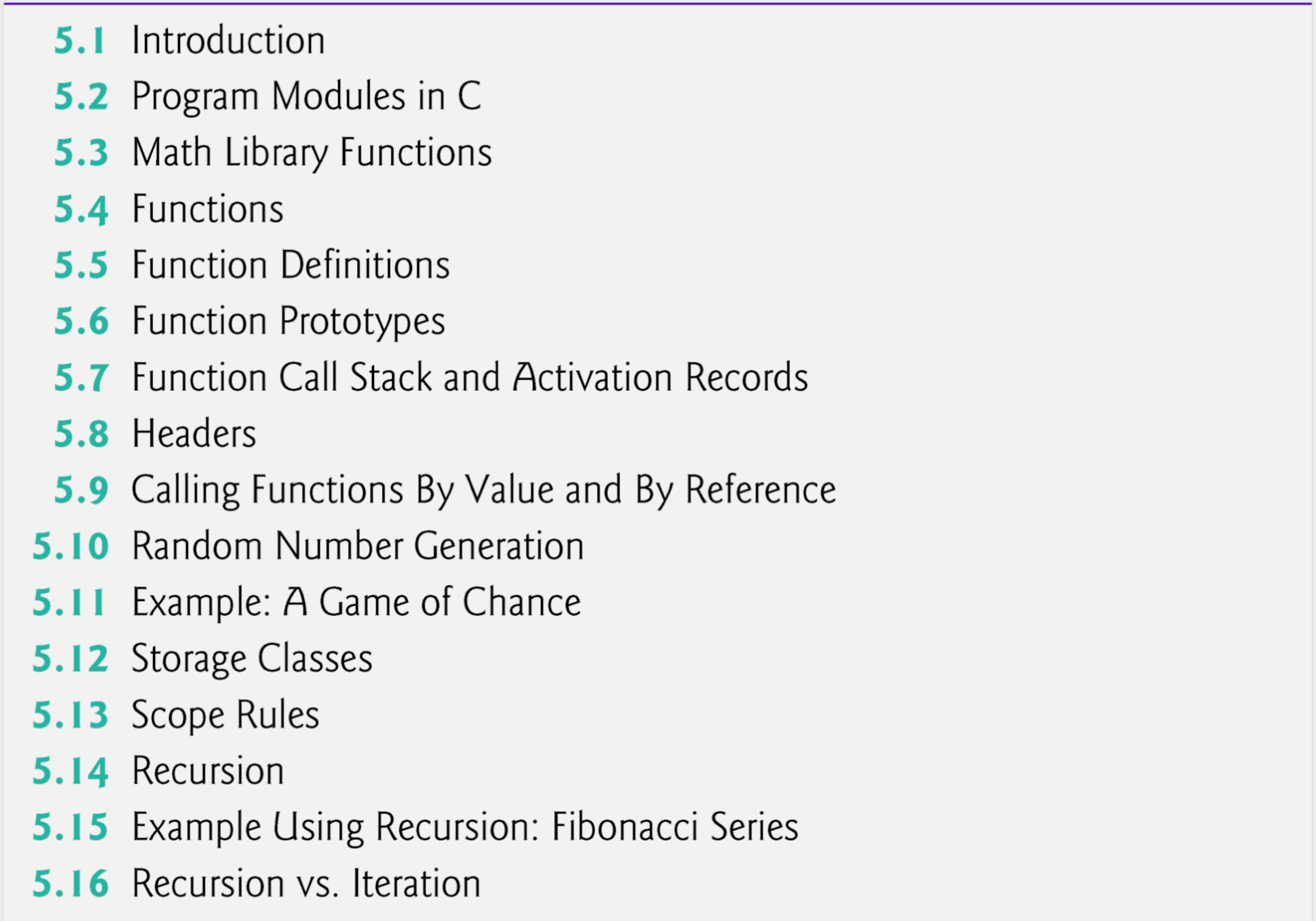
Computer Programming I

Ming-Feng Tsai (Victor Tsai)

Dept. of Computer Science
National Chengchi University

C Functions



5.1 Introduction
5.2 Program Modules in C
5.3 Math Library Functions
5.4 Functions
5.5 Function Definitions
5.6 Function Prototypes
5.7 Function Call Stack and Activation Records
5.8 Headers
5.9 Calling Functions By Value and By Reference
5.10 Random Number Generation
5.11 Example: A Game of Chance
5.12 Storage Classes
5.13 Scope Rules
5.14 Recursion
5.15 Example Using Recursion: Fibonacci Series
5.16 Recursion vs. Iteration

Call Functions By Value and By Reference

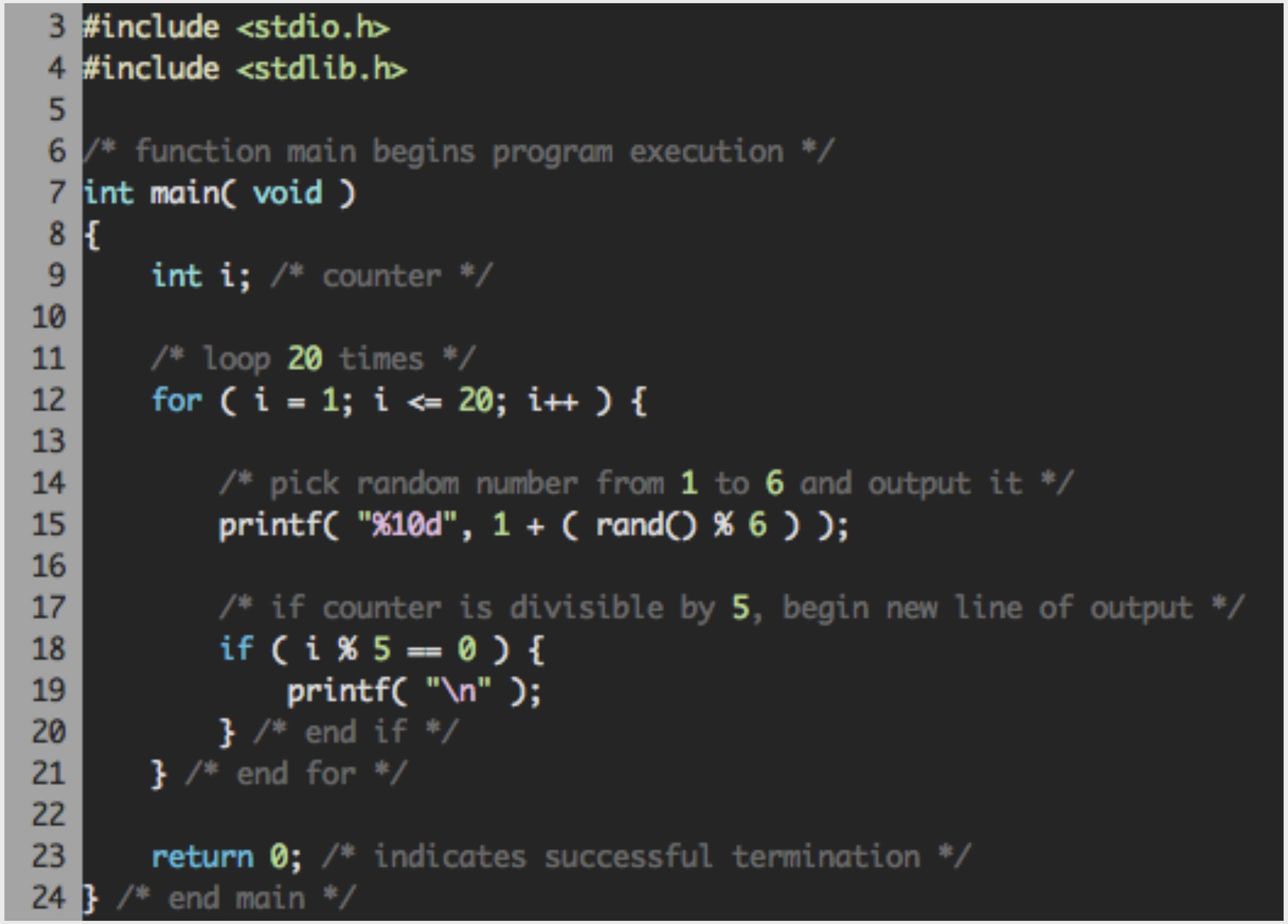
- There are two ways to invoke functions in many programming languages:
- **Call-by-Value**
  - A copy of the argument's value is made and passed to the called function
  - Changes to the copy do not affect an original variable's value in the caller.
- **Call-by-Reference**
  - The caller allows the called function to modify the original variable's value.

Call Functions By Value and By Reference (Cont.)

- We have to wait until Chapter 7 for a full understanding of the call-by-reference
- For now, we concentrate on call-by-value

Random Number Generation

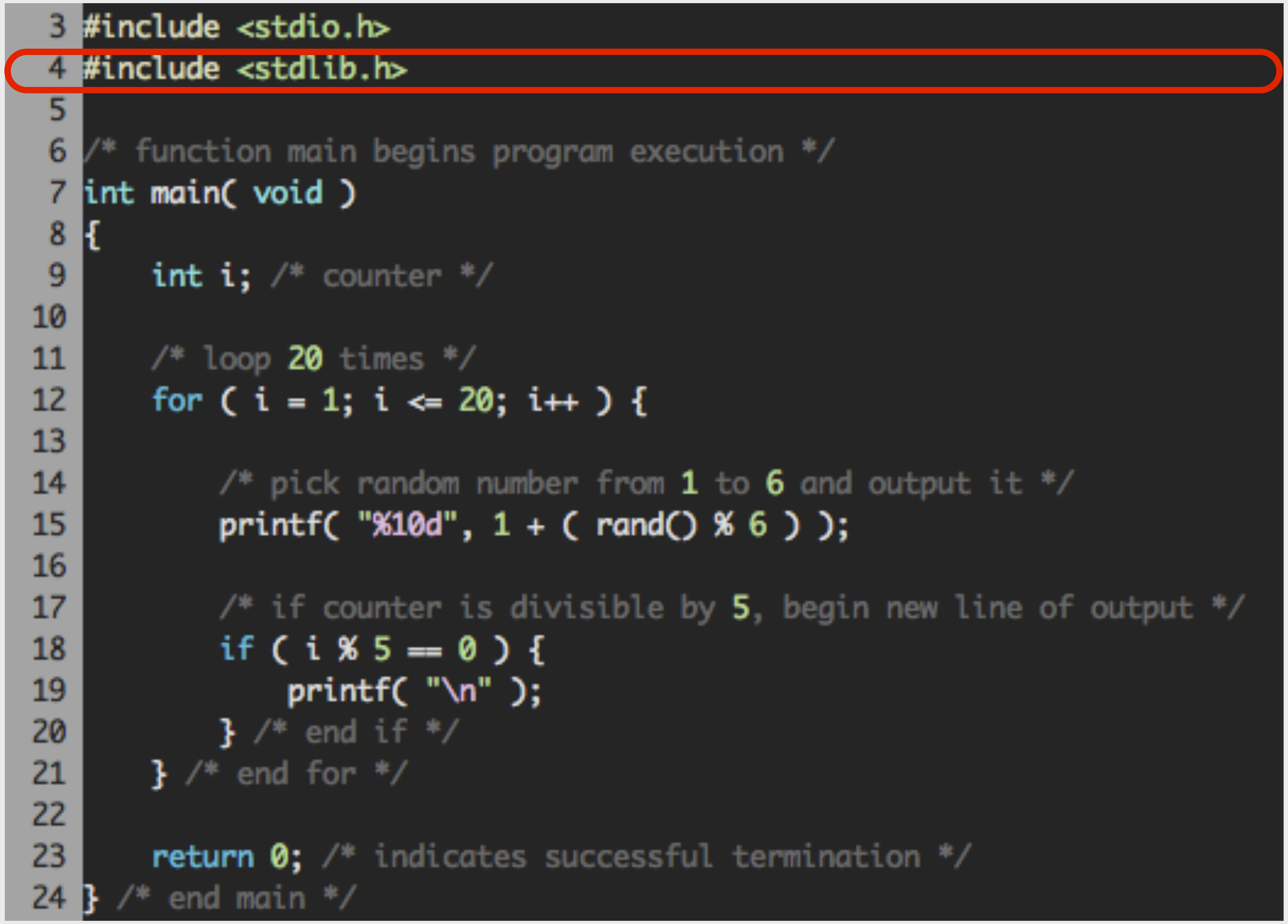
- Example: fig05_07.c



```c
#include <stdio.h>
#include <stdlib.h>

/* function main begins program execution */
int main( void )
{
    int i; /* counter */

    /* loop 20 times */
    for ( i = 1; i <= 20; i++ ) {

        /* pick random number from 1 to 6 and output it */
        printf( "%10d", 1 + ( rand() % 6 ) );

        /* if counter is divisible by 5, begin new line of output */
        if ( i % 5 == 0 ) {
            printf( "\n" );
        } /* end if */
    } /* end for */

    return 0; /* indicates successful termination */
} /* end main */
```

Random Number Generation

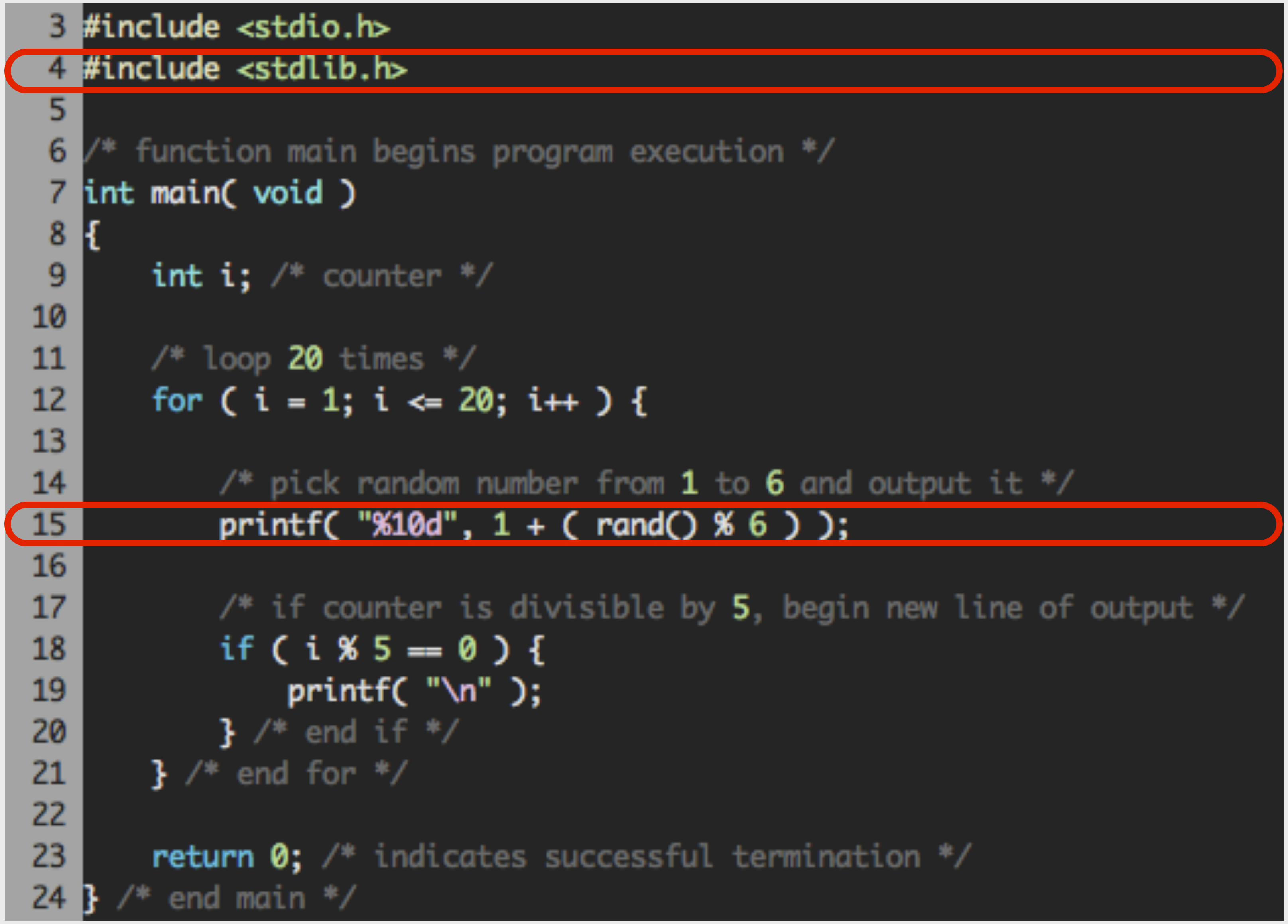
- Example: fig05_07.c



```c
#include <stdio.h>
#include <stdlib.h>

/* function main begins program execution */
int main( void )
{
    int i; /* counter */

    /* loop 20 times */
    for ( i = 1; i <= 20; i++ ) {

        /* pick random number from 1 to 6 and output it */
        printf( "%10d", 1 + ( rand() % 6 ) );

        /* if counter is divisible by 5, begin new line of output */
        if ( i % 5 == 0 ) {
            printf( "\n" );
        } /* end if */
    } /* end for */

    return 0; /* indicates successful termination */
} /* end main */
```

Random Number Generation

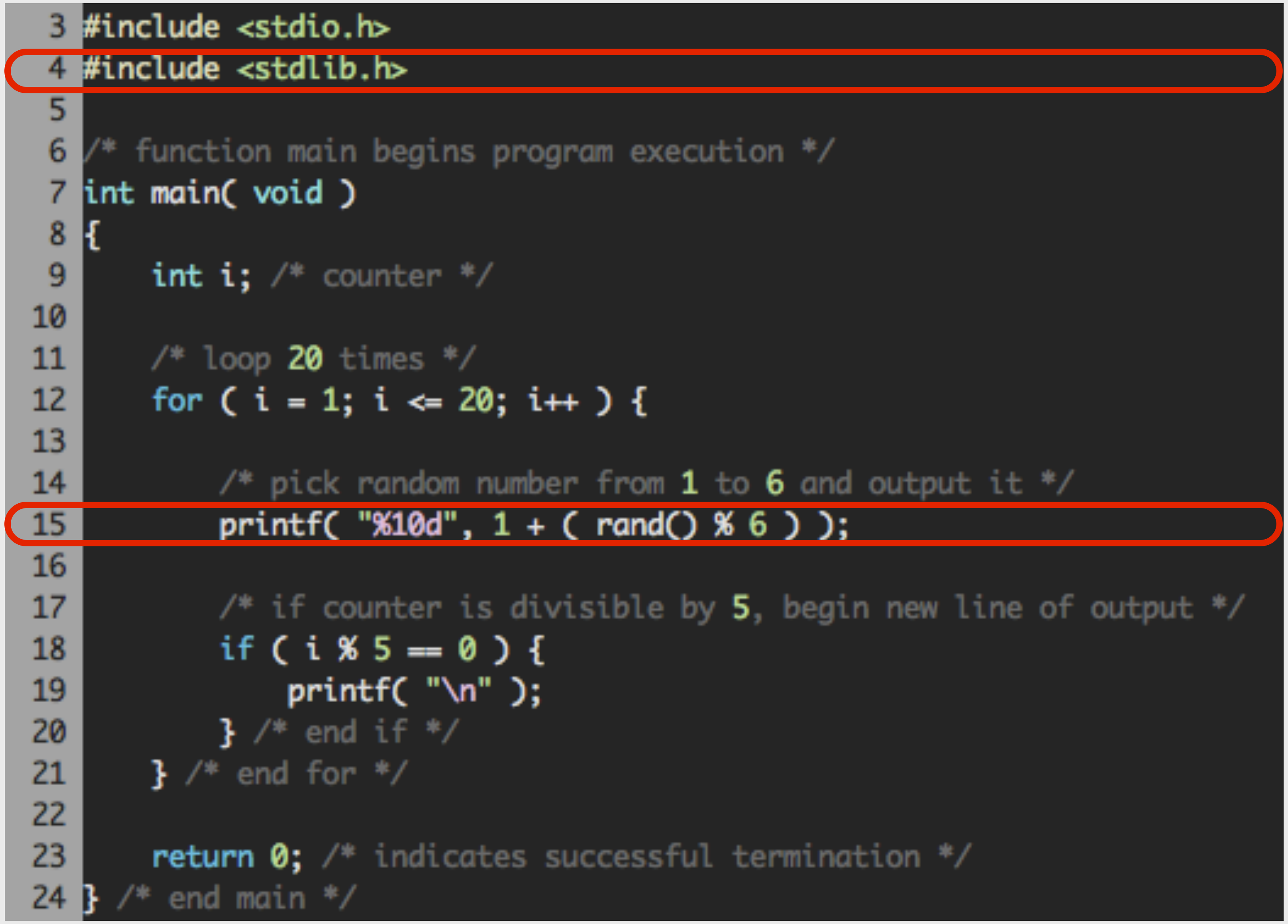
- Example: fig05_07.c



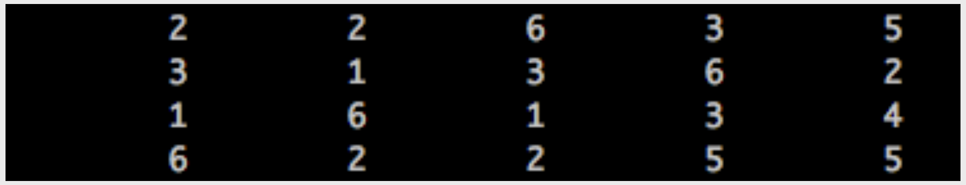
```c
#include <stdio.h>
#include <stdlib.h>

/* function main begins program execution */
int main( void )
{
    int i; /* counter */

    /* loop 20 times */
    for ( i = 1; i <= 20; i++ ) {

        /* pick random number from 1 to 6 and output it */
        printf( "%10d", 1 + ( rand() % 6 ) );

        /* if counter is divisible by 5, begin new line of output */
        if ( i % 5 == 0 ) {
            printf( "\n" );
        } /* end if */
    } /* end for */

    return 0; /* indicates successful termination */
} /* end main */
```

Random Number Generation

- Example: fig05_07.c



```c
#include <stdio.h>
#include <stdlib.h>

/* function main begins program execution */
int main( void )
{
    int i; /* counter */

    /* loop 20 times */
    for ( i = 1; i <= 20; i++ ) {

        /* pick random number from 1 to 6 and output it */
        printf( "%10d", 1 + ( rand() % 6 ) );

        /* if counter is divisible by 5, begin new line of output */
        if ( i % 5 == 0 ) {
            printf( "\n" );
        } /* end if */
    } /* end for */

    return 0; /* indicates successful termination */
} /* end main */
```



```
         2         2         6         3         5
         3         1         3         6         2
         1         6         1         3         4
         6         2         2         5         5
```

Random Number Generation (Cont.)

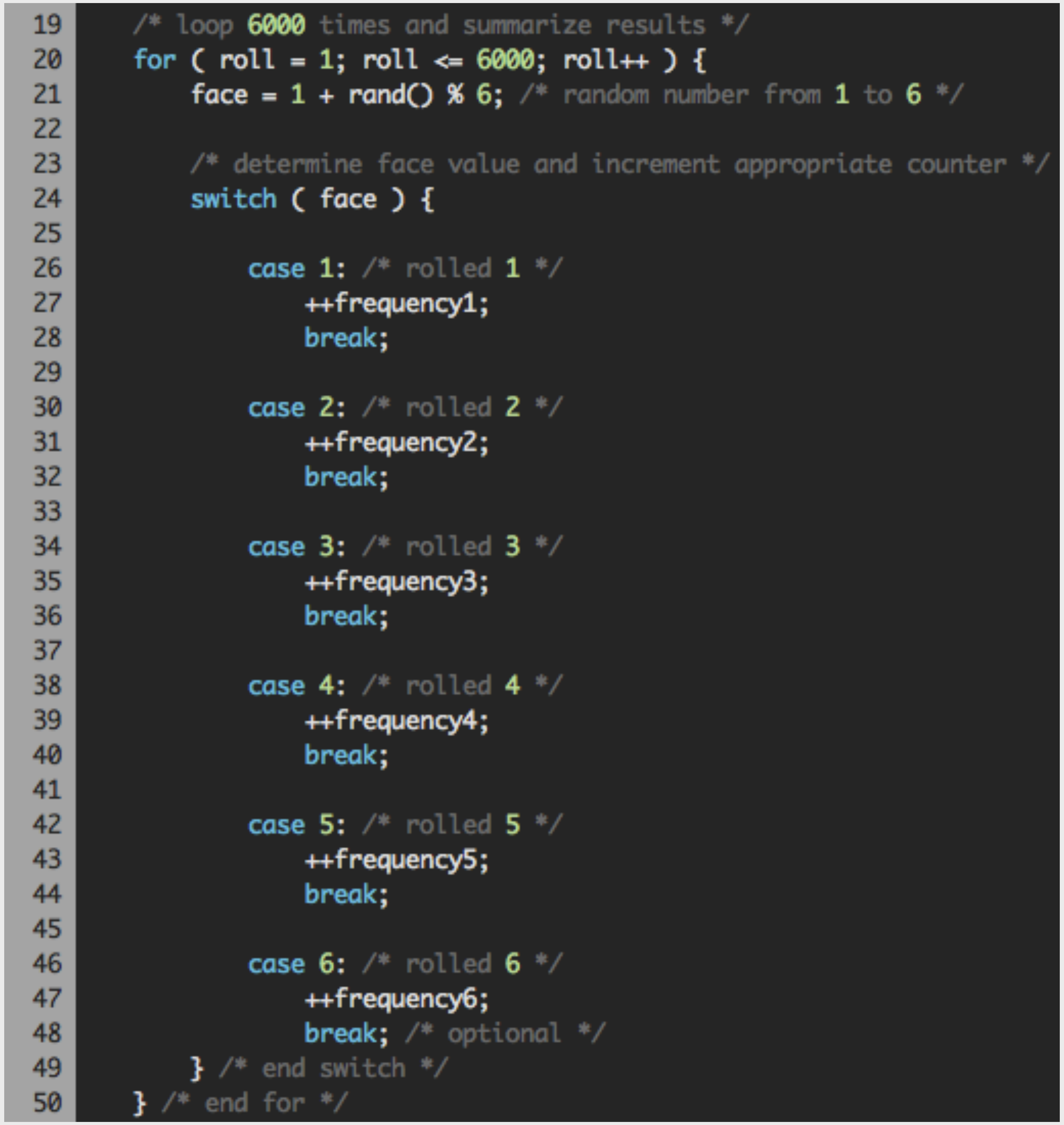
- The function prototype for function **`rand`** is in **`<stdlib.h>`**.
- The **`rand`** function generates an integer between **0 and `RAND_MAX`**
- We use the remainder operator (`%`) in conjunction with rand as follows

```
    rand() % 6
```

  - to produce integers in the **range 0 to 5**.
- This is called **scaling**.
- The number 6 is called the **scaling factor**.

Random Number Generation (Cont.)

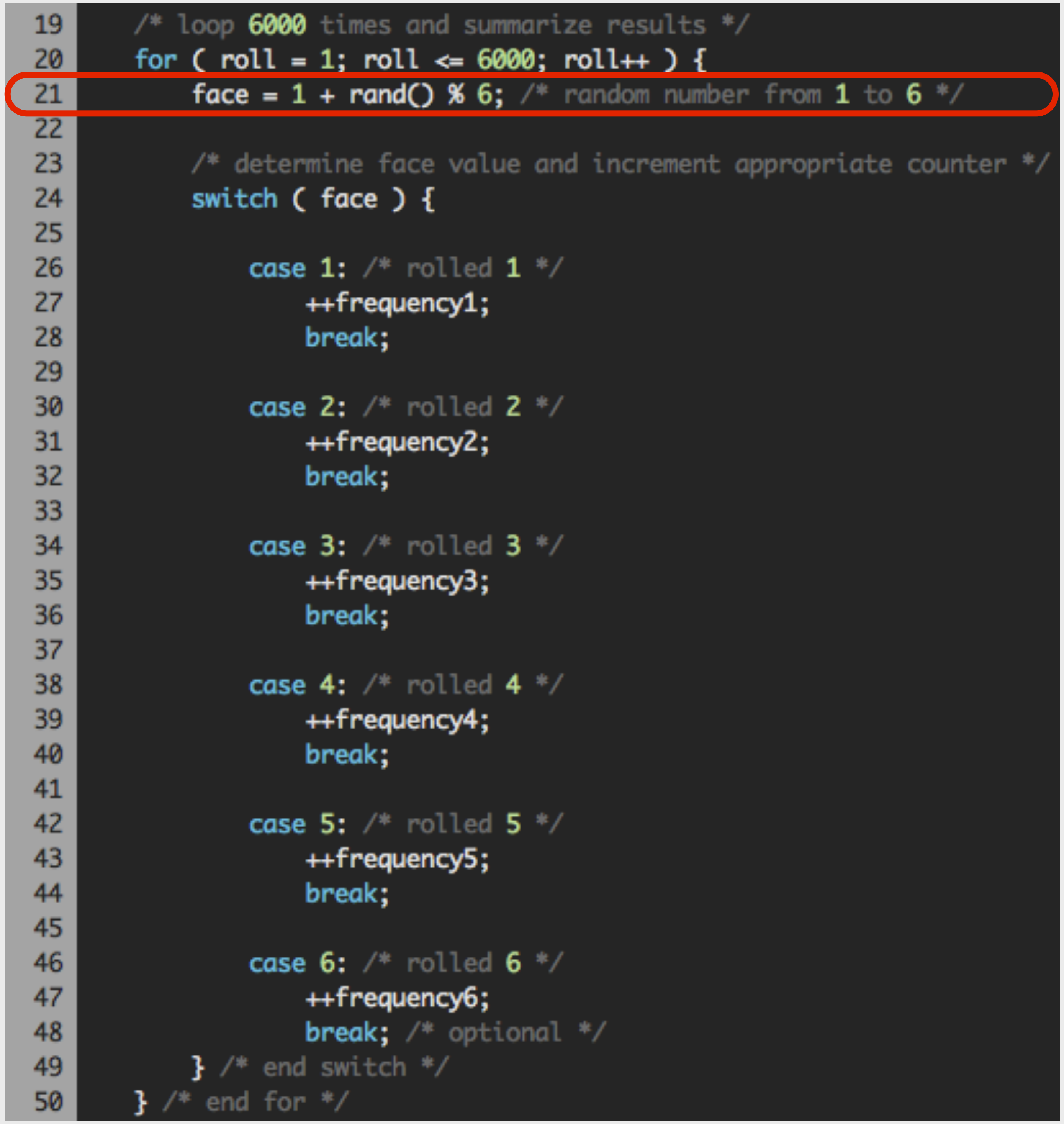
- Example: fig05_08.c



```c
    /* loop 6000 times and summarize results */
    for ( roll = 1; roll <= 6000; roll++ ) {
        face = 1 + rand() % 6; /* random number from 1 to 6 */

        /* determine face value and increment appropriate counter */
        switch ( face ) {

            case 1: /* rolled 1 */
                ++frequency1;
                break;

            case 2: /* rolled 2 */
                ++frequency2;
                break;

            case 3: /* rolled 3 */
                ++frequency3;
                break;

            case 4: /* rolled 4 */
                ++frequency4;
                break;

            case 5: /* rolled 5 */
                ++frequency5;
                break;

            case 6: /* rolled 6 */
                ++frequency6;
                break; /* optional */
        } /* end switch */
    } /* end for */
```

Random Number Generation (Cont.)

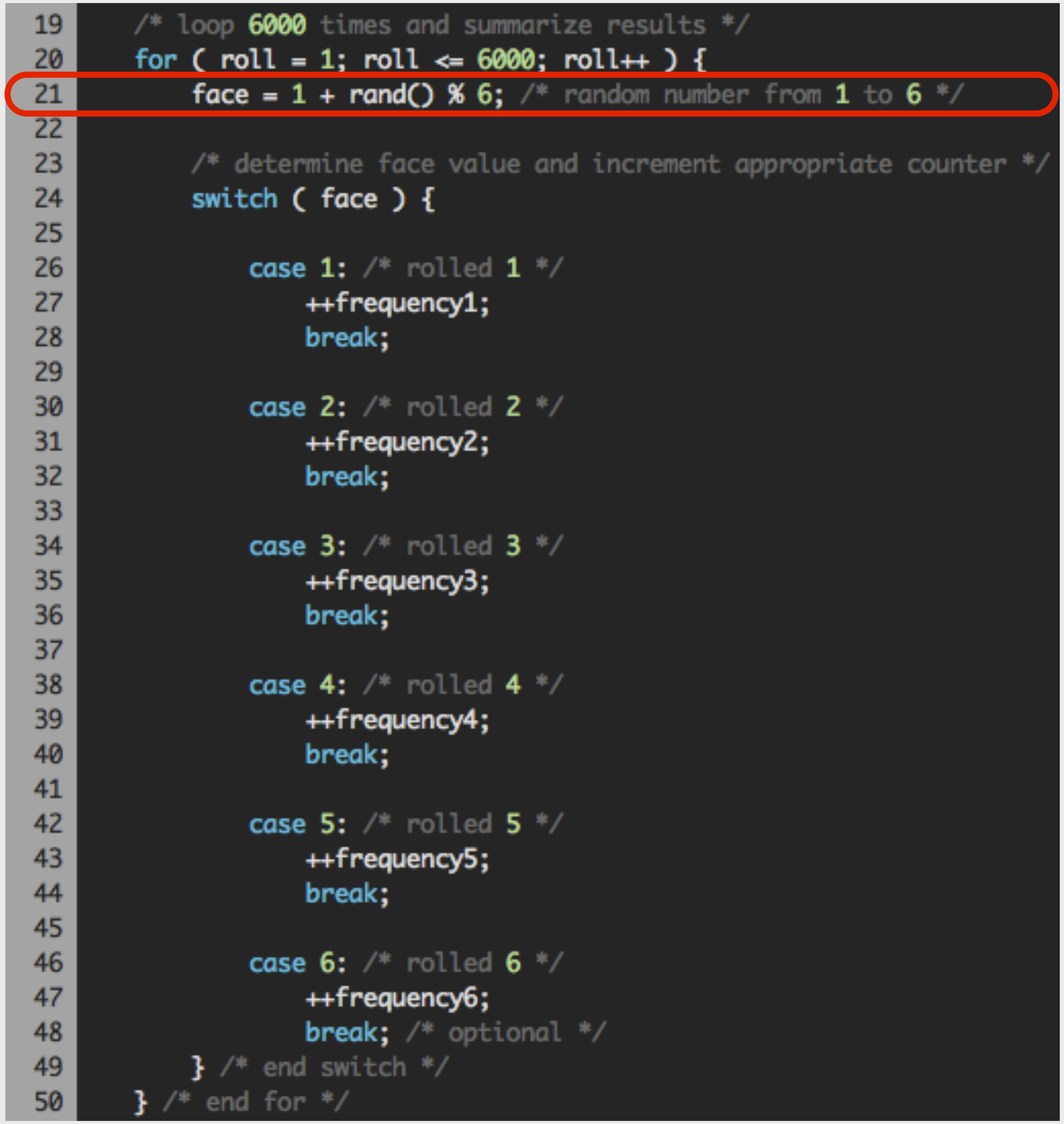
- Example: fig05_08.c



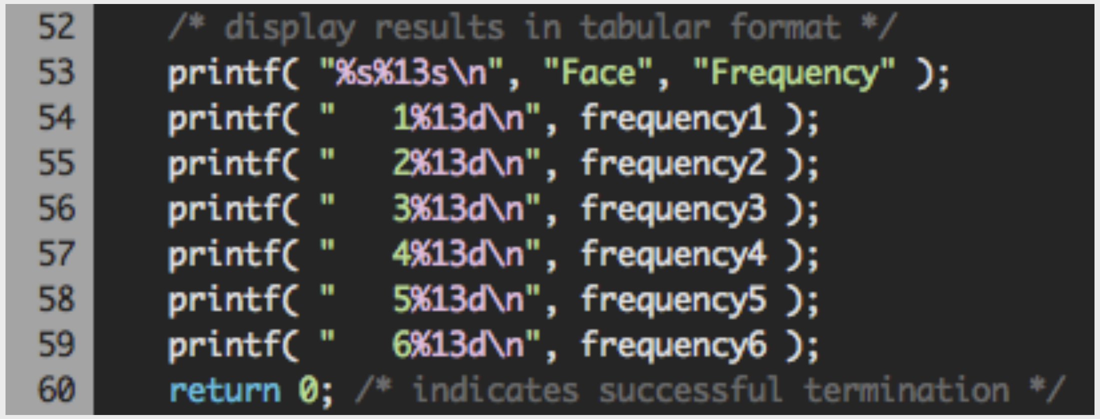
```
    /* loop 6000 times and summarize results */
    for ( roll = 1; roll <= 6000; roll++ ) {
        face = 1 + rand() % 6; /* random number from 1 to 6 */

        /* determine face value and increment appropriate counter */
        switch ( face ) {

            case 1: /* rolled 1 */
                ++frequency1;
                break;

            case 2: /* rolled 2 */
                ++frequency2;
                break;

            case 3: /* rolled 3 */
                ++frequency3;
                break;

            case 4: /* rolled 4 */
                ++frequency4;
                break;

            case 5: /* rolled 5 */
                ++frequency5;
                break;

            case 6: /* rolled 6 */
                ++frequency6;
                break; /* optional */
        } /* end switch */
    } /* end for */
```

Random Number Generation (Cont.)

- Example: fig05_08.c



```c
    /* loop 6000 times and summarize results */
    for ( roll = 1; roll <= 6000; roll++ ) {
        face = 1 + rand() % 6; /* random number from 1 to 6 */

        /* determine face value and increment appropriate counter */
        switch ( face ) {

            case 1: /* rolled 1 */
                ++frequency1;
                break;

            case 2: /* rolled 2 */
                ++frequency2;
                break;

            case 3: /* rolled 3 */
                ++frequency3;
                break;

            case 4: /* rolled 4 */
                ++frequency4;
                break;

            case 5: /* rolled 5 */
                ++frequency5;
                break;

            case 6: /* rolled 6 */
                ++frequency6;
                break; /* optional */
        } /* end switch */
    } /* end for */
```



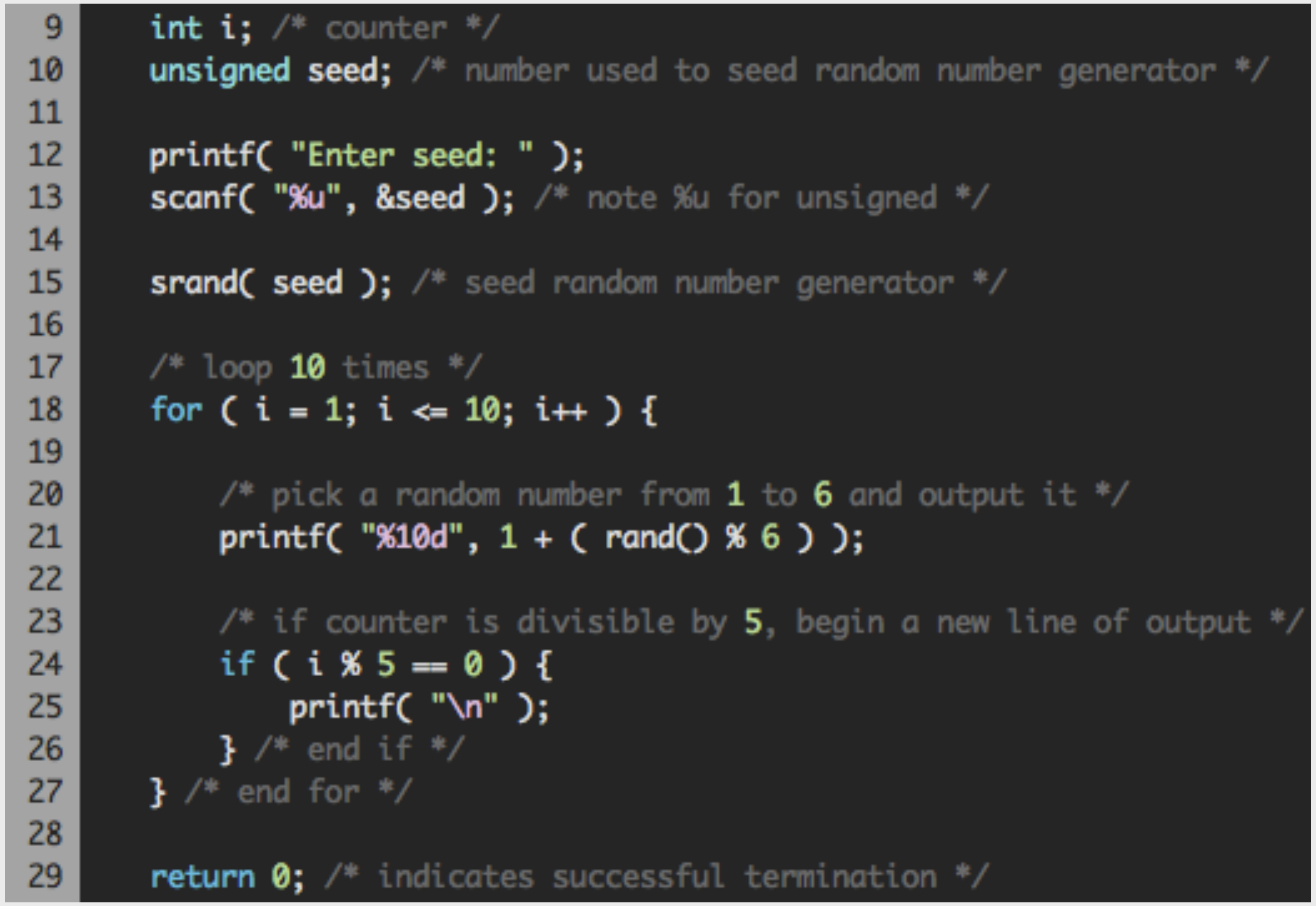
```c
    /* display results in tabular format */
    printf( "%s%13s\n", "Face", "Frequency" );
    printf( "   1%13d\n", frequency1 );
    printf( "   2%13d\n", frequency2 );
    printf( "   3%13d\n", frequency3 );
    printf( "   4%13d\n", frequency4 );
    printf( "   5%13d\n", frequency5 );
    printf( "   6%13d\n", frequency6 );
    return 0; /* indicates successful termination */
```

Random Number Generation (Cont.)

- Executing the program of fig05_07.c again produces **exactly** the same sequence of values.
- How can these be random numbers? Ironically, this repeatability is an important characteristic of function **`rand`**.
  - Calling **`rand`** repeatedly produces a sequence of numbers that appears to be random
- Another randomization is accomplished by **`srand`**

Random Number Generation (Cont.)

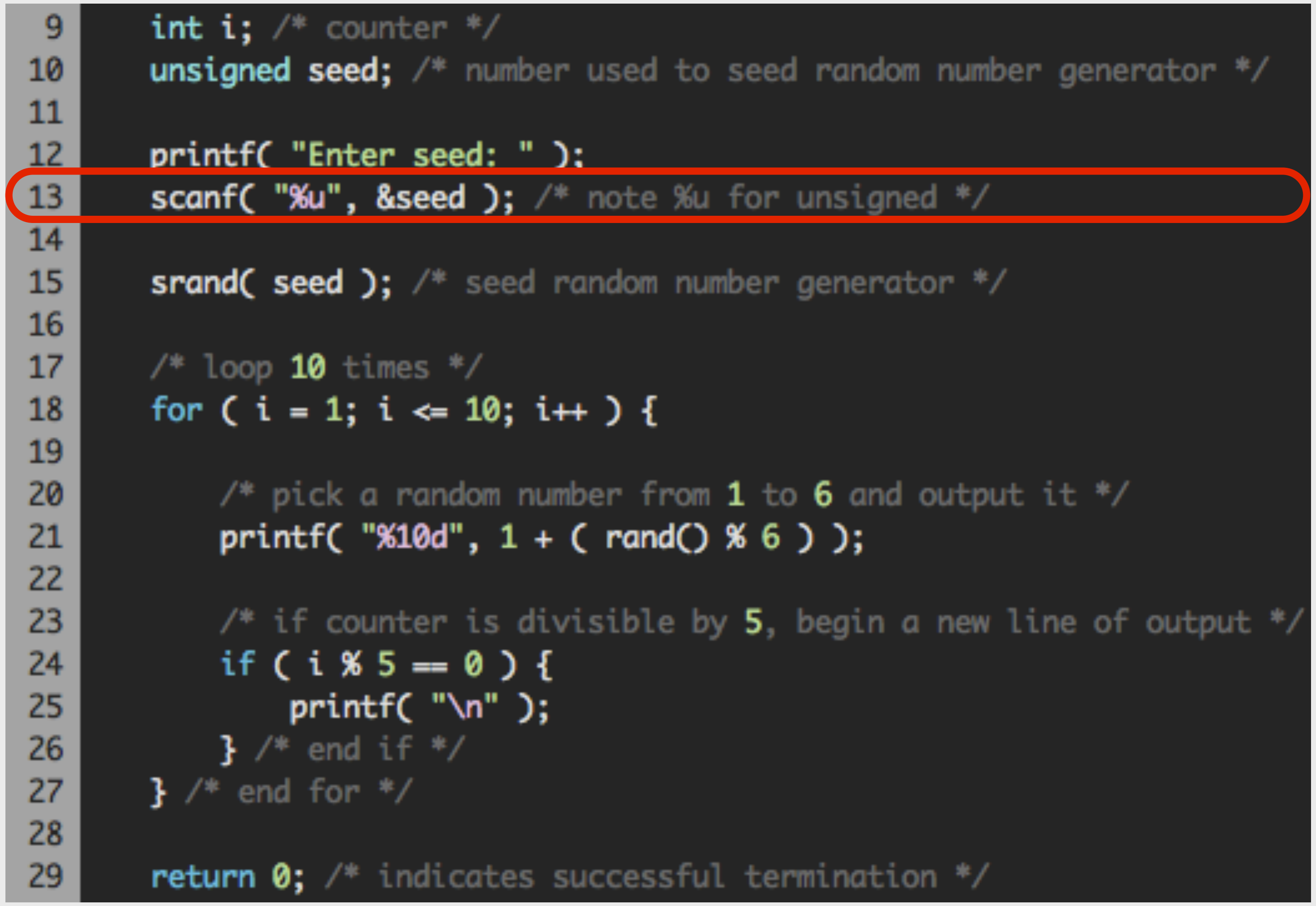
- Example: fig05_09.c



```c
    int i; /* counter */
    unsigned seed; /* number used to seed random number generator */

    printf( "Enter seed: " );
    scanf( "%u", &seed ); /* note %u for unsigned */

    srand( seed ); /* seed random number generator */

    /* loop 10 times */
    for ( i = 1; i <= 10; i++ ) {

        /* pick a random number from 1 to 6 and output it */
        printf( "%10d", 1 + ( rand() % 6 ) );

        /* if counter is divisible by 5, begin a new line of output */
        if ( i % 5 == 0 ) {
            printf( "\n" );
        } /* end if */
    } /* end for */

    return 0; /* indicates successful termination */
```

Random Number Generation (Cont.)

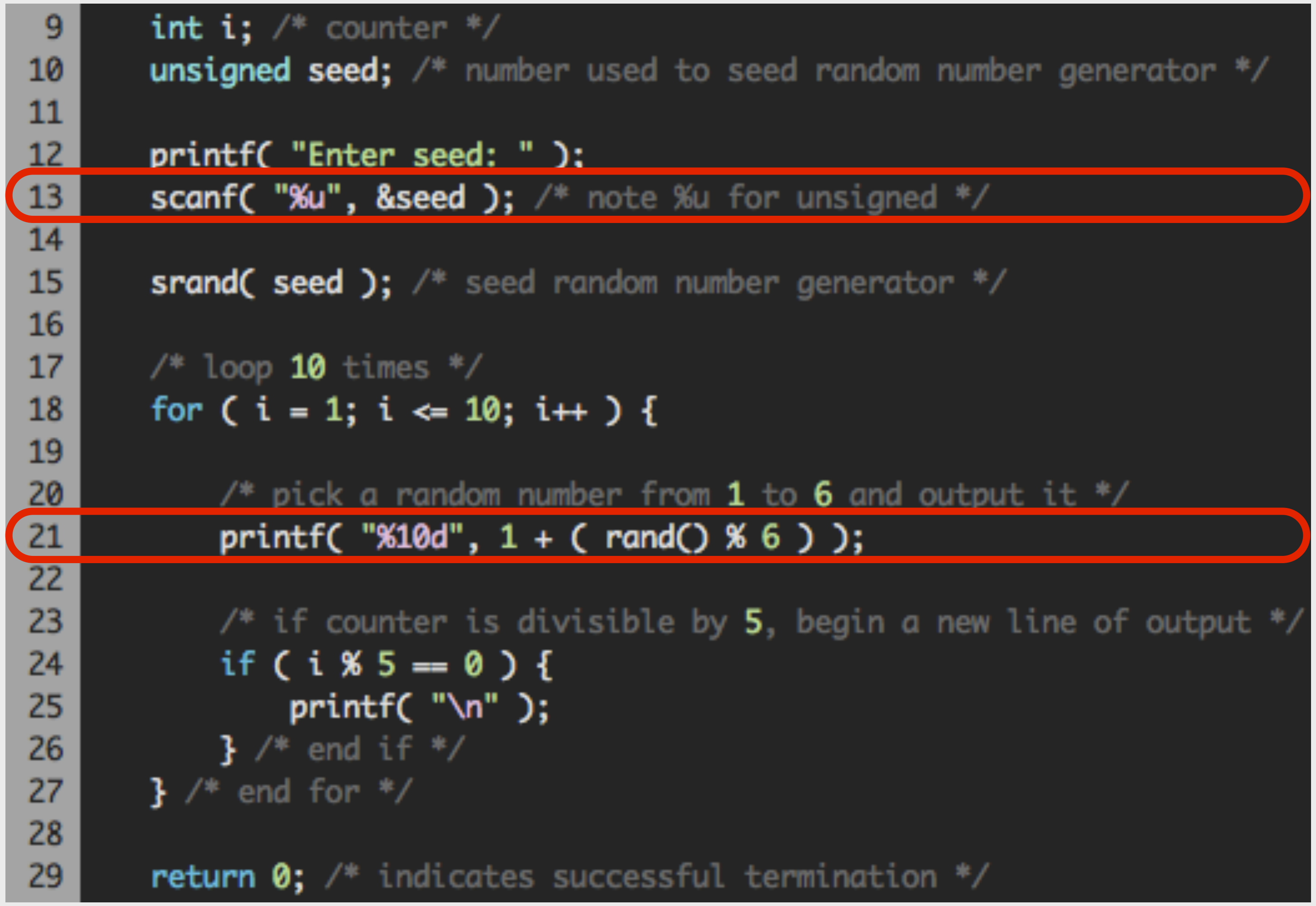
- Example: fig05_09.c



```
    int i; /* counter */
    unsigned seed; /* number used to seed random number generator */

    printf( "Enter seed: " );
    scanf( "%u", &seed ); /* note %u for unsigned */

    srand( seed ); /* seed random number generator */

    /* loop 10 times */
    for ( i = 1; i <= 10; i++ ) {

        /* pick a random number from 1 to 6 and output it */
        printf( "%10d", 1 + ( rand() % 6 ) );

        /* if counter is divisible by 5, begin a new line of output */
        if ( i % 5 == 0 ) {
            printf( "\n" );
        } /* end if */
    } /* end for */

    return 0; /* indicates successful termination */
```

Random Number Generation (Cont.)

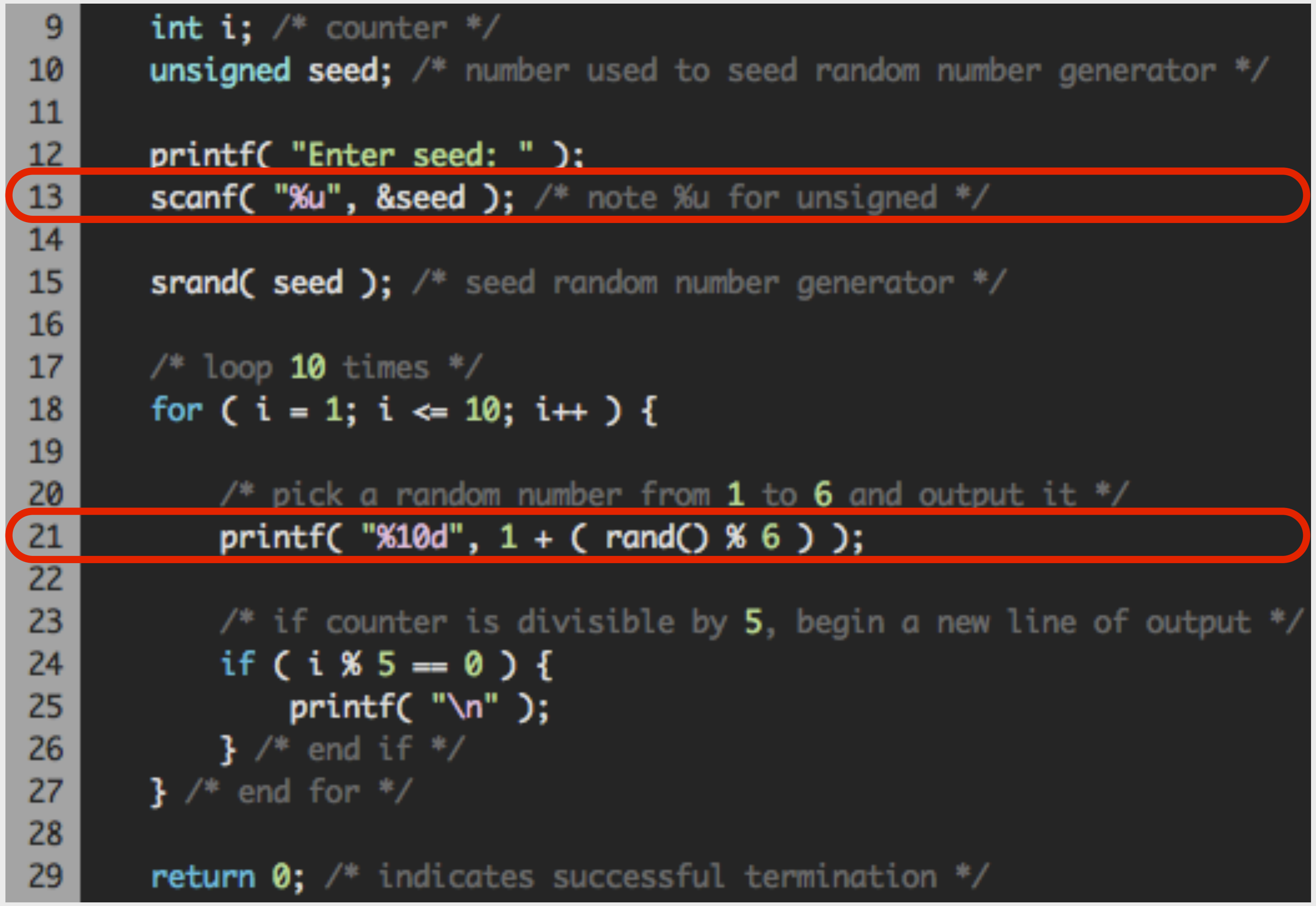
- Example: fig05_09.c



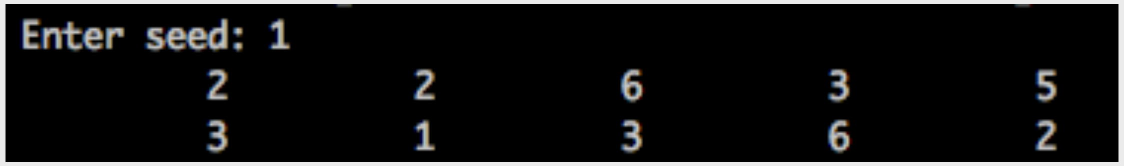
```
   int i; /* counter */
   unsigned seed; /* number used to seed random number generator */

   printf( "Enter seed: " );
   scanf( "%u", &seed ); /* note %u for unsigned */

   srand( seed ); /* seed random number generator */

   /* loop 10 times */
   for ( i = 1; i <= 10; i++ ) {

      /* pick a random number from 1 to 6 and output it */
      printf( "%10d", 1 + ( rand() % 6 ) );

      /* if counter is divisible by 5, begin a new line of output */
      if ( i % 5 == 0 ) {
         printf( "\n" );
      } /* end if */
   } /* end for */

   return 0; /* indicates successful termination */
```

Random Number Generation (Cont.)

- Example: fig05_09.c



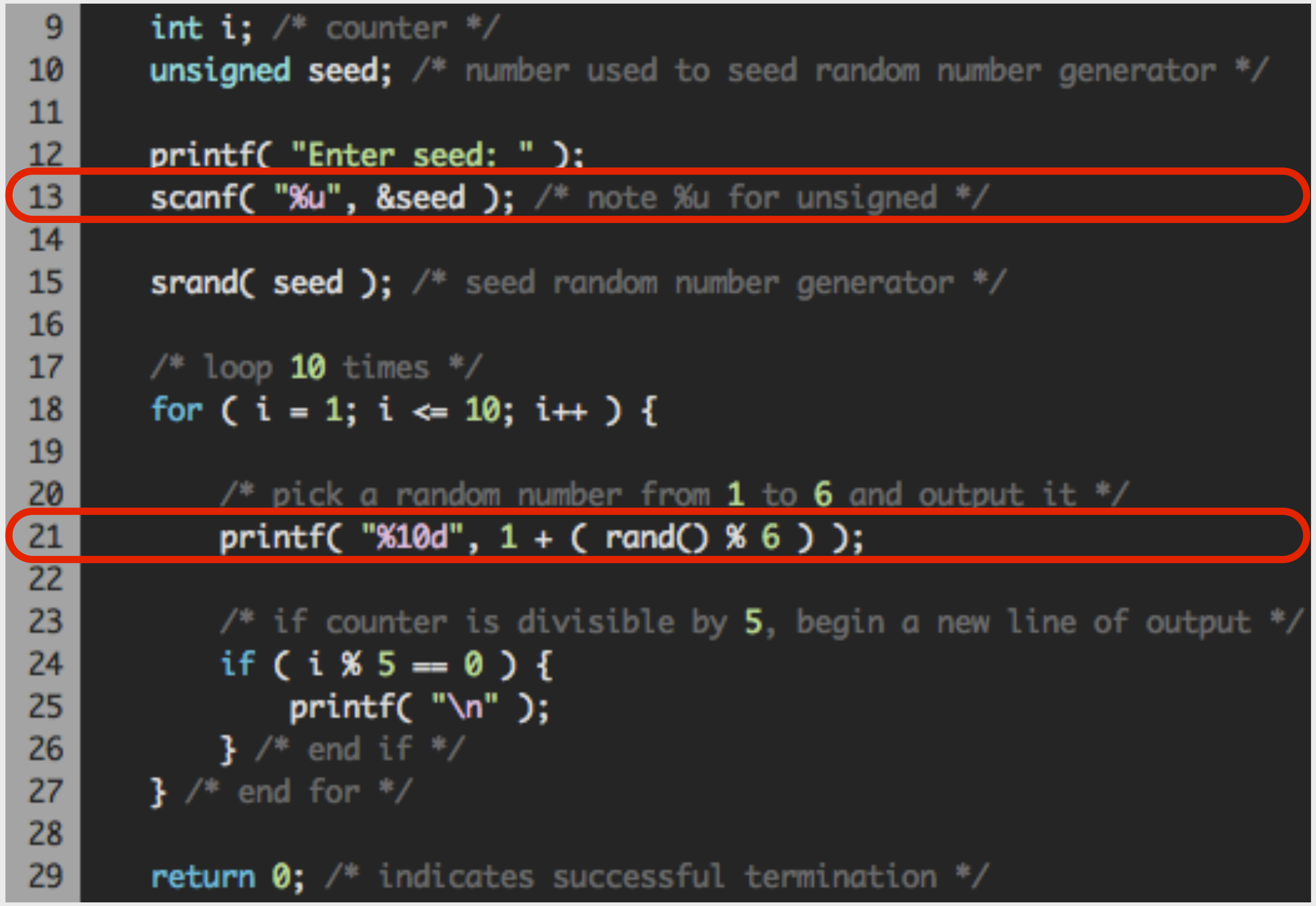
```c
    int i; /* counter */
    unsigned seed; /* number used to seed random number generator */

    printf( "Enter seed: " );
    scanf( "%u", &seed ); /* note %u for unsigned */

    srand( seed ); /* seed random number generator */

    /* loop 10 times */
    for ( i = 1; i <= 10; i++ ) {

        /* pick a random number from 1 to 6 and output it */
        printf( "%10d", 1 + ( rand() % 6 ) );

        /* if counter is divisible by 5, begin a new line of output */
        if ( i % 5 == 0 ) {
            printf( "\n" );
        } /* end if */
    } /* end for */

    return 0; /* indicates successful termination */
```



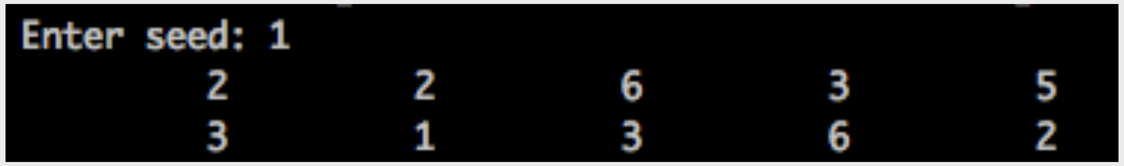
```
Enter seed: 1
         2         2         6         3         5
         3         1         3         6         2
```

Random Number Generation (Cont.)

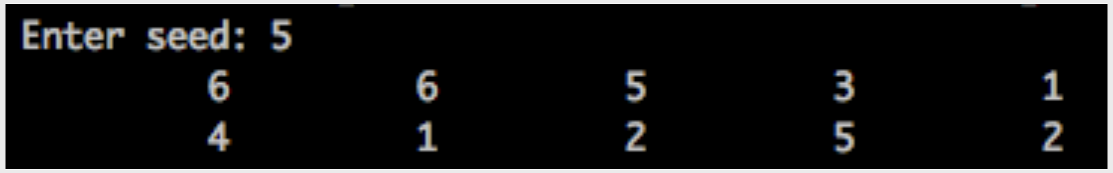
- Example: fig05_09.c



```c
    int i; /* counter */
    unsigned seed; /* number used to seed random number generator */

    printf( "Enter seed: " );
    scanf( "%u", &seed ); /* note %u for unsigned */

    srand( seed ); /* seed random number generator */

    /* loop 10 times */
    for ( i = 1; i <= 10; i++ ) {

        /* pick a random number from 1 to 6 and output it */
        printf( "%10d", 1 + ( rand() % 6 ) );

        /* if counter is divisible by 5, begin a new line of output */
        if ( i % 5 == 0 ) {
            printf( "\n" );
        } /* end if */
    } /* end for */

    return 0; /* indicates successful termination */
```



```
Enter seed: 1
         2         2         6         3         5
         3         1         3         6         2
```



```
Enter seed: 5
         6         6         5         3         1
         4         1         2         5         2
```

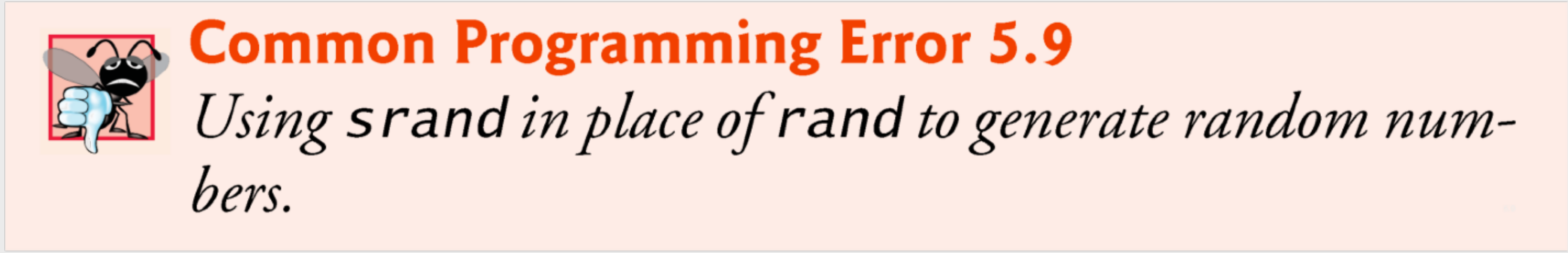
Random Number Generation (Cont.)

- To randomize without entering a seed each time, use a statement like

```
srand( time( NULL ) );
```

  - This causes the computer to read its clock to obtain the value for the seed automatically.
  - Function time takes **`NULL`** as an argument (time is capable of providing you with a string representing the value it returns; **`NULL`** disables this capability for a specific call to time).
  - The function prototype for time is in **`<time.h>`**.

Random Number Generation (Cont.)



Common Programming Error 5.9

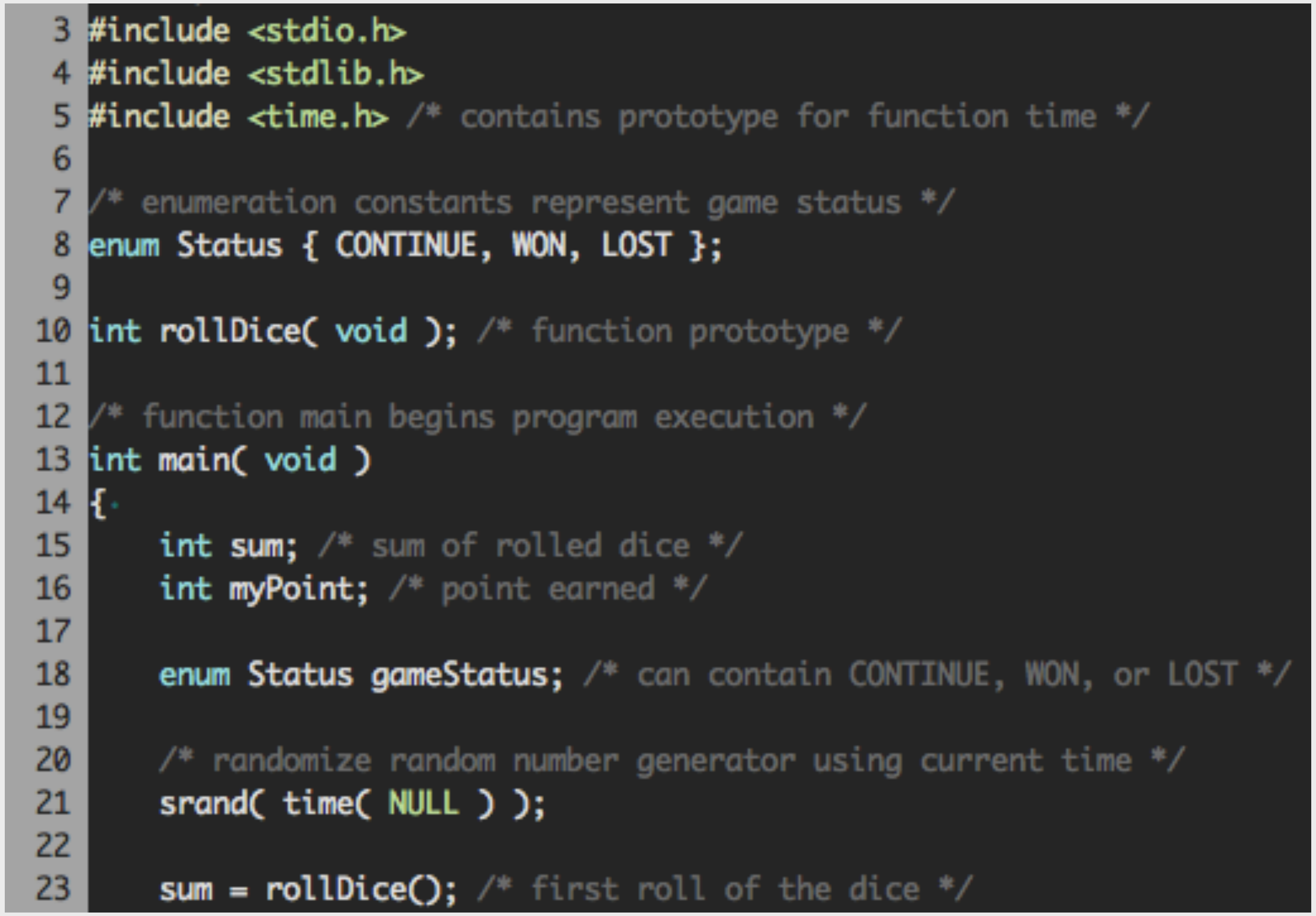
Using `srand` *in place of* `rand` *to generate random numbers.*

Example: A Game of Chance

- One of the most popular games of chance is a dice game known as "**craps**." The rules of the game are simple.
  - A player rolls two dice. Each die has six faces. These faces contain 1, 2, 3, 4, 5, and 6 spots.
  - If the sum is **7 or 11 on the first throw**, the player **wins**.
  - If the sum is **2, 3, or 12 on the first throw**, the player **loses**.
  - If the sum is **4, 5, 6, 8, 9, or 10 on the first throw**, then that sum becomes the player's "point."
    - To **win**, you must continue rolling the dice until you "**make your point**." The player **loses by rolling a 7** before making the point.

Example: A Game of Chance (Cont.)

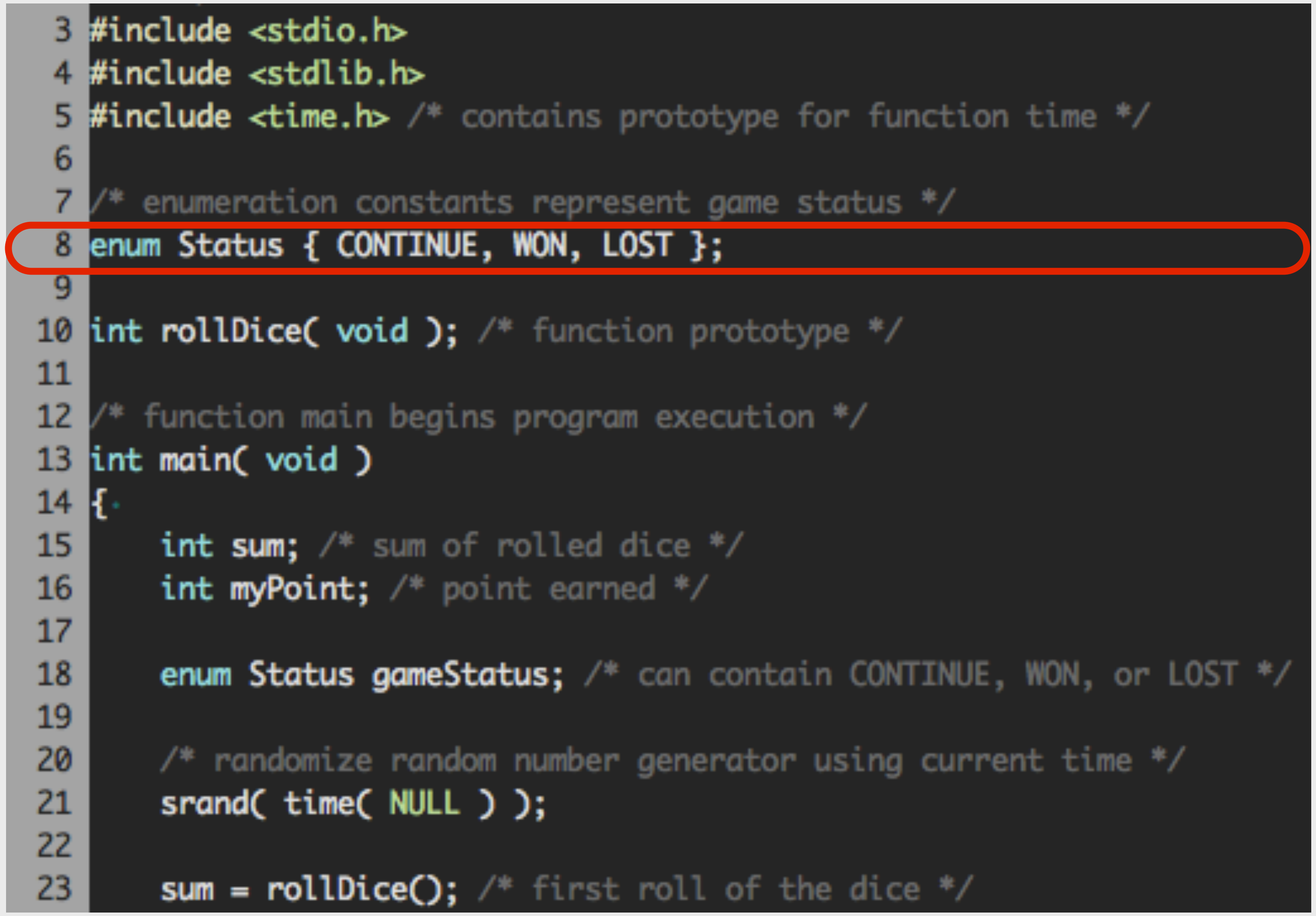
- Example: fig05_10.c



```
#include <stdio.h>
#include <stdlib.h>
#include <time.h> /* contains prototype for function time */

/* enumeration constants represent game status */
enum Status { CONTINUE, WON, LOST };

int rollDice( void ); /* function prototype */

/* function main begins program execution */
int main( void )
{
    int sum; /* sum of rolled dice */
    int myPoint; /* point earned */

    enum Status gameStatus; /* can contain CONTINUE, WON, or LOST */

    /* randomize random number generator using current time */
    srand( time( NULL ) );

    sum = rollDice(); /* first roll of the dice */
```

Example: A Game of Chance (Cont.)

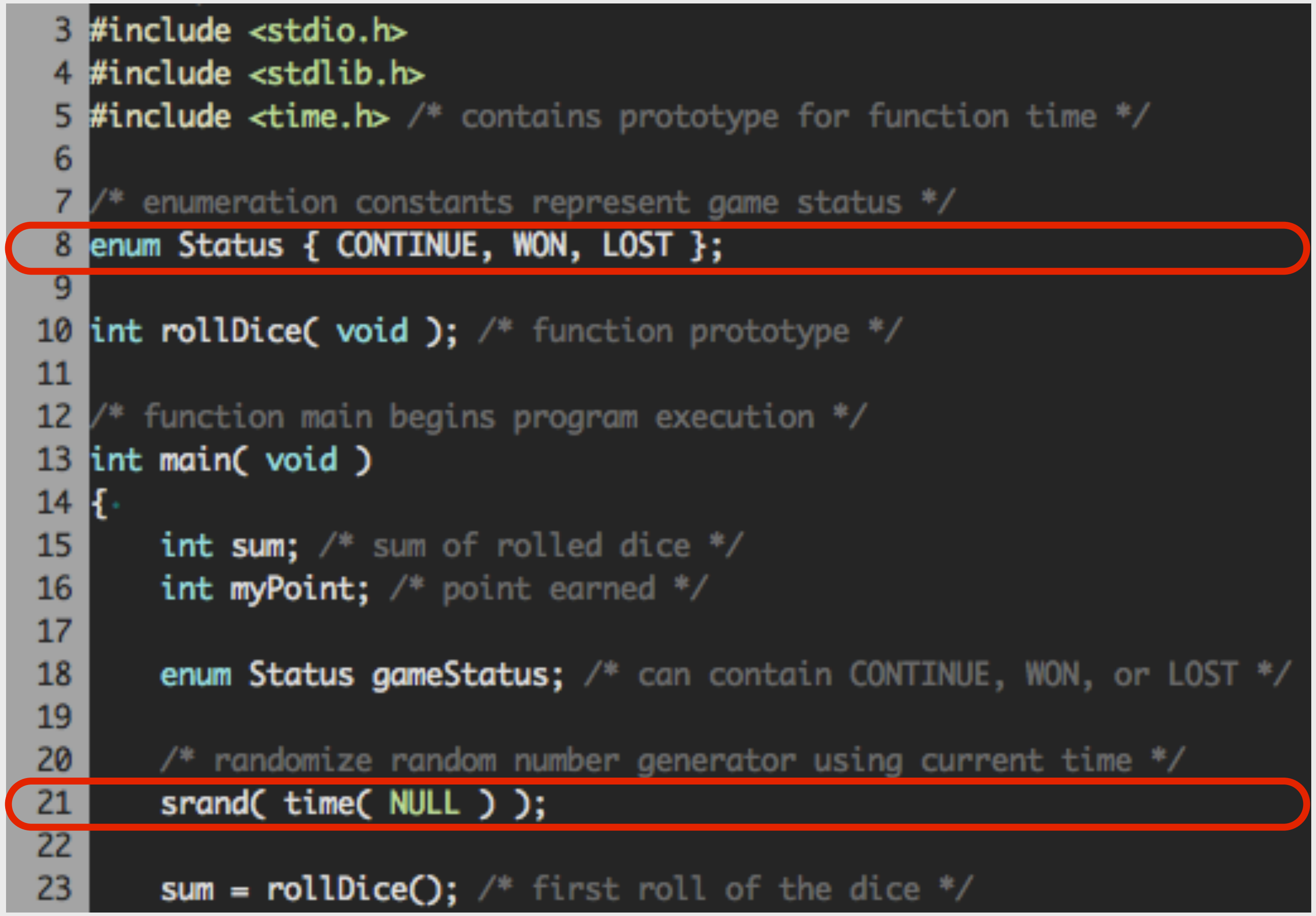
- Example: fig05_10.c



```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h> /* contains prototype for function time */

/* enumeration constants represent game status */
enum Status { CONTINUE, WON, LOST };

int rollDice( void ); /* function prototype */

/* function main begins program execution */
int main( void )
{
    int sum; /* sum of rolled dice */
    int myPoint; /* point earned */

    enum Status gameStatus; /* can contain CONTINUE, WON, or LOST */

    /* randomize random number generator using current time */
    srand( time( NULL ) );

    sum = rollDice(); /* first roll of the dice */
```

Example: A Game of Chance (Cont.)

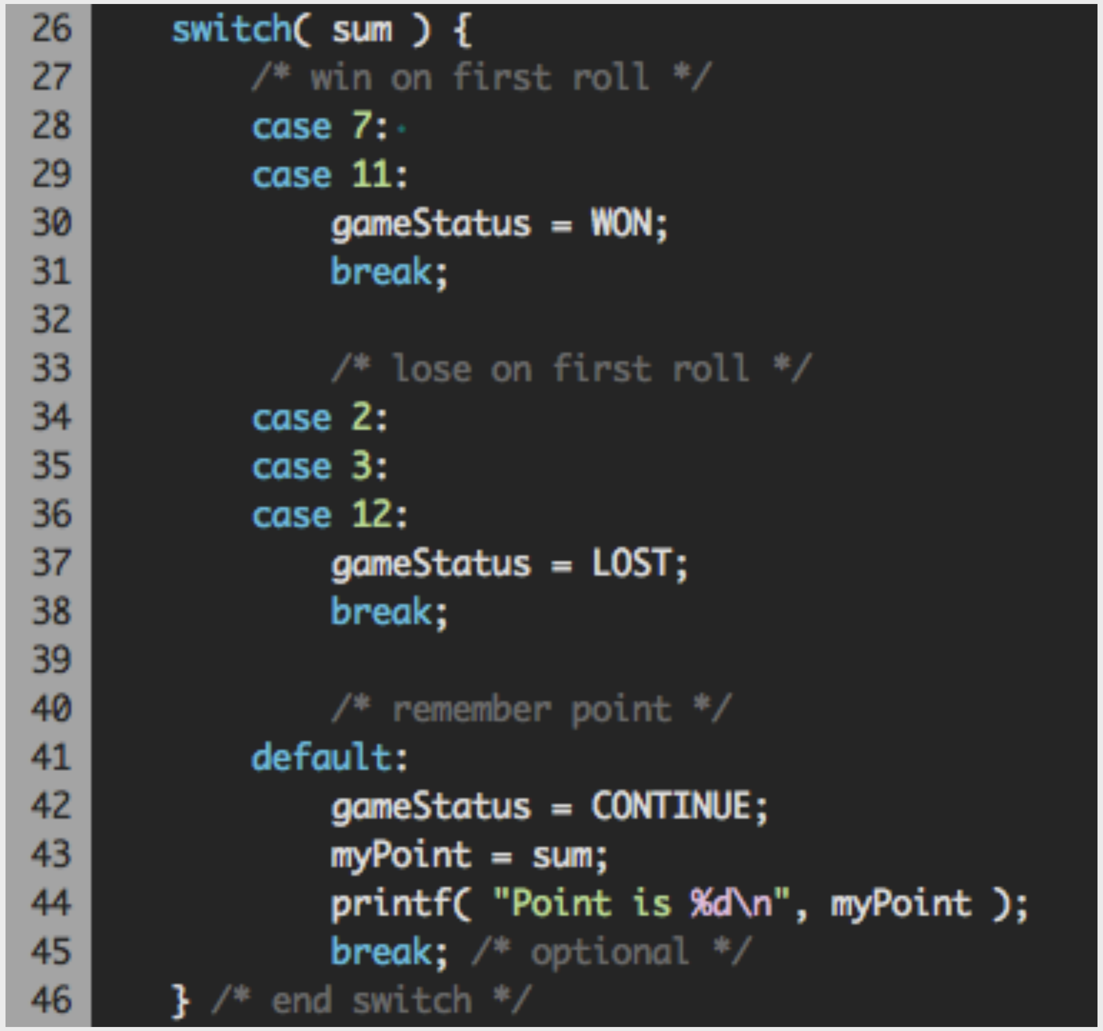
- Example: fig05_10.c



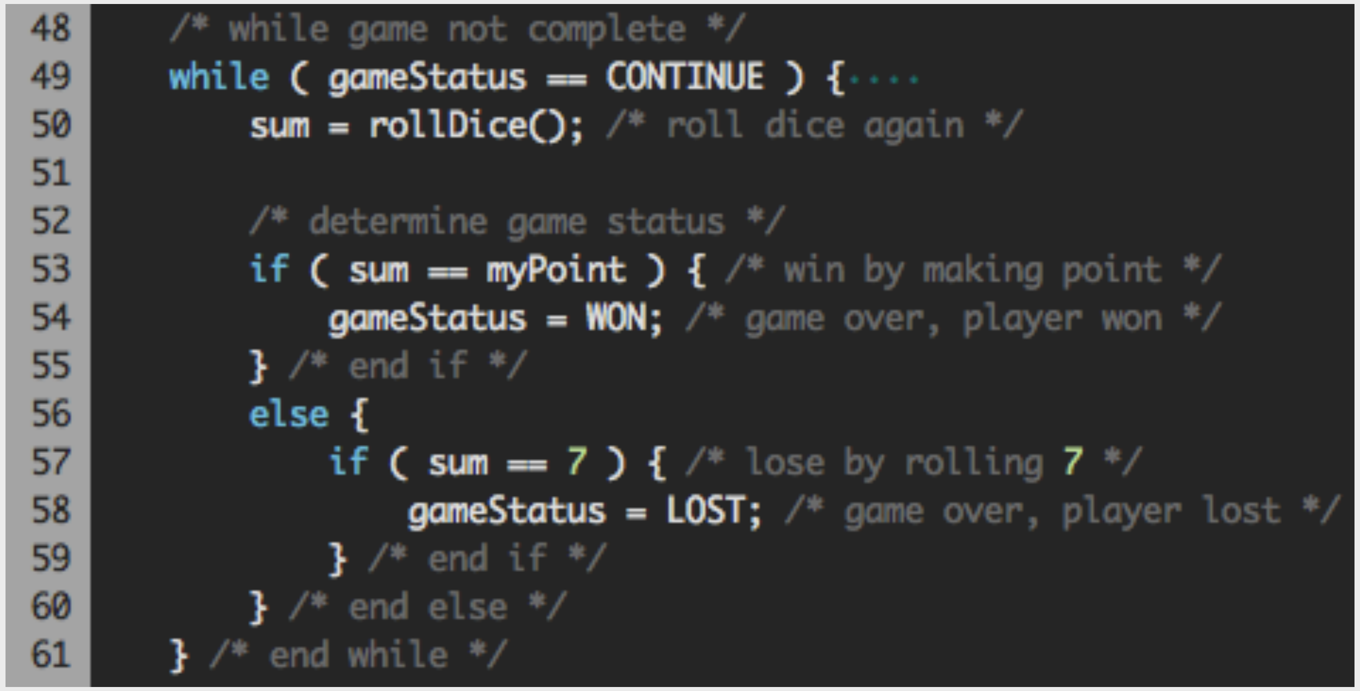
```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h> /* contains prototype for function time */

/* enumeration constants represent game status */
enum Status { CONTINUE, WON, LOST };

int rollDice( void ); /* function prototype */

/* function main begins program execution */
int main( void )
{
    int sum; /* sum of rolled dice */
    int myPoint; /* point earned */

    enum Status gameStatus; /* can contain CONTINUE, WON, or LOST */

    /* randomize random number generator using current time */
    srand( time( NULL ) );

    sum = rollDice(); /* first roll of the dice */
```

Example: A Game of Chance (Cont.)

- Example: fig05_10.c



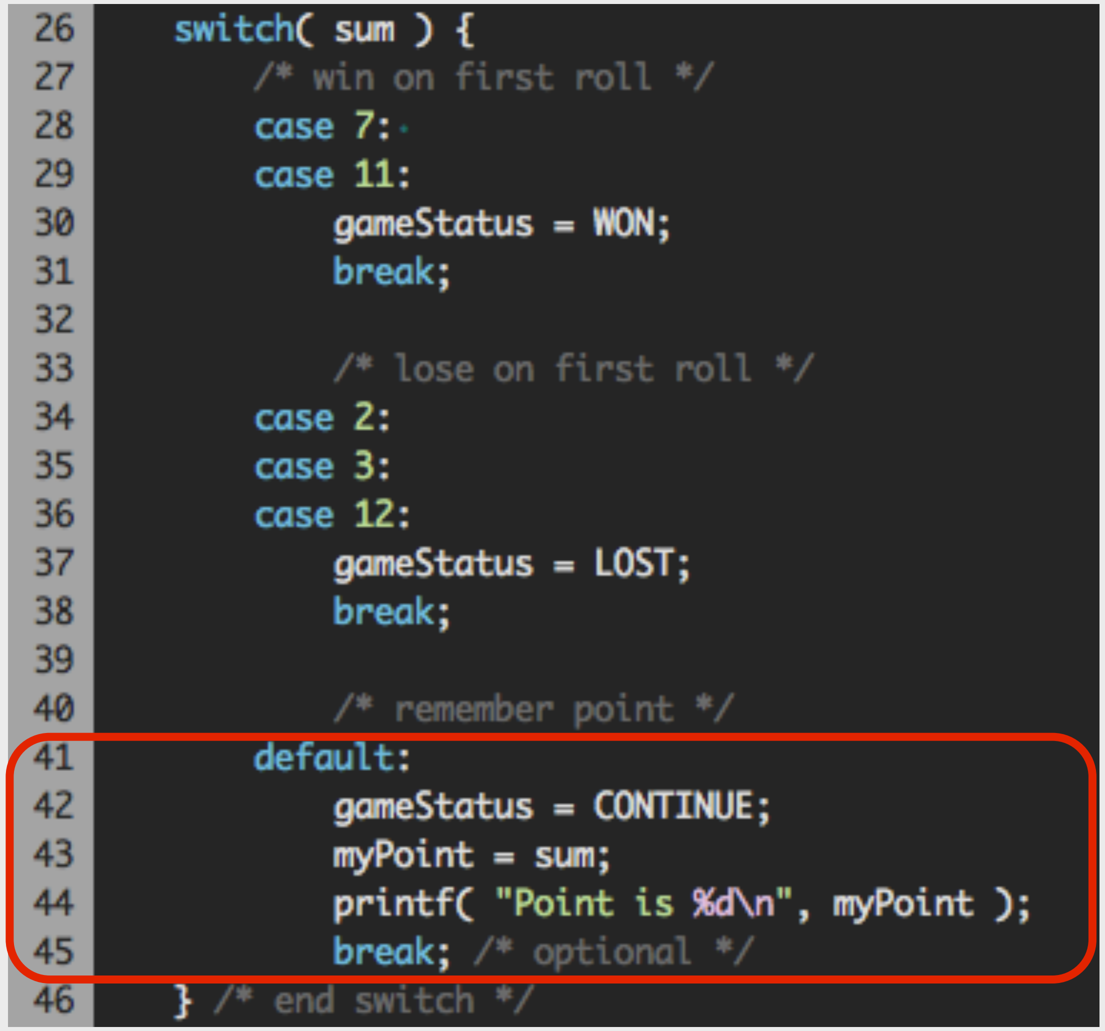
```c
    switch( sum ) {
        /* win on first roll */
        case 7:
        case 11:
            gameStatus = WON;
            break;

            /* lose on first roll */
        case 2:
        case 3:
        case 12:
            gameStatus = LOST;
            break;

            /* remember point */
        default:
            gameStatus = CONTINUE;
            myPoint = sum;
            printf( "Point is %d\n", myPoint );
            break; /* optional */
    } /* end switch */
```



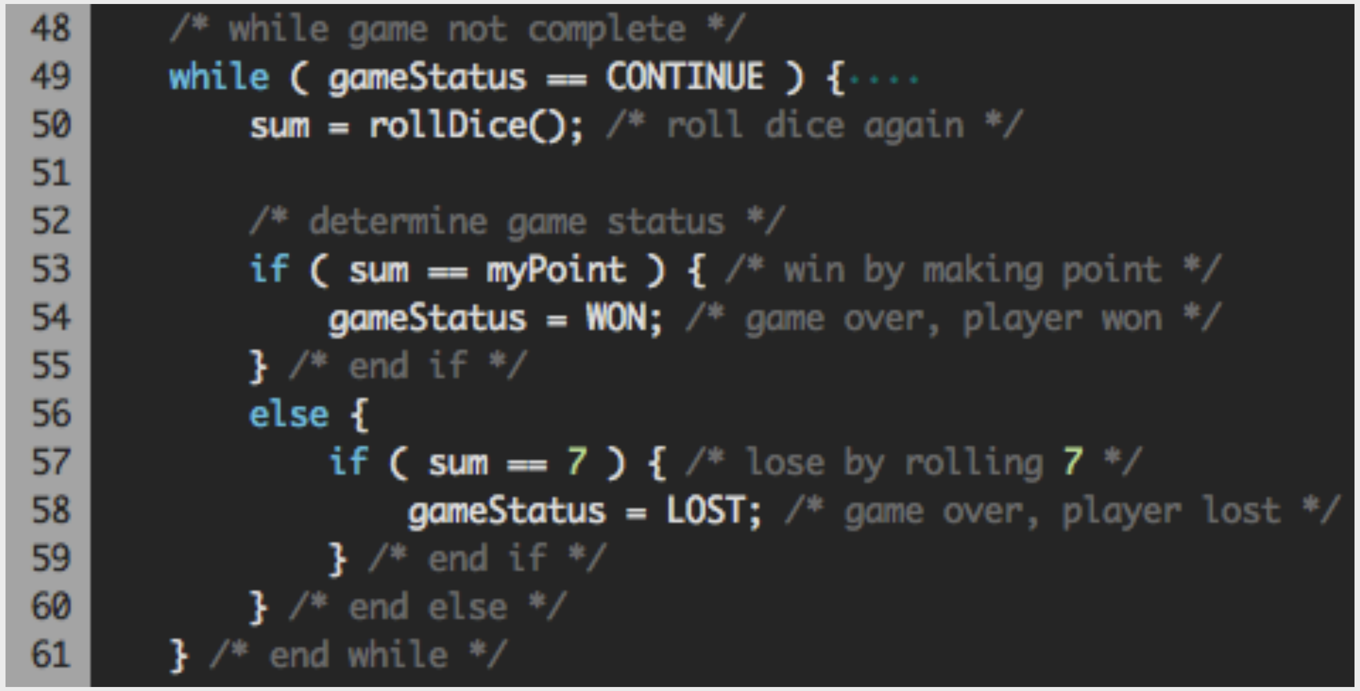
```c
    /* while game not complete */
    while ( gameStatus == CONTINUE ) {
        sum = rollDice(); /* roll dice again */

        /* determine game status */
        if ( sum == myPoint ) { /* win by making point */
            gameStatus = WON; /* game over, player won */
        } /* end if */
        else {
            if ( sum == 7 ) { /* lose by rolling 7 */
                gameStatus = LOST; /* game over, player lost */
            } /* end if */
        } /* end else */
    } /* end while */
```

Example: A Game of Chance (Cont.)

- Example: fig05_10.c



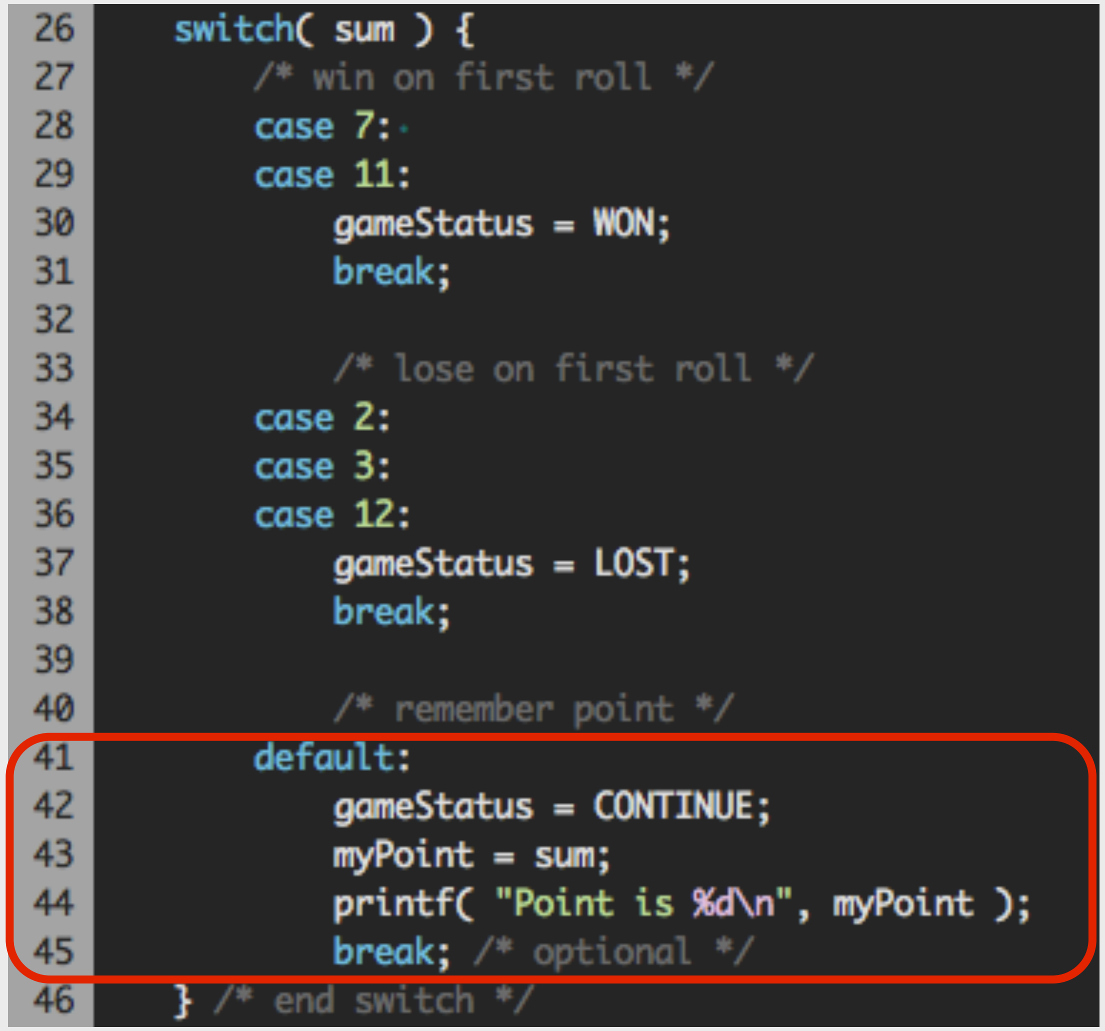
```
    switch( sum ) {
        /* win on first roll */
        case 7:
        case 11:
            gameStatus = WON;
            break;

            /* lose on first roll */
        case 2:
        case 3:
        case 12:
            gameStatus = LOST;
            break;

            /* remember point */
        default:
            gameStatus = CONTINUE;
            myPoint = sum;
            printf( "Point is %d\n", myPoint );
            break; /* optional */
    } /* end switch */
```



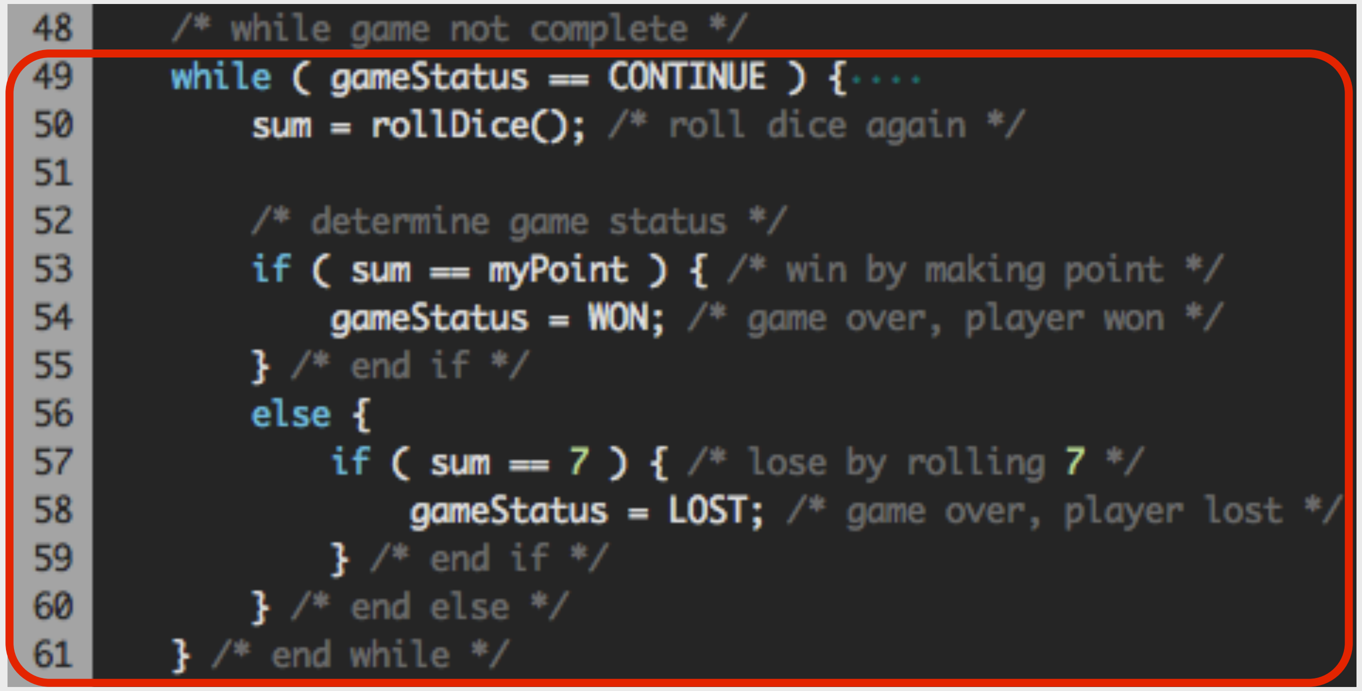
```
    /* while game not complete */
    while ( gameStatus == CONTINUE ) {
        sum = rollDice(); /* roll dice again */

        /* determine game status */
        if ( sum == myPoint ) { /* win by making point */
            gameStatus = WON; /* game over, player won */
        } /* end if */
        else {
            if ( sum == 7 ) { /* lose by rolling 7 */
                gameStatus = LOST; /* game over, player lost */
            } /* end if */
        } /* end else */
    } /* end while */
```

Example: A Game of Chance (Cont.)

- Example: fig05_10.c



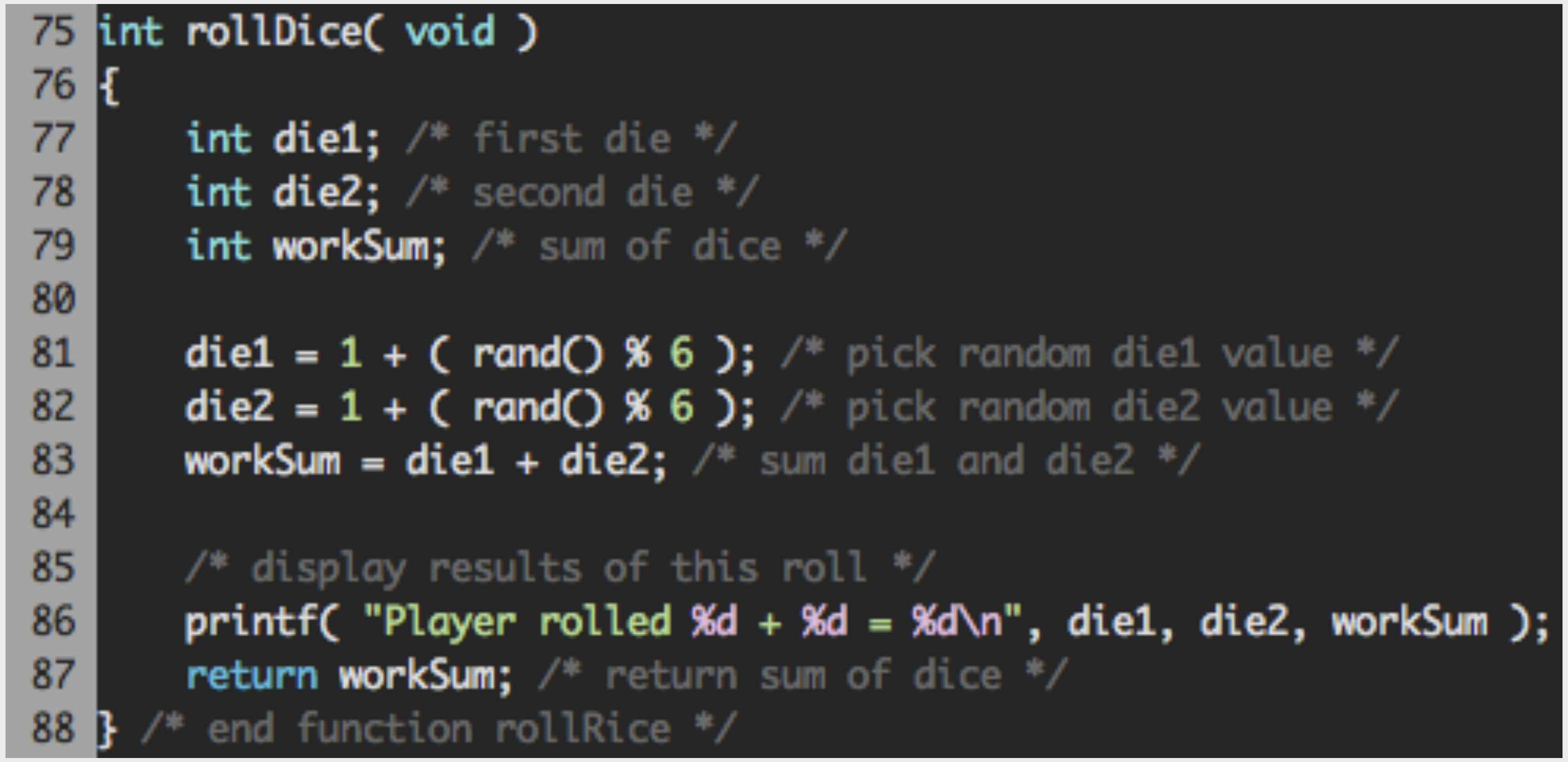
```c
    switch( sum ) {
        /* win on first roll */
        case 7:
        case 11:
            gameStatus = WON;
            break;

            /* lose on first roll */
        case 2:
        case 3:
        case 12:
            gameStatus = LOST;
            break;

            /* remember point */
        default:
            gameStatus = CONTINUE;
            myPoint = sum;
            printf( "Point is %d\n", myPoint );
            break; /* optional */
    } /* end switch */
```



```c
    /* while game not complete */
    while ( gameStatus == CONTINUE ) {
        sum = rollDice(); /* roll dice again */

        /* determine game status */
        if ( sum == myPoint ) { /* win by making point */
            gameStatus = WON; /* game over, player won */
        } /* end if */
        else {
            if ( sum == 7 ) { /* lose by rolling 7 */
                gameStatus = LOST; /* game over, player lost */
            } /* end if */
        } /* end else */
    } /* end while */
```

Example: A Game of Chance (Cont.)

- Example: fig05_10.c



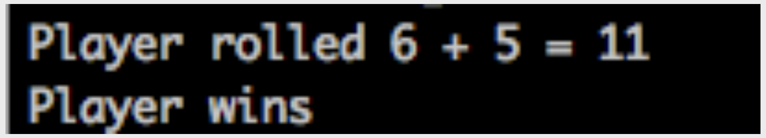
```c
int rollDice( void )
{
    int die1; /* first die */
    int die2; /* second die */
    int workSum; /* sum of dice */

    die1 = 1 + ( rand() % 6 ); /* pick random die1 value */
    die2 = 1 + ( rand() % 6 ); /* pick random die2 value */
    workSum = die1 + die2; /* sum die1 and die2 */

    /* display results of this roll */
    printf( "Player rolled %d + %d = %d\n", die1, die2, workSum );
    return workSum; /* return sum of dice */
} /* end function rollRice */
```

Example: A Game of Chance (Cont.)

- Example: fig05_10.c

Example: A Game of Chance (Cont.)

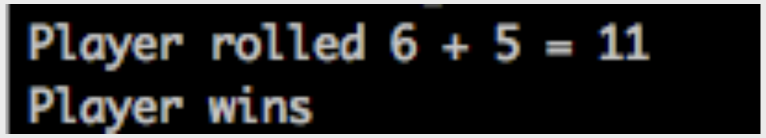
- Example: fig05_10.c



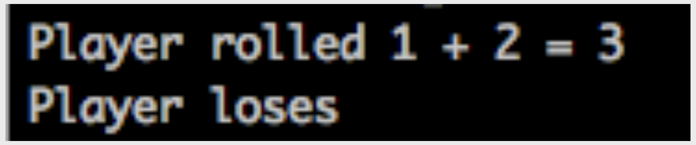
```
Player rolled 6 + 5 = 11
Player wins
```

Example: A Game of Chance (Cont.)

- Example: fig05_10.c



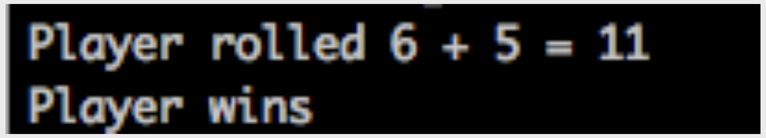
```
Player rolled 6 + 5 = 11
Player wins
```



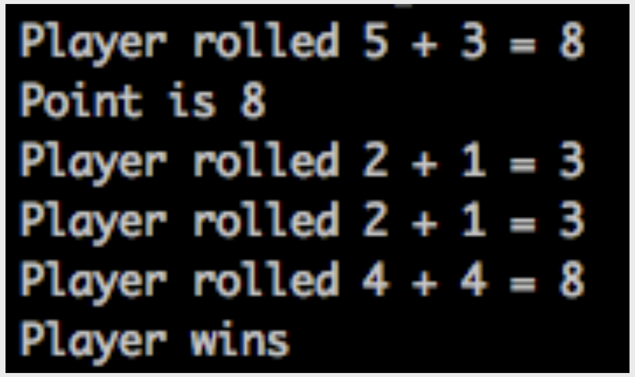
```
Player rolled 1 + 2 = 3
Player loses
```

Example: A Game of Chance (Cont.)

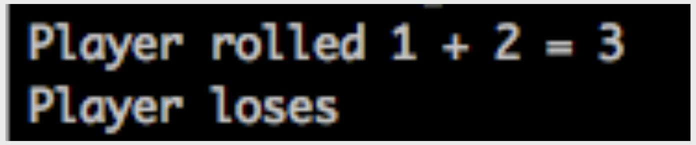
- Example: fig05_10.c



```
Player rolled 6 + 5 = 11
Player wins
```



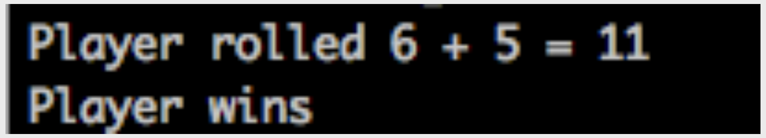
```
Player rolled 5 + 3 = 8
Point is 8
Player rolled 2 + 1 = 3
Player rolled 2 + 1 = 3
Player rolled 4 + 4 = 8
Player wins
```



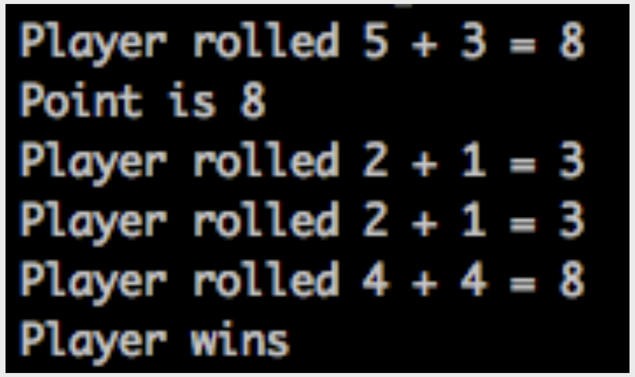
```
Player rolled 1 + 2 = 3
Player loses
```

Example: A Game of Chance (Cont.)

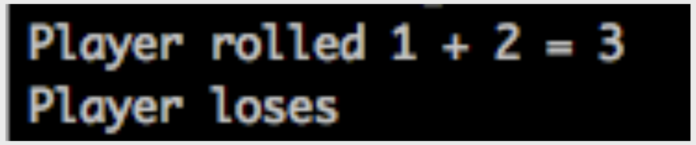
- Example: fig05_10.c



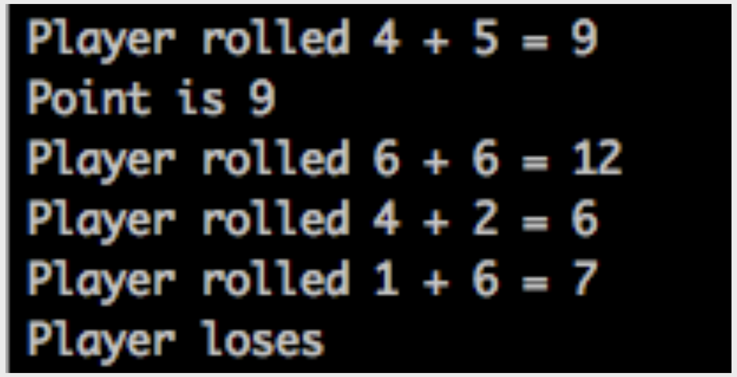
```
Player rolled 6 + 5 = 11
Player wins
```



```
Player rolled 5 + 3 = 8
Point is 8
Player rolled 2 + 1 = 3
Player rolled 2 + 1 = 3
Player rolled 4 + 4 = 8
Player wins
```



```
Player rolled 1 + 2 = 3
Player loses
```



```
Player rolled 4 + 5 = 9
Point is 9
Player rolled 6 + 6 = 12
Player rolled 4 + 2 = 6
Player rolled 1 + 6 = 7
Player loses
```

Example: A Game of Chance (Cont.)

- An enumeration, introduced by the keyword **`enum`**, is a set of integer constants represented by identifiers.
    - Enumeration constants are sometimes called **symbolic constants**.
    - The constant **`CONTINUE`** has the value 0, **`WON`** has the value 1 and **`LOST`** has the value 2.
    - It’s also possible to assign an integer value to each identifier in an enum (see Chapter 10).

Storage Classes

- Actually, each identifier in a program has other attributes, including **storage class**, **storage duration**, **scope** and **linkage**.
  - **storage class** determines its storage duration, scope and linkage.
  - **storage duration** is the period during which the identifier exists in memory.
  - **scope** is where the identifier can be referenced in a program.
  - **linkage** determines for a multiple-source-file program

Storage Classes (Cont.)

- An identifier’s **scope** is where the identifier can be referenced in a program.
- An identifier’s **linkage** determines for a **multiple-source-file program**
- This section discusses **storage classes** and **storage duration**. (Chap 5.12)

Storage Classes (Cont.)

- The four storage-class specifiers can be split into two storage durations: **automatic storage duration** and **static storage duration**.

- Keywords **auto** and **register** are used to declare variables of **automatic storage duration**.

Storage Classes (Cont.)

- A function's **local variables** (those declared in the parameter list or function body) normally have **automatic** storage duration.
- Local variables have automatic storage duration **by default**, so keyword **auto** is rarely used.
- For example

  auto double x, y;

Storage Classes (Cont.)

- **register** variables
  - Loaded into registers for calculations and other processing.
  - The following declaration suggests that the integer variable counter be placed in one of the computer's registers and initialized to 1:

    register int counter = 1;

Storage Classes (Cont.)

- Keywords **extern** and **static** are used in the declarations of identifiers for variables and functions of **static storage duration.**
  - Identifiers of static storage duration exist from the time at which the program begins execution.
  - For static variables, storage is allocated and initialized once, when the program begins execution. For example:

    static int count = 1;
- In Chapter 14 we discuss the explicit use of **extern** and **static** with **external identifiers** and **multiple-source-file programs**.

Scope Rules

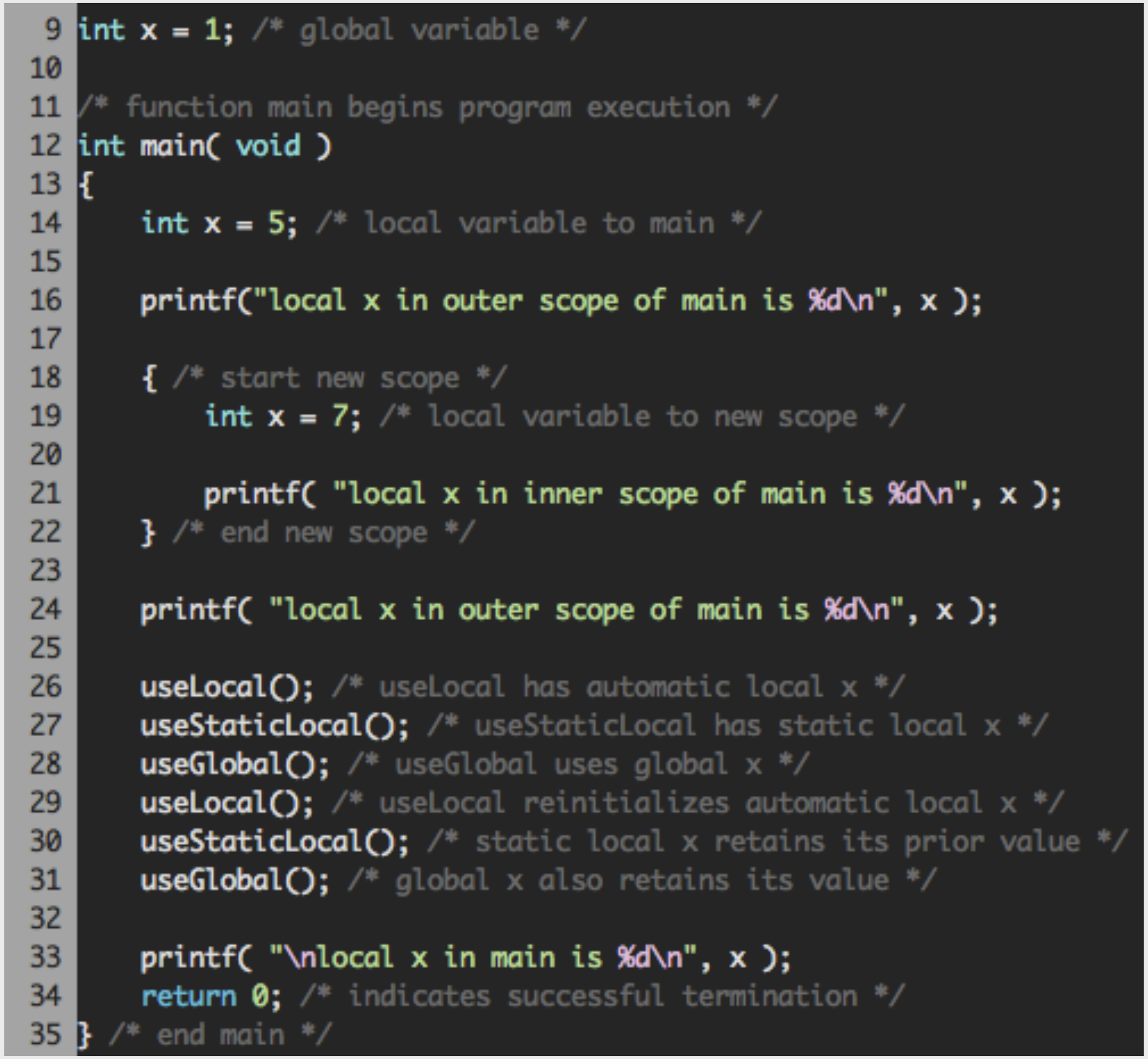
- The four identifier scopes are
  - function scope
  - file scope
  - block scope
  - function-prototype scope

Scope Rules (Cont.)

- Labels (an identifier followed by a colon such as **start:**) are the only identifiers with **function scope**.
- An identifier declared outside any function has **file scope**.
- Identifiers defined inside a block ({}) have **block scope**.
- The only identifiers with **function-prototype scope** are those used in the parameter list of a function prototype.

Scope Rules (Cont.)

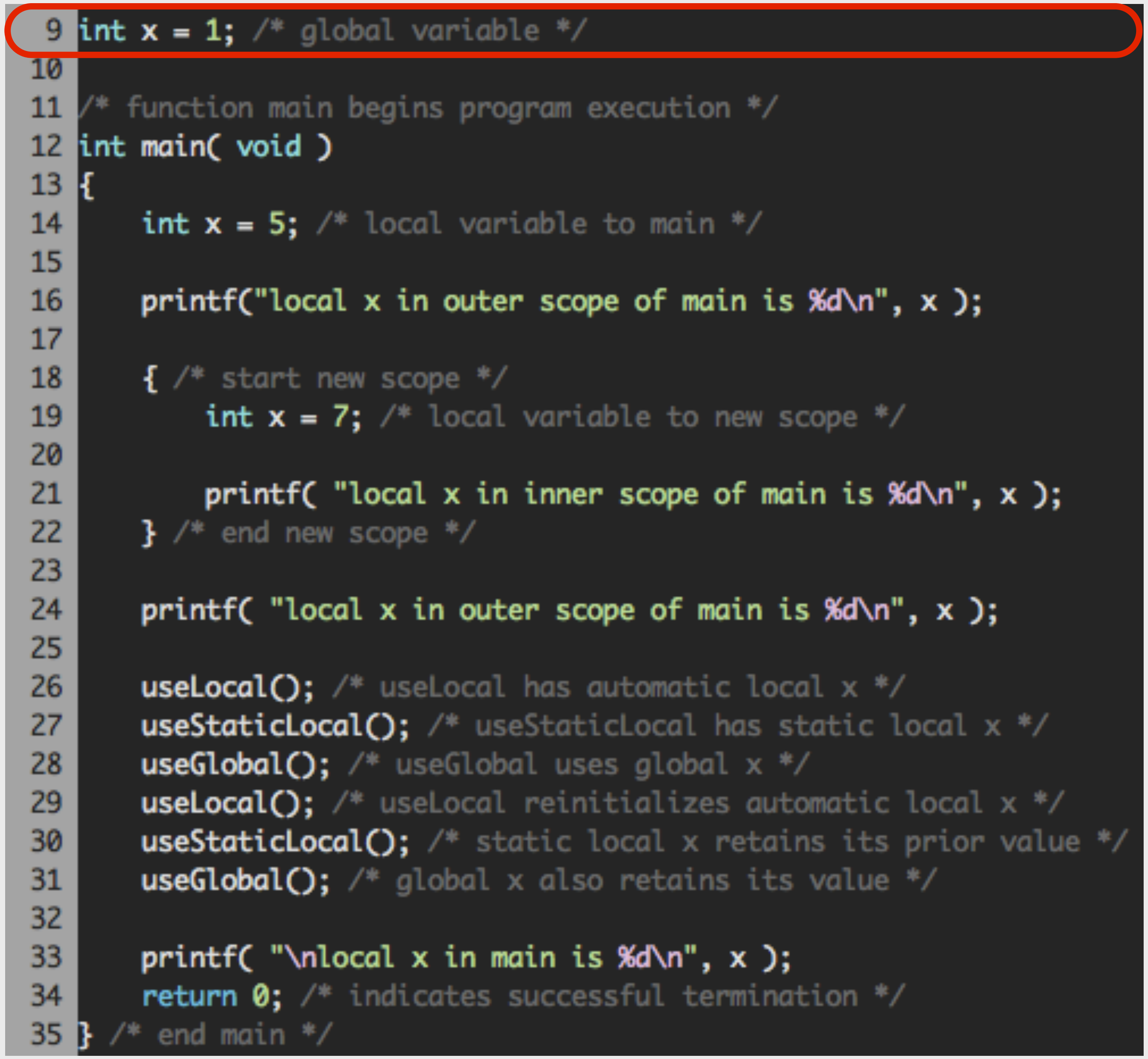
- Example: fig05_12.c



```c
int x = 1; /* global variable */

/* function main begins program execution */
int main( void )
{
    int x = 5; /* local variable to main */

    printf("local x in outer scope of main is %d\n", x );

    { /* start new scope */
        int x = 7; /* local variable to new scope */

        printf( "local x in inner scope of main is %d\n", x );
    } /* end new scope */

    printf( "local x in outer scope of main is %d\n", x );

    useLocal(); /* useLocal has automatic local x */
    useStaticLocal(); /* useStaticLocal has static local x */
    useGlobal(); /* useGlobal uses global x */
    useLocal(); /* useLocal reinitializes automatic local x */
    useStaticLocal(); /* static local x retains its prior value */
    useGlobal(); /* global x also retains its value */

    printf( "\nlocal x in main is %d\n", x );
    return 0; /* indicates successful termination */
} /* end main */
```

Scope Rules (Cont.)

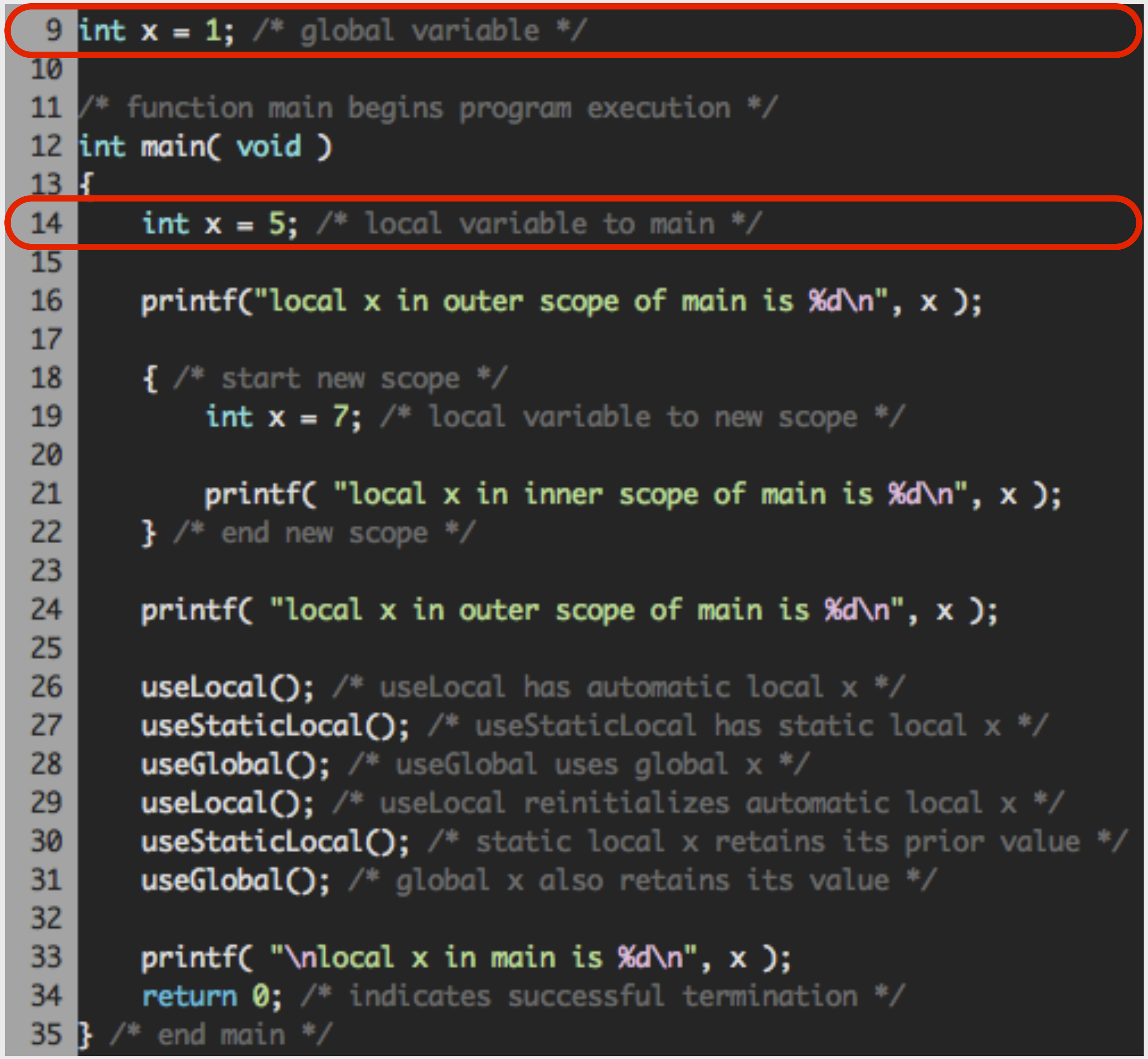
- Example: fig05_12.c



```c
int x = 1; /* global variable */

/* function main begins program execution */
int main( void )
{
    int x = 5; /* local variable to main */

    printf("local x in outer scope of main is %d\n", x );

    { /* start new scope */
        int x = 7; /* local variable to new scope */

        printf( "local x in inner scope of main is %d\n", x );
    } /* end new scope */

    printf( "local x in outer scope of main is %d\n", x );

    useLocal(); /* useLocal has automatic local x */
    useStaticLocal(); /* useStaticLocal has static local x */
    useGlobal(); /* useGlobal uses global x */
    useLocal(); /* useLocal reinitializes automatic local x */
    useStaticLocal(); /* static local x retains its prior value */
    useGlobal(); /* global x also retains its value */

    printf( "\nlocal x in main is %d\n", x );
    return 0; /* indicates successful termination */
} /* end main */
```

Scope Rules (Cont.)

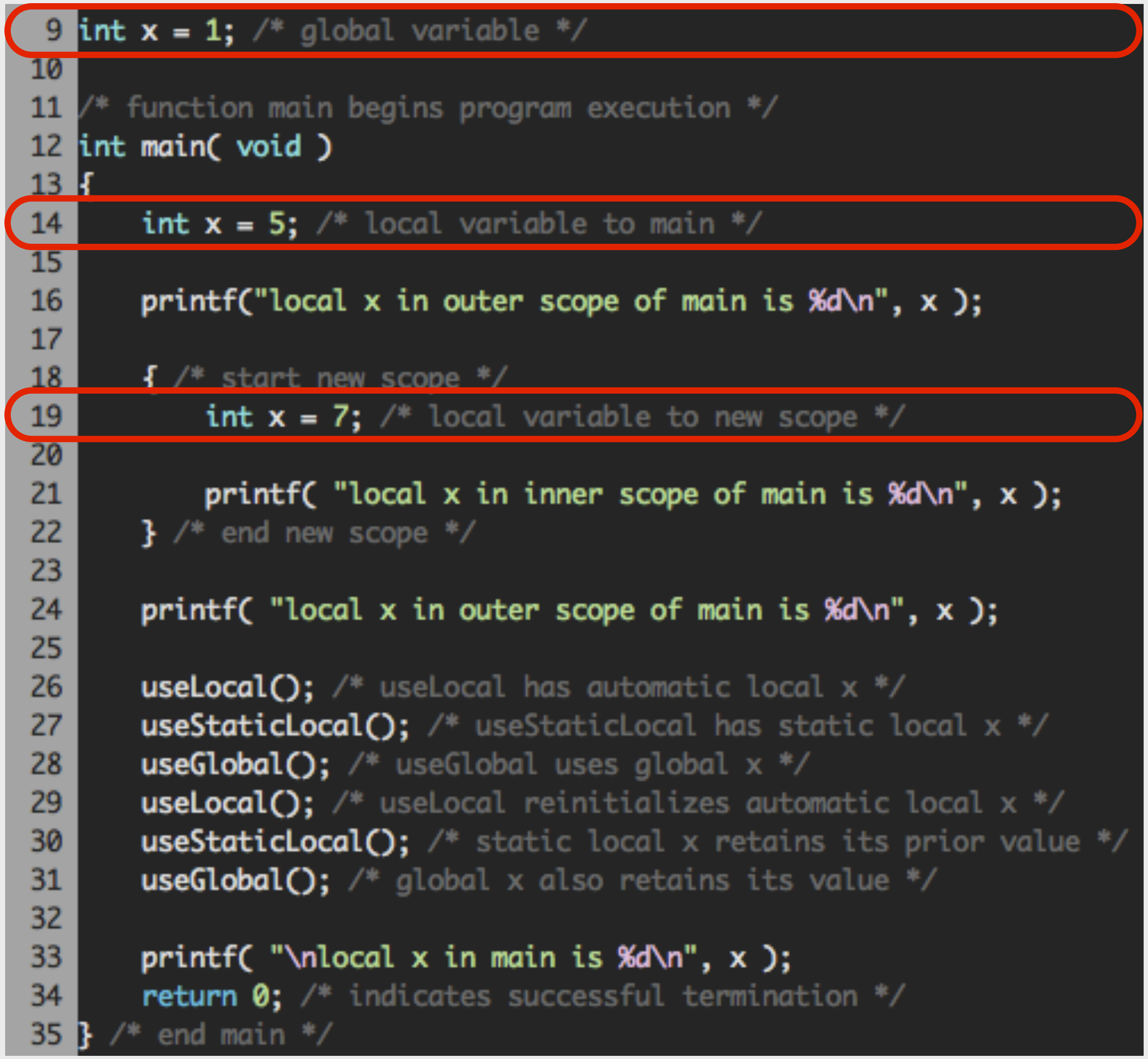
- Example: fig05_12.c



```c
int x = 1; /* global variable */

/* function main begins program execution */
int main( void )
{
    int x = 5; /* local variable to main */

    printf("local x in outer scope of main is %d\n", x );

    { /* start new scope */
        int x = 7; /* local variable to new scope */

        printf( "local x in inner scope of main is %d\n", x );
    } /* end new scope */

    printf( "local x in outer scope of main is %d\n", x );

    useLocal(); /* useLocal has automatic local x */
    useStaticLocal(); /* useStaticLocal has static local x */
    useGlobal(); /* useGlobal uses global x */
    useLocal(); /* useLocal reinitializes automatic local x */
    useStaticLocal(); /* static local x retains its prior value */
    useGlobal(); /* global x also retains its value */

    printf( "\nlocal x in main is %d\n", x );
    return 0; /* indicates successful termination */
} /* end main */
```

Scope Rules (Cont.)

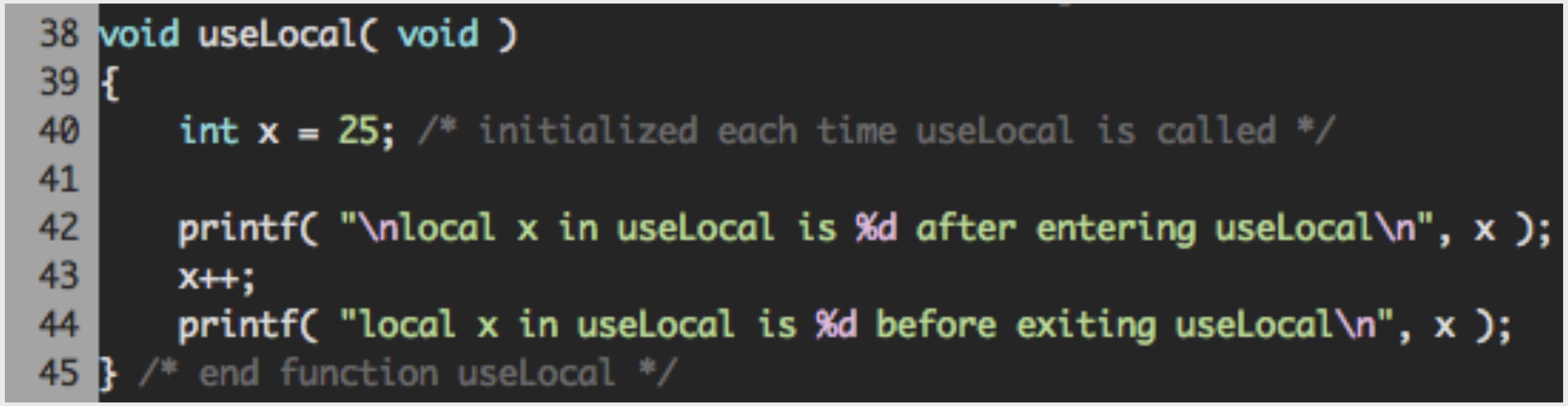
- Example: fig05_12.c



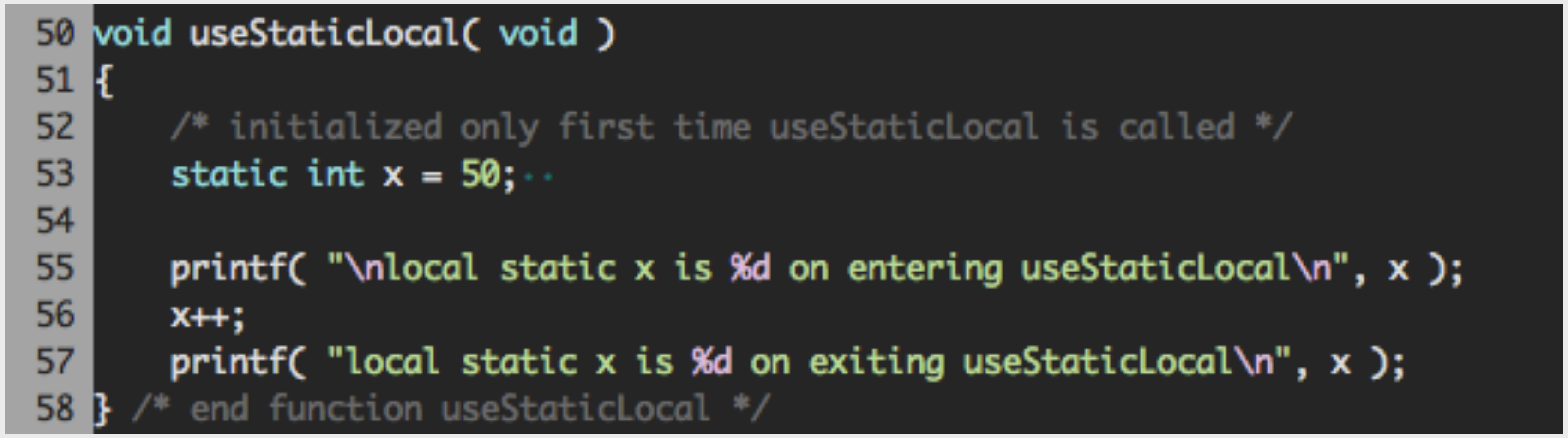
```c
int x = 1; /* global variable */

/* function main begins program execution */
int main( void )
{
    int x = 5; /* local variable to main */

    printf("local x in outer scope of main is %d\n", x );

    { /* start new scope */
        int x = 7; /* local variable to new scope */

        printf( "local x in inner scope of main is %d\n", x );
    } /* end new scope */

    printf( "local x in outer scope of main is %d\n", x );

    useLocal(); /* useLocal has automatic local x */
    useStaticLocal(); /* useStaticLocal has static local x */
    useGlobal(); /* useGlobal uses global x */
    useLocal(); /* useLocal reinitializes automatic local x */
    useStaticLocal(); /* static local x retains its prior value */
    useGlobal(); /* global x also retains its value */

    printf( "\nlocal x in main is %d\n", x );
    return 0; /* indicates successful termination */
} /* end main */
```

Scope Rules (Cont.)

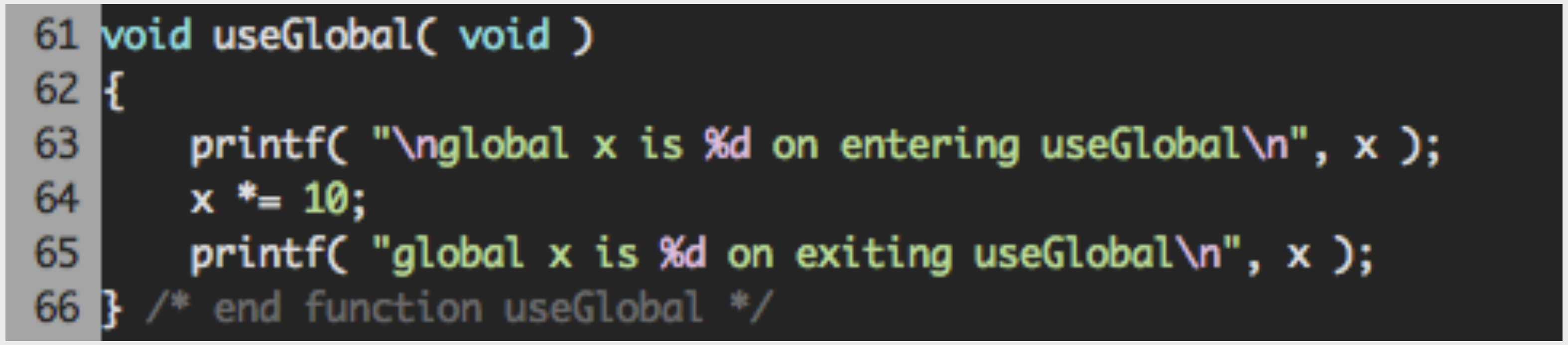
- Example: fig05_12.c



```c
void useLocal( void )
{
    int x = 25; /* initialized each time useLocal is called */

    printf( "\nlocal x in useLocal is %d after entering useLocal\n", x );
    x++;
    printf( "local x in useLocal is %d before exiting useLocal\n", x );
} /* end function useLocal */
```



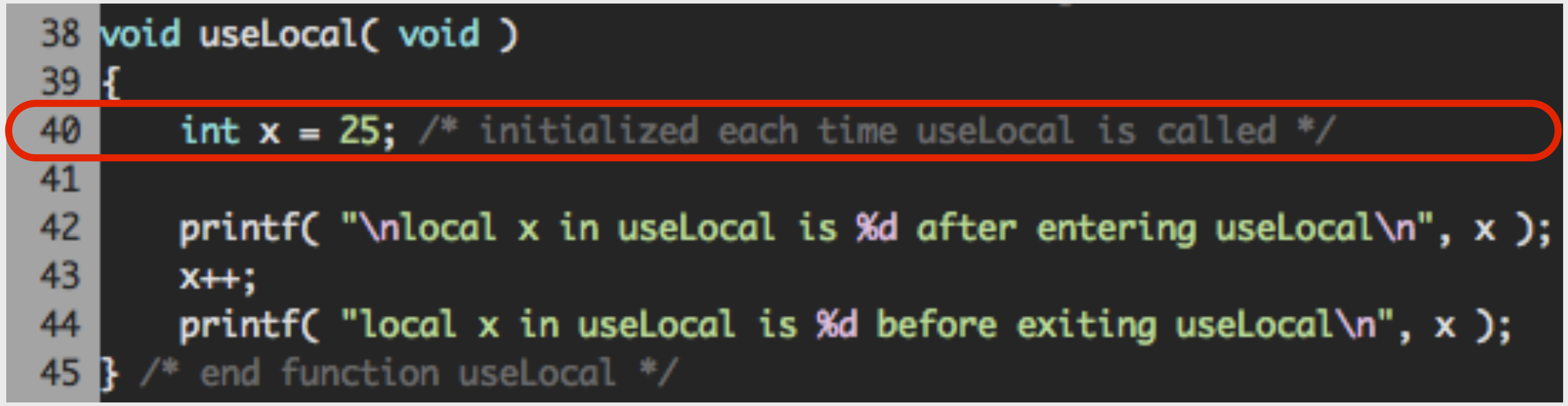
```c
void useStaticLocal( void )
{
    /* initialized only first time useStaticLocal is called */
    static int x = 50;

    printf( "\nlocal static x is %d on entering useStaticLocal\n", x );
    x++;
    printf( "local static x is %d on exiting useStaticLocal\n", x );
} /* end function useStaticLocal */
```



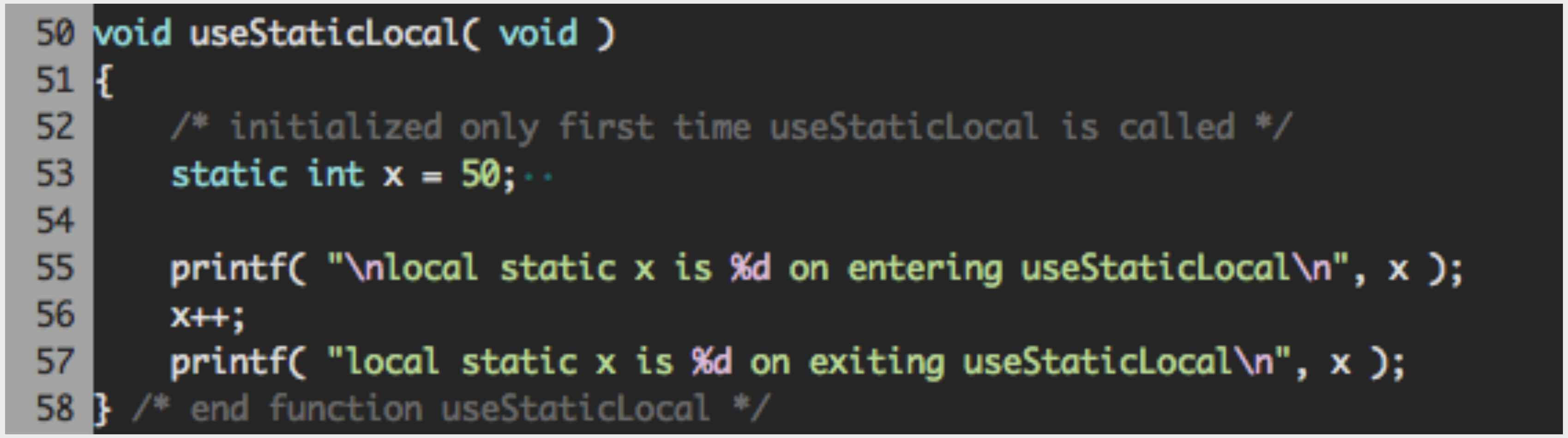
```c
void useGlobal( void )
{
    printf( "\nglobal x is %d on entering useGlobal\n", x );
    x *= 10;
    printf( "global x is %d on exiting useGlobal\n", x );
} /* end function useGlobal */
```

Scope Rules (Cont.)

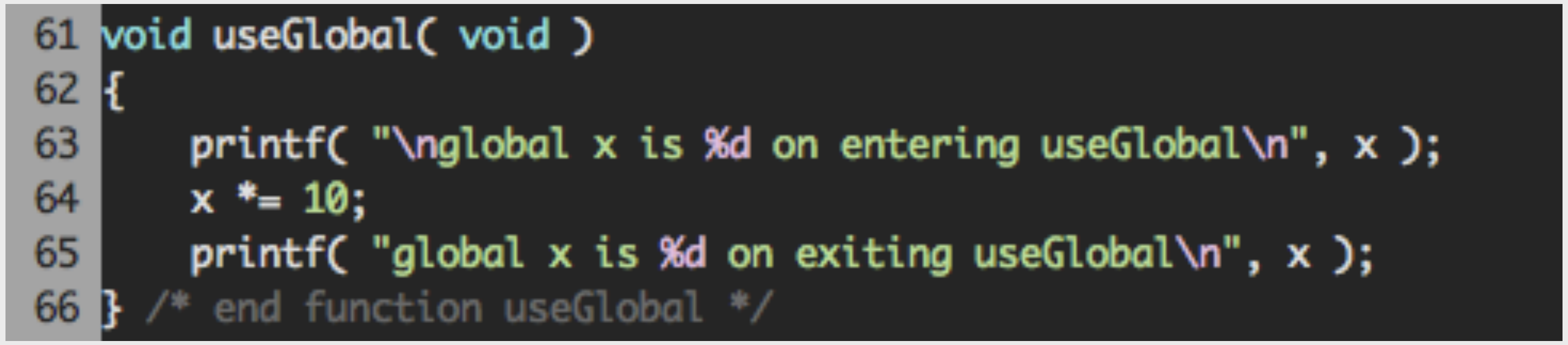
- Example: fig05_12.c



```
void useLocal( void )
{
    int x = 25; /* initialized each time useLocal is called */

    printf( "\nlocal x in useLocal is %d after entering useLocal\n", x );
    x++;
    printf( "local x in useLocal is %d before exiting useLocal\n", x );
} /* end function useLocal */
```



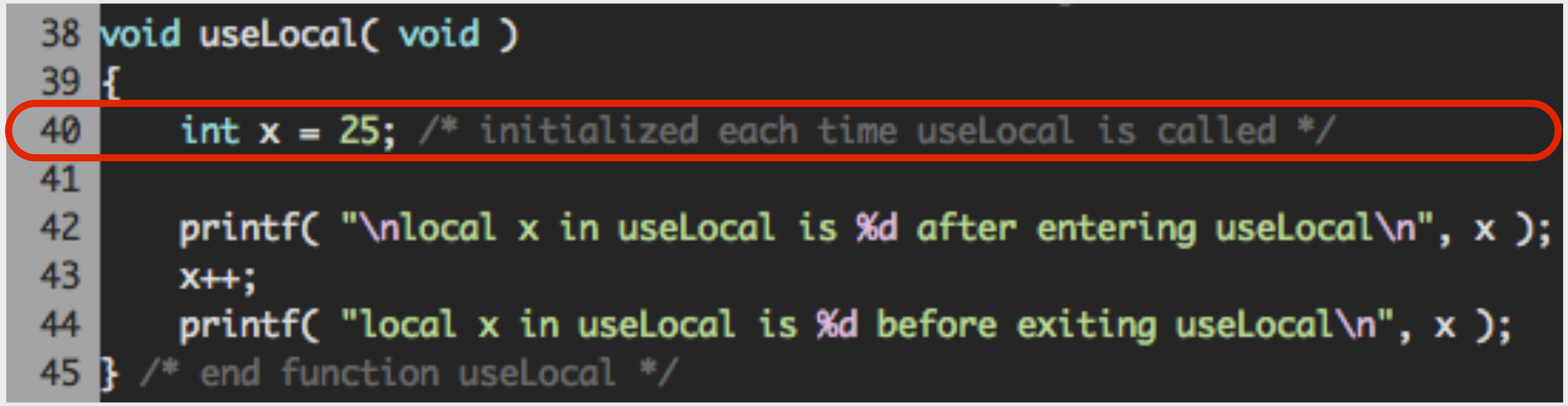
```
void useStaticLocal( void )
{
    /* initialized only first time useStaticLocal is called */
    static int x = 50;

    printf( "\nlocal static x is %d on entering useStaticLocal\n", x );
    x++;
    printf( "local static x is %d on exiting useStaticLocal\n", x );
} /* end function useStaticLocal */
```



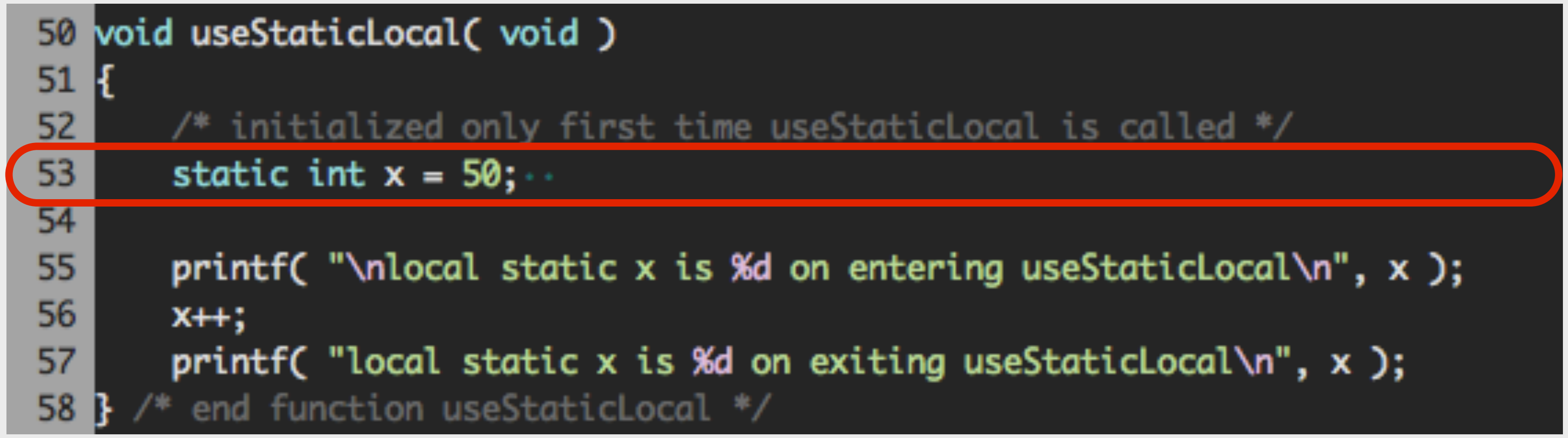
```
void useGlobal( void )
{
    printf( "\nglobal x is %d on entering useGlobal\n", x );
    x *= 10;
    printf( "global x is %d on exiting useGlobal\n", x );
} /* end function useGlobal */
```

Scope Rules (Cont.)

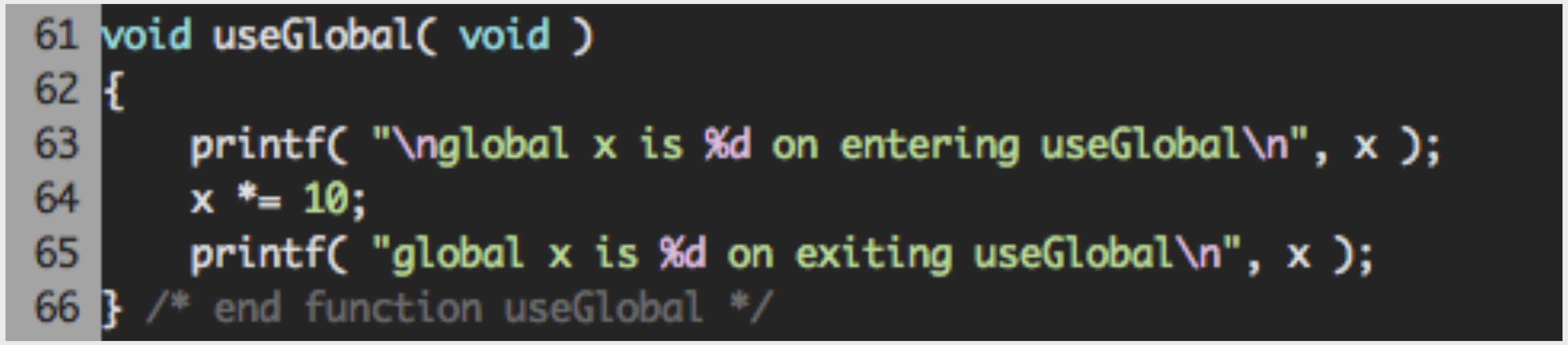
- Example: fig05_12.c



```c
void useLocal( void )
{
   int x = 25; /* initialized each time useLocal is called */

   printf( "\nlocal x in useLocal is %d after entering useLocal\n", x );
   x++;
   printf( "local x in useLocal is %d before exiting useLocal\n", x );
} /* end function useLocal */
```



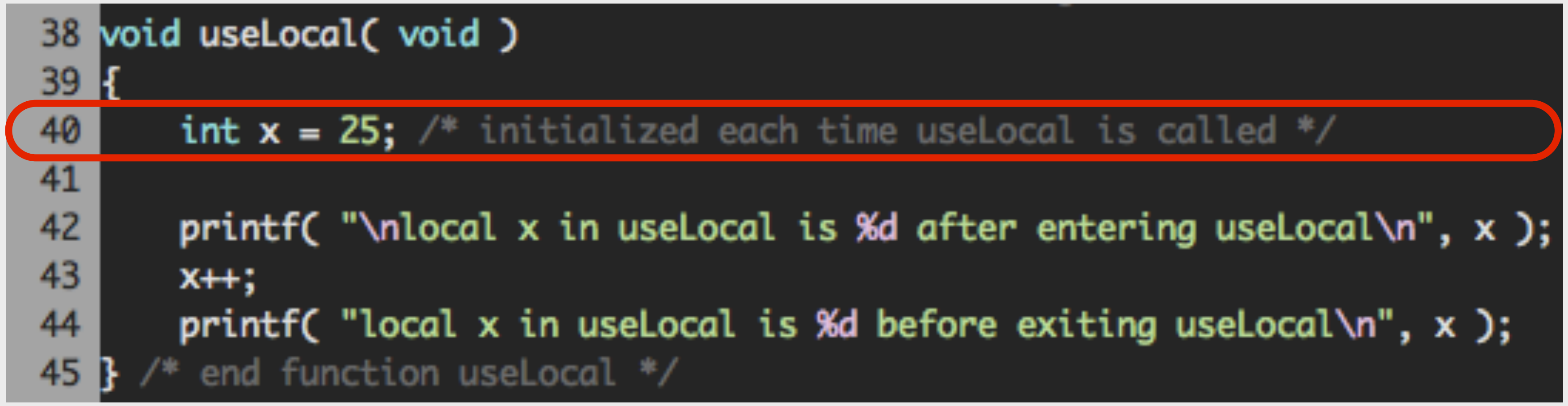
```c
void useStaticLocal( void )
{
   /* initialized only first time useStaticLocal is called */
   static int x = 50;

   printf( "\nlocal static x is %d on entering useStaticLocal\n", x );
   x++;
   printf( "local static x is %d on exiting useStaticLocal\n", x );
} /* end function useStaticLocal */
```



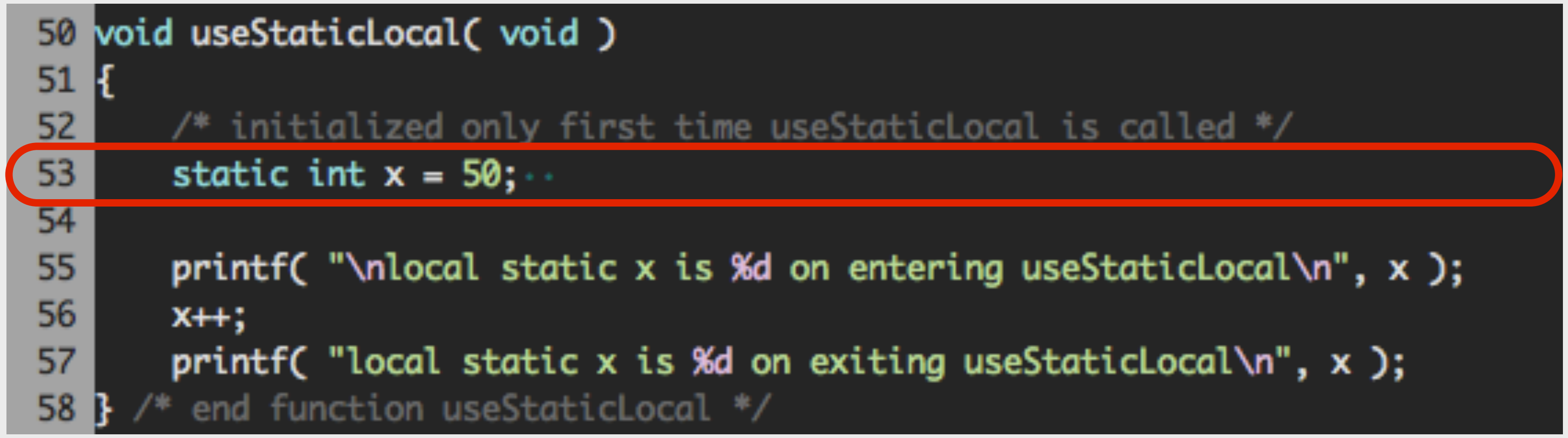
```c
void useGlobal( void )
{
   printf( "\nglobal x is %d on entering useGlobal\n", x );
   x *= 10;
   printf( "global x is %d on exiting useGlobal\n", x );
} /* end function useGlobal */
```

Scope Rules (Cont.)

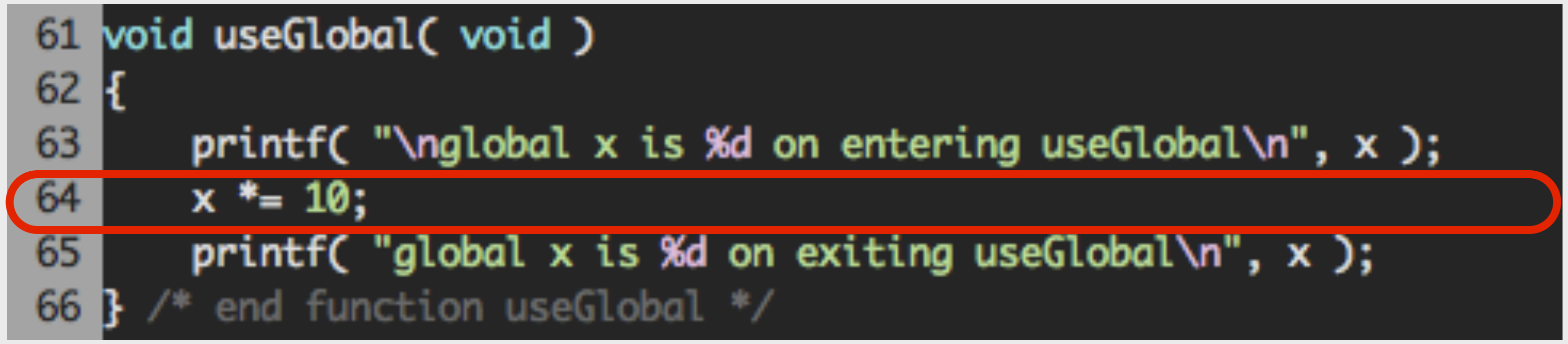
- Example: fig05_12.c



```c
void useLocal( void )
{
   int x = 25; /* initialized each time useLocal is called */

   printf( "\nlocal x in useLocal is %d after entering useLocal\n", x );
   x++;
   printf( "local x in useLocal is %d before exiting useLocal\n", x );
} /* end function useLocal */
```



```c
void useStaticLocal( void )
{
   /* initialized only first time useStaticLocal is called */
   static int x = 50;

   printf( "\nlocal static x is %d on entering useStaticLocal\n", x );
   x++;
   printf( "local static x is %d on exiting useStaticLocal\n", x );
} /* end function useStaticLocal */
```



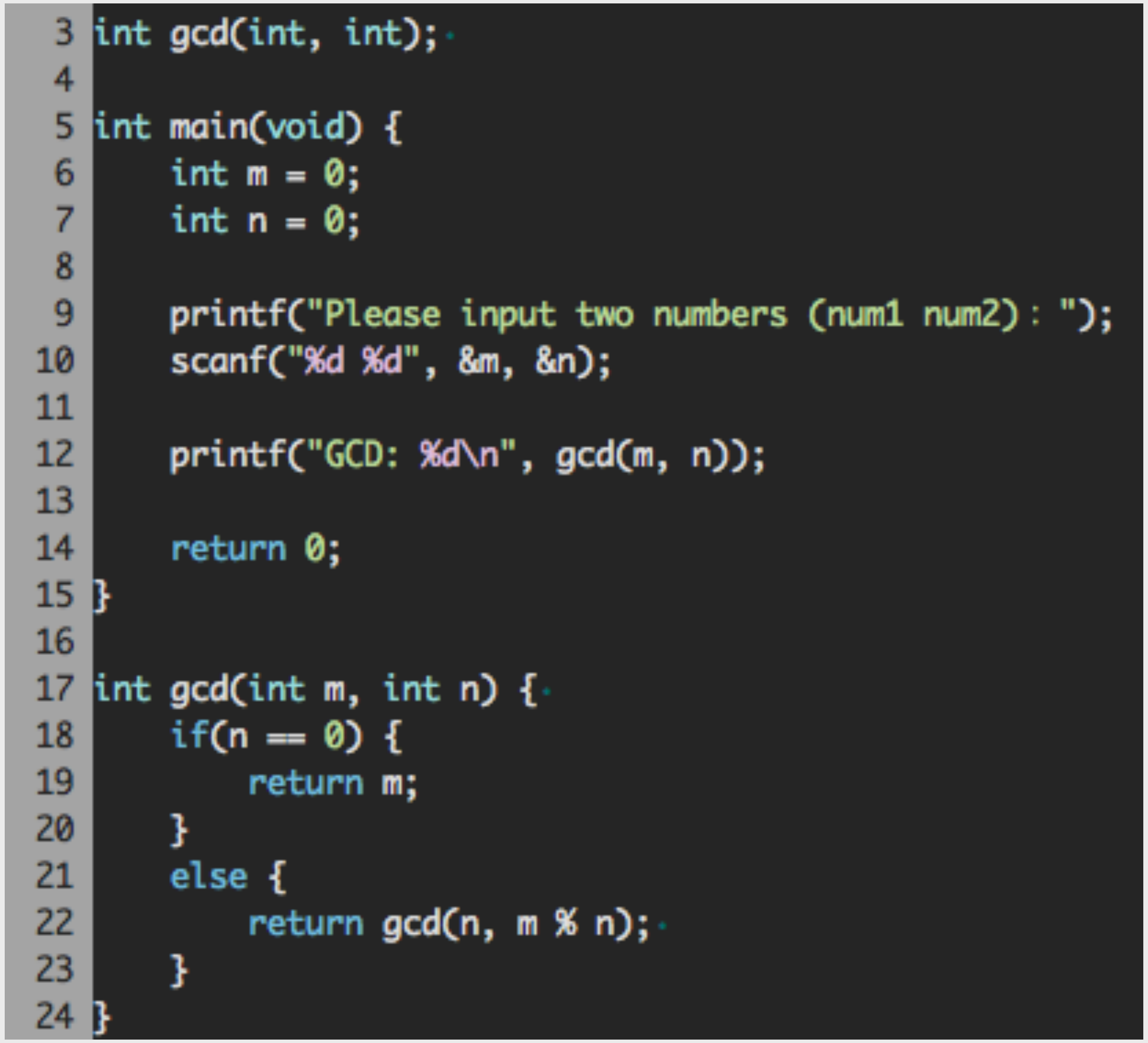
```c
void useGlobal( void )
{
   printf( "\nglobal x is %d on entering useGlobal\n", x );
   x *= 10;
   printf( "global x is %d on exiting useGlobal\n", x );
} /* end function useGlobal */
```

Recursion vs. Iteration

- Each **recursive** call causes another copy of the function (actually only the function’s variables) to be created; this can consume considerable memory.
- **Iteration** normally occurs within a function, so the overhead of repeated function calls and extra memory assignment is omitted.
- So why choose recursion?

Greatest common divisor

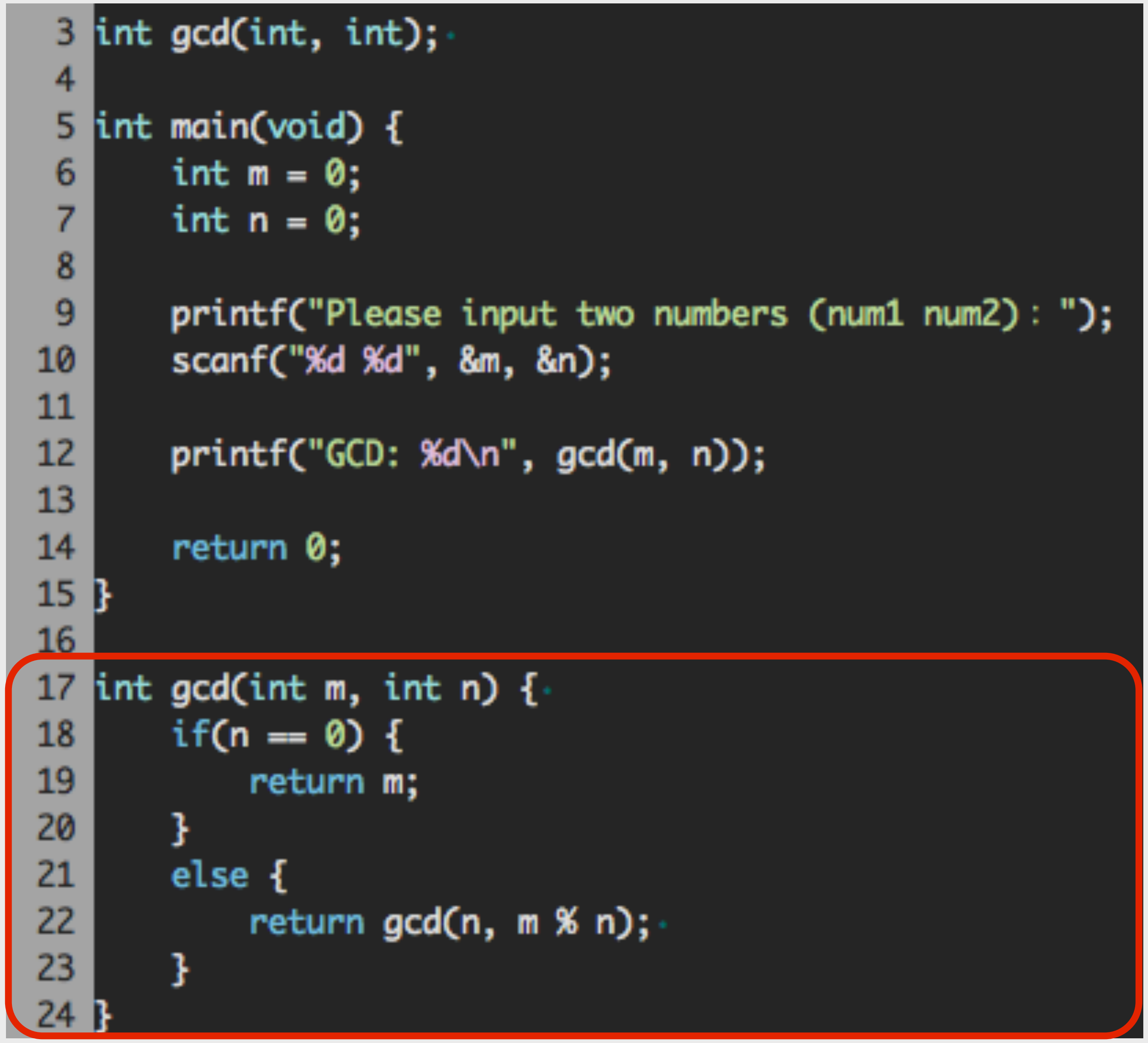
- Example: fig05_16-1.c



```c
int gcd(int, int);

int main(void) {
    int m = 0;
    int n = 0;

    printf("Please input two numbers (num1 num2): ");
    scanf("%d %d", &m, &n);

    printf("GCD: %d\n", gcd(m, n));

    return 0;
}

int gcd(int m, int n) {
    if(n == 0) {
        return m;
    }
    else {
        return gcd(n, m % n);
    }
}
```

Greatest common divisor

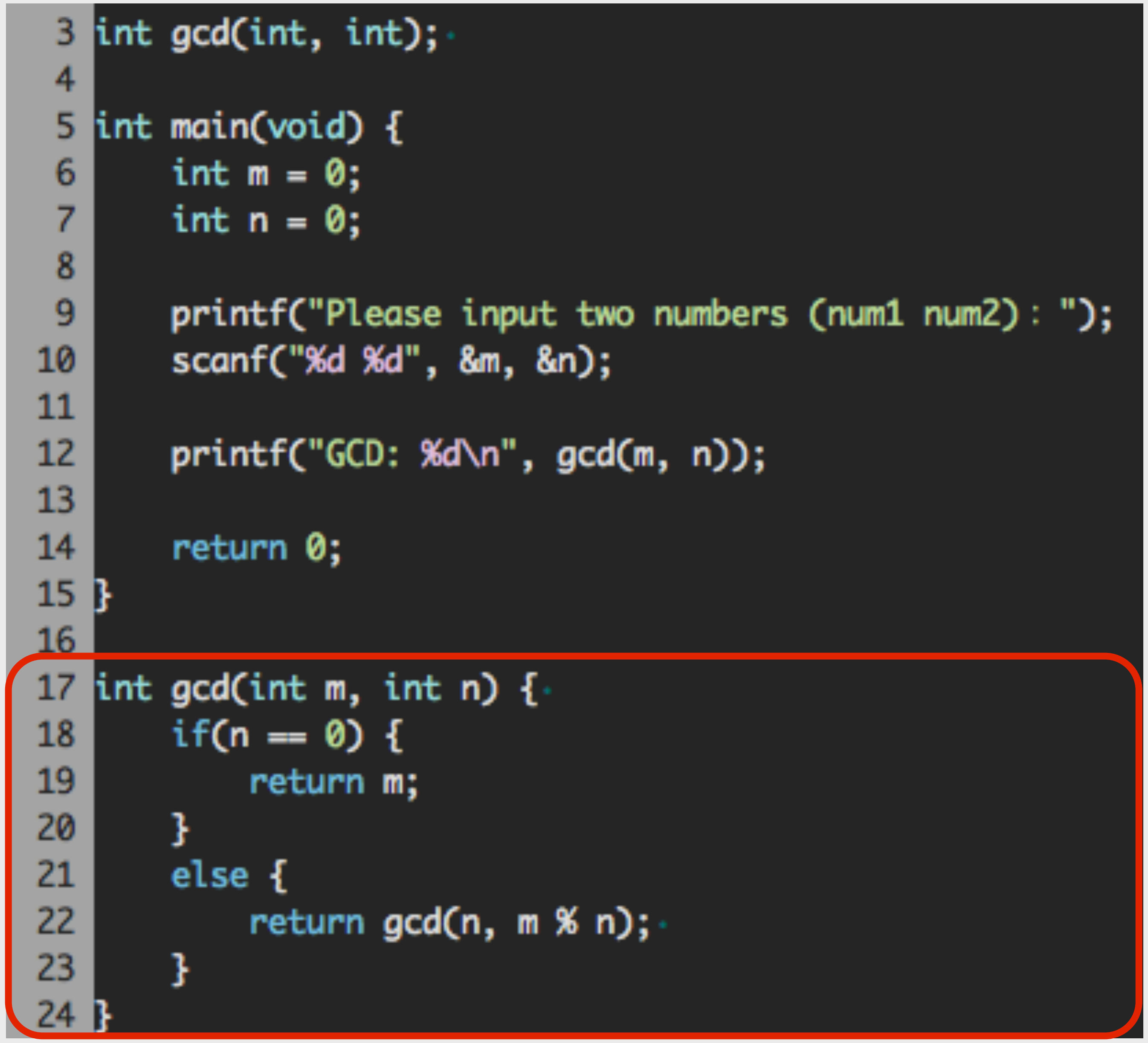
- Example: fig05_16-1.c



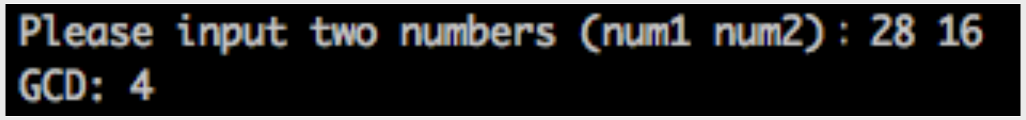
```c
int gcd(int, int);

int main(void) {
    int m = 0;
    int n = 0;

    printf("Please input two numbers (num1 num2): ");
    scanf("%d %d", &m, &n);

    printf("GCD: %d\n", gcd(m, n));

    return 0;
}

int gcd(int m, int n) {
    if(n == 0) {
        return m;
    }
    else {
        return gcd(n, m % n);
    }
}
```

Greatest common divisor

- Example: fig05_16-1.c



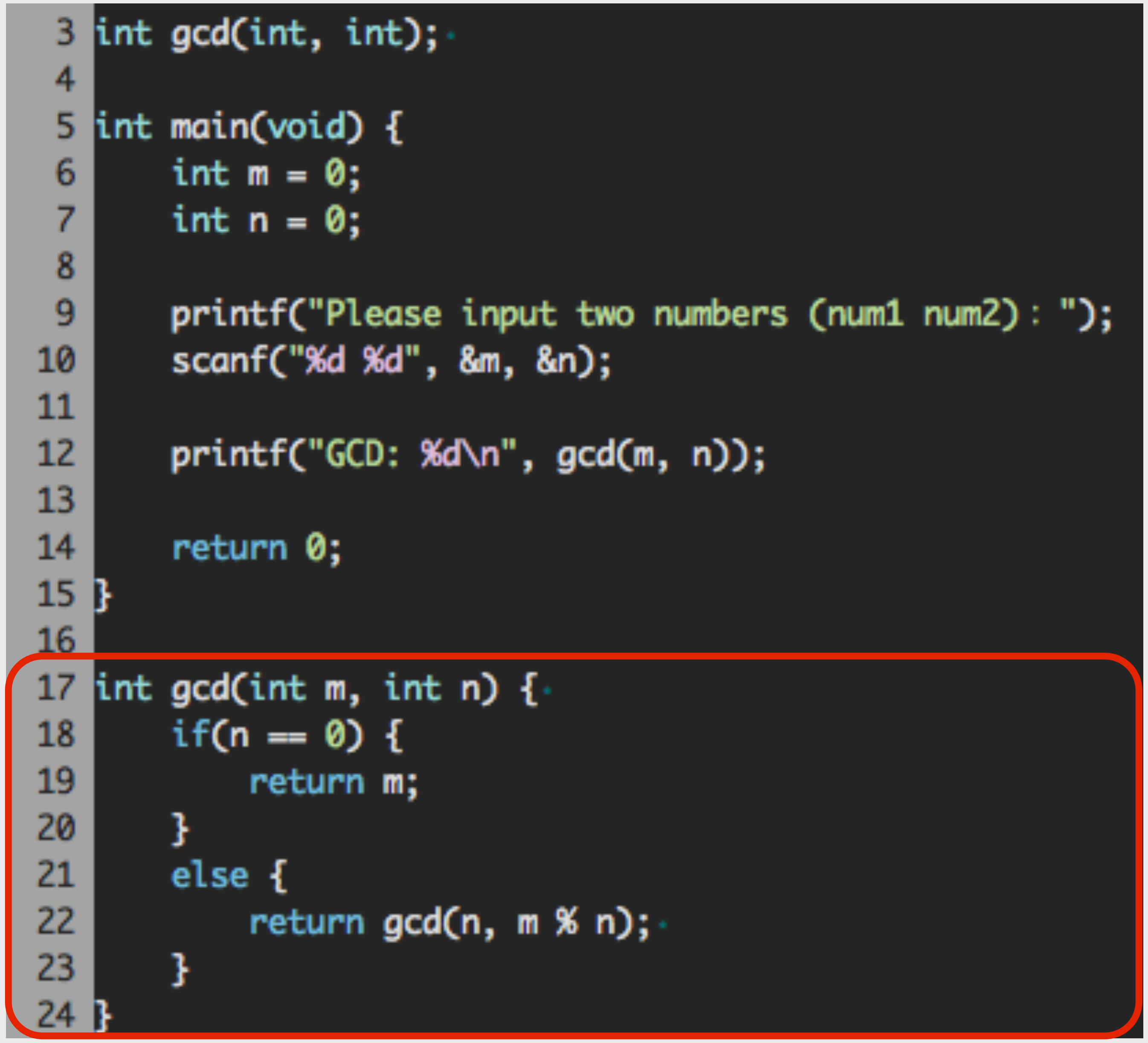
```c
int gcd(int, int);

int main(void) {
    int m = 0;
    int n = 0;

    printf("Please input two numbers (num1 num2): ");
    scanf("%d %d", &m, &n);

    printf("GCD: %d\n", gcd(m, n));

    return 0;
}

int gcd(int m, int n) {
    if(n == 0) {
        return m;
    }
    else {
        return gcd(n, m % n);
    }
}
```



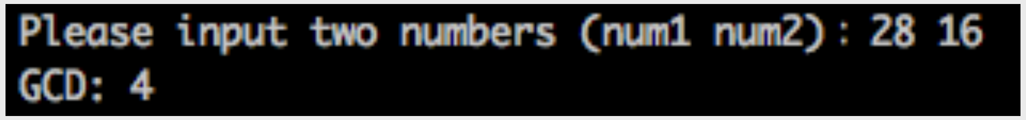
```
Please input two numbers (num1 num2): 28 16
GCD: 4
```

Greatest common divisor

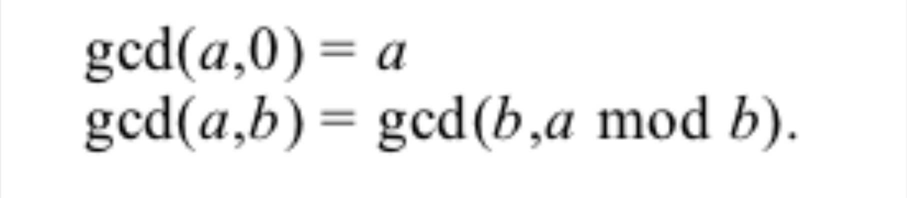
- Example: fig05_16-1.c



```c
int gcd(int, int);

int main(void) {
    int m = 0;
    int n = 0;

    printf("Please input two numbers (num1 num2): ");
    scanf("%d %d", &m, &n);

    printf("GCD: %d\n", gcd(m, n));

    return 0;
}

int gcd(int m, int n) {
    if(n == 0) {
        return m;
    }
    else {
        return gcd(n, m % n);
    }
}
```



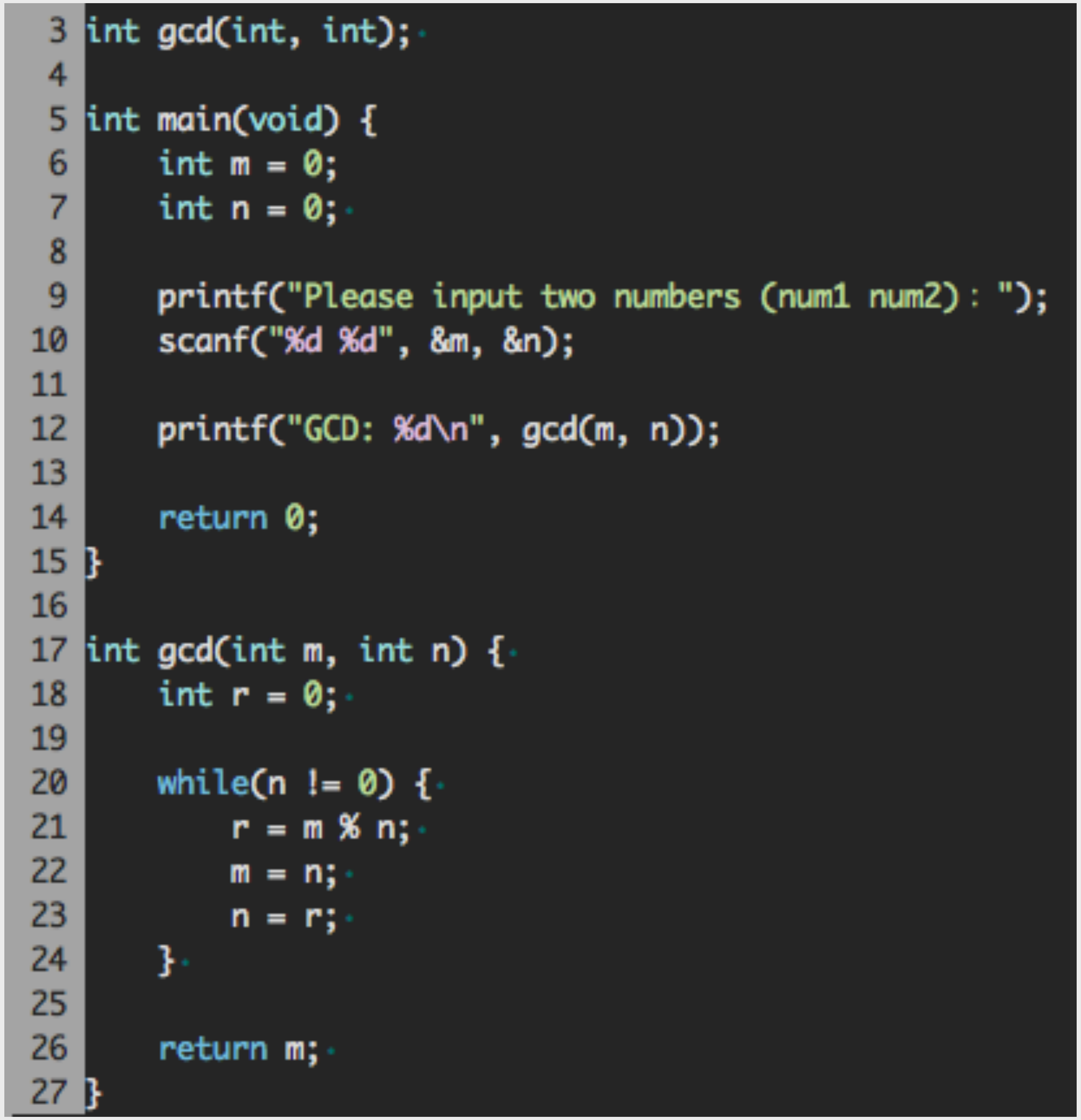
```
Please input two numbers (num1 num2): 28 16
GCD: 4
```



$$\gcd(a,0) = a$$
$$\gcd(a,b) = \gcd(b, a \bmod b).$$

Greatest common divisor

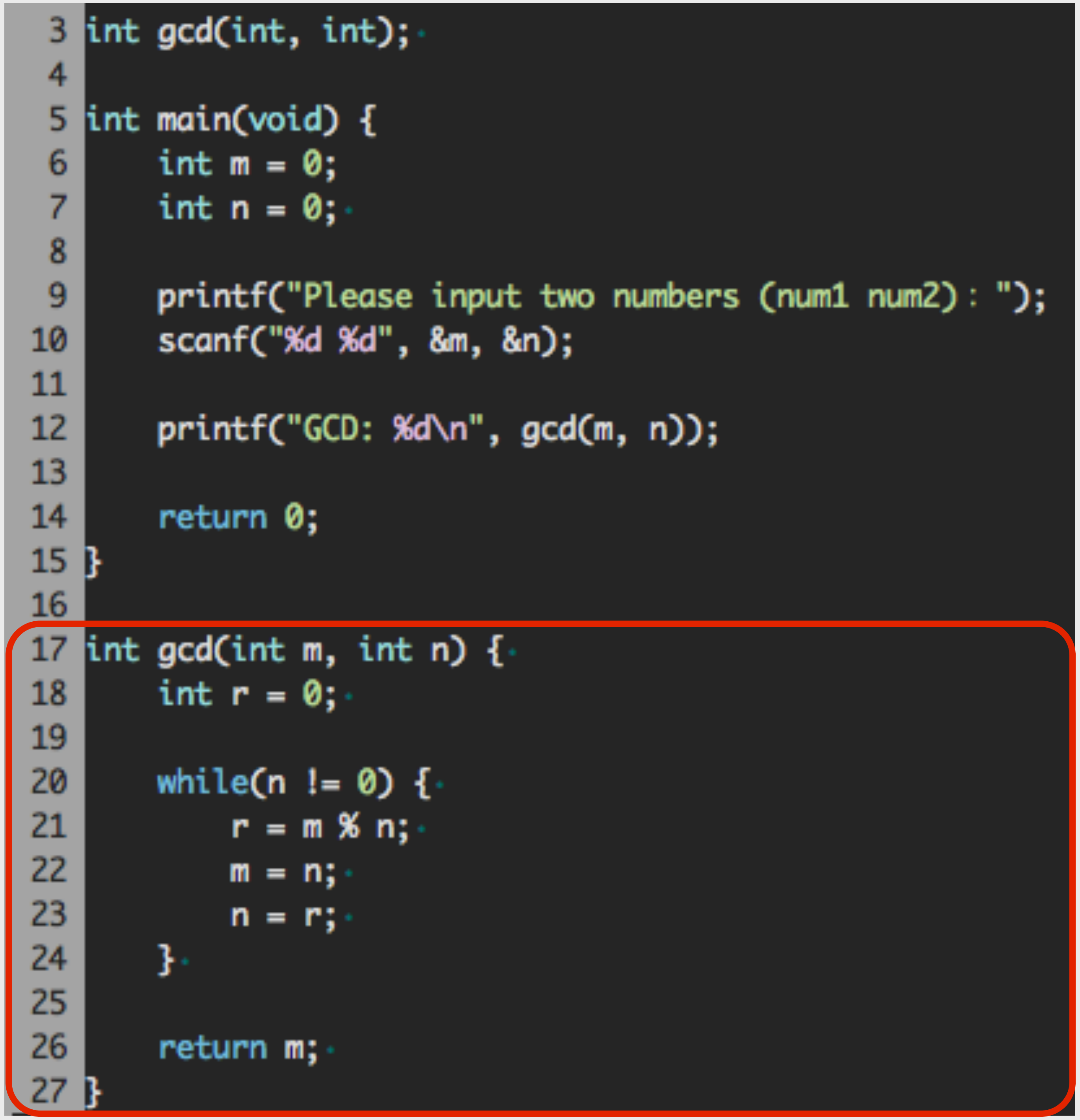
- Example: fig05_16-2.c



```c
int gcd(int, int);

int main(void) {
    int m = 0;
    int n = 0;

    printf("Please input two numbers (num1 num2)： ");
    scanf("%d %d", &m, &n);

    printf("GCD: %d\n", gcd(m, n));

    return 0;
}

int gcd(int m, int n) {
    int r = 0;

    while(n != 0) {
        r = m % n;
        m = n;
        n = r;
    }

    return m;
}
```

Greatest common divisor

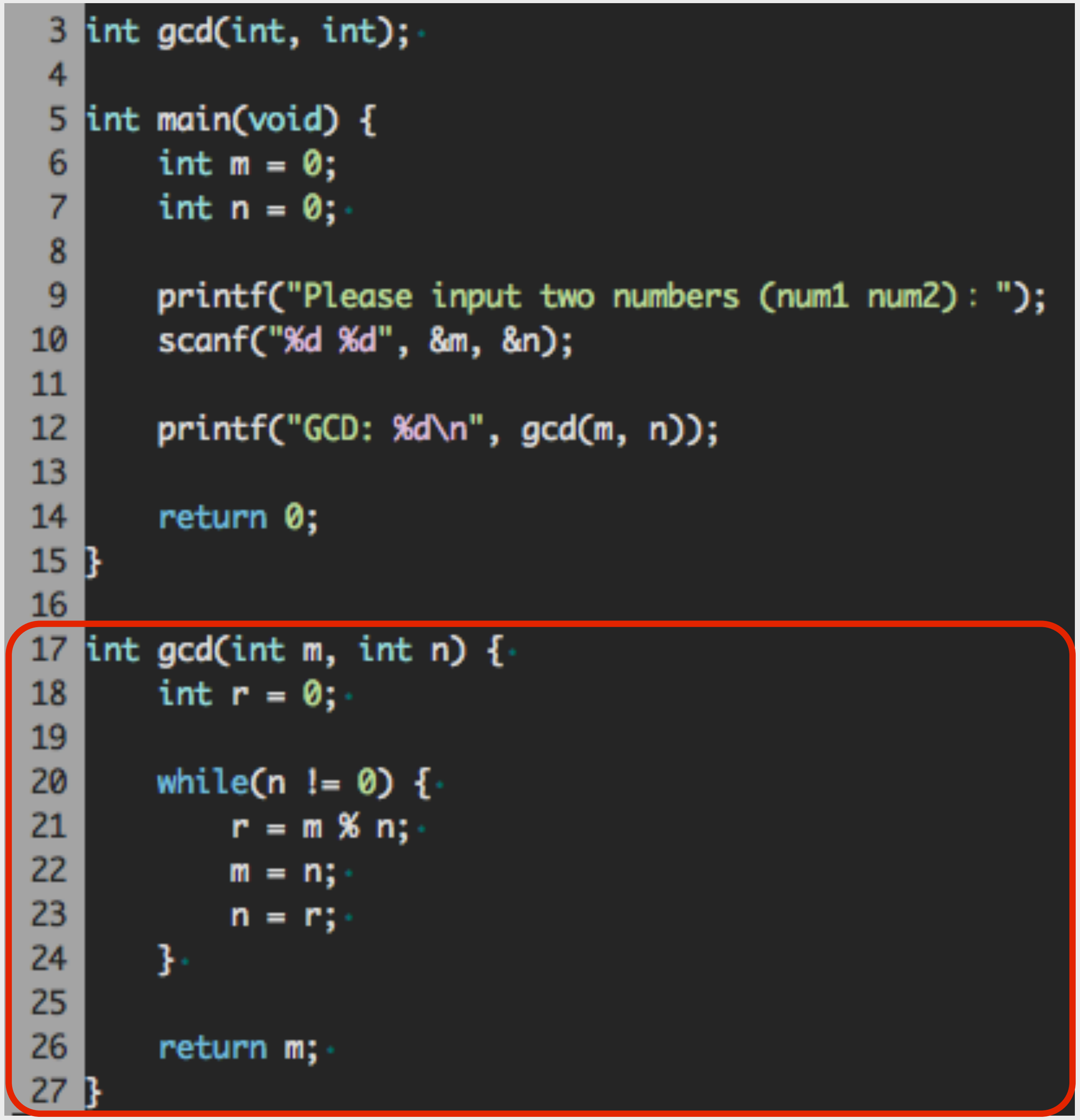
- Example: fig05_16-2.c



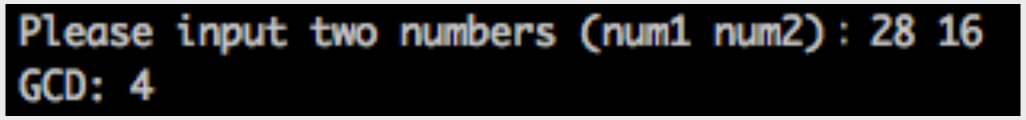
```c
int gcd(int, int);

int main(void) {
    int m = 0;
    int n = 0;

    printf("Please input two numbers (num1 num2): ");
    scanf("%d %d", &m, &n);

    printf("GCD: %d\n", gcd(m, n));

    return 0;
}

int gcd(int m, int n) {
    int r = 0;

    while(n != 0) {
        r = m % n;
        m = n;
        n = r;
    }

    return m;
}
```

Greatest common divisor

- Example: fig05_16-2.c

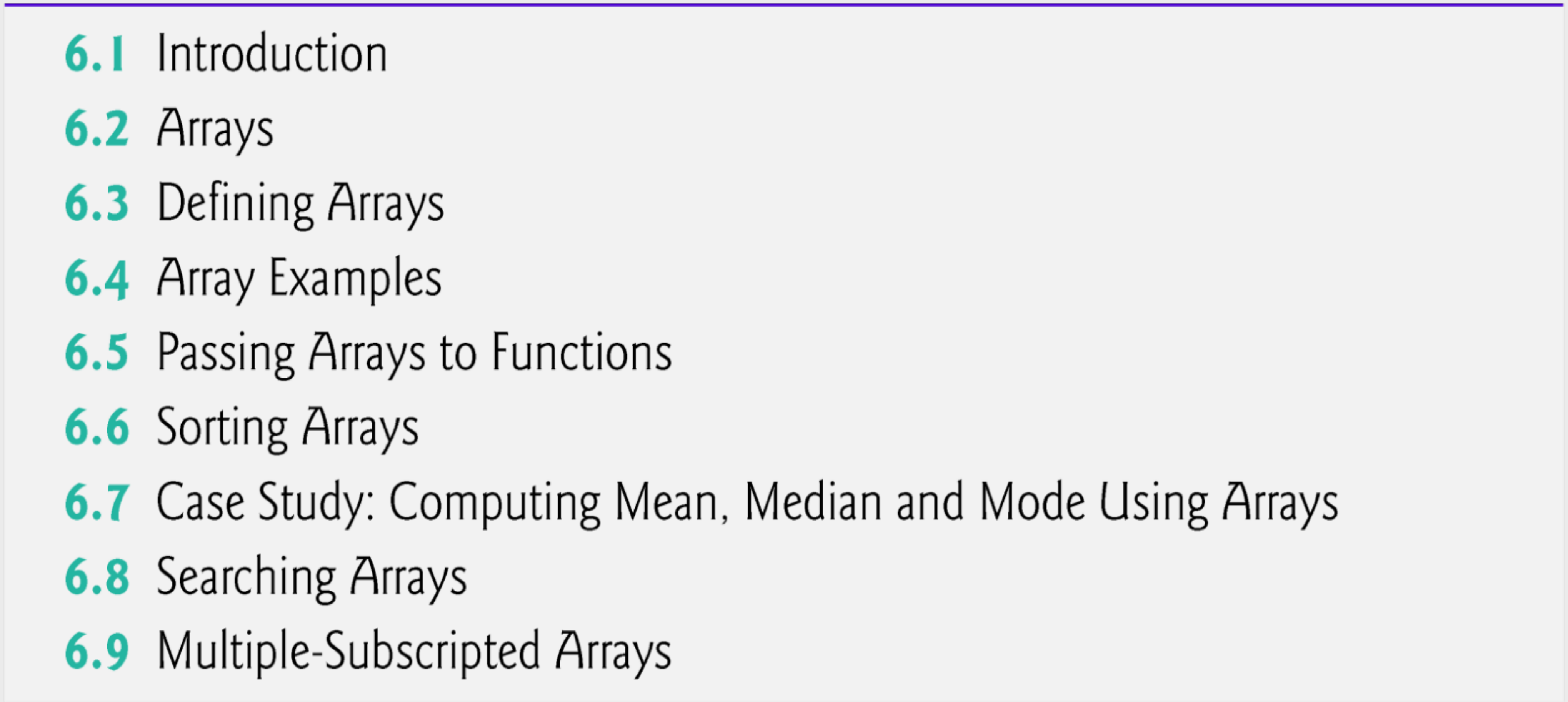


```c
int gcd(int, int);

int main(void) {
    int m = 0;
    int n = 0;

    printf("Please input two numbers (num1 num2): ");
    scanf("%d %d", &m, &n);

    printf("GCD: %d\n", gcd(m, n));

    return 0;
}

int gcd(int m, int n) {
    int r = 0;

    while(n != 0) {
        r = m % n;
        m = n;
        n = r;
    }

    return m;
}
```



```
Please input two numbers (num1 num2): 28 16
GCD: 4
```

C Arrays



6.1 Introduction

6.2 Arrays

6.3 Defining Arrays

6.4 Array Examples

6.5 Passing Arrays to Functions

6.6 Sorting Arrays

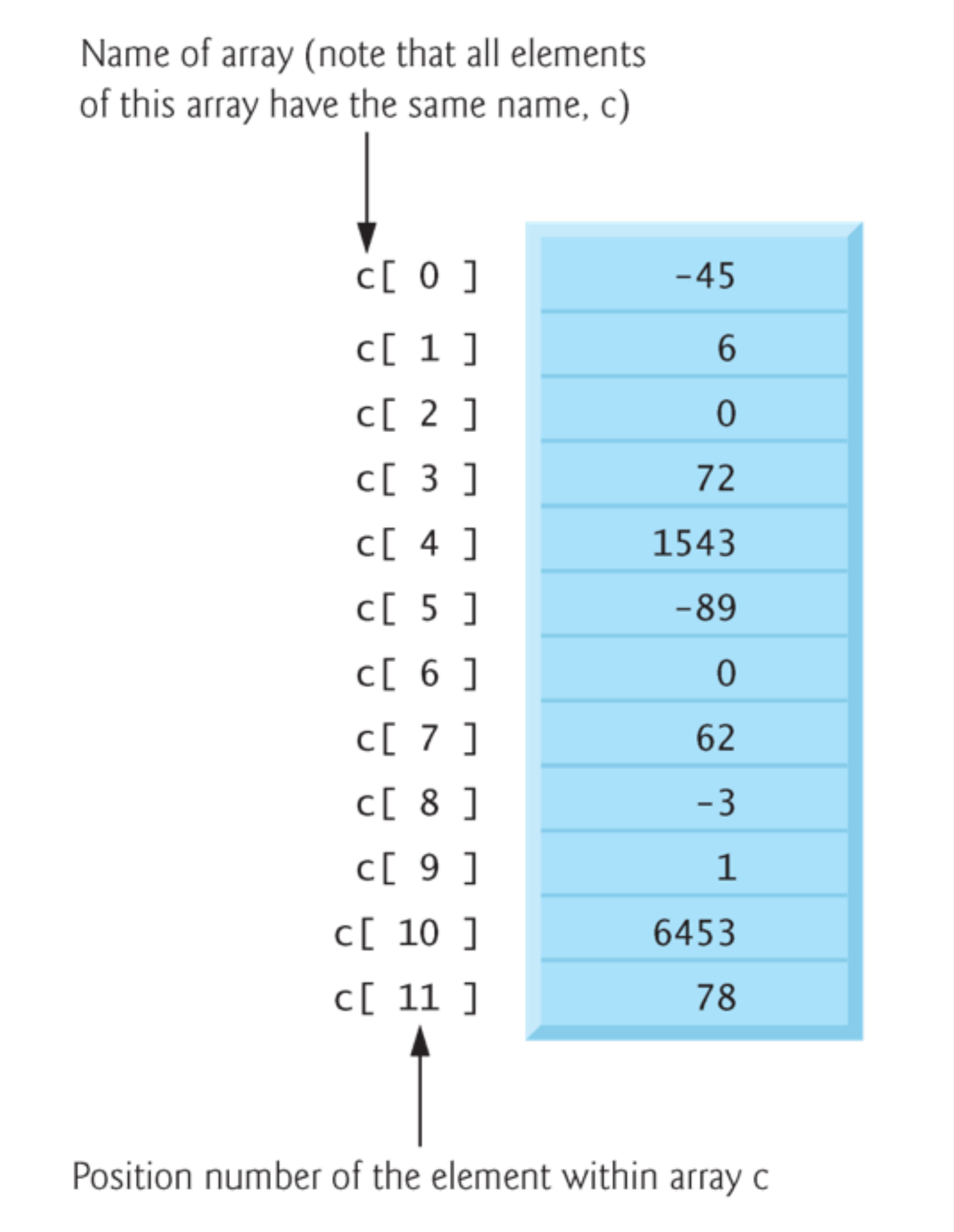
6.7 Case Study: Computing Mean, Median and Mode Using Arrays

6.8 Searching Arrays

6.9 Multiple-Subscripted Arrays

Arrays

- An array is **a group of memory** locations related by the fact that they all have **the same name and the same type**.
- To refer to a particular location or element in the array, we specify **the name of the array** and **the position number** of the particular element in the array.
- The position number contained within square brackets is more formally called a **subscript** (or **index**).



Name of array (note that all elements of this array have the same name, c)

c[0]	-45
c[1]	6
c[2]	0
c[3]	72
c[4]	1543
c[5]	-89
c[6]	0
c[7]	62
c[8]	-3
c[9]	1
c[10]	6453
c[11]	78

Position number of the element within array c

Defining Arrays

- Arrays occupy space in memory.
- To tell the computer to reserve 12 elements for **integer array c**, use the definition

```
int c[12];
```

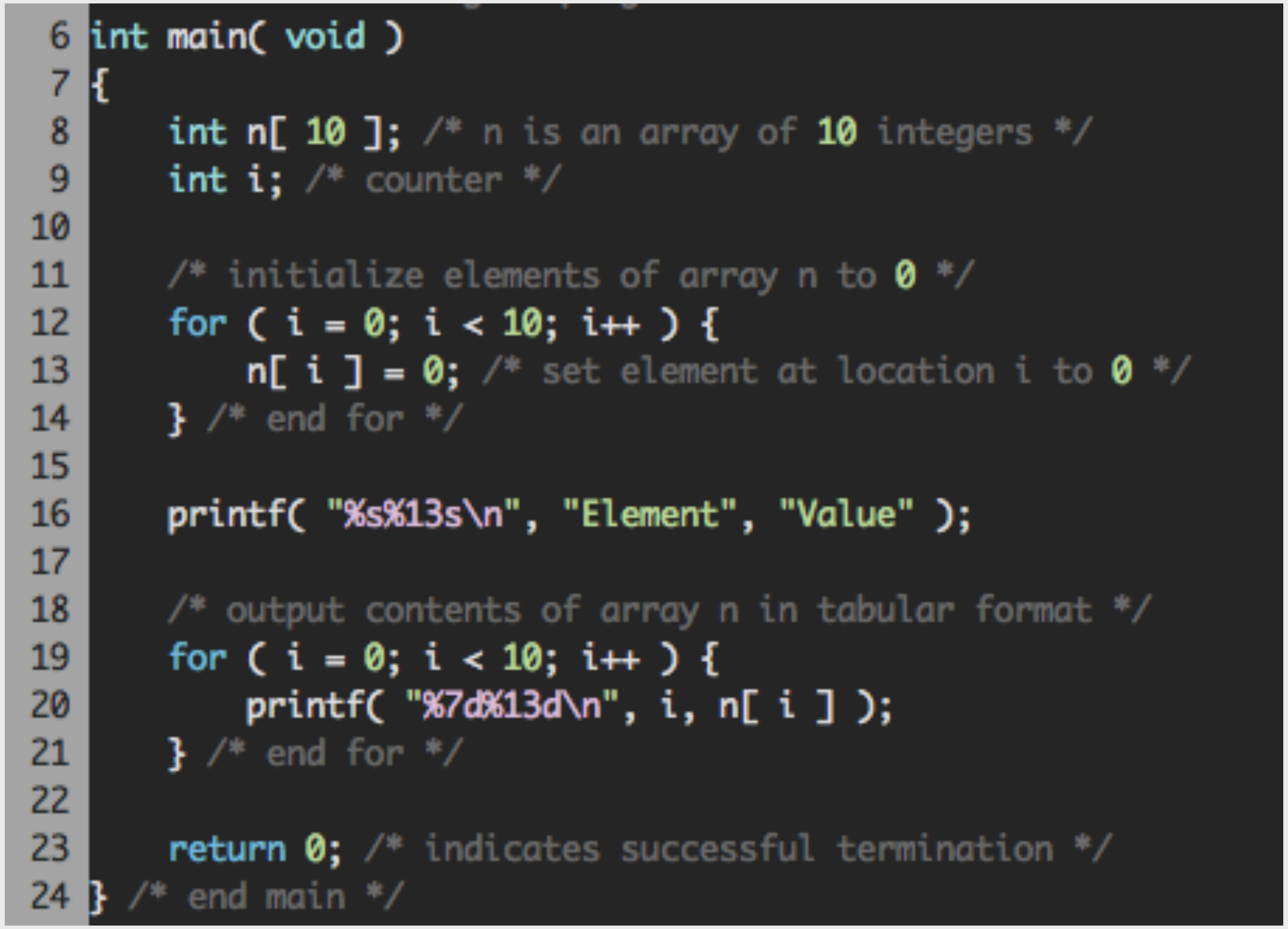
- The following definition

```
int b[ 100 ], x[ 27 ];
```

  - reserves 100 elements for integer array b and 27 elements for integer array x.

Array Examples

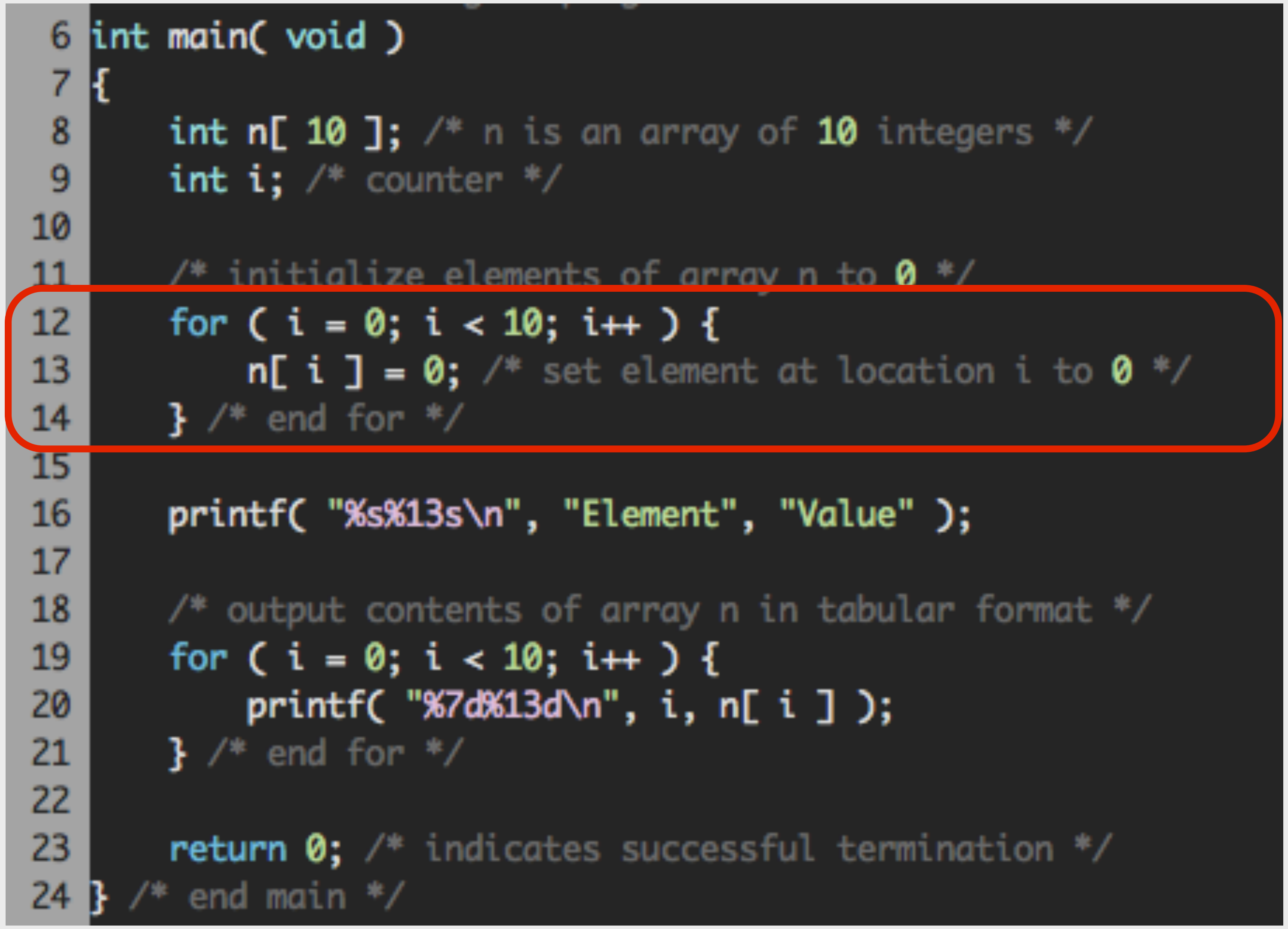
- Example: fig06_03.c



```c
int main( void )
{
    int n[ 10 ]; /* n is an array of 10 integers */
    int i; /* counter */

    /* initialize elements of array n to 0 */
    for ( i = 0; i < 10; i++ ) {
        n[ i ] = 0; /* set element at location i to 0 */
    } /* end for */

    printf( "%s%13s\n", "Element", "Value" );

    /* output contents of array n in tabular format */
    for ( i = 0; i < 10; i++ ) {
        printf( "%7d%13d\n", i, n[ i ] );
    } /* end for */

    return 0; /* indicates successful termination */
} /* end main */
```

Array Examples

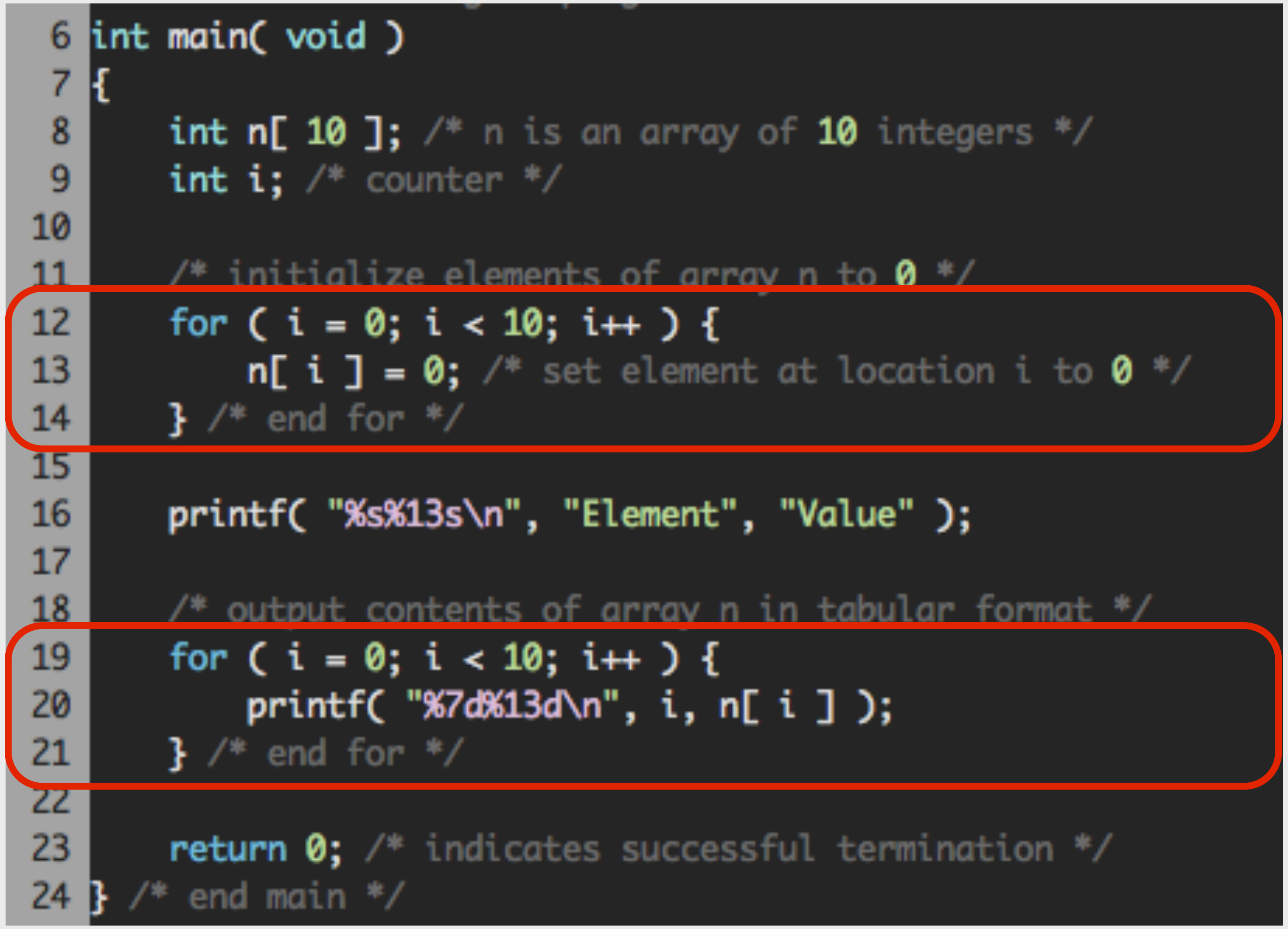
- Example: fig06_03.c



```
int main( void )
{
    int n[ 10 ]; /* n is an array of 10 integers */
    int i; /* counter */

    /* initialize elements of array n to 0 */
    for ( i = 0; i < 10; i++ ) {
        n[ i ] = 0; /* set element at location i to 0 */
    } /* end for */

    printf( "%s%13s\n", "Element", "Value" );

    /* output contents of array n in tabular format */
    for ( i = 0; i < 10; i++ ) {
        printf( "%7d%13d\n", i, n[ i ] );
    } /* end for */

    return 0; /* indicates successful termination */
} /* end main */
```

Array Examples

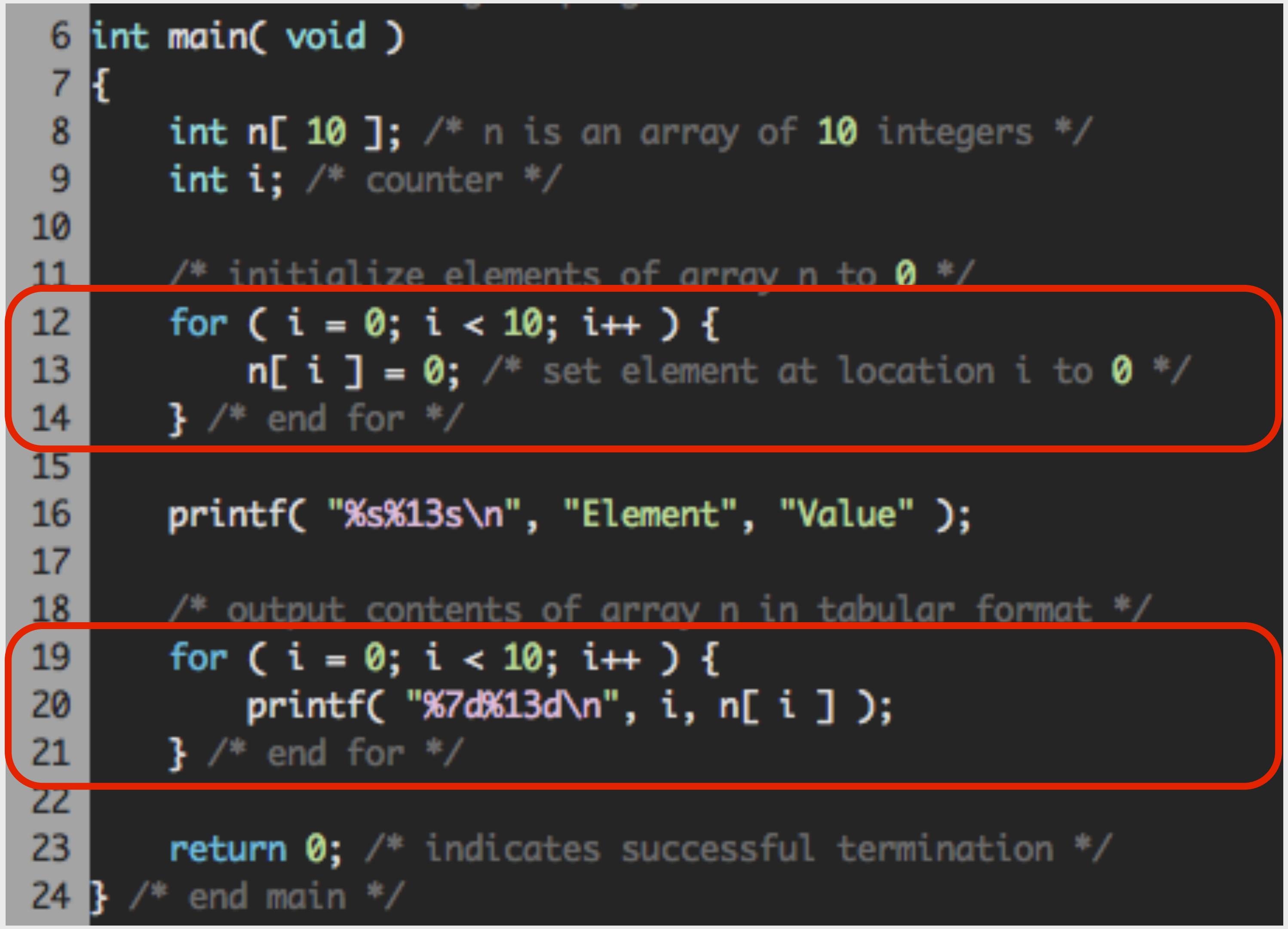
- Example: fig06_03.c



```c
int main( void )
{
    int n[ 10 ]; /* n is an array of 10 integers */
    int i; /* counter */

    /* initialize elements of array n to 0 */
    for ( i = 0; i < 10; i++ ) {
        n[ i ] = 0; /* set element at location i to 0 */
    } /* end for */

    printf( "%s%13s\n", "Element", "Value" );

    /* output contents of array n in tabular format */
    for ( i = 0; i < 10; i++ ) {
        printf( "%7d%13d\n", i, n[ i ] );
    } /* end for */

    return 0; /* indicates successful termination */
} /* end main */
```

Array Examples

- Example: fig06_03.c

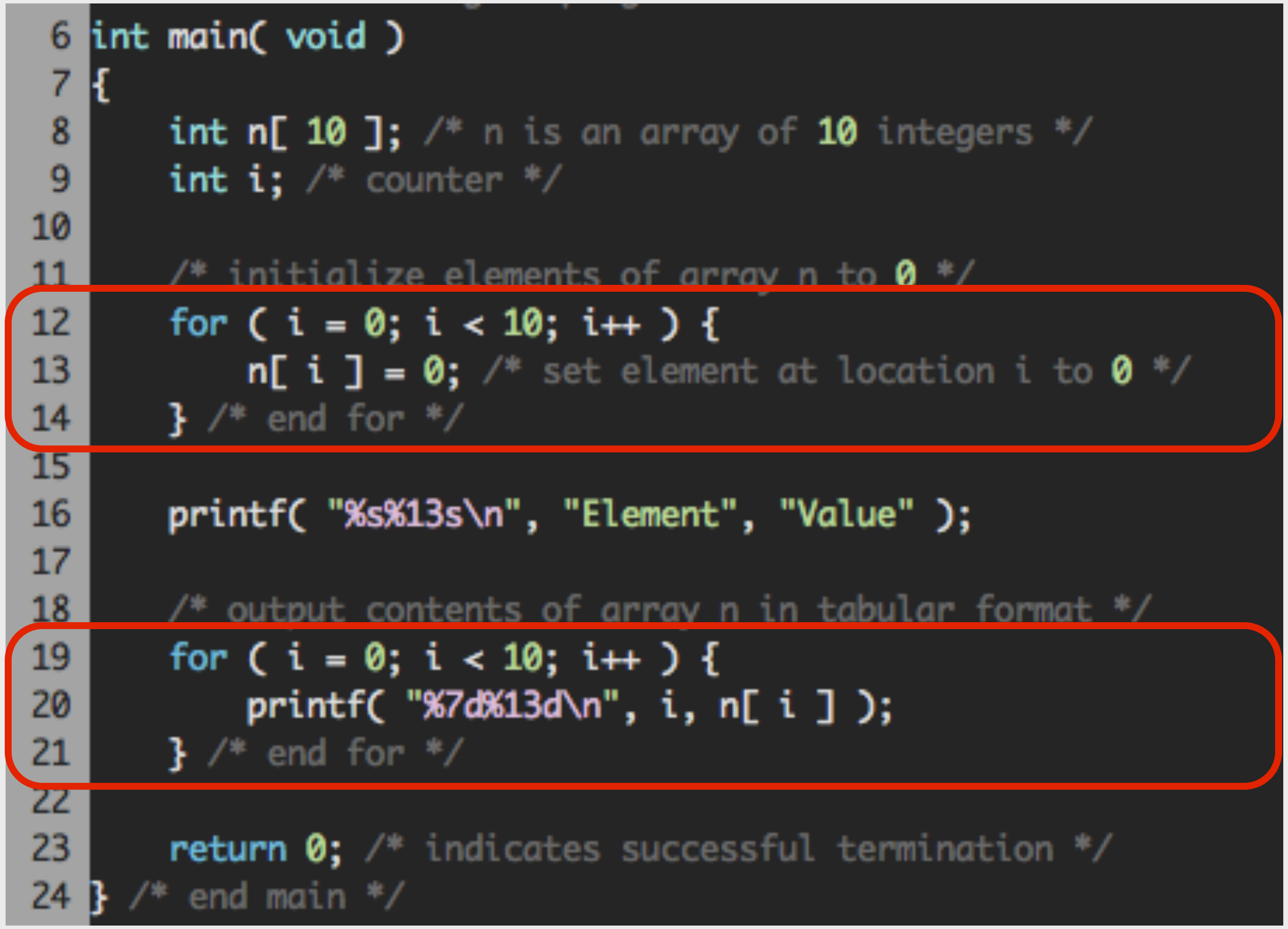


```
int main( void )
{
    int n[ 10 ]; /* n is an array of 10 integers */
    int i; /* counter */

    /* initialize elements of array n to 0 */
    for ( i = 0; i < 10; i++ ) {
        n[ i ] = 0; /* set element at location i to 0 */
    } /* end for */

    printf( "%s%13s\n", "Element", "Value" );

    /* output contents of array n in tabular format */
    for ( i = 0; i < 10; i++ ) {
        printf( "%7d%13d\n", i, n[ i ] );
    } /* end for */

    return 0; /* indicates successful termination */
} /* end main */
```

use for loop to initialize the array

Array Examples

- Example: fig06_03.c



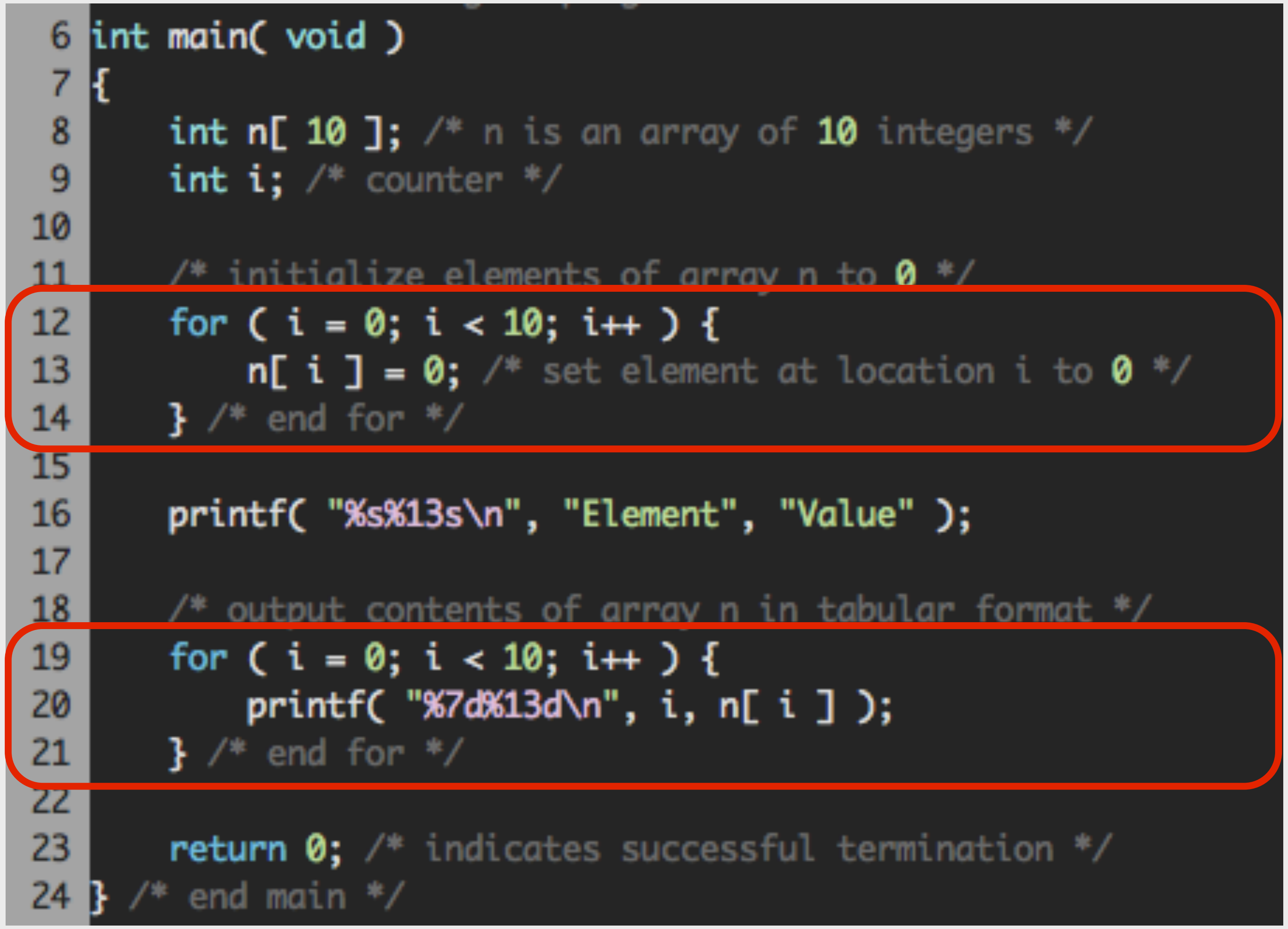
```
int main( void )
{
    int n[ 10 ]; /* n is an array of 10 integers */
    int i; /* counter */

    /* initialize elements of array n to 0 */
    for ( i = 0; i < 10; i++ ) {
        n[ i ] = 0; /* set element at location i to 0 */
    } /* end for */

    printf( "%s%13s\n", "Element", "Value" );

    /* output contents of array n in tabular format */
    for ( i = 0; i < 10; i++ ) {
        printf( "%7d%13d\n", i, n[ i ] );
    } /* end for */

    return 0; /* indicates successful termination */
} /* end main */
```

use for loop to initialize the array

use for loop to access the array

Array Examples

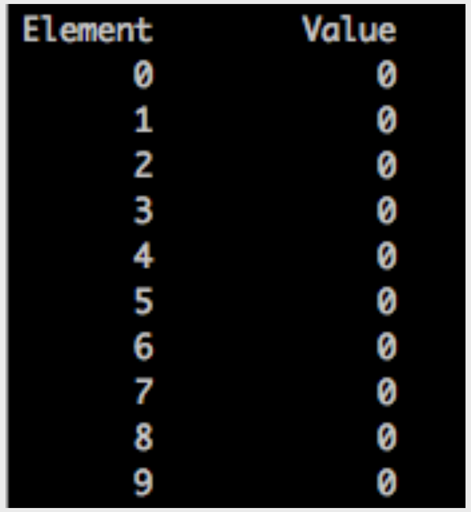
- Example: fig06_03.c



```c
int main( void )
{
    int n[ 10 ]; /* n is an array of 10 integers */
    int i; /* counter */

    /* initialize elements of array n to 0 */
    for ( i = 0; i < 10; i++ ) {
        n[ i ] = 0; /* set element at location i to 0 */
    } /* end for */

    printf( "%s%13s\n", "Element", "Value" );

    /* output contents of array n in tabular format */
    for ( i = 0; i < 10; i++ ) {
        printf( "%7d%13d\n", i, n[ i ] );
    } /* end for */

    return 0; /* indicates successful termination */
} /* end main */
```

use for loop to initialize the array

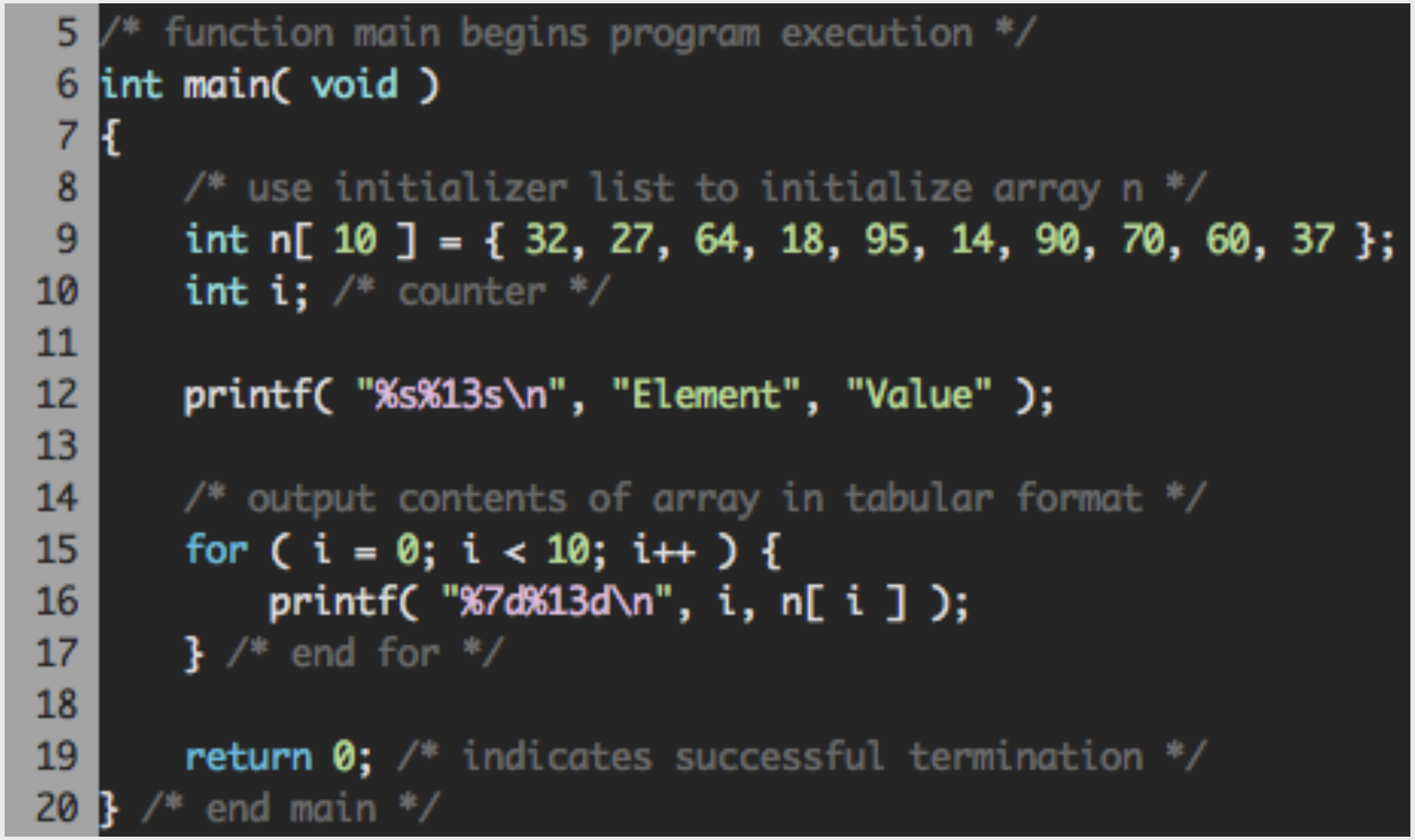
use for loop to access the array



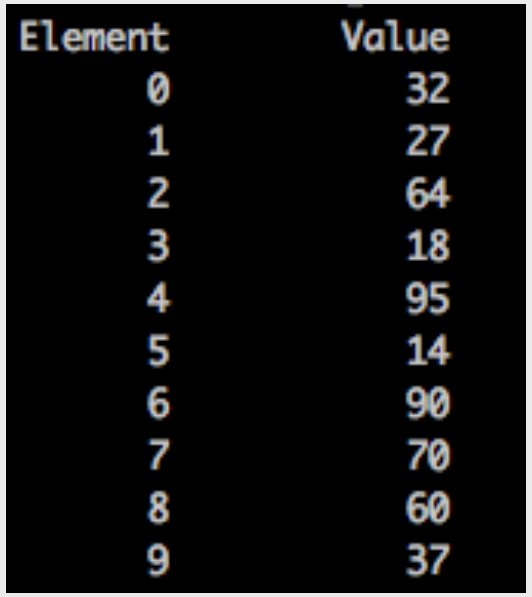
```
Element        Value
      0            0
      1            0
      2            0
      3            0
      4            0
      5            0
      6            0
      7            0
      8            0
      9            0
```

Array Examples (Cont.)

- Example: fig06_04.c



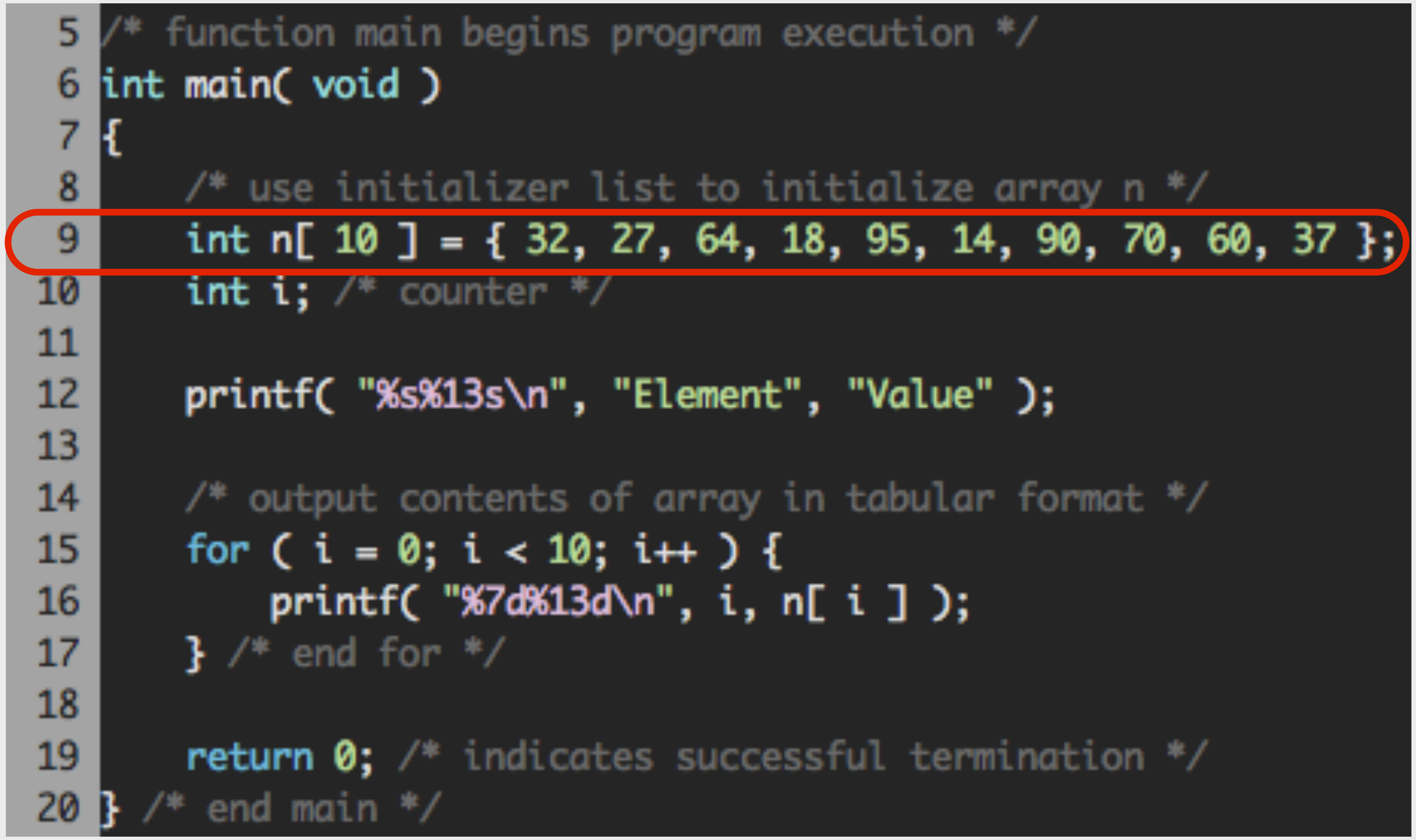
```c
/* function main begins program execution */
int main( void )
{
    /* use initializer list to initialize array n */
    int n[ 10 ] = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };
    int i; /* counter */

    printf( "%s%13s\n", "Element", "Value" );

    /* output contents of array in tabular format */
    for ( i = 0; i < 10; i++ ) {
        printf( "%7d%13d\n", i, n[ i ] );
    } /* end for */

    return 0; /* indicates successful termination */
} /* end main */
```



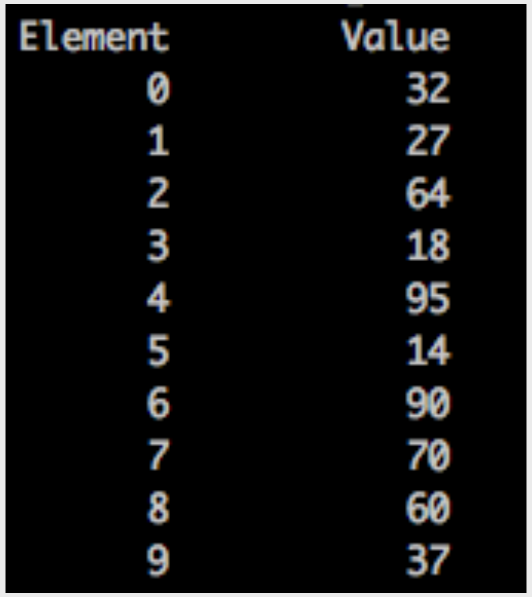
```
Element        Value
      0           32
      1           27
      2           64
      3           18
      4           95
      5           14
      6           90
      7           70
      8           60
      9           37
```

Array Examples (Cont.)

- Example: fig06_04.c



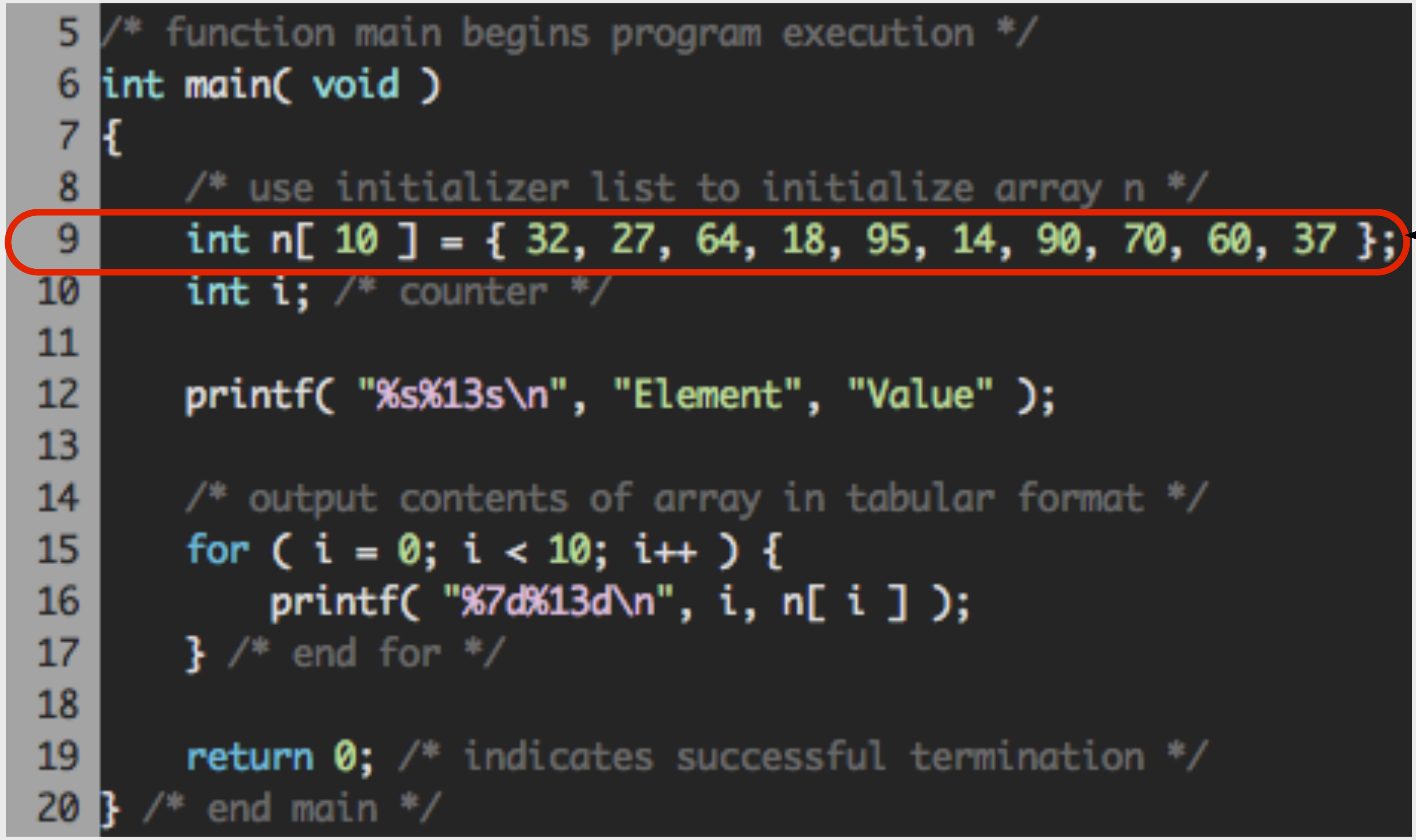
```c
/* function main begins program execution */
int main( void )
{
    /* use initializer list to initialize array n */
    int n[ 10 ] = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };
    int i; /* counter */

    printf( "%s%13s\n", "Element", "Value" );

    /* output contents of array in tabular format */
    for ( i = 0; i < 10; i++ ) {
        printf( "%7d%13d\n", i, n[ i ] );
    } /* end for */

    return 0; /* indicates successful termination */
} /* end main */
```



```
Element        Value
      0           32
      1           27
      2           64
      3           18
      4           95
      5           14
      6           90
      7           70
      8           60
      9           37
```

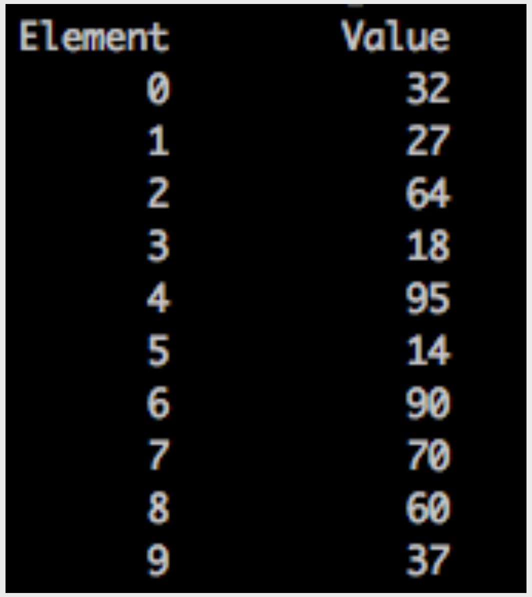
Array Examples (Cont.)

- Example: fig06_04.c



```c
/* function main begins program execution */
int main( void )
{
    /* use initializer list to initialize array n */
    int n[ 10 ] = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };
    int i; /* counter */

    printf( "%s%13s\n", "Element", "Value" );

    /* output contents of array in tabular format */
    for ( i = 0; i < 10; i++ ) {
        printf( "%7d%13d\n", i, n[ i ] );
    } /* end for */

    return 0; /* indicates successful termination */
} /* end main */
```

use { } to initialize values



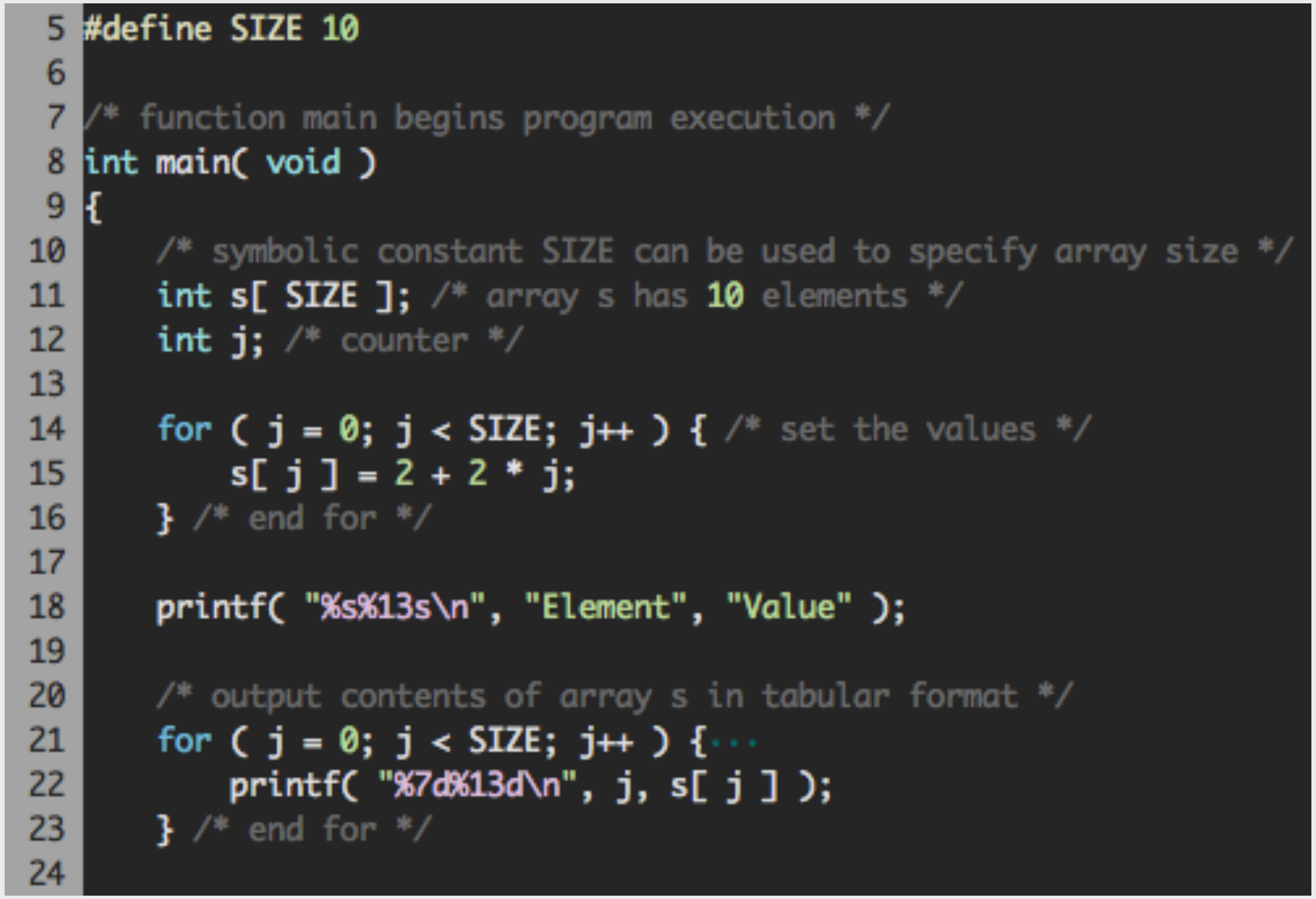
```
Element        Value
      0           32
      1           27
      2           64
      3           18
      4           95
      5           14
      6           90
      7           70
      8           60
      9           37
```

Array Examples (Cont.)

- **`int n[ 10 ] = { 0 };`**
  - initialize to all zeros
- **`int n[5] = {32,27,64,18,95,14};`**
  - syntax error
- **`int n[] = { 1, 2, 3, 4, 5 };`**
  - array size can be omitted if using {} to initialize

Array Examples (Cont.)

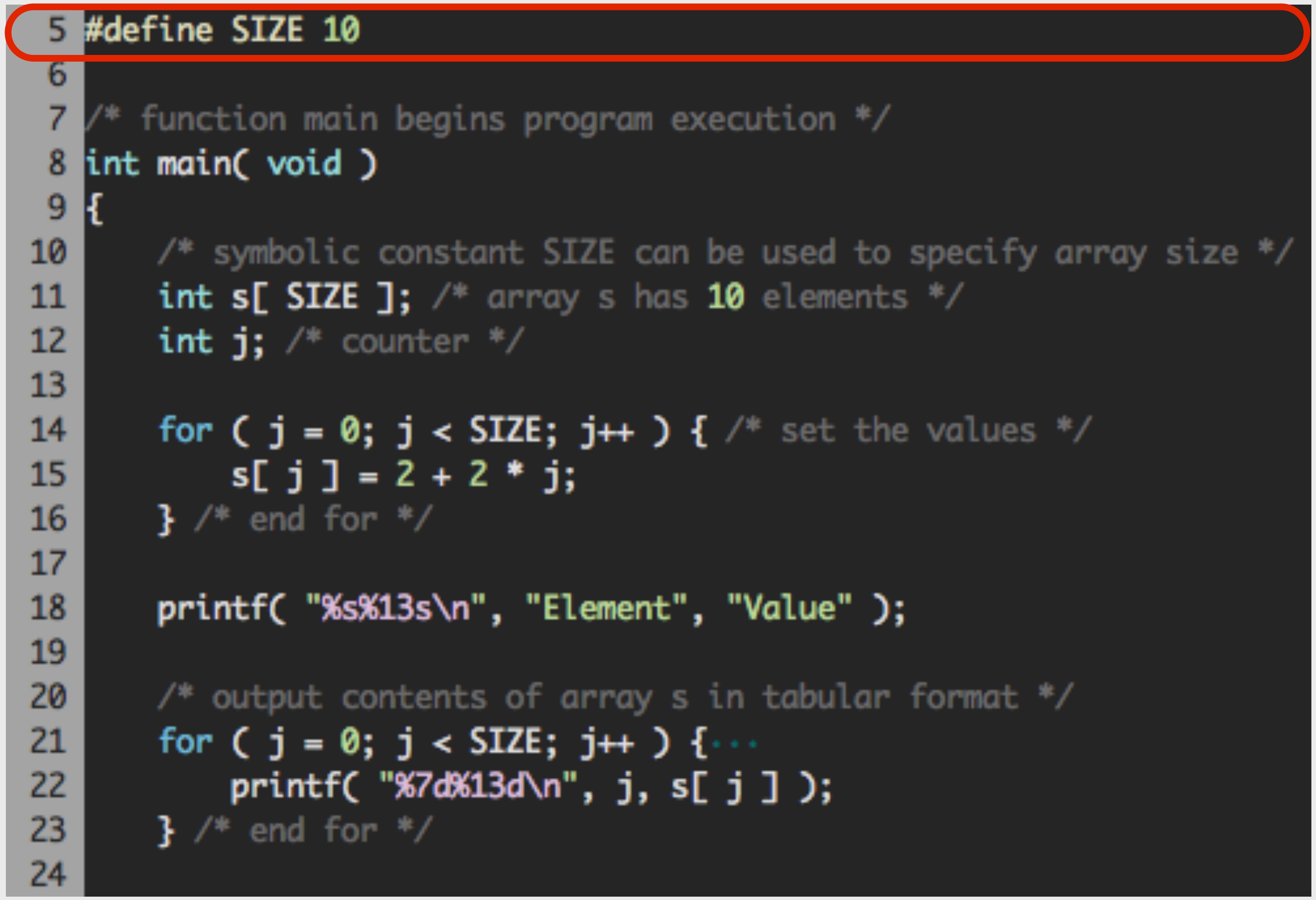
- Example: fig06_05.c



```c
#define SIZE 10

/* function main begins program execution */
int main( void )
{
    /* symbolic constant SIZE can be used to specify array size */
    int s[ SIZE ]; /* array s has 10 elements */
    int j; /* counter */

    for ( j = 0; j < SIZE; j++ ) { /* set the values */
        s[ j ] = 2 + 2 * j;
    } /* end for */

    printf( "%s%13s\n", "Element", "Value" );

    /* output contents of array s in tabular format */
    for ( j = 0; j < SIZE; j++ ) {···
        printf( "%7d%13d\n", j, s[ j ] );
    } /* end for */

```

Array Examples (Cont.)

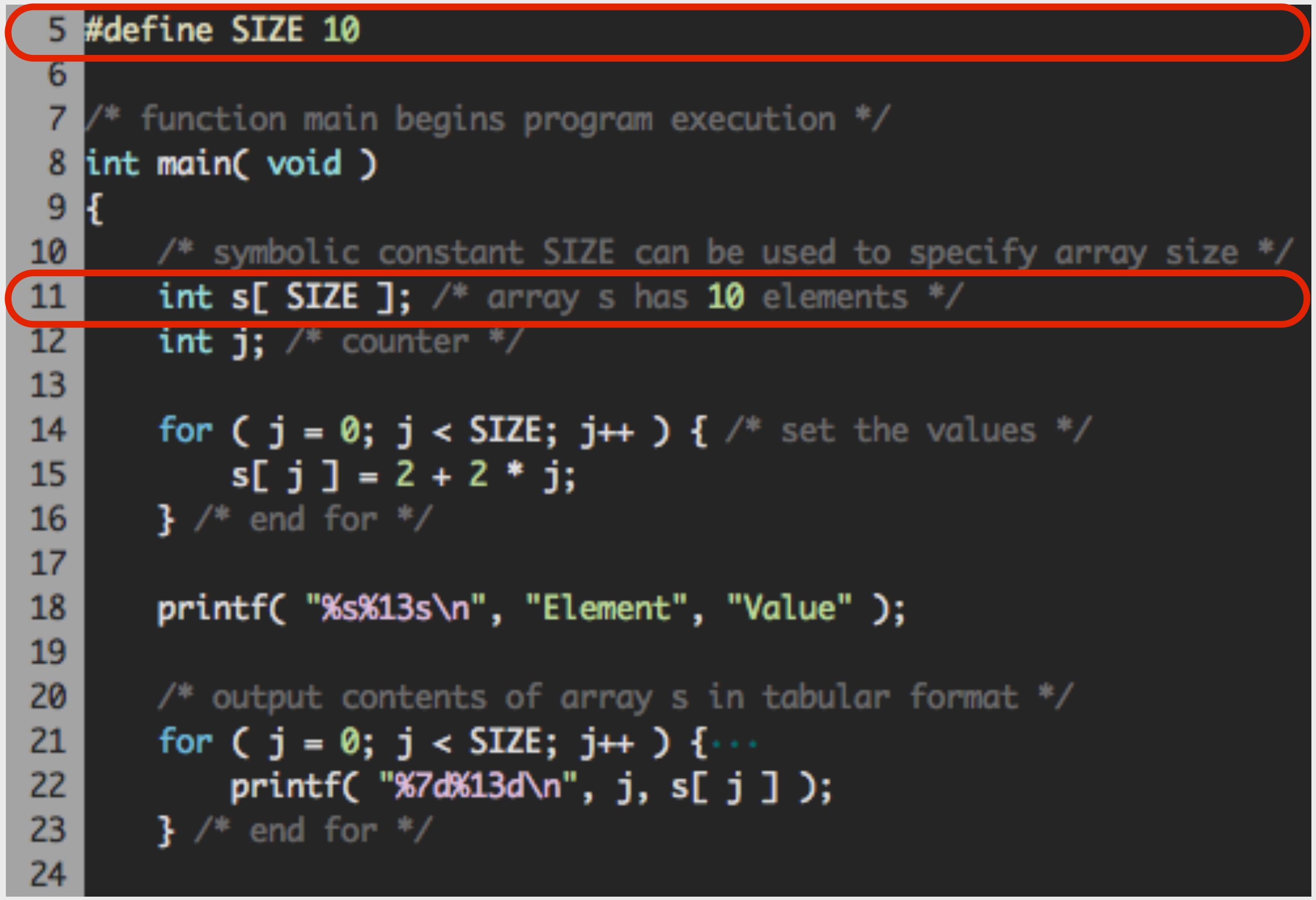
- Example: fig06_05.c



```
#define SIZE 10

/* function main begins program execution */
int main( void )
{
    /* symbolic constant SIZE can be used to specify array size */
    int s[ SIZE ]; /* array s has 10 elements */
    int j; /* counter */

    for ( j = 0; j < SIZE; j++ ) { /* set the values */
        s[ j ] = 2 + 2 * j;
    } /* end for */

    printf( "%s%13s\n", "Element", "Value" );

    /* output contents of array s in tabular format */
    for ( j = 0; j < SIZE; j++ ) {
        printf( "%7d%13d\n", j, s[ j ] );
    } /* end for */

```

Array Examples (Cont.)

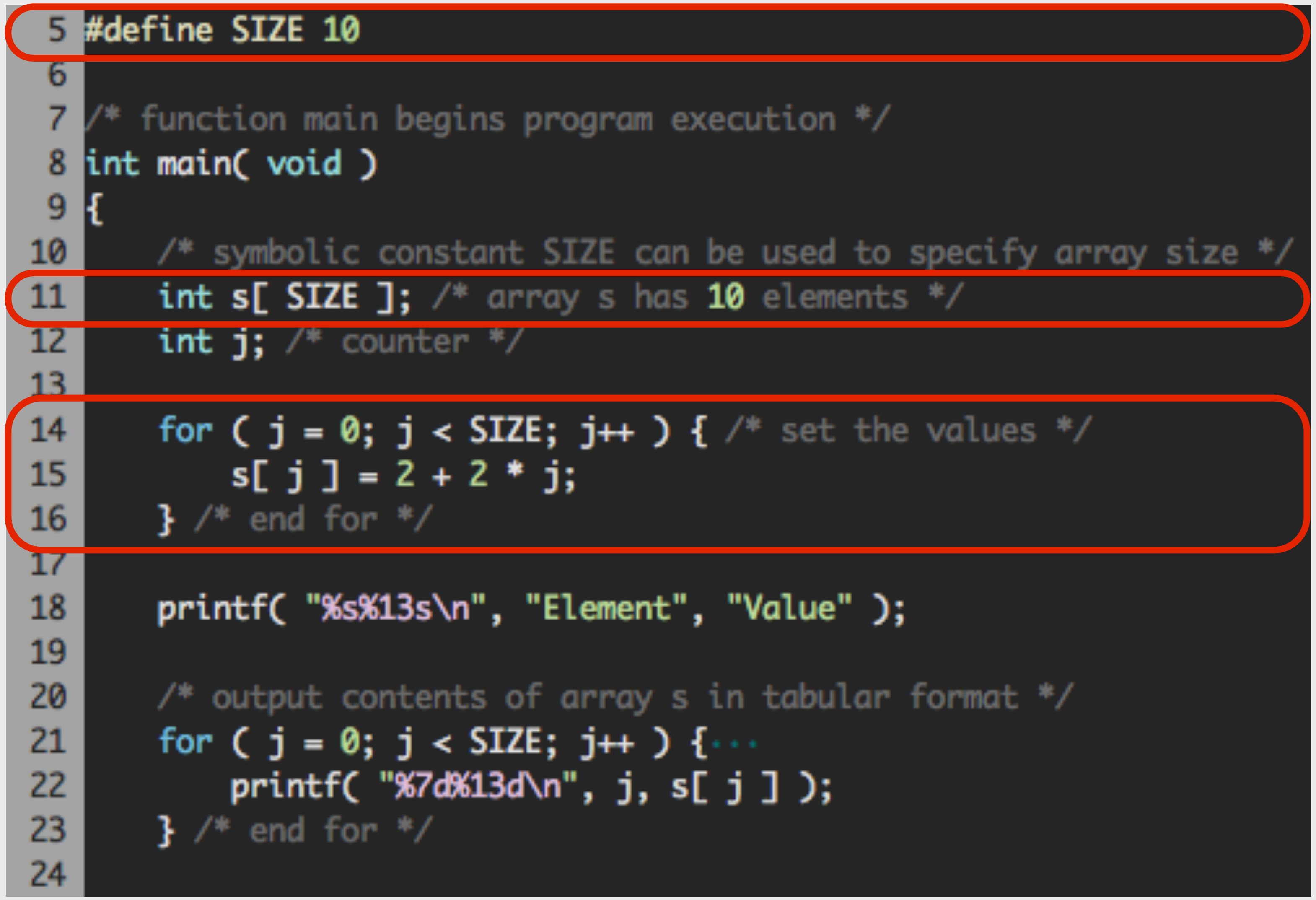
- Example: fig06_05.c



```c
#define SIZE 10

/* function main begins program execution */
int main( void )
{
    /* symbolic constant SIZE can be used to specify array size */
    int s[ SIZE ]; /* array s has 10 elements */
    int j; /* counter */

    for ( j = 0; j < SIZE; j++ ) { /* set the values */
        s[ j ] = 2 + 2 * j;
    } /* end for */

    printf( "%s%13s\n", "Element", "Value" );

    /* output contents of array s in tabular format */
    for ( j = 0; j < SIZE; j++ ) {
        printf( "%7d%13d\n", j, s[ j ] );
    } /* end for */

```

Array Examples (Cont.)

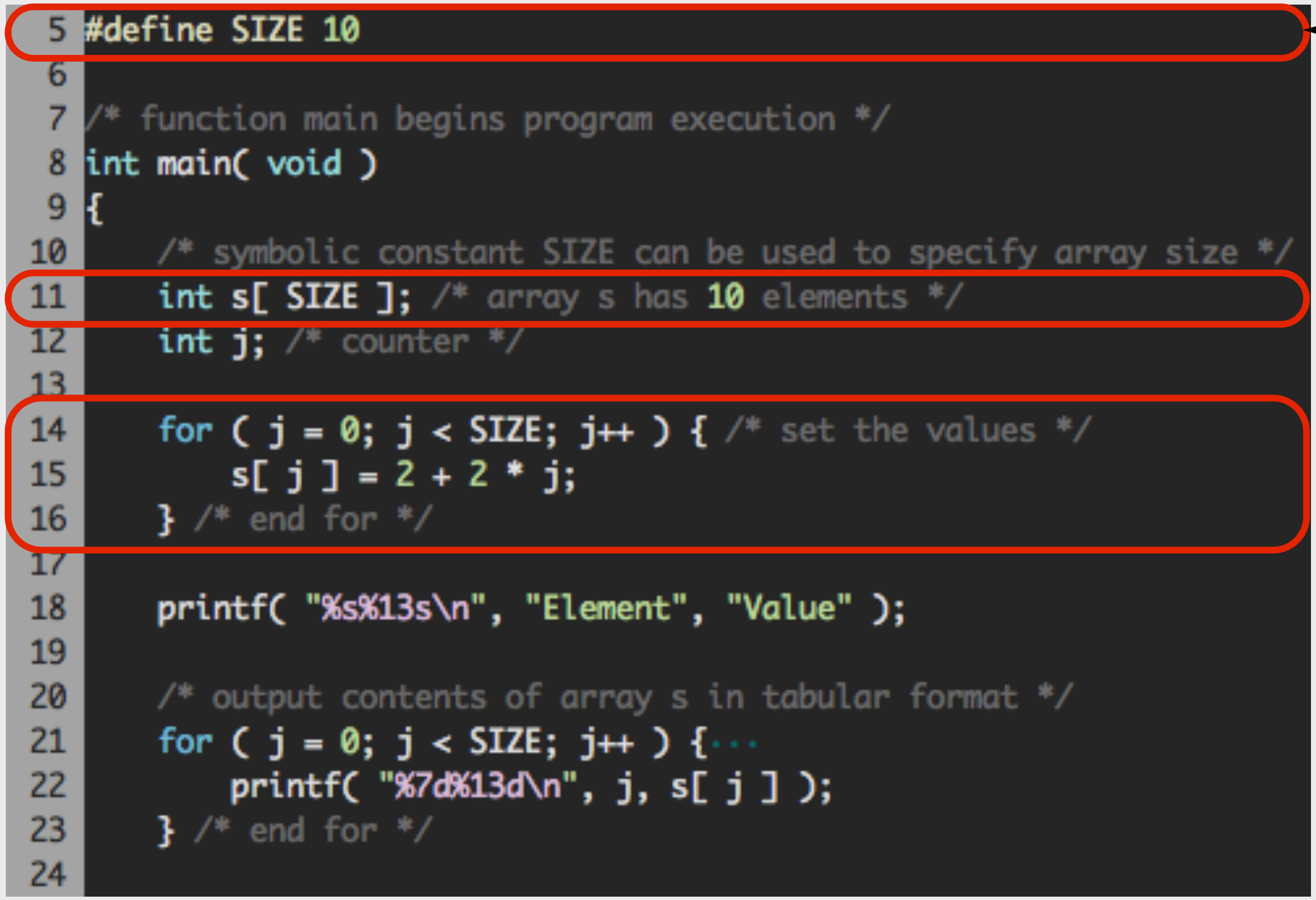
- Example: fig06_05.c



```c
#define SIZE 10

/* function main begins program execution */
int main( void )
{
    /* symbolic constant SIZE can be used to specify array size */
    int s[ SIZE ]; /* array s has 10 elements */
    int j; /* counter */

    for ( j = 0; j < SIZE; j++ ) { /* set the values */
        s[ j ] = 2 + 2 * j;
    } /* end for */

    printf( "%s%13s\n", "Element", "Value" );

    /* output contents of array s in tabular format */
    for ( j = 0; j < SIZE; j++ ) {
        printf( "%7d%13d\n", j, s[ j ] );
    } /* end for */

```

Array Examples (Cont.)

- Example: fig06_05.c

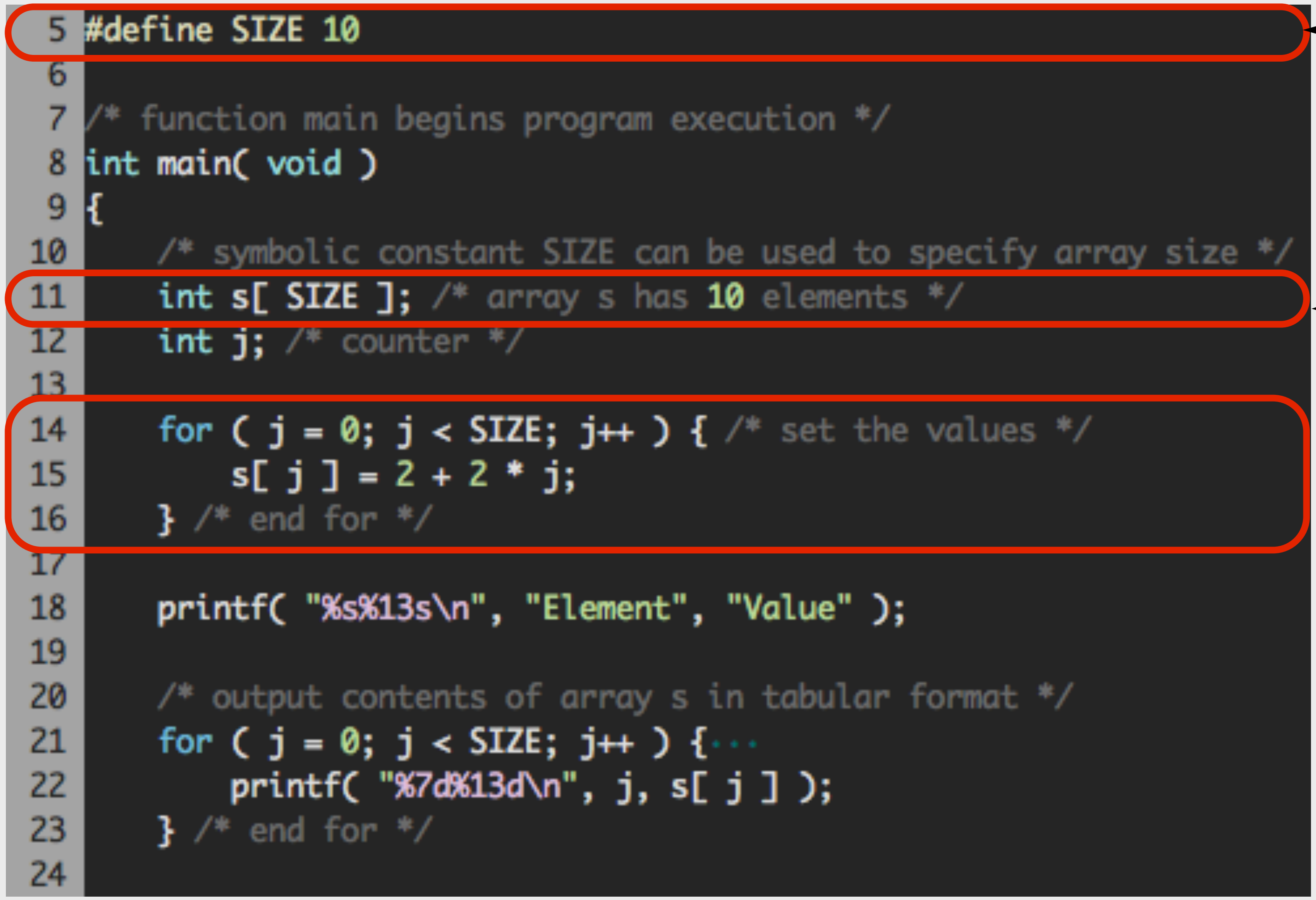


```
#define SIZE 10

/* function main begins program execution */
int main( void )
{
    /* symbolic constant SIZE can be used to specify array size */
    int s[ SIZE ]; /* array s has 10 elements */
    int j; /* counter */

    for ( j = 0; j < SIZE; j++ ) { /* set the values */
        s[ j ] = 2 + 2 * j;
    } /* end for */

    printf( "%s%13s\n", "Element", "Value" );

    /* output contents of array s in tabular format */
    for ( j = 0; j < SIZE; j++ ) {
        printf( "%7d%13d\n", j, s[ j ] );
    } /* end for */

```

use **#define** preprocessor to define a symbolic constant SIZE

Array Examples (Cont.)

- Example: fig06_05.c



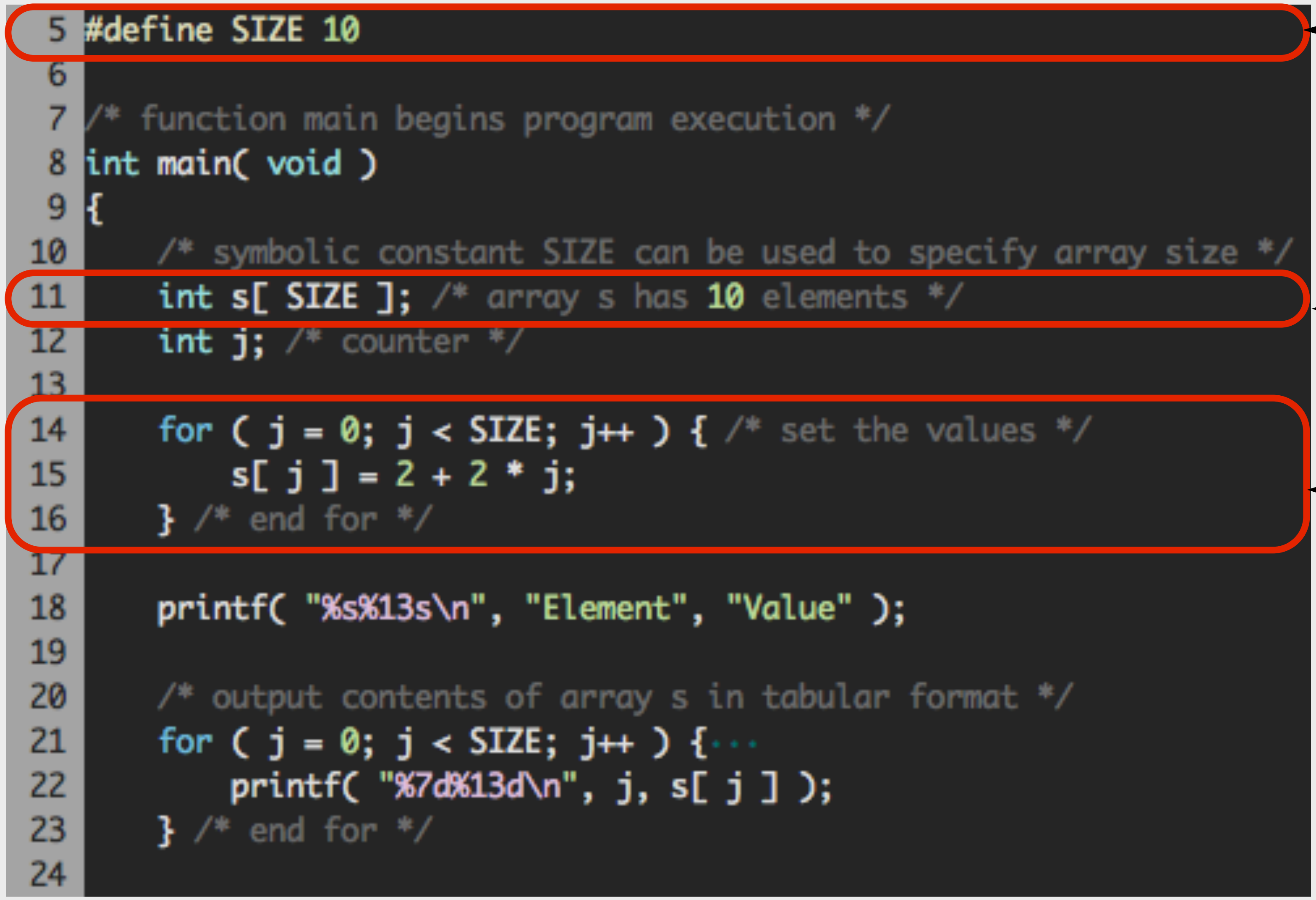
```c
#define SIZE 10

/* function main begins program execution */
int main( void )
{
    /* symbolic constant SIZE can be used to specify array size */
    int s[ SIZE ]; /* array s has 10 elements */
    int j; /* counter */

    for ( j = 0; j < SIZE; j++ ) { /* set the values */
        s[ j ] = 2 + 2 * j;
    } /* end for */

    printf( "%s%13s\n", "Element", "Value" );

    /* output contents of array s in tabular format */
    for ( j = 0; j < SIZE; j++ ) {
        printf( "%7d%13d\n", j, s[ j ] );
    } /* end for */

```

use **#define** preprocessor to define a symbolic constant SIZE

equal to
int s[10];

Array Examples (Cont.)

- Example: fig06_05.c



```
#define SIZE 10

/* function main begins program execution */
int main( void )
{
    /* symbolic constant SIZE can be used to specify array size */
    int s[ SIZE ]; /* array s has 10 elements */
    int j; /* counter */

    for ( j = 0; j < SIZE; j++ ) { /* set the values */
        s[ j ] = 2 + 2 * j;
    } /* end for */

    printf( "%s%13s\n", "Element", "Value" );

    /* output contents of array s in tabular format */
    for ( j = 0; j < SIZE; j++ ) {
        printf( "%7d%13d\n", j, s[ j ] );
    } /* end for */

```

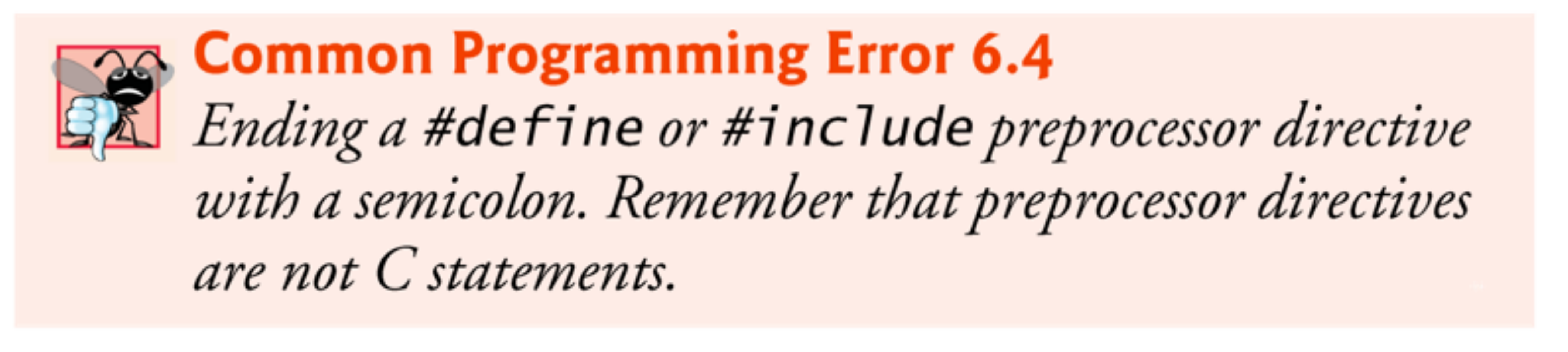
use **#define** preprocessor to define a symbolic constant SIZE

equal to
int s[10];

The values are generated by multiplying the loop counter by 2 and adding 2

Array Examples (Cont.)

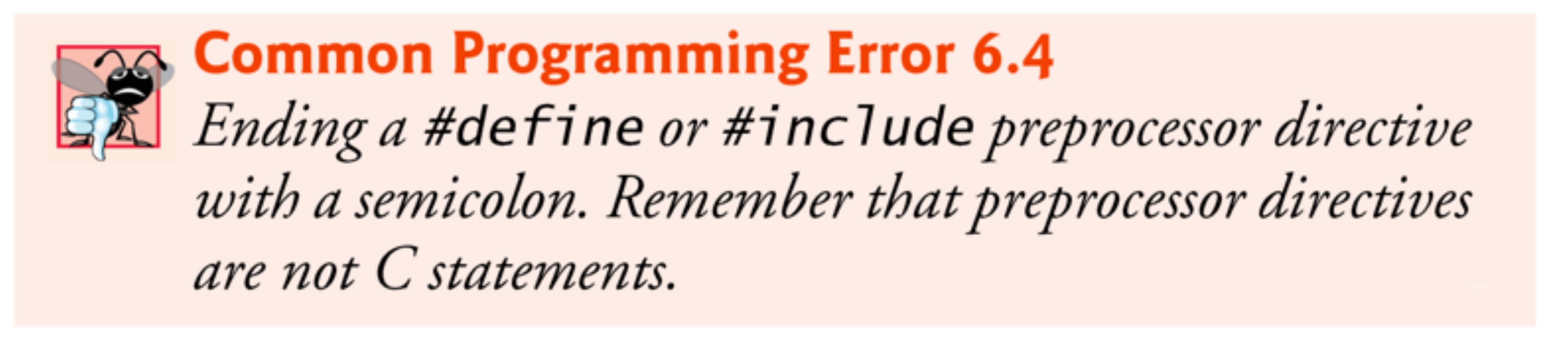
Array Examples (Cont.)



Common Programming Error 6.4

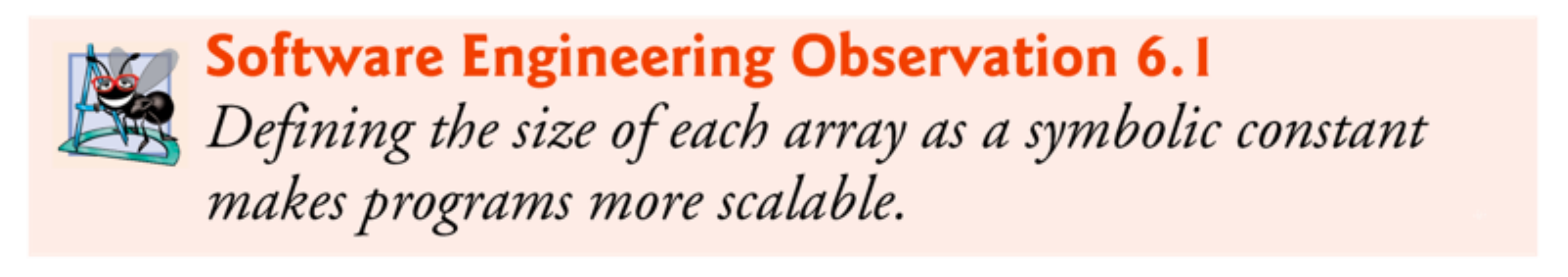
Ending a `#define` *or* `#include` *preprocessor directive with a semicolon. Remember that preprocessor directives are not C statements.*

Array Examples (Cont.)



Common Programming Error 6.4

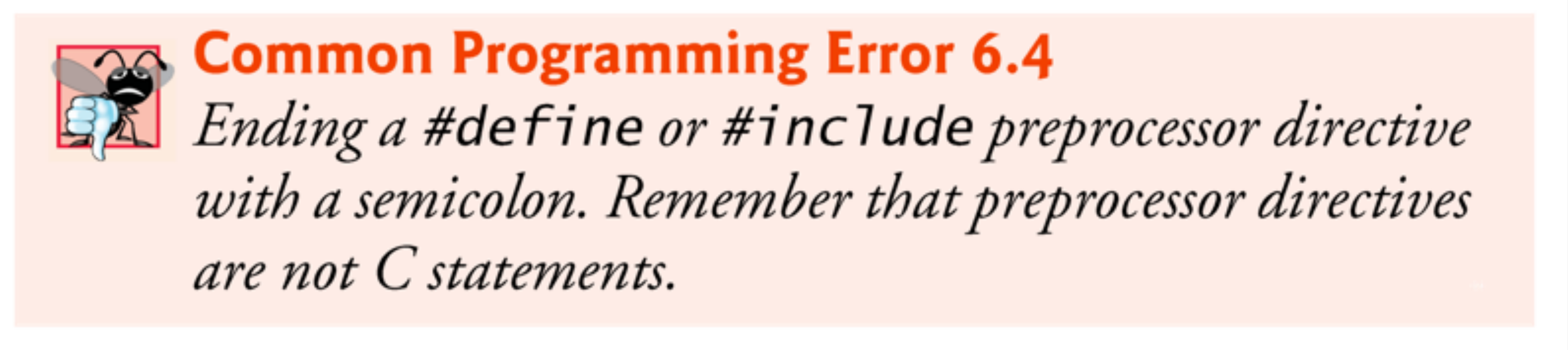
Ending a `#define` or `#include` preprocessor directive with a semicolon. Remember that preprocessor directives are not C statements.



Software Engineering Observation 6.1

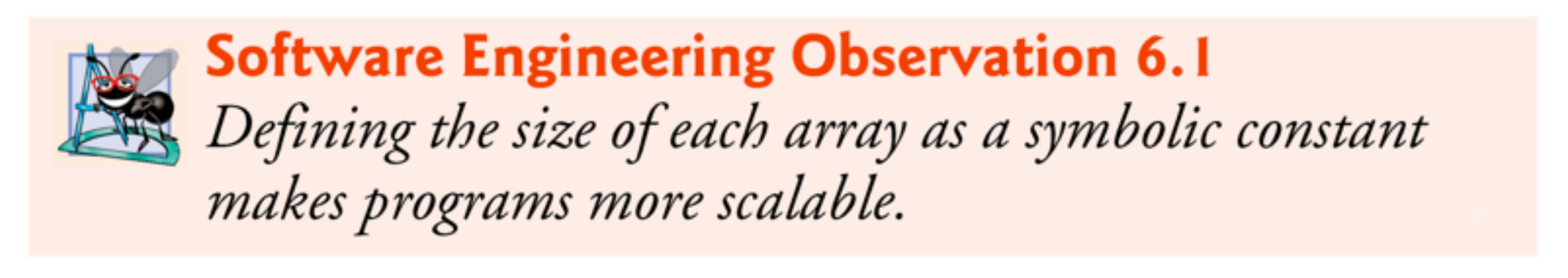
Defining the size of each array as a symbolic constant makes programs more scalable.

Array Examples (Cont.)



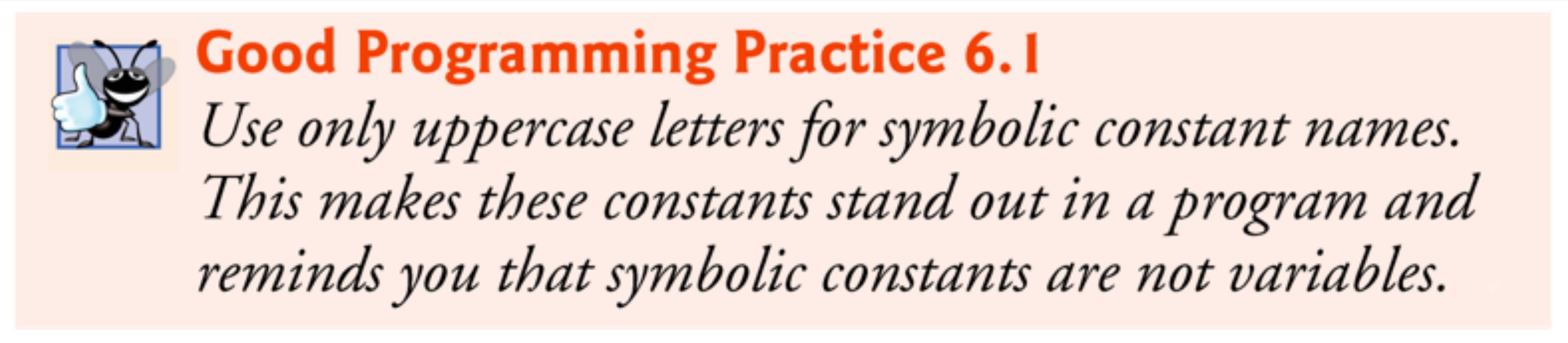
Common Programming Error 6.4

Ending a `#define` or `#include` preprocessor directive with a semicolon. Remember that preprocessor directives are not C statements.



Software Engineering Observation 6.1

Defining the size of each array as a symbolic constant makes programs more scalable.

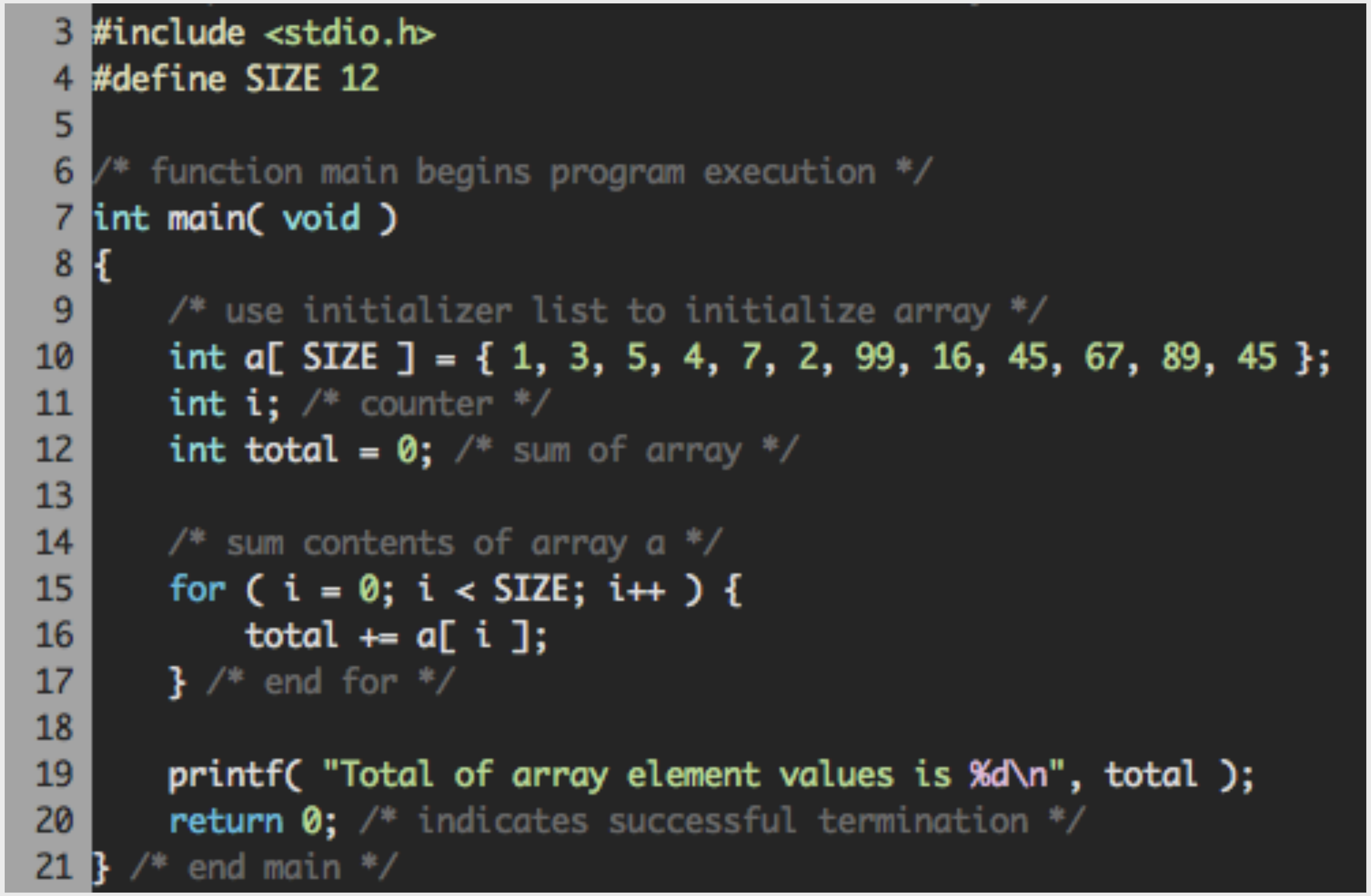


Good Programming Practice 6.1

Use only uppercase letters for symbolic constant names. This makes these constants stand out in a program and reminds you that symbolic constants are not variables.

Array Examples (Cont.)

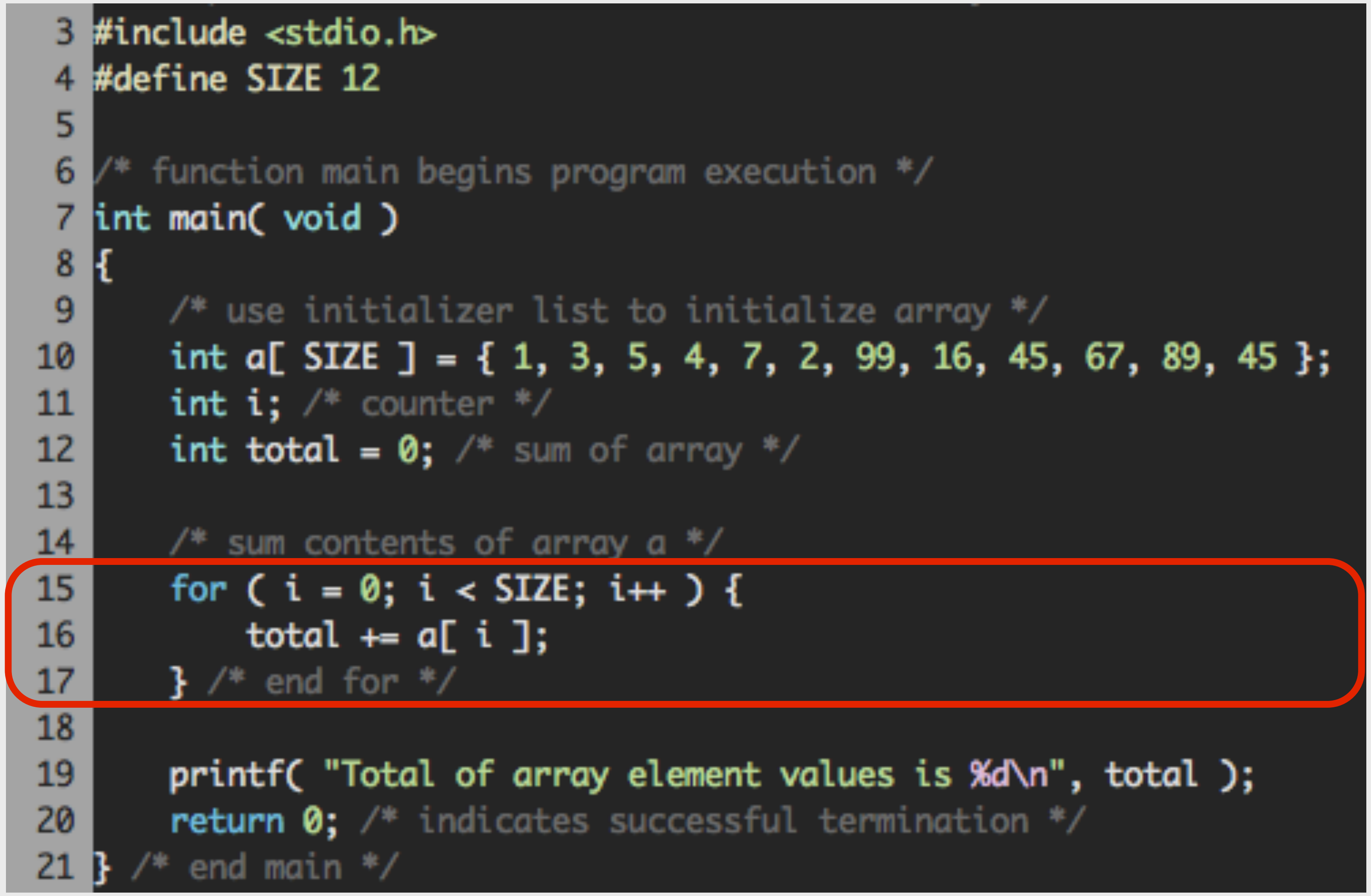
- Ex: fig06_06.c



```c
#include <stdio.h>
#define SIZE 12

/* function main begins program execution */
int main( void )
{
    /* use initializer list to initialize array */
    int a[ SIZE ] = { 1, 3, 5, 4, 7, 2, 99, 16, 45, 67, 89, 45 };
    int i; /* counter */
    int total = 0; /* sum of array */

    /* sum contents of array a */
    for ( i = 0; i < SIZE; i++ ) {
        total += a[ i ];
    } /* end for */

    printf( "Total of array element values is %d\n", total );
    return 0; /* indicates successful termination */
} /* end main */
```

Array Examples (Cont.)

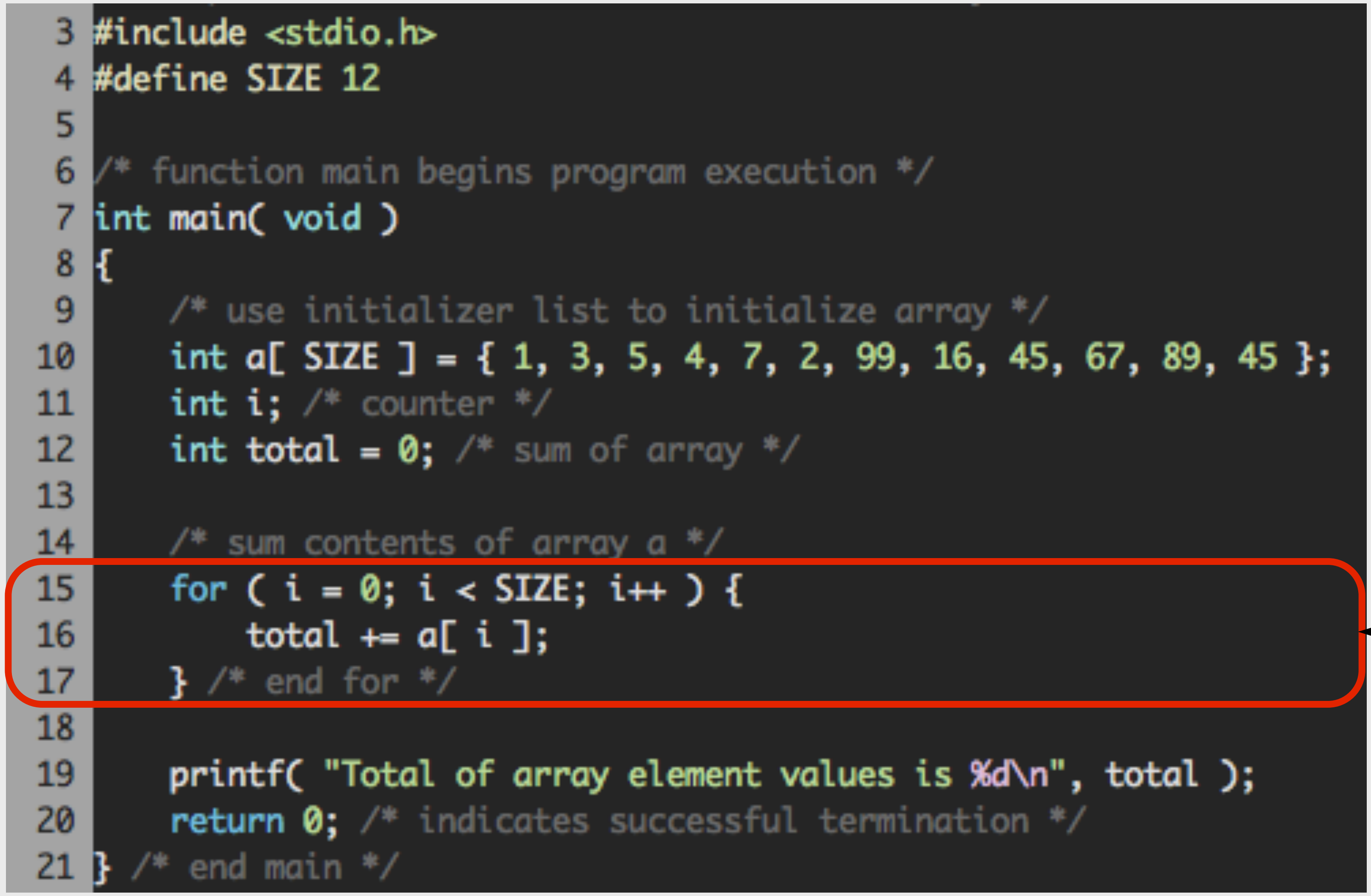
- Ex: fig06_06.c



```c
#include <stdio.h>
#define SIZE 12

/* function main begins program execution */
int main( void )
{
    /* use initializer list to initialize array */
    int a[ SIZE ] = { 1, 3, 5, 4, 7, 2, 99, 16, 45, 67, 89, 45 };
    int i; /* counter */
    int total = 0; /* sum of array */

    /* sum contents of array a */
    for ( i = 0; i < SIZE; i++ ) {
        total += a[ i ];
    } /* end for */

    printf( "Total of array element values is %d\n", total );
    return 0; /* indicates successful termination */
} /* end main */
```

Array Examples (Cont.)

- Ex: fig06_06.c

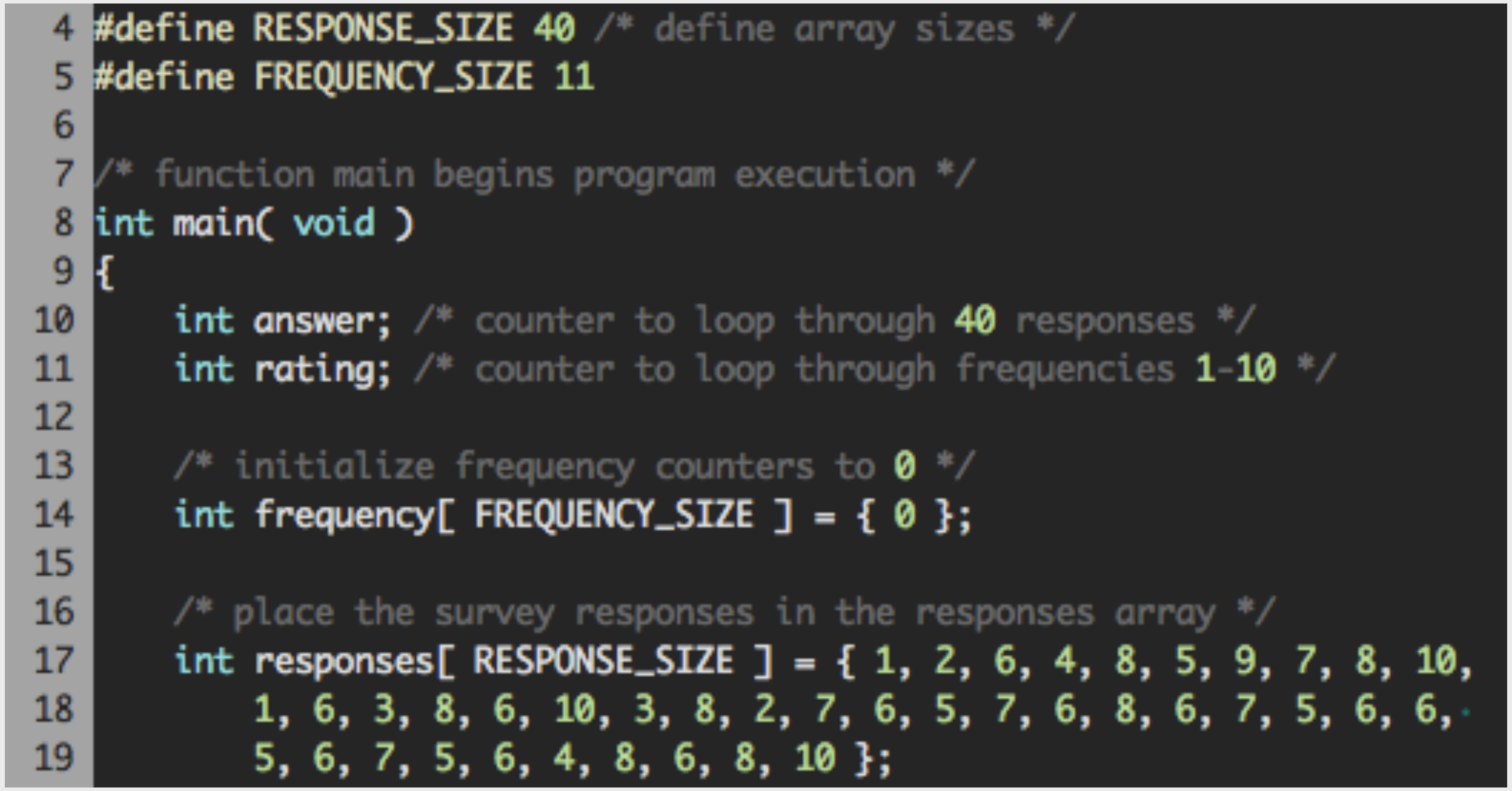


```
#include <stdio.h>
#define SIZE 12

/* function main begins program execution */
int main( void )
{
    /* use initializer list to initialize array */
    int a[ SIZE ] = { 1, 3, 5, 4, 7, 2, 99, 16, 45, 67, 89, 45 };
    int i; /* counter */
    int total = 0; /* sum of array */

    /* sum contents of array a */
    for ( i = 0; i < SIZE; i++ ) {
        total += a[ i ];
    } /* end for */

    printf( "Total of array element values is %d\n", total );
    return 0; /* indicates successful termination */
} /* end main */
```

Sum the values contained in the array **a**

Array Examples (Cont.)

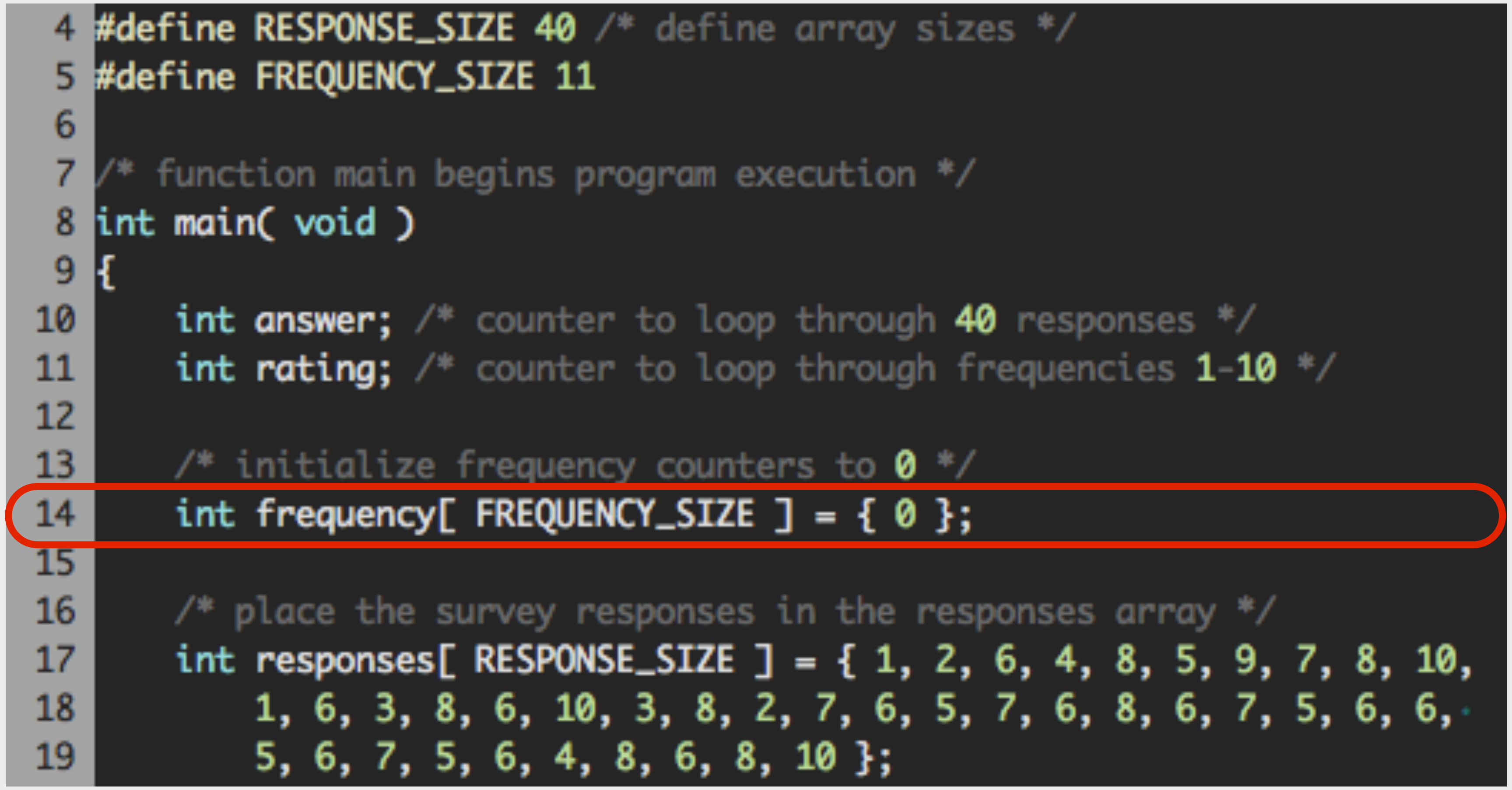
- Ex: fig06_07.c



```c
#define RESPONSE_SIZE 40 /* define array sizes */
#define FREQUENCY_SIZE 11

/* function main begins program execution */
int main( void )
{
    int answer; /* counter to loop through 40 responses */
    int rating; /* counter to loop through frequencies 1-10 */

    /* initialize frequency counters to 0 */
    int frequency[ FREQUENCY_SIZE ] = { 0 };

    /* place the survey responses in the responses array */
    int responses[ RESPONSE_SIZE ] = { 1, 2, 6, 4, 8, 5, 9, 7, 8, 10,
        1, 6, 3, 8, 6, 10, 3, 8, 2, 7, 6, 5, 7, 6, 8, 6, 7, 5, 6, 6,
        5, 6, 7, 5, 6, 4, 8, 6, 8, 10 };
```

Array Examples (Cont.)

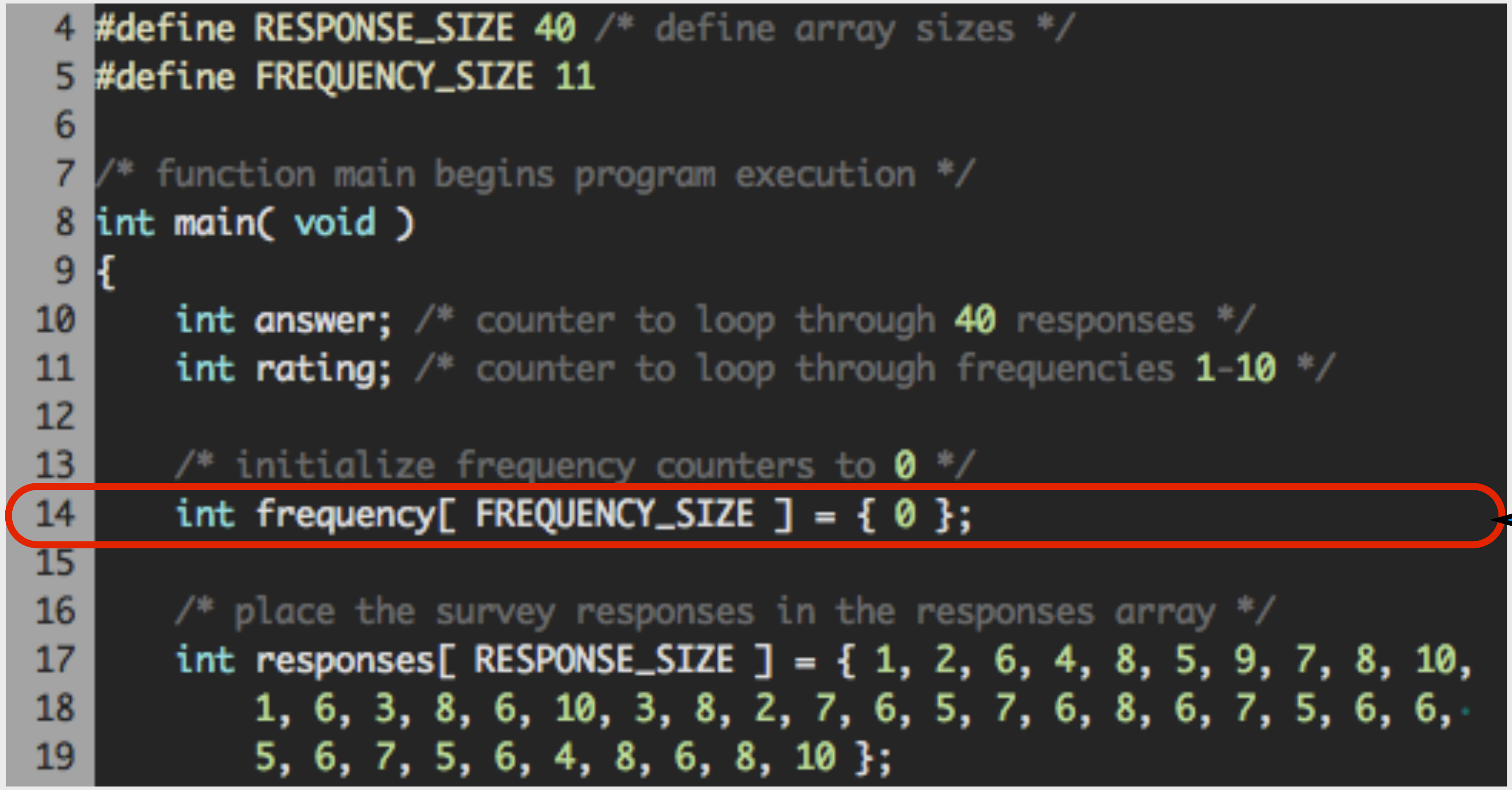
- Ex: fig06_07.c



```c
#define RESPONSE_SIZE 40 /* define array sizes */
#define FREQUENCY_SIZE 11

/* function main begins program execution */
int main( void )
{
    int answer; /* counter to loop through 40 responses */
    int rating; /* counter to loop through frequencies 1-10 */

    /* initialize frequency counters to 0 */
    int frequency[ FREQUENCY_SIZE ] = { 0 };

    /* place the survey responses in the responses array */
    int responses[ RESPONSE_SIZE ] = { 1, 2, 6, 4, 8, 5, 9, 7, 8, 10,
        1, 6, 3, 8, 6, 10, 3, 8, 2, 7, 6, 5, 7, 6, 8, 6, 7, 5, 6, 6,
        5, 6, 7, 5, 6, 4, 8, 6, 8, 10 };
```

Array Examples (Cont.)

- Ex: fig06_07.c

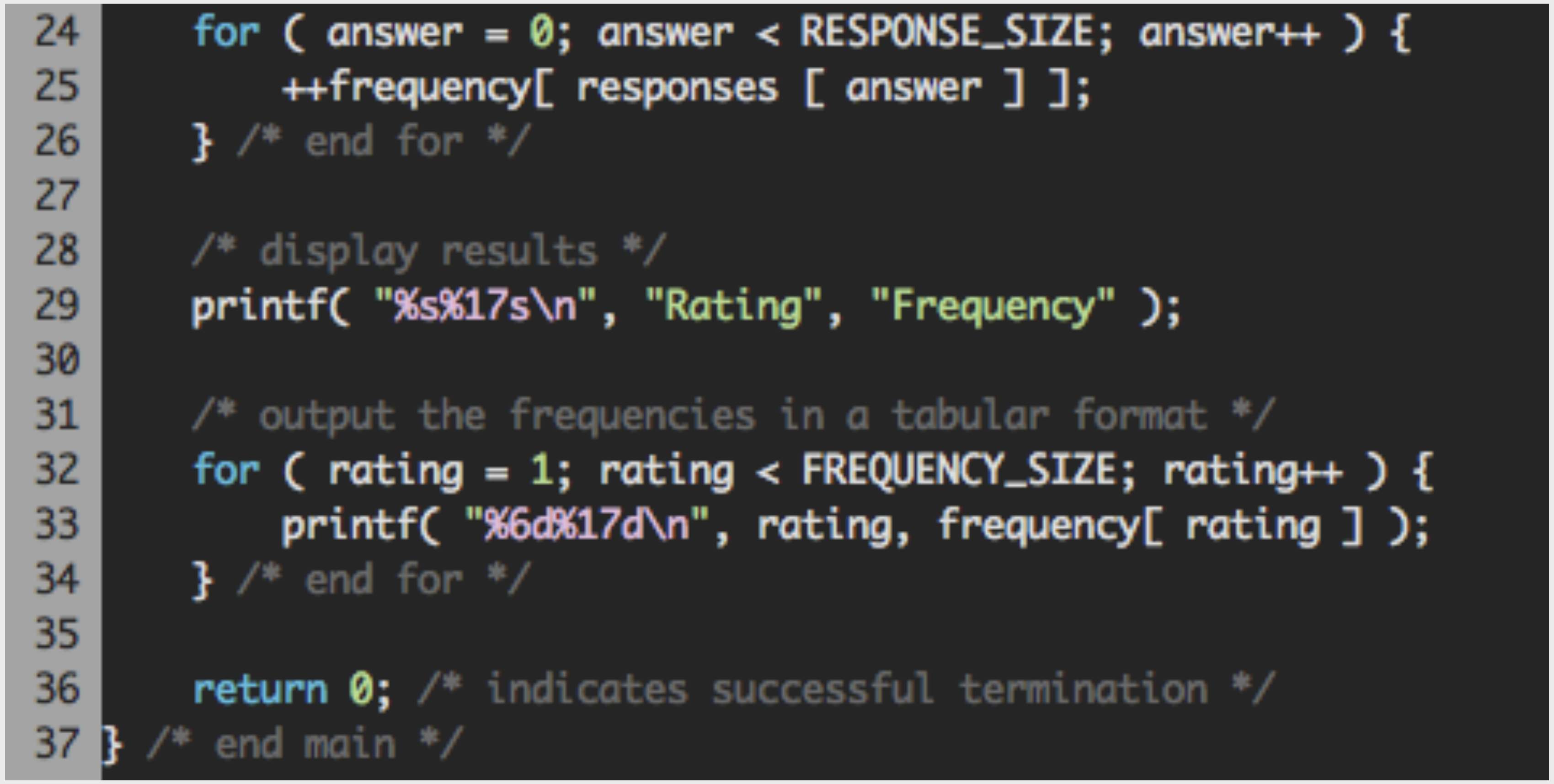


```
#define RESPONSE_SIZE 40 /* define array sizes */
#define FREQUENCY_SIZE 11

/* function main begins program execution */
int main( void )
{
    int answer; /* counter to loop through 40 responses */
    int rating; /* counter to loop through frequencies 1-10 */

    /* initialize frequency counters to 0 */
    int frequency[ FREQUENCY_SIZE ] = { 0 };

    /* place the survey responses in the responses array */
    int responses[ RESPONSE_SIZE ] = { 1, 2, 6, 4, 8, 5, 9, 7, 8, 10,
        1, 6, 3, 8, 6, 10, 3, 8, 2, 7, 6, 5, 7, 6, 8, 6, 7, 5, 6, 6,
        5, 6, 7, 5, 6, 4, 8, 6, 8, 10 };
```

Use **frequency** to count the number of occurrences of each response

Array Examples (Cont.)

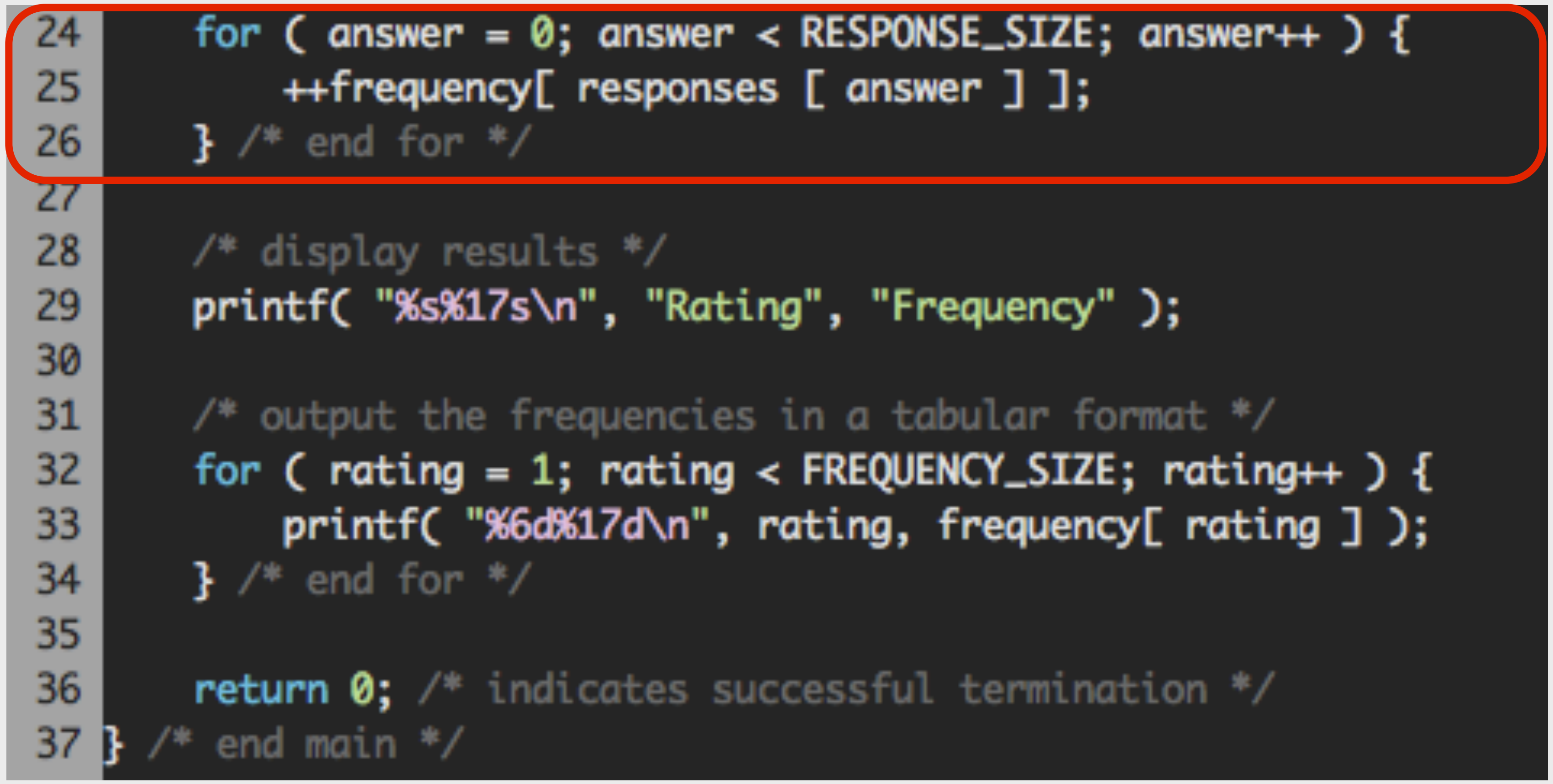
- Example: fig06_07.c



```c
    for ( answer = 0; answer < RESPONSE_SIZE; answer++ ) {
        ++frequency[ responses [ answer ] ];
    } /* end for */

    /* display results */
    printf( "%s%17s\n", "Rating", "Frequency" );

    /* output the frequencies in a tabular format */
    for ( rating = 1; rating < FREQUENCY_SIZE; rating++ ) {
        printf( "%6d%17d\n", rating, frequency[ rating ] );
    } /* end for */

    return 0; /* indicates successful termination */
} /* end main */
```

Array Examples (Cont.)

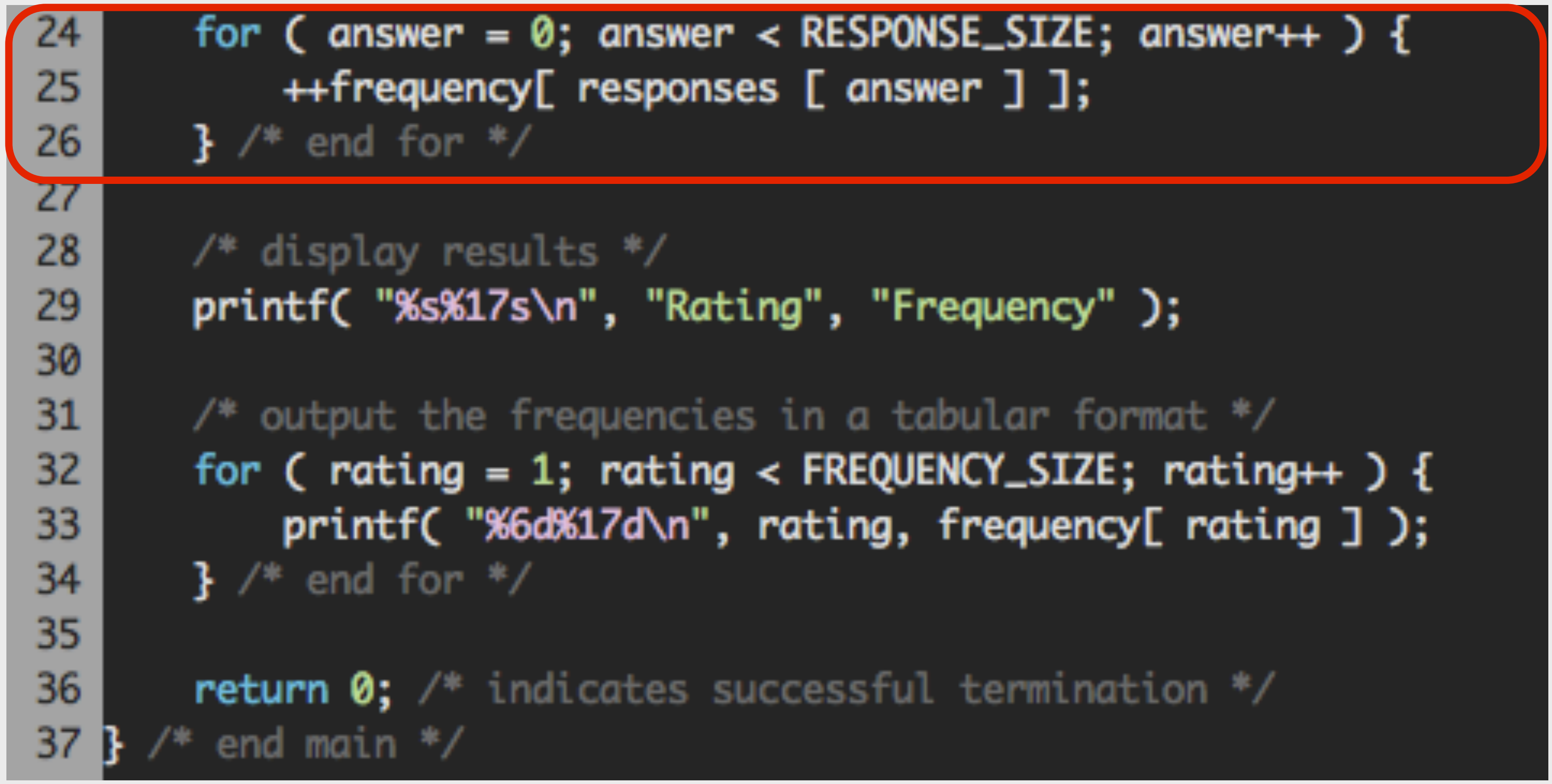
- Example: fig06_07.c



```
    for ( answer = 0; answer < RESPONSE_SIZE; answer++ ) {
        ++frequency[ responses [ answer ] ];
    } /* end for */

    /* display results */
    printf( "%s%17s\n", "Rating", "Frequency" );

    /* output the frequencies in a tabular format */
    for ( rating = 1; rating < FREQUENCY_SIZE; rating++ ) {
        printf( "%6d%17d\n", rating, frequency[ rating ] );
    } /* end for */

    return 0; /* indicates successful termination */
} /* end main */
```

Array Examples (Cont.)

- Example: fig06_07.c

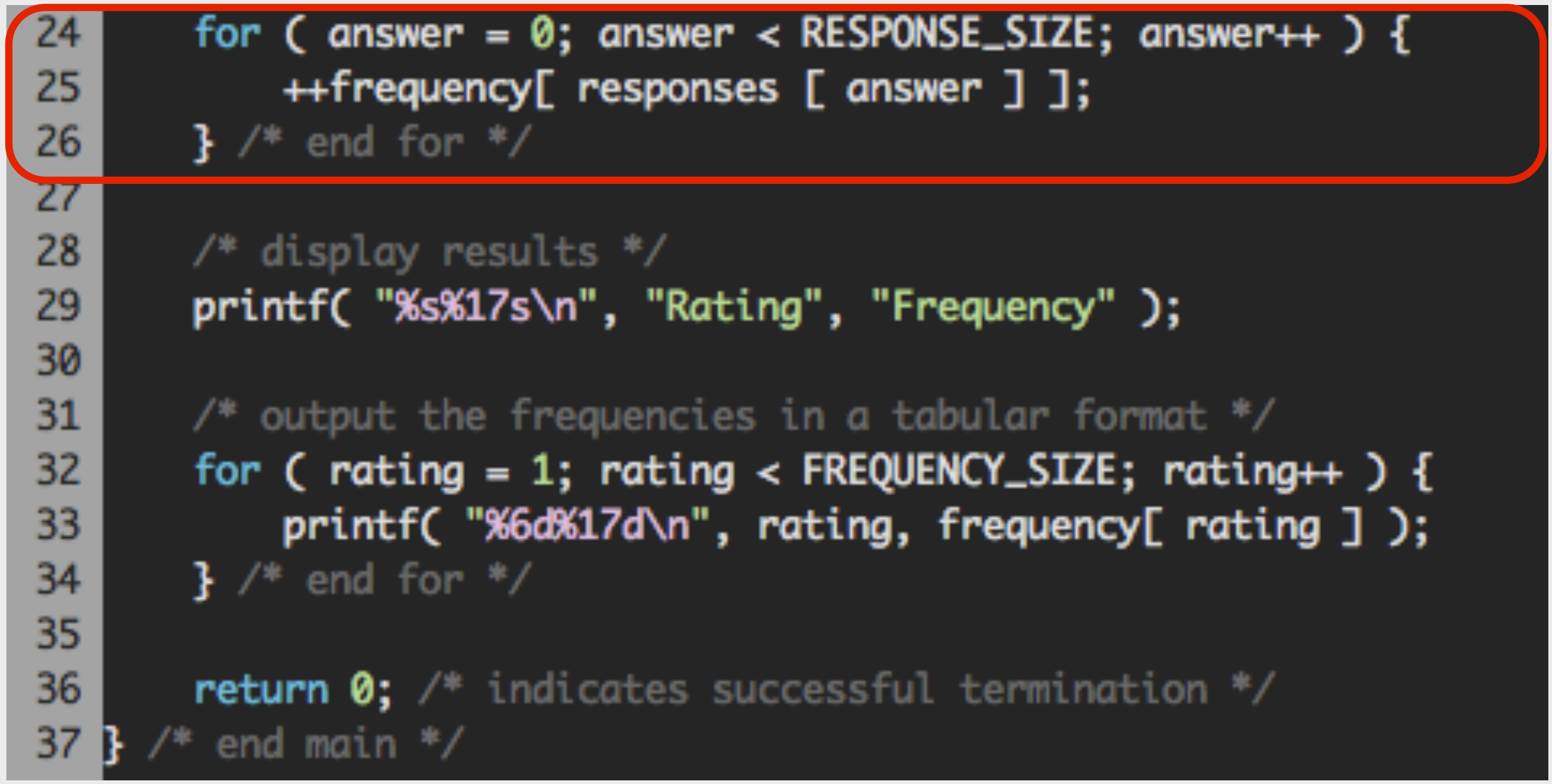


```c
    for ( answer = 0; answer < RESPONSE_SIZE; answer++ ) {
        ++frequency[ responses [ answer ] ];
    } /* end for */

    /* display results */
    printf( "%s%17s\n", "Rating", "Frequency" );

    /* output the frequencies in a tabular format */
    for ( rating = 1; rating < FREQUENCY_SIZE; rating++ ) {
        printf( "%6d%17d\n", rating, frequency[ rating ] );
    } /* end for */

    return 0; /* indicates successful termination */
} /* end main */
```

Allows us to use each response directly as the subscript in the **frequency** array.

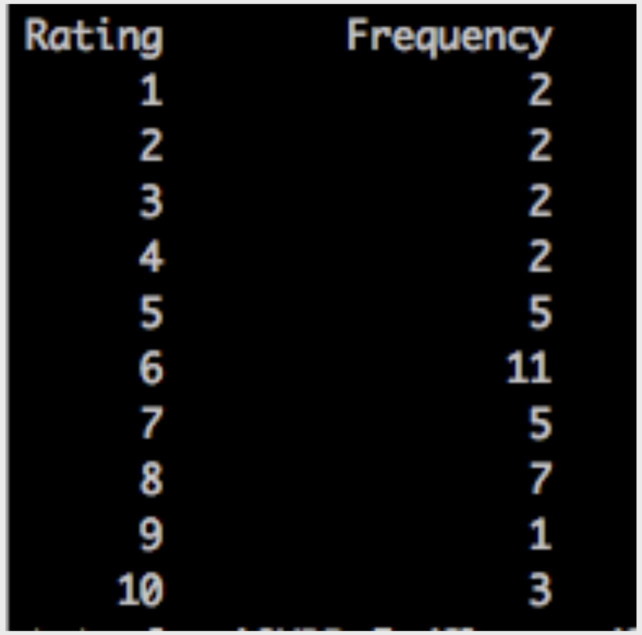
Array Examples (Cont.)

- Example: fig06_07.c



```c
    for ( answer = 0; answer < RESPONSE_SIZE; answer++ ) {
        ++frequency[ responses [ answer ] ];
    } /* end for */

    /* display results */
    printf( "%s%17s\n", "Rating", "Frequency" );

    /* output the frequencies in a tabular format */
    for ( rating = 1; rating < FREQUENCY_SIZE; rating++ ) {
        printf( "%6d%17d\n", rating, frequency[ rating ] );
    } /* end for */

    return 0; /* indicates successful termination */
} /* end main */
```

Allows us to use each response directly as the subscript in the **frequency** array.



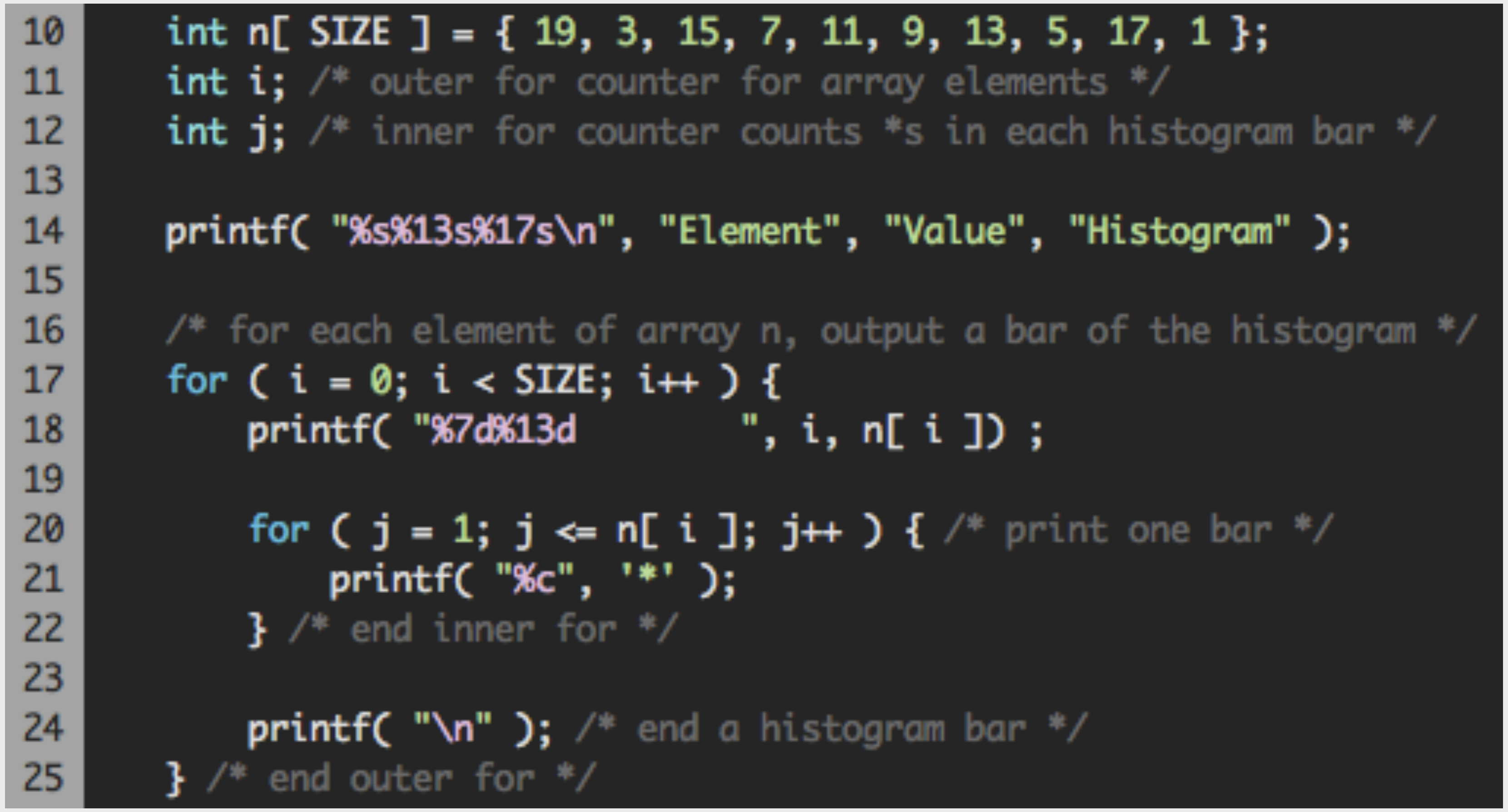
```
Rating        Frequency
     1                2
     2                2
     3                2
     4                2
     5                5
     6               11
     7                5
     8                7
     9                1
    10                3
```

Array Examples (Cont.)

- *C has no array bounds checking to prevent the program from referring to an element that does not exist.*
- Thus, an executing program can “walk off” the end of an array without warning.
- You should **ensure that all array references remain within the bounds of the array.**

Array Examples (Cont.)

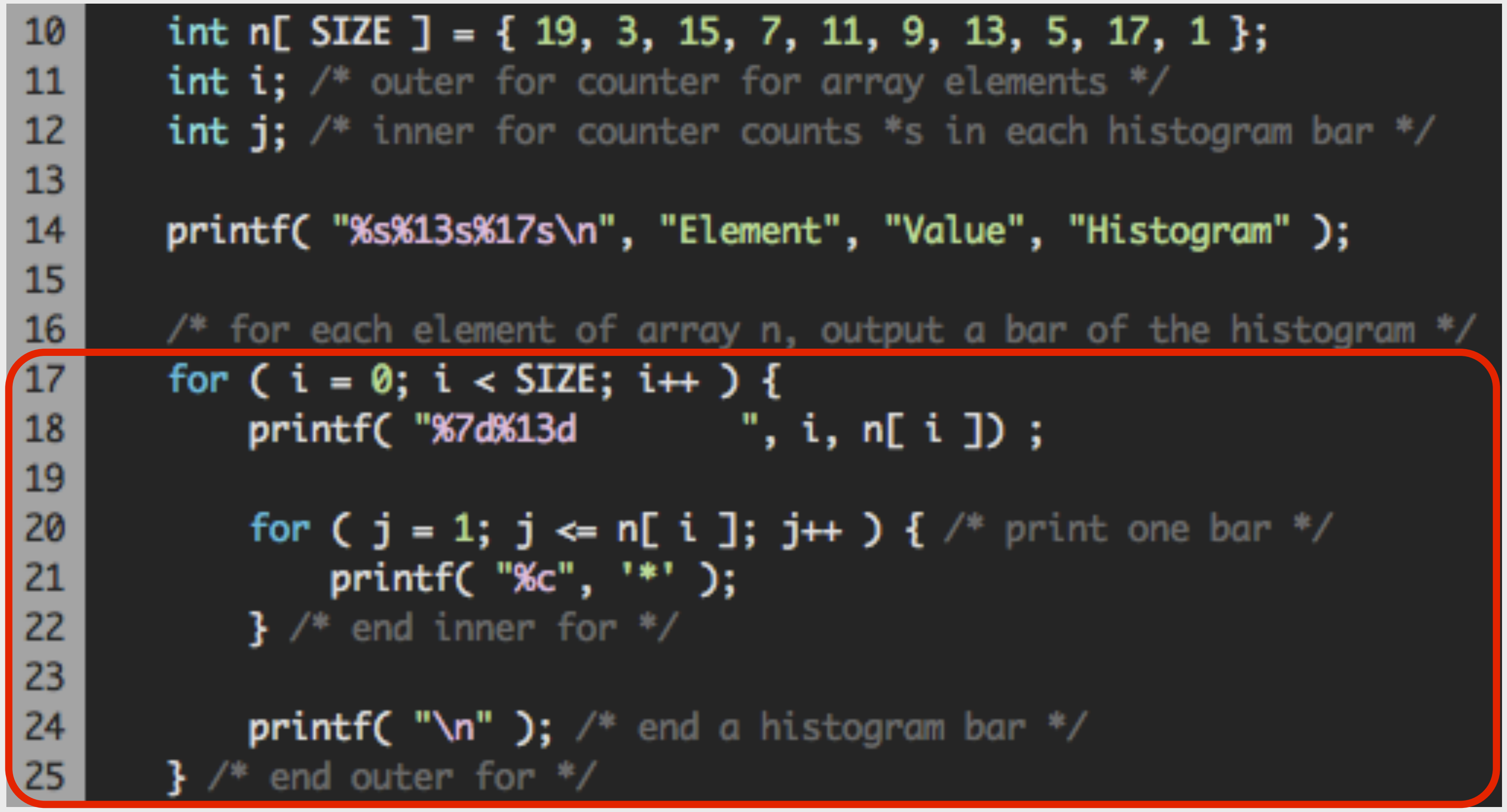
- Example: fig06_08.c



```c
int n[ SIZE ] = { 19, 3, 15, 7, 11, 9, 13, 5, 17, 1 };
int i; /* outer for counter for array elements */
int j; /* inner for counter counts *s in each histogram bar */

printf( "%s%13s%17s\n", "Element", "Value", "Histogram" );

/* for each element of array n, output a bar of the histogram */
for ( i = 0; i < SIZE; i++ ) {
    printf( "%7d%13d        ", i, n[ i ]) ;

    for ( j = 1; j <= n[ i ]; j++ ) { /* print one bar */
        printf( "%c", '*' );
    } /* end inner for */

    printf( "\n" ); /* end a histogram bar */
} /* end outer for */
```

Array Examples (Cont.)

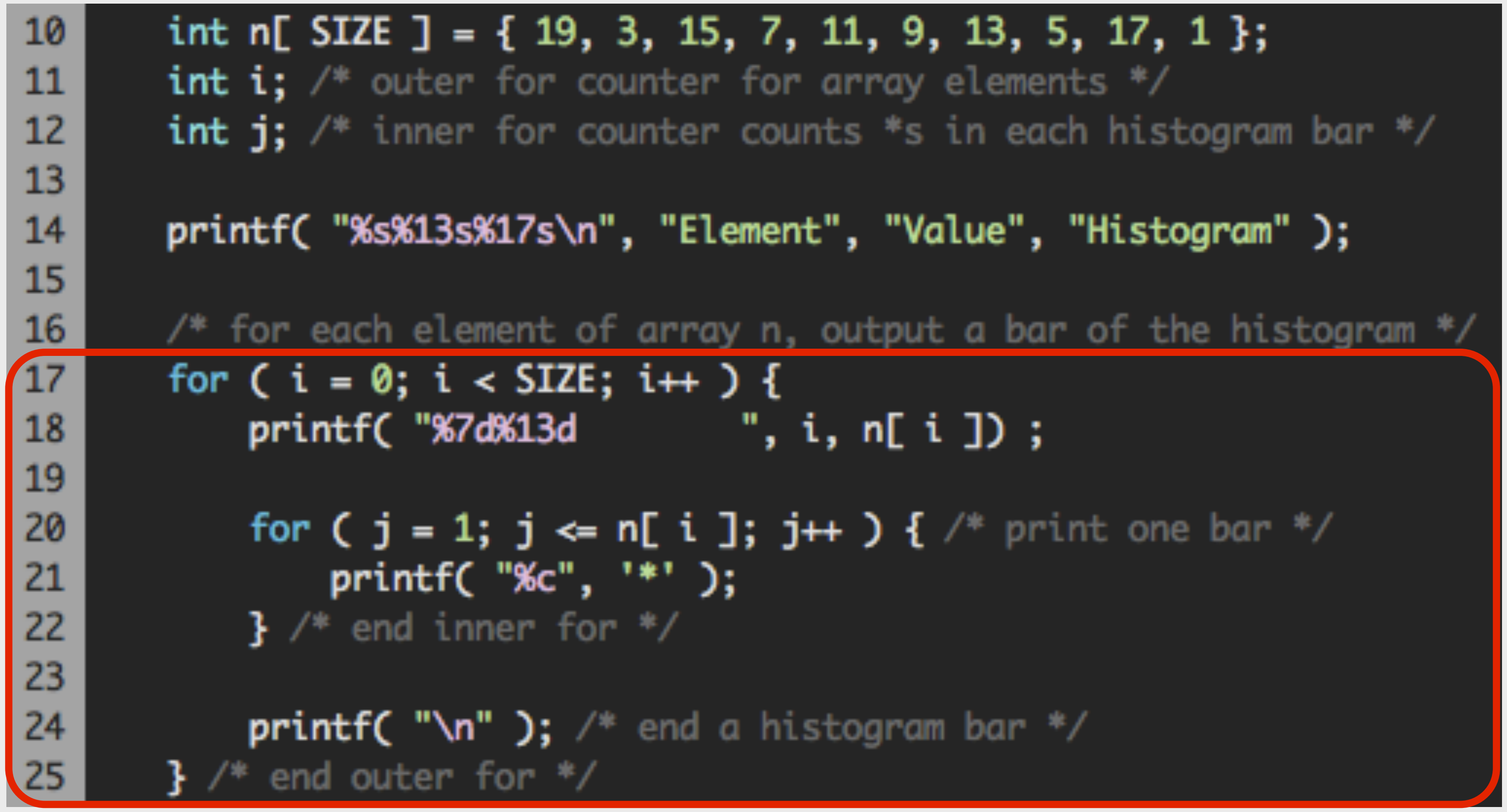
- Example: fig06_08.c



```c
    int n[ SIZE ] = { 19, 3, 15, 7, 11, 9, 13, 5, 17, 1 };
    int i; /* outer for counter for array elements */
    int j; /* inner for counter counts *s in each histogram bar */

    printf( "%s%13s%17s\n", "Element", "Value", "Histogram" );

    /* for each element of array n, output a bar of the histogram */
    for ( i = 0; i < SIZE; i++ ) {
        printf( "%7d%13d        ", i, n[ i ]) ;

        for ( j = 1; j <= n[ i ]; j++ ) { /* print one bar */
            printf( "%c", '*' );
        } /* end inner for */

        printf( "\n" ); /* end a histogram bar */
    } /* end outer for */
```

Array Examples (Cont.)

- Example: fig06_08.c

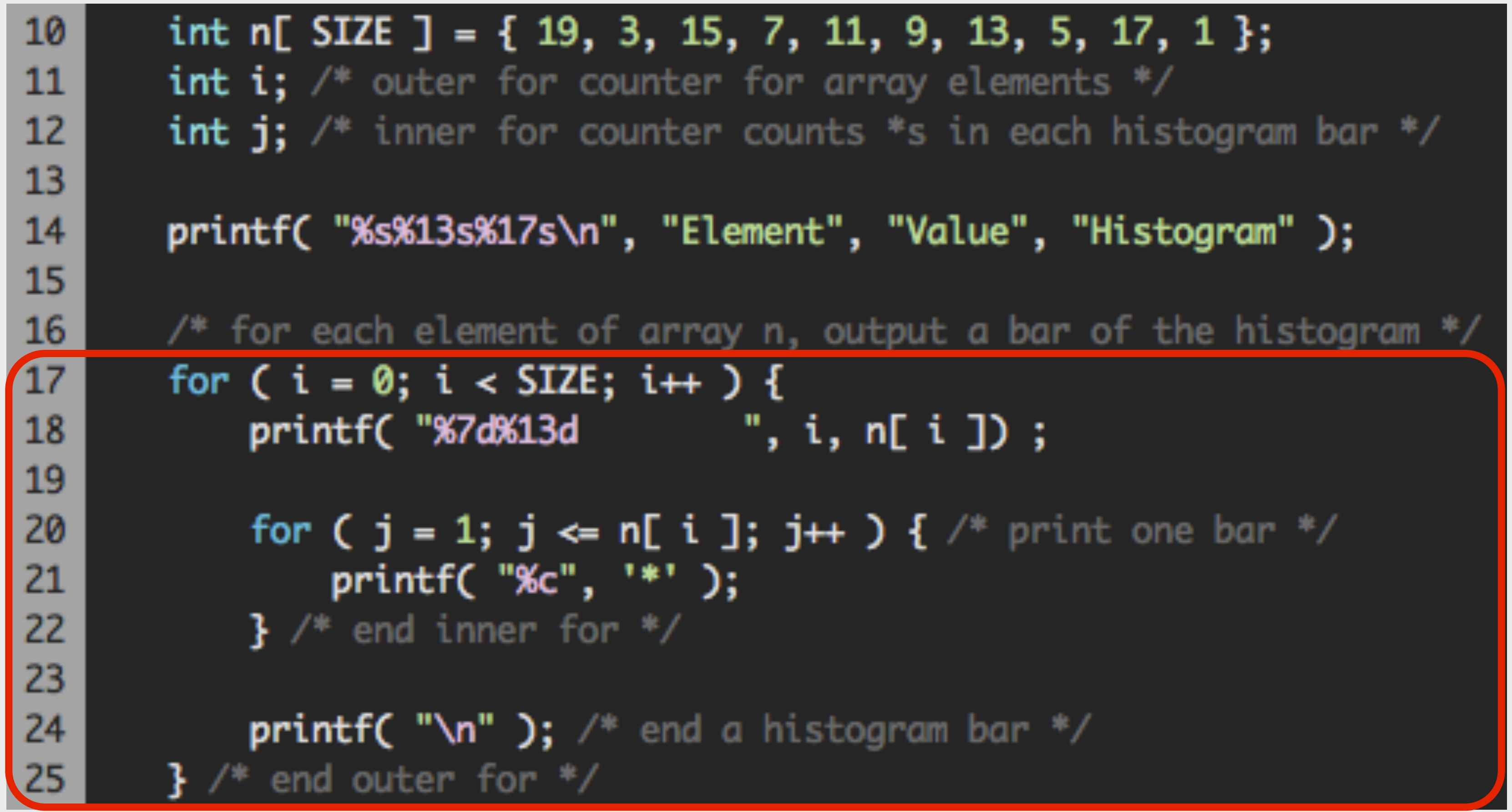


```
int n[ SIZE ] = { 19, 3, 15, 7, 11, 9, 13, 5, 17, 1 };
int i; /* outer for counter for array elements */
int j; /* inner for counter counts *s in each histogram bar */

printf( "%s%13s%17s\n", "Element", "Value", "Histogram" );

/* for each element of array n, output a bar of the histogram */
for ( i = 0; i < SIZE; i++ ) {
    printf( "%7d%13d        ", i, n[ i ]) ;

    for ( j = 1; j <= n[ i ]; j++ ) { /* print one bar */
        printf( "%c", '*' );
    } /* end inner for */

    printf( "\n" ); /* end a histogram bar */
} /* end outer for */
```

Reads numbers from an array and graphs the information in the form of a bar chart or histogram; the **for** statement draws the bars

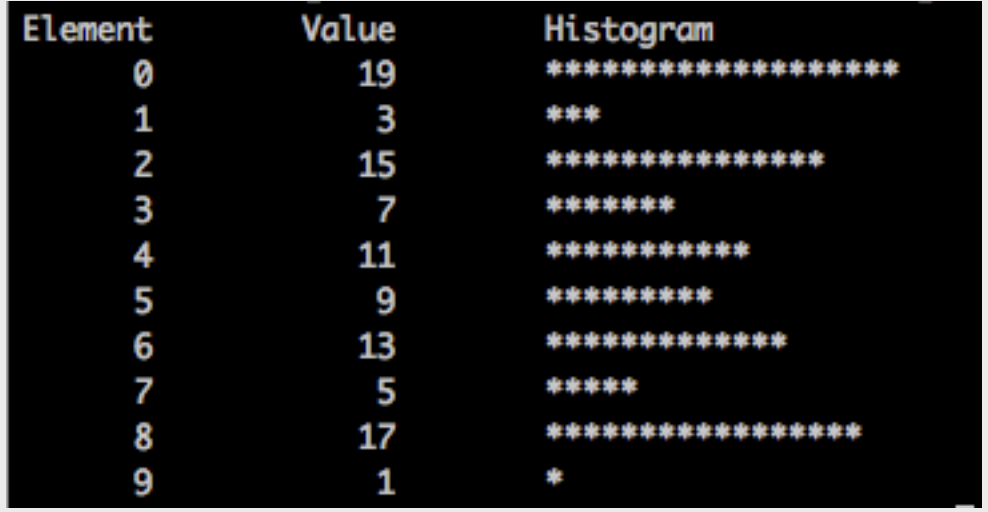
Array Examples (Cont.)

- Example: fig06_08.c



```c
int n[ SIZE ] = { 19, 3, 15, 7, 11, 9, 13, 5, 17, 1 };
int i; /* outer for counter for array elements */
int j; /* inner for counter counts *s in each histogram bar */

printf( "%s%13s%17s\n", "Element", "Value", "Histogram" );

/* for each element of array n, output a bar of the histogram */
for ( i = 0; i < SIZE; i++ ) {
    printf( "%7d%13d        ", i, n[ i ]) ;

    for ( j = 1; j <= n[ i ]; j++ ) { /* print one bar */
        printf( "%c", '*' );
    } /* end inner for */

    printf( "\n" ); /* end a histogram bar */
} /* end outer for */
```

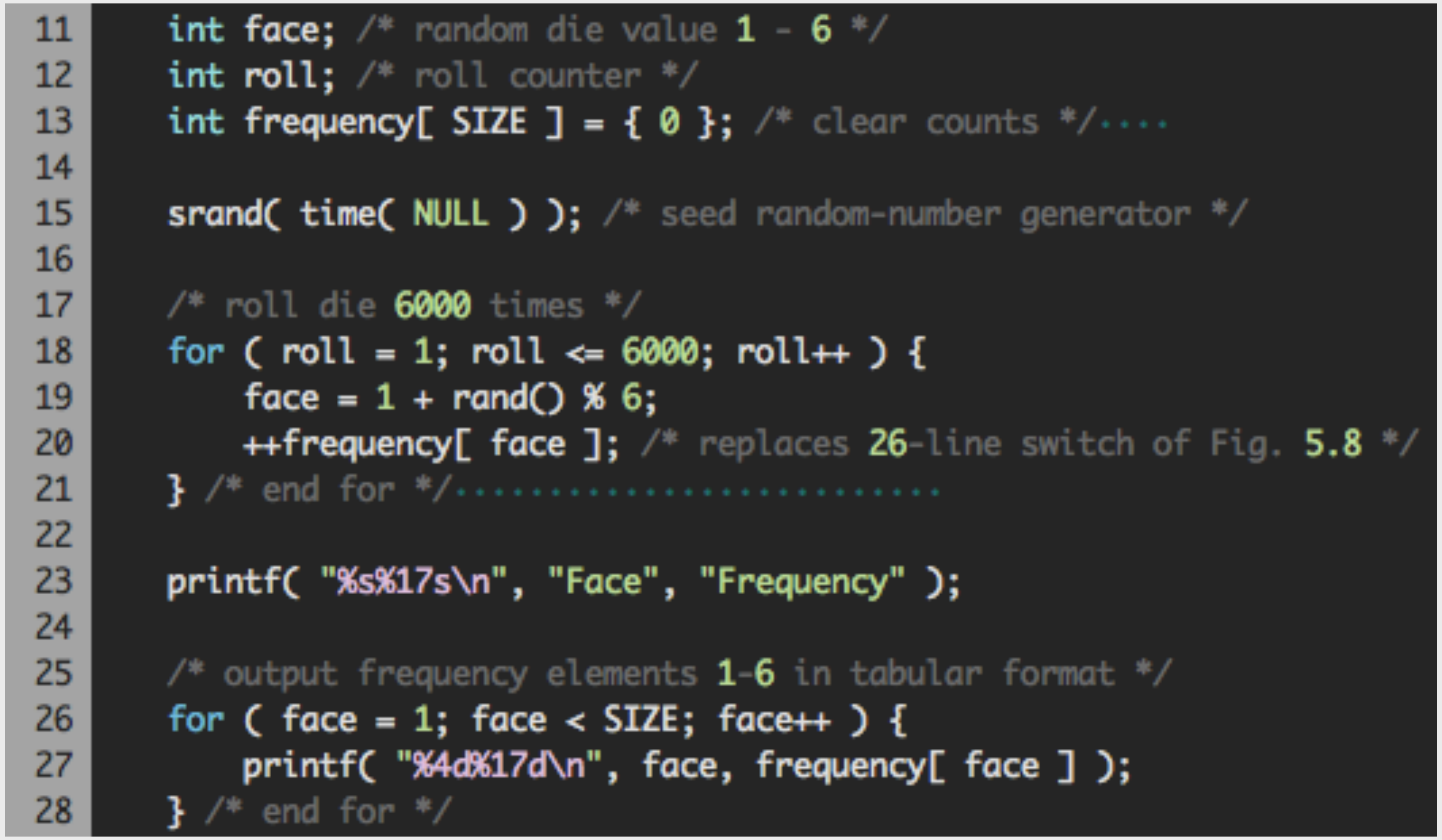
Reads numbers from an array and graphs the information in the form of a bar chart or histogram; the **for** statement draws the bars



```
Element        Value        Histogram
      0           19        *******************
      1            3        ***
      2           15        ***************
      3            7        *******
      4           11        ***********
      5            9        *********
      6           13        *************
      7            5        *****
      8           17        *****************
      9            1        *
```

Array Examples (Cont.)

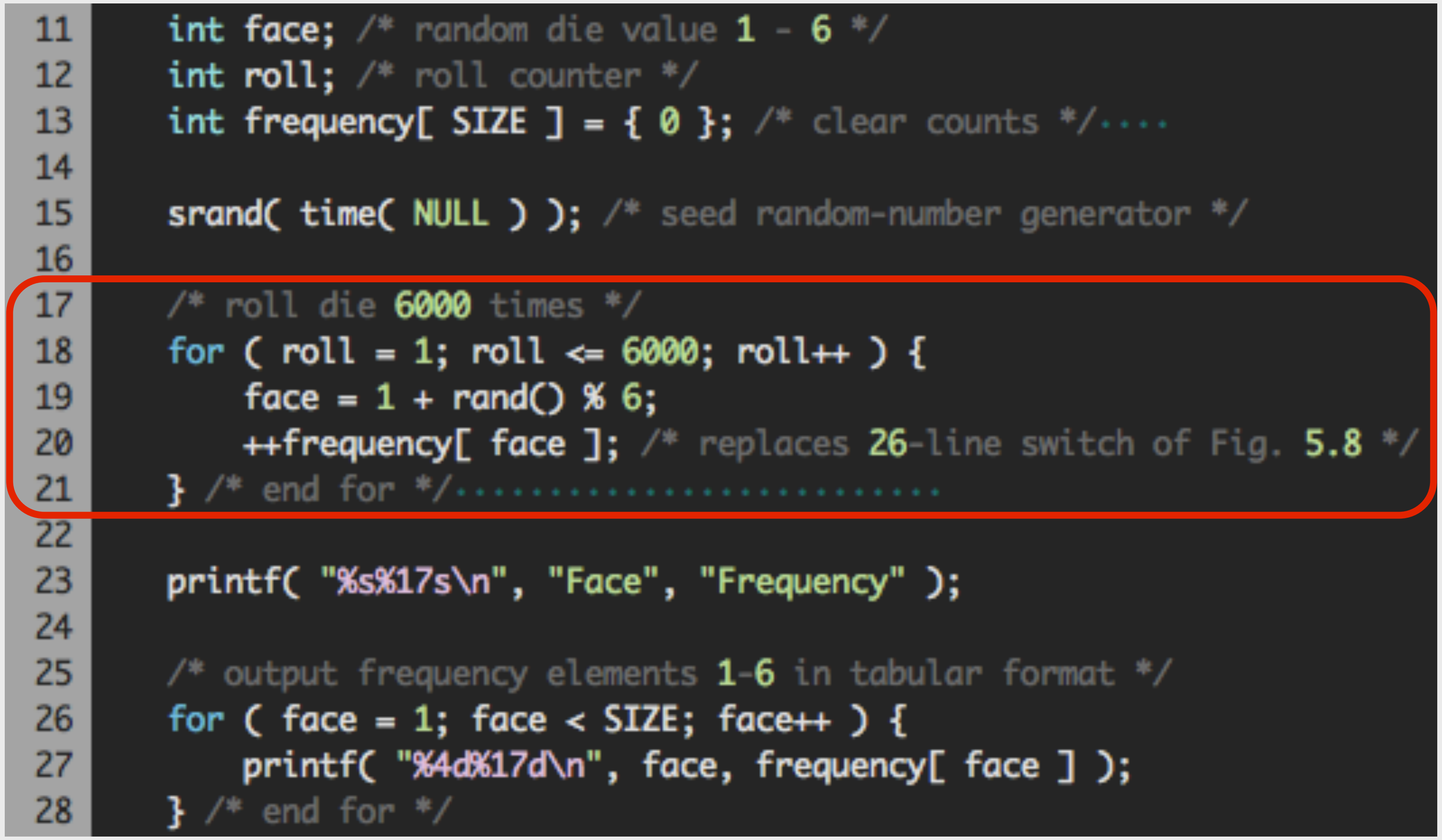
- Ex: fig06_09.c



```
    int face; /* random die value 1 - 6 */
    int roll; /* roll counter */
    int frequency[ SIZE ] = { 0 }; /* clear counts */

    srand( time( NULL ) ); /* seed random-number generator */

    /* roll die 6000 times */
    for ( roll = 1; roll <= 6000; roll++ ) {
        face = 1 + rand() % 6;
        ++frequency[ face ]; /* replaces 26-line switch of Fig. 5.8 */
    } /* end for */

    printf( "%s%17s\n", "Face", "Frequency" );

    /* output frequency elements 1-6 in tabular format */
    for ( face = 1; face < SIZE; face++ ) {
        printf( "%4d%17d\n", face, frequency[ face ] );
    } /* end for */
```

Array Examples (Cont.)

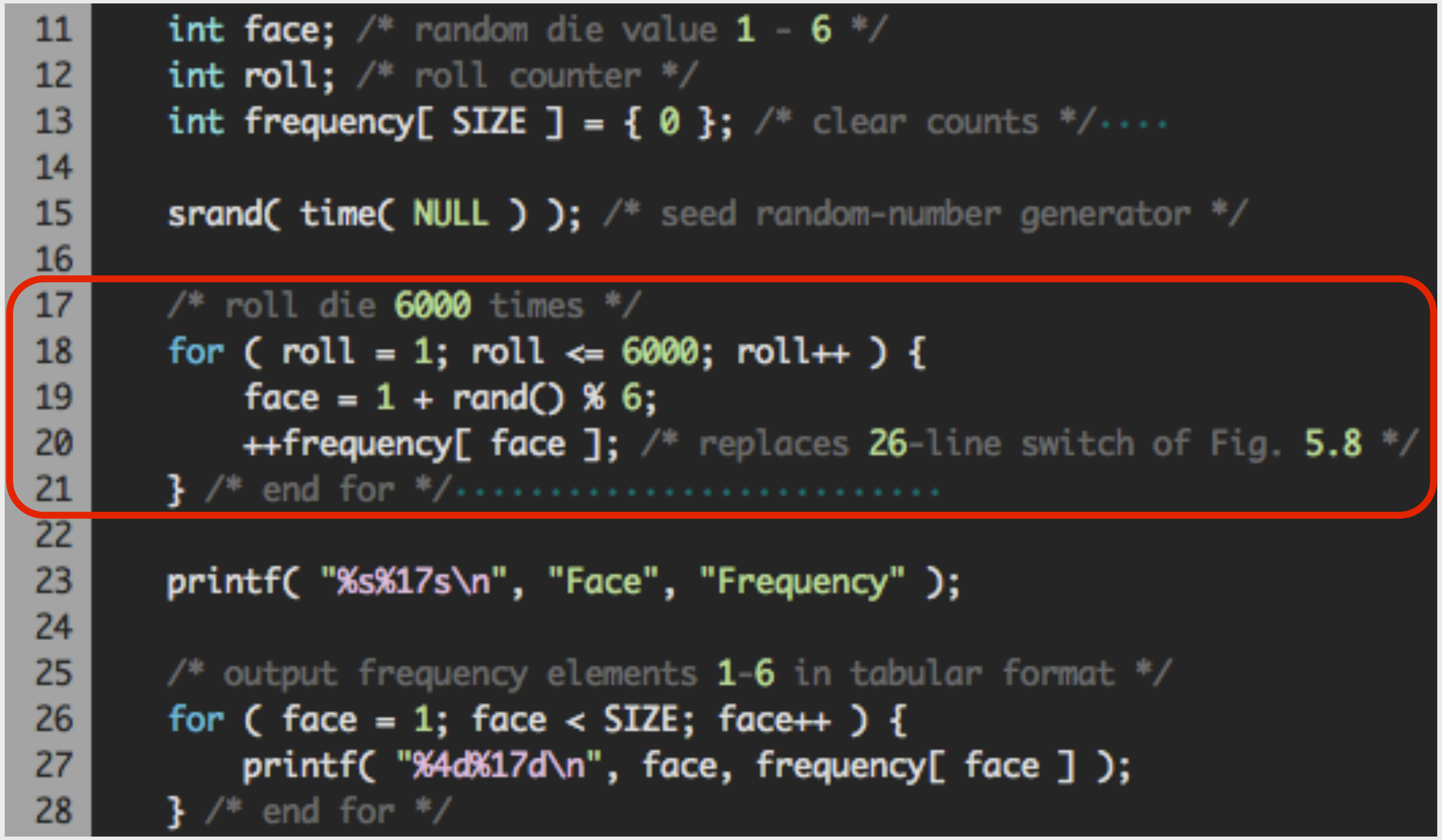
- Ex: fig06_09.c



```c
    int face; /* random die value 1 - 6 */
    int roll; /* roll counter */
    int frequency[ SIZE ] = { 0 }; /* clear counts */

    srand( time( NULL ) ); /* seed random-number generator */

    /* roll die 6000 times */
    for ( roll = 1; roll <= 6000; roll++ ) {
        face = 1 + rand() % 6;
        ++frequency[ face ]; /* replaces 26-line switch of Fig. 5.8 */
    } /* end for */

    printf( "%s%17s\n", "Face", "Frequency" );

    /* output frequency elements 1-6 in tabular format */
    for ( face = 1; face < SIZE; face++ ) {
        printf( "%4d%17d\n", face, frequency[ face ] );
    } /* end for */
```

Array Examples (Cont.)

- Ex: fig06_09.c

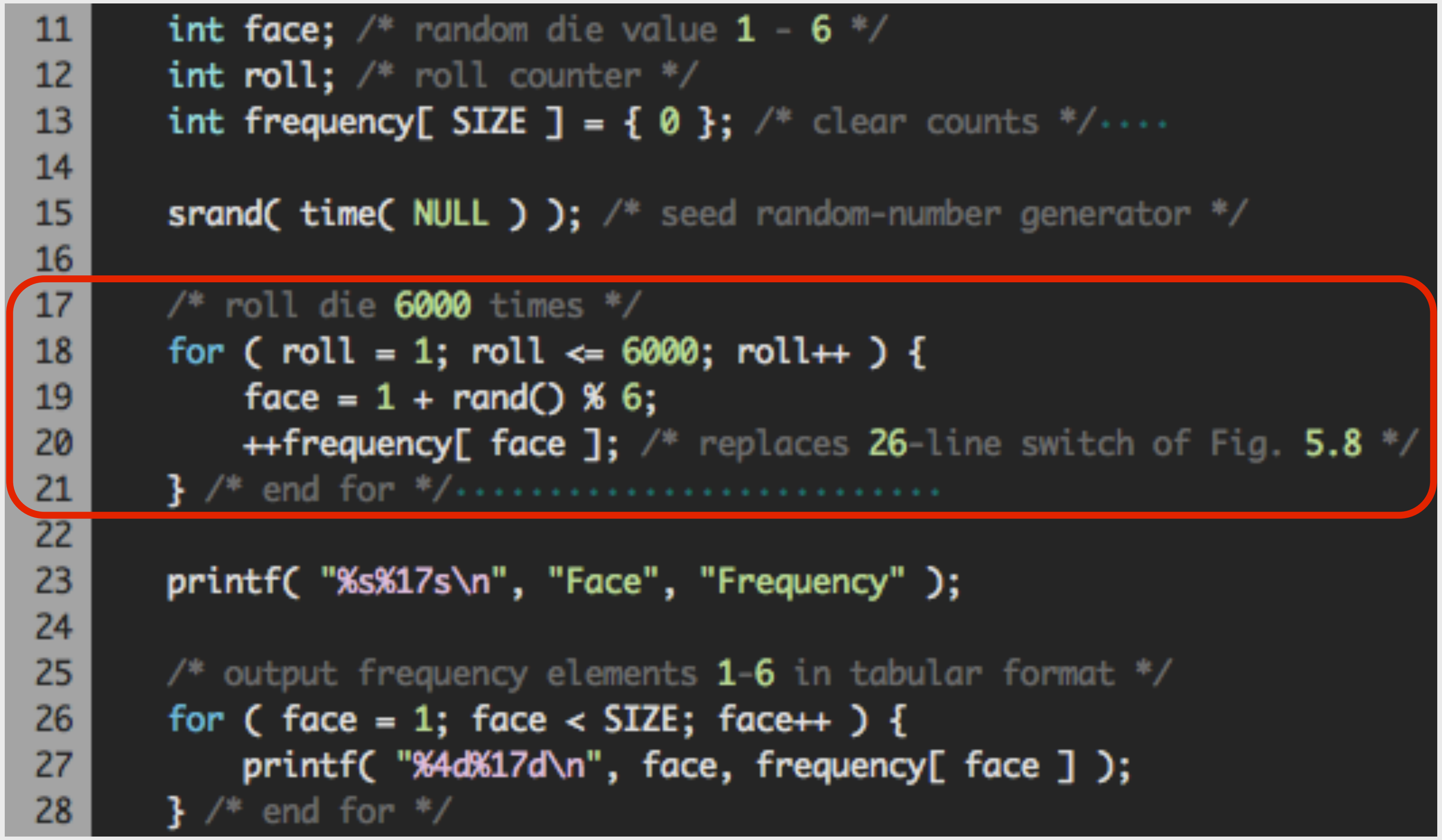


```
   int face; /* random die value 1 - 6 */
   int roll; /* roll counter */
   int frequency[ SIZE ] = { 0 }; /* clear counts */

   srand( time( NULL ) ); /* seed random-number generator */

   /* roll die 6000 times */
   for ( roll = 1; roll <= 6000; roll++ ) {
      face = 1 + rand() % 6;
      ++frequency[ face ]; /* replaces 26-line switch of Fig. 5.8 */
   } /* end for */

   printf( "%s%17s\n", "Face", "Frequency" );

   /* output frequency elements 1-6 in tabular format */
   for ( face = 1; face < SIZE; face++ ) {
      printf( "%4d%17d\n", face, frequency[ face ] );
   } /* end for */
```

Roll a single six-sided die 6000 times to test whether the random number generator actually produces random numbers.

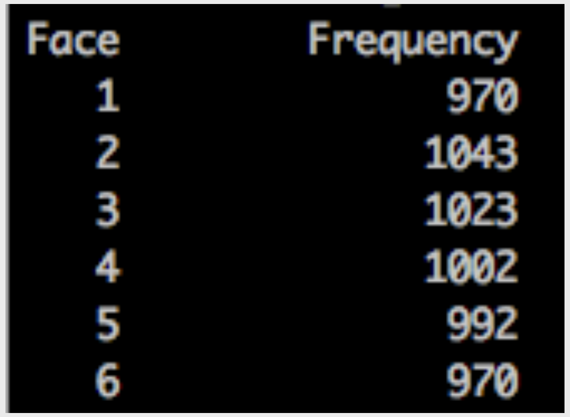
Array Examples (Cont.)

- Ex: fig06_09.c



```c
   int face; /* random die value 1 - 6 */
   int roll; /* roll counter */
   int frequency[ SIZE ] = { 0 }; /* clear counts */

   srand( time( NULL ) ); /* seed random-number generator */

   /* roll die 6000 times */
   for ( roll = 1; roll <= 6000; roll++ ) {
      face = 1 + rand() % 6;
      ++frequency[ face ]; /* replaces 26-line switch of Fig. 5.8 */
   } /* end for */

   printf( "%s%17s\n", "Face", "Frequency" );

   /* output frequency elements 1-6 in tabular format */
   for ( face = 1; face < SIZE; face++ ) {
      printf( "%4d%17d\n", face, frequency[ face ] );
   } /* end for */
```

Roll a single six-sided die 6000 times to test whether the random number generator actually produces random numbers.



```
Face        Frequency
   1              970
   2             1043
   3             1023
   4             1002
   5              992
   6              970
```