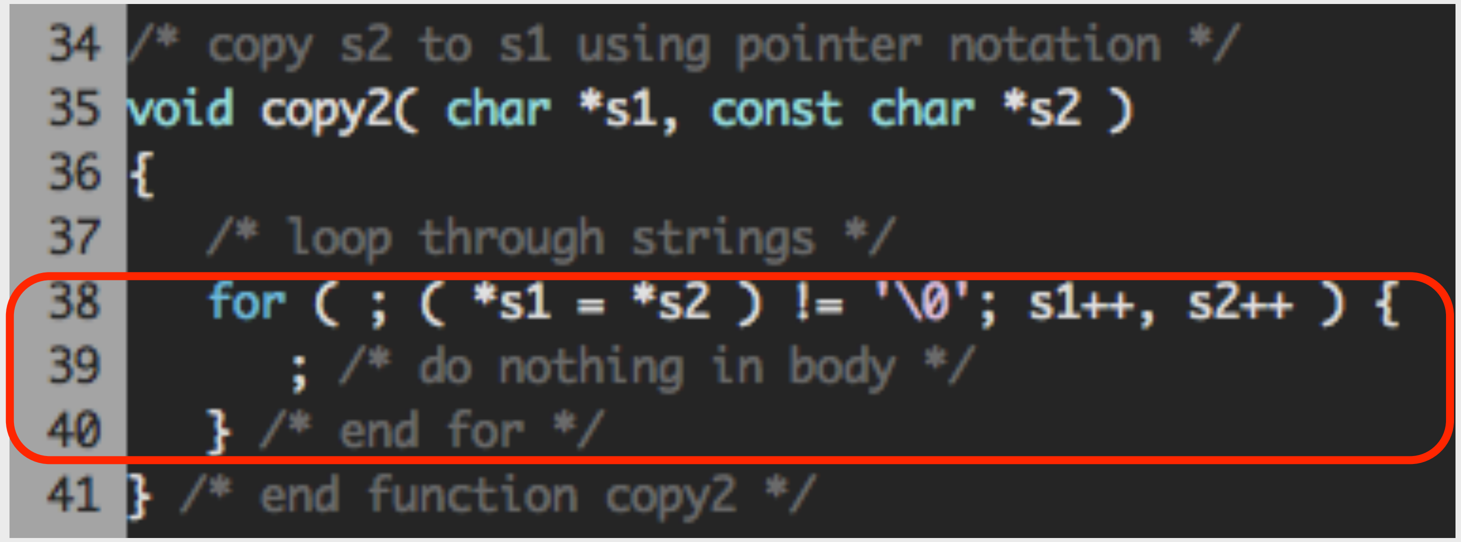
Relationship between Pointers and Arrays (Cont.)

- Example: fig07_21.c



```
/* copy s2 to s1 using array notation */
void copy1( char *s1, const char *s2 )
{
   int i; /* counter */

   /* loop through strings */
   for ( i = 0; ( s1[ i ] = s2[ i ] ) != '\0'; i++ ) {
      ; /* do nothing in body */
   } /* end for */
} /* end function copy1 */
```

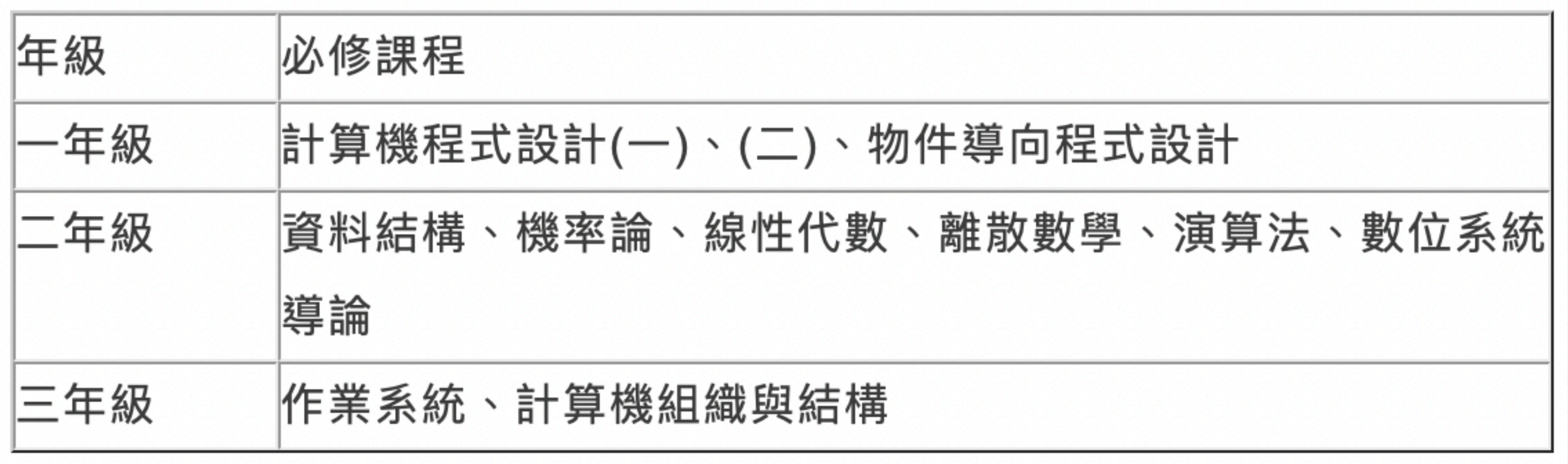
use array subscription to copy the string



```
/* copy s2 to s1 using pointer notation */
void copy2( char *s1, const char *s2 )
{
   /* loop through strings */
   for ( ; ( *s1 = *s2 ) != '\0'; s1++, s2++ ) {
      ; /* do nothing in body */
   } /* end for */
} /* end function copy2 */
```

use pointer offset to copy the string

國立政治大學資訊科學系
必修課程成績優異獎勵辦法



年級	必修課程
一年級	計算機程式設計(一)、(二)、物件導向程式設計
二年級	資料結構、機率論、線性代數、離散數學、演算法、數位系統導論
三年級	作業系統、計算機組織與結構

- 本獎學金評審原則及核發人數如下：
    - 由上述表列獎勵課程之授課教師依該班修課成績前 5 名為原則，推薦 3 名並予以排序後送至行政暨學生事務組審核。
    - 每班給予獎學金 2 名、依排序核撥 10,000 及5,000元。
    - 每班依該科修課成績前 3 名於次學期網頁公告成績優異名單。

https://www.cs.nccu.edu.tw/web/winning/winning.jsp