Chapter 3: Top-Down Design with Functions

Problem Solving & Program Design in C

Eighth Edition

By Jeri R. Hanly &
Elliot B. Koffman

Addison Wesley
is an imprint of

Outline

3.1 BUILDING PROGRAMS FROM EXISING INFORMATION

- *CASE STUDY:*
  - *FINDING THE AREA AND CIRCUMFERENCE OF A CIRCLE*
- *CASE STUDY:*
  - *FINDING THE WEIGHT OF A BATCH OF FLAT WASHERS*

3.2 LIBRARY FUNCTION

3.3 TOP-DOWN DESIGN AND STRUCTURE CHARTS

- *CASE STUDY:DRAWING SIMPLE DIAGRAMS*

3.4 FUNCTIONS WITHOUT ARGUMENTS

3.5 FUNCTIONS WITH INPUT ARGUMENTS

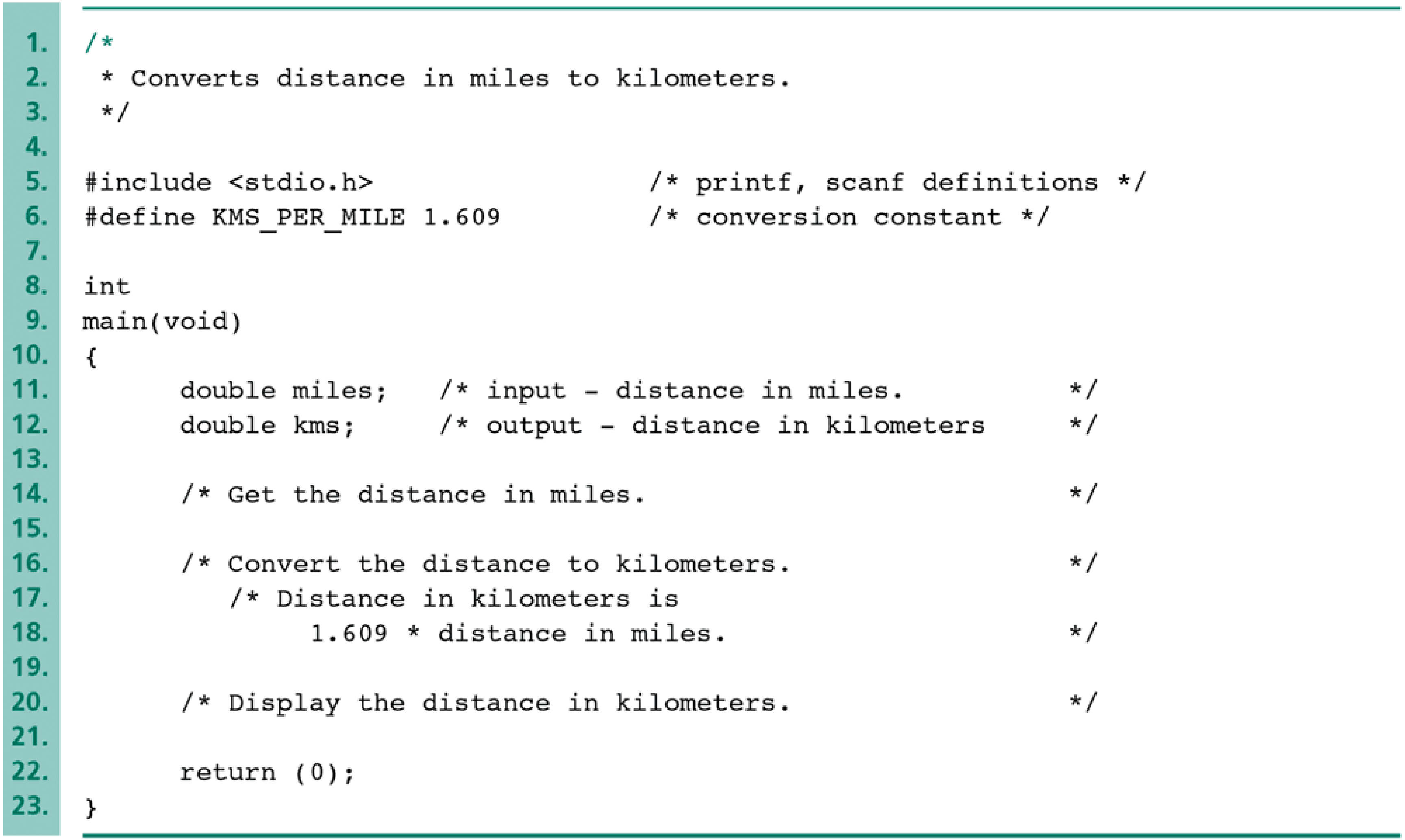
3.6 INTRODUCITON TO COMPUTER GRAPHICS

3.7 COMMON PROGRAMMING ERRORS

3.1 Building Programs from Existing Information

- Code a program steps
  - Software method generate a system documents
  - Coding (ex. Figure 3.1)
    - First editing data requirements
    - Second conform to C syntax for constant macro and variable declarations
  - Develop executable statements
    - First: use the initial algorithm and refinements as program comments
    - Second: write C statements for unrefined step
    - Third: editing refinement to replace it with C code

Figure 3.1 Edited Data Requirements and Algorithm for Conversion Program



```
/*
 * Converts distance in miles to kilometers.
 */

#include <stdio.h>                 /* printf, scanf definitions */
#define KMS_PER_MILE 1.609         /* conversion constant */

int
main(void)
{
      double miles;   /* input - distance in miles.         */
      double kms;     /* output - distance in kilometers    */

      /* Get the distance in miles.                         */

      /* Convert the distance to kilometers.                */
         /* Distance in kilometers is
              1.609 * distance in miles.                    */

      /* Display the distance in kilometers.                */

      return (0);
}
```

Case Study
Finding the Area and Circumference of a circle(1/4)

Step 1: Problem

- Get the radius of a circle. Compute and display the circle's area and circumference.

Case Study
Finding the Area and Circumference of a circle(2/4)

Step 2: Analysis

- Problem Constant
  - PI 3.14159
- Problem Inputs
  - radius
- Problem Outputs
  - area
  - circum
- Relevant Formulas
  - *Area of a circle = π x radius*^{2}
  - *Circumference of a circle = 2 π x radius*

Case Study
Finding the Area and Circumference of a circle(3/4)

Step 3: Design

- Initial Algorithm
  1. Get the circle radius
  2. Calculate the area
  3. Calculate the circumference
  4. Display the area and the circumference
- Algorithm Refinement

  2.1 Assign PI * radius * radius to area

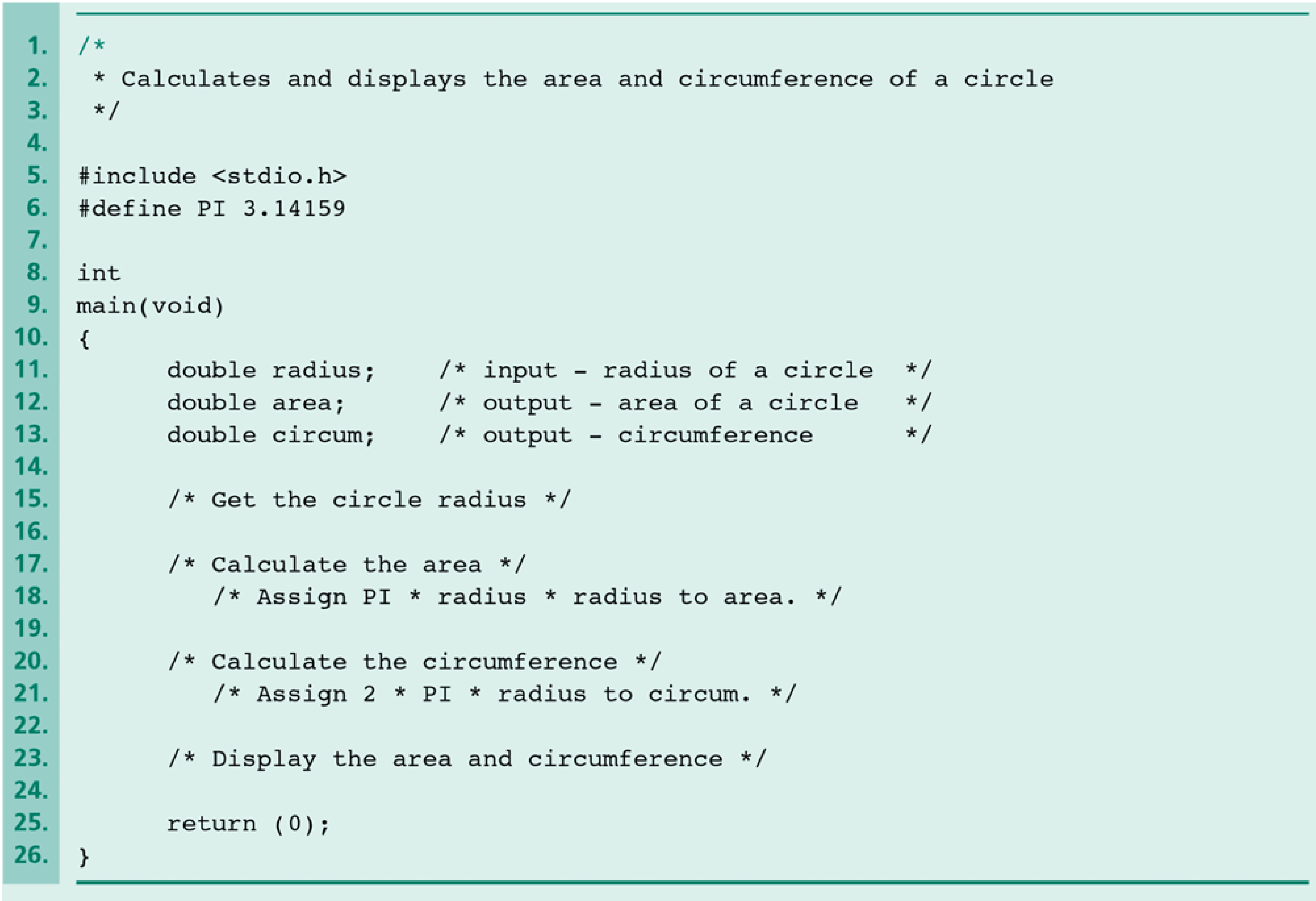
  3.1 Assign 2 * PI * radius to circum

Case Study
Finding the Area and Circumference of a circle(4/4)

Step 4: Implementation (Figure 3.2、Figure 3.3)

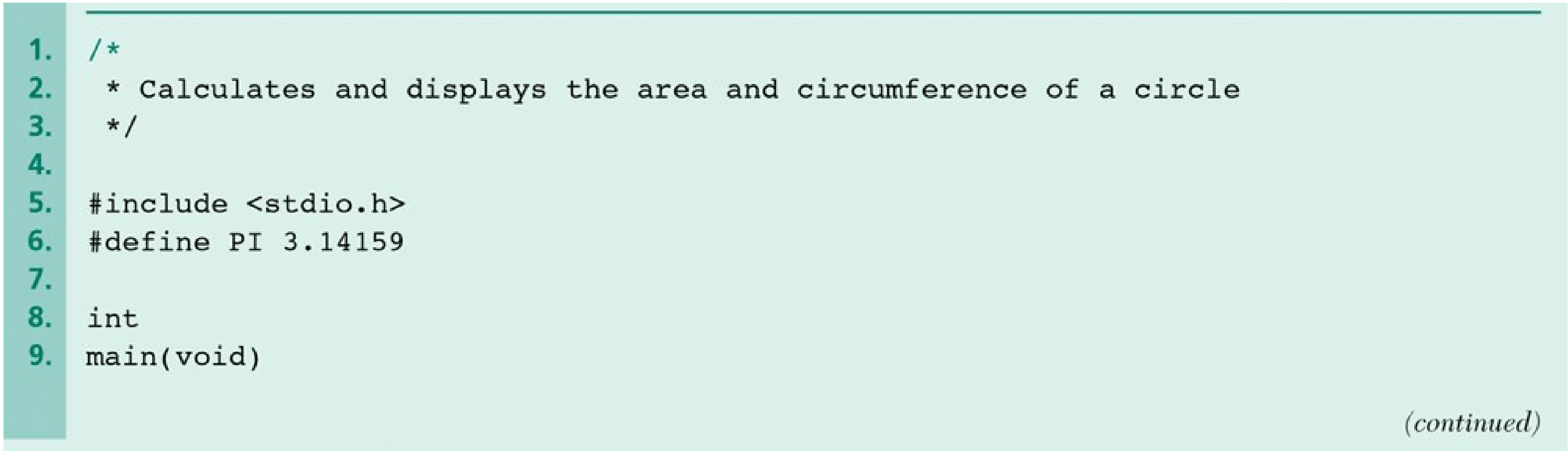
Step 5: Testing

Figure 3.2 Outline of Program Circle



```
/*
 * Calculates and displays the area and circumference of a circle
 */

#include <stdio.h>
#define PI 3.14159

int
main(void)
{
      double radius;    /* input - radius of a circle  */
      double area;      /* output - area of a circle   */
      double circum;    /* output - circumference      */

      /* Get the circle radius */

      /* Calculate the area */
         /* Assign PI * radius * radius to area. */

      /* Calculate the circumference */
         /* Assign 2 * PI * radius to circum. */

      /* Display the area and circumference */

      return (0);
}
```

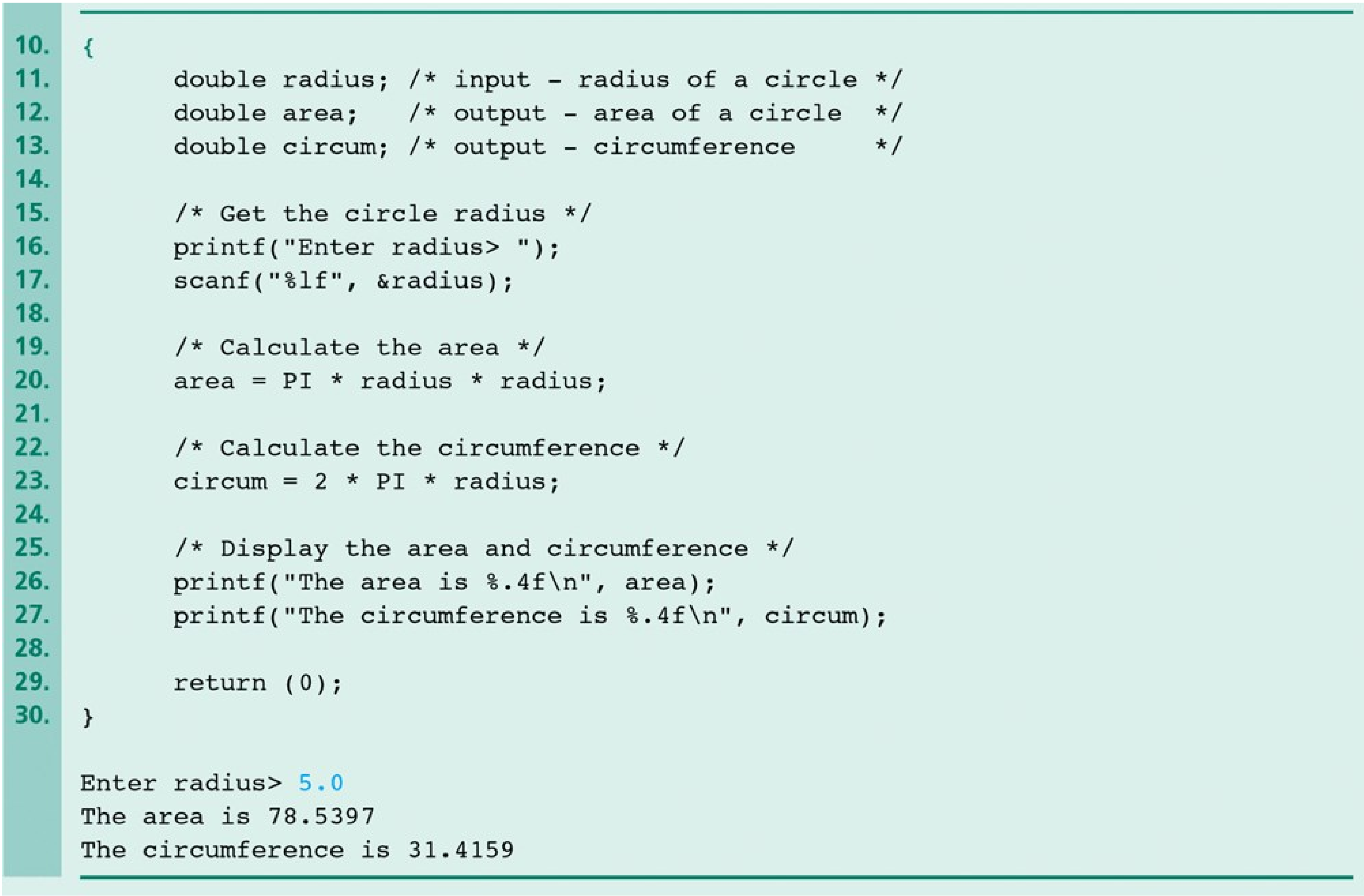
Figure 3.3 Calculating the Area and the Circumference of a Circle



```
/*
 * Calculates and displays the area and circumference of a circle
 */

#include <stdio.h>
#define PI 3.14159

int
main(void)
```

(continued)

Figure 3.3 Calculating the Area and the Circumference of a Circle (cont'd)



```
{
      double radius; /* input - radius of a circle */
      double area;   /* output - area of a circle  */
      double circum; /* output - circumference     */

      /* Get the circle radius */
      printf("Enter radius> ");
      scanf("%lf", &radius);

      /* Calculate the area */
      area = PI * radius * radius;

      /* Calculate the circumference */
      circum = 2 * PI * radius;

      /* Display the area and circumference */
      printf("The area is %.4f\n", area);
      printf("The circumference is %.4f\n", circum);

      return (0);
}
```

```
Enter radius> 5.0
The area is 78.5397
The circumference is 31.4159
```

Case Study

Computing the Weight of a Batch of Flat Washers(1/5)

Step 1: Problem

- You work for a hardware company that manufactures flat washers. To estimate shipping costs, your company needs a program that computes the weight of a specified quality of flat washers.

Case Study
Computing the Weight of a Batch of Flat Washers(2/5)

Step 2: Analysis

- Problem Constant
  - PI 3.14159
- Problem Inputs
  - double hole_diameter
  - double edge_diameter
  - double thickness
  - double density
  - double quantity
- Problem Outputs
  - double weight

Case Study
Computing the Weight of a Batch of Flat Washers(3/5)

- Program Variables
  - double hole_radius
  - double edge_radius
  - double rim_area
  - double unit_weight
- Relevant Formulas
  - area of a circle = π x radius^2
  - radius of a circle = diameter / 2
  - rim area = area of outer circle – area of hole
  - unit weight = rim area x thickness x density

Case Study

Computing the Weight of a Batch of Flat Washers(4/5)

Step 3: Design

- Initial algorithm
  1. Get the washer's inner diameter, outer diameter, and thickness
  2. Get the material density and quantity of washer's manufactured
  3. Compute the rim area
  4. Compute the weight of one flat washer
  5. Compute the weight of the batch of washers
  6. Display the weight of he batch of washers
- Refinement

  3.1 Compute hole_rasius and edge_radius

  3.2 rim_area is PI * edge_radius * edge_radius – PI * hole_radius * hole_radius

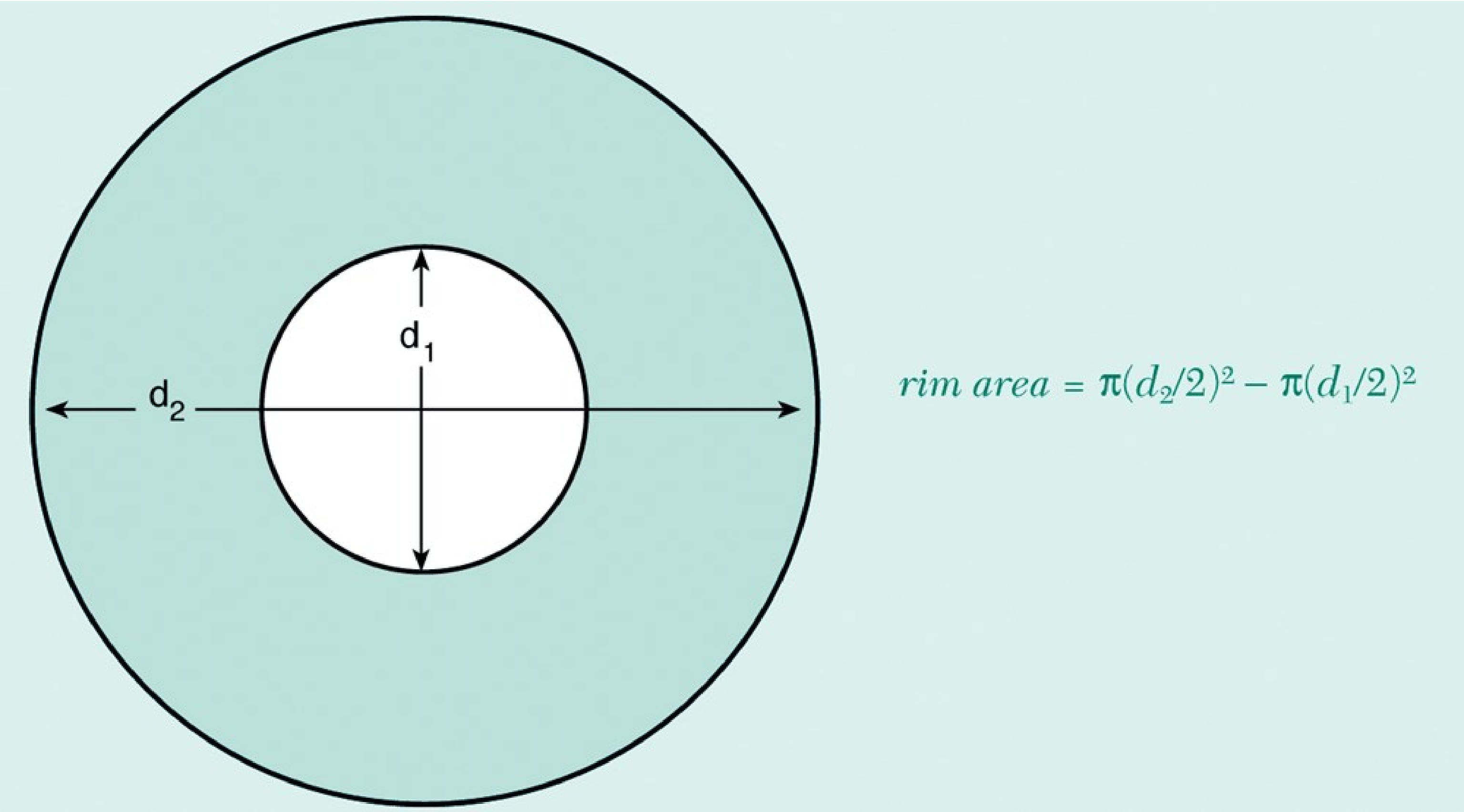
  4.1 unit_weight is rim_area * thickness * density

Case Study
Computing the Weight of a Batch of Flat Washers(5/5)

Step 4: Implementation (Figure 3.4 、 Figure 3.5)

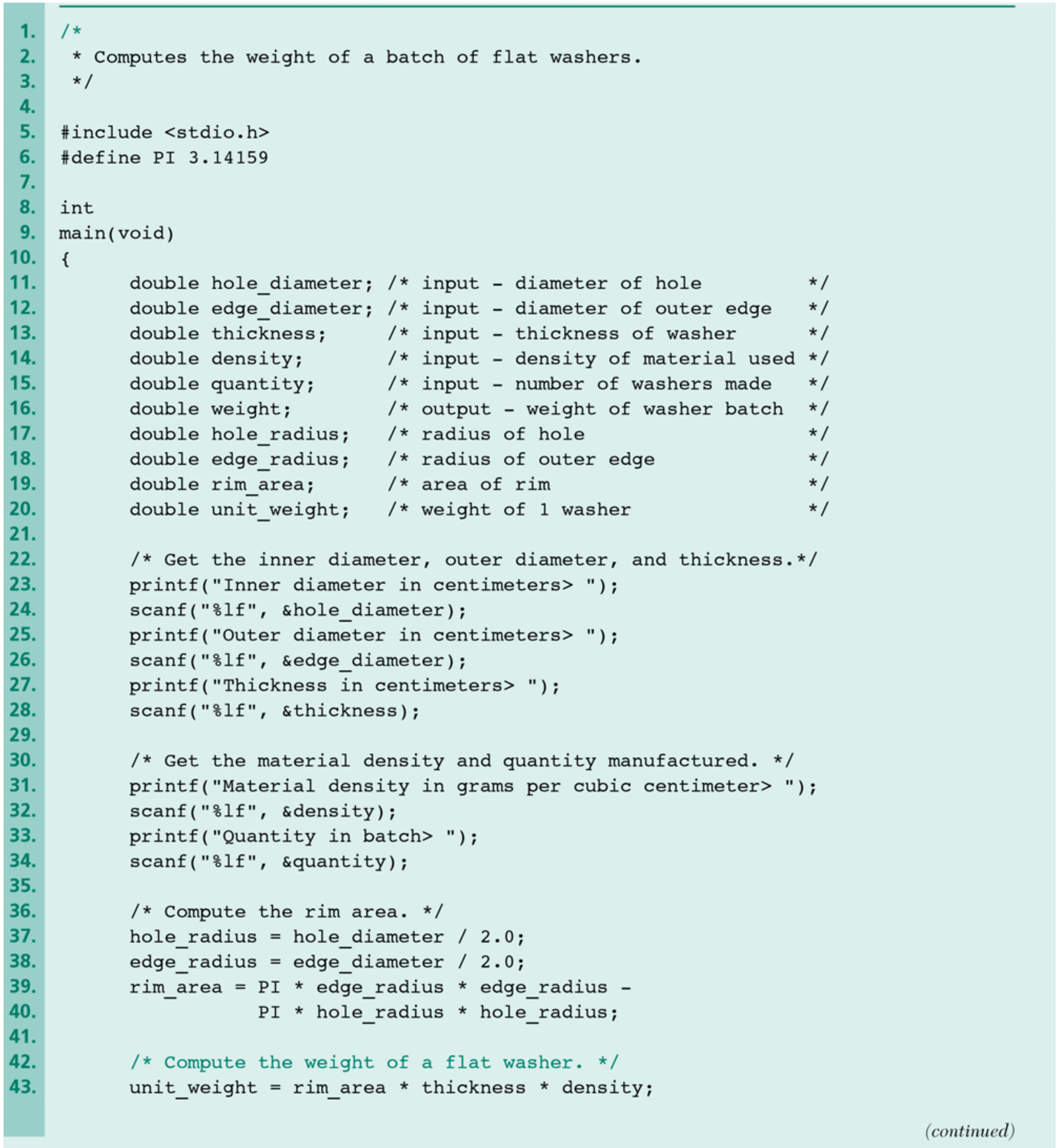
Step 5: Testing

Figure 3.4 Computing the Rim Area of a Flat Washer



$$\textit{rim area} = \pi(d_2/2)^2 - \pi(d_1/2)^2$$

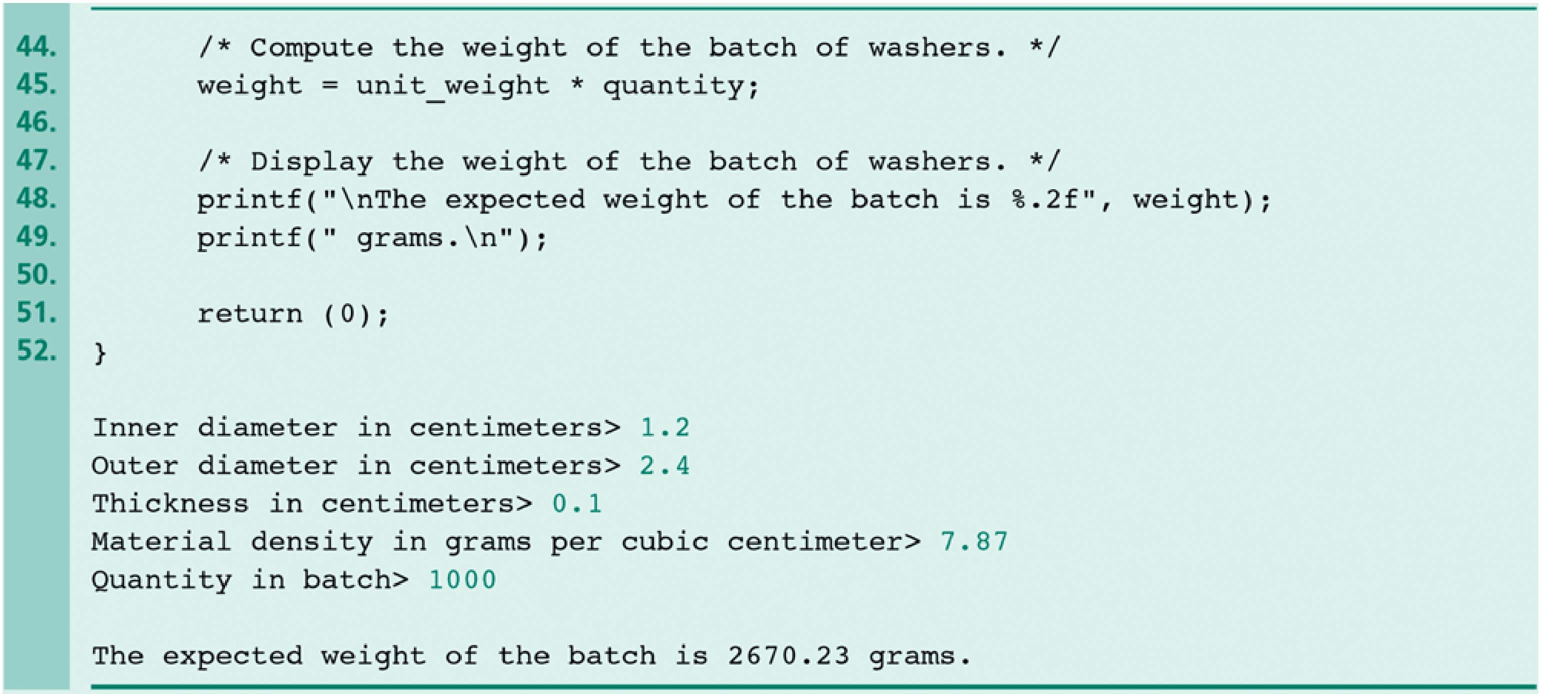
Figure 3.5 Flat Washer Program



```c
/*
 * Computes the weight of a batch of flat washers.
 */

#include <stdio.h>
#define PI 3.14159

int
main(void)
{
      double hole_diameter; /* input - diameter of hole          */
      double edge_diameter; /* input - diameter of outer edge    */
      double thickness;     /* input - thickness of washer       */
      double density;       /* input - density of material used */
      double quantity;      /* input - number of washers made    */
      double weight;        /* output - weight of washer batch  */
      double hole_radius;   /* radius of hole                    */
      double edge_radius;   /* radius of outer edge              */
      double rim_area;      /* area of rim                       */
      double unit_weight;   /* weight of 1 washer                */

      /* Get the inner diameter, outer diameter, and thickness.*/
      printf("Inner diameter in centimeters> ");
      scanf("%lf", &hole_diameter);
      printf("Outer diameter in centimeters> ");
      scanf("%lf", &edge_diameter);
      printf("Thickness in centimeters> ");
      scanf("%lf", &thickness);

      /* Get the material density and quantity manufactured. */
      printf("Material density in grams per cubic centimeter> ");
      scanf("%lf", &density);
      printf("Quantity in batch> ");
      scanf("%lf", &quantity);

      /* Compute the rim area. */
      hole_radius = hole_diameter / 2.0;
      edge_radius = edge_diameter / 2.0;
      rim_area = PI * edge_radius * edge_radius -
                 PI * hole_radius * hole_radius;

      /* Compute the weight of a flat washer. */
      unit_weight = rim_area * thickness * density;
```

(continued)

Figure 3.5 Flat Washer Program (cont'd)



```
     /* Compute the weight of the batch of washers. */
     weight = unit_weight * quantity;

     /* Display the weight of the batch of washers. */
     printf("\nThe expected weight of the batch is %.2f", weight);
     printf(" grams.\n");

     return (0);
}

Inner diameter in centimeters> 1.2
Outer diameter in centimeters> 2.4
Thickness in centimeters> 0.1
Material density in grams per cubic centimeter> 7.87
Quantity in batch> 1000

The expected weight of the batch is 2670.23 grams.
```

A Program using given function

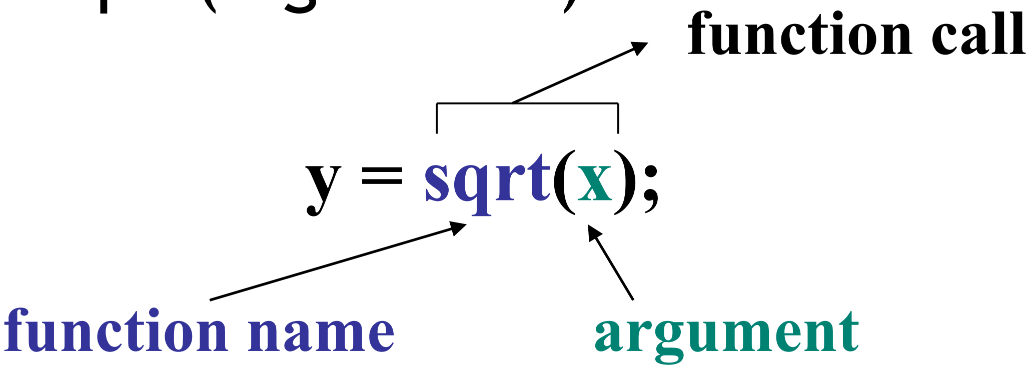
- Funtion find_area(r): return the area of a circle
- For the falt washer problem

  rim_area

  = find_area(edge_radius)-find_area(hole_radius)

3.2 Library Functions

- Code reuse
  - one way to accomplish writing error-free code
- C provides predefined functions
  - used to perform mathematical computations
- C standard math library
  - ex. sqrt (Figure 3.6)



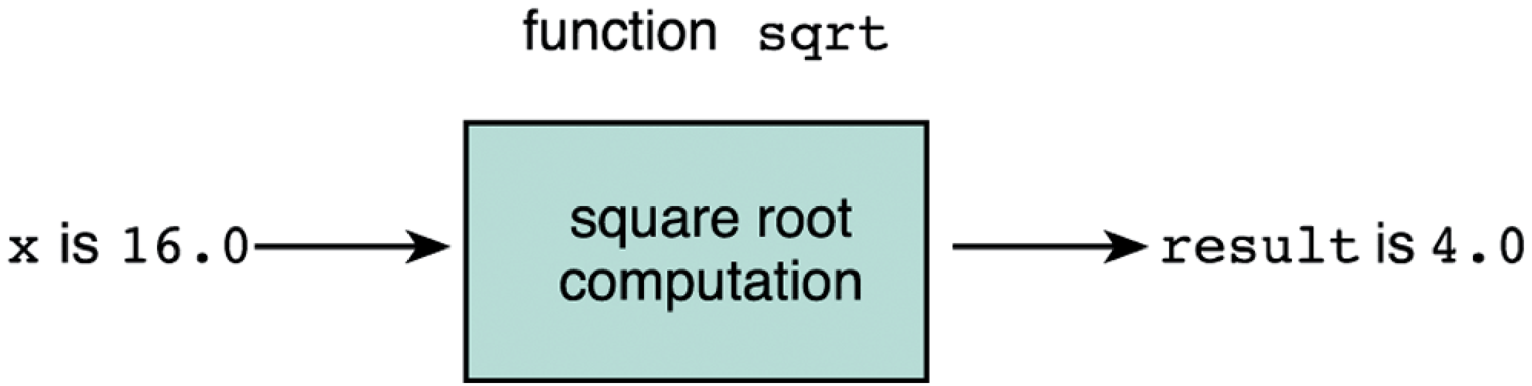
function call

y = sqrt(x);

function name

argument

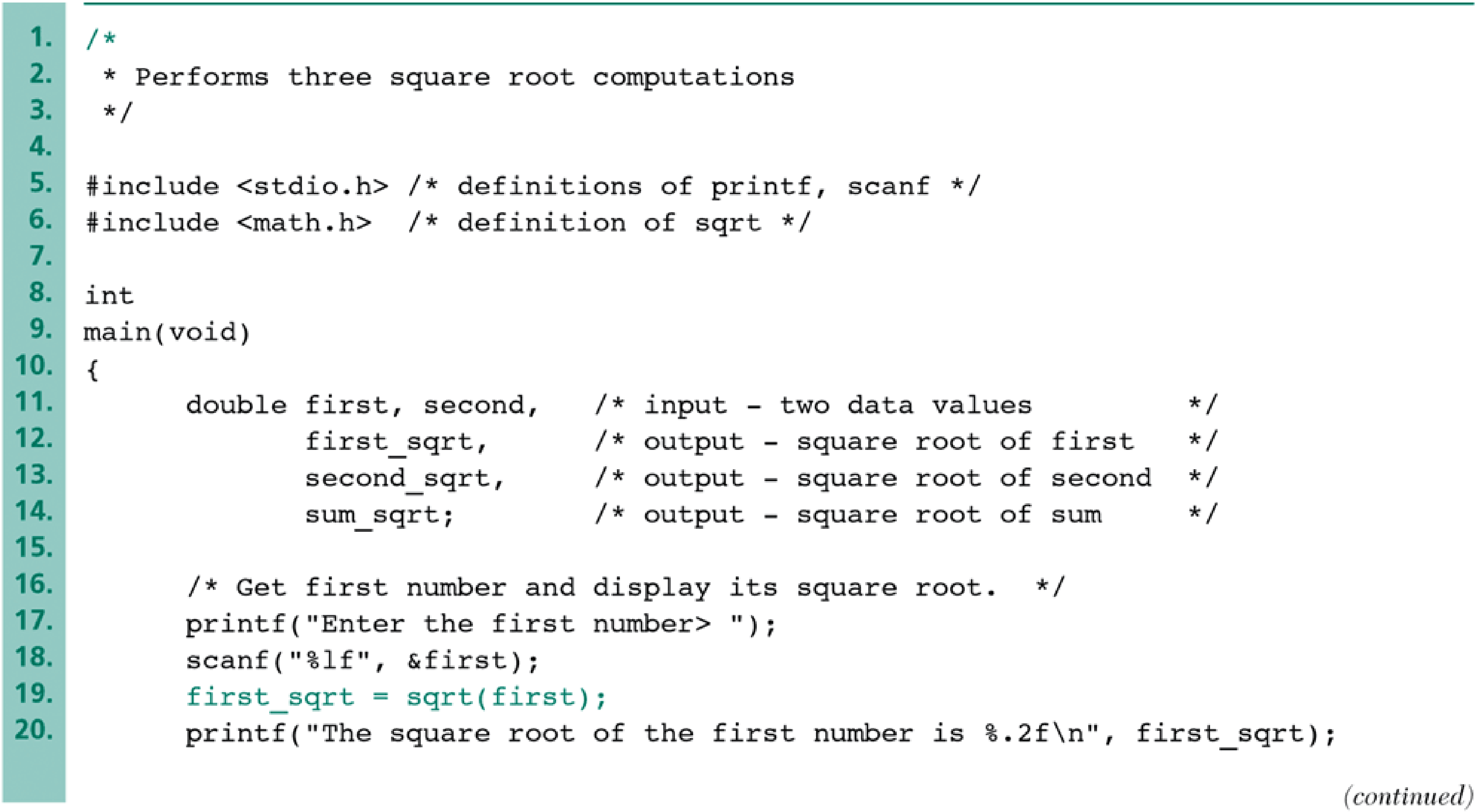
Figure 3.6 Function sqrt as a “Black Box”



Example

- Example 3.1 (Figure 3.7)
  - The square root of two numbers provided as input data (first and second) and the square root of their sum. To do so, it must call the C function sqrt three times
    - first_sqrt = sqrt (first);
    - second_sqrt = sqrt(second);
    - sum_sqrt = sqrt (first + second);

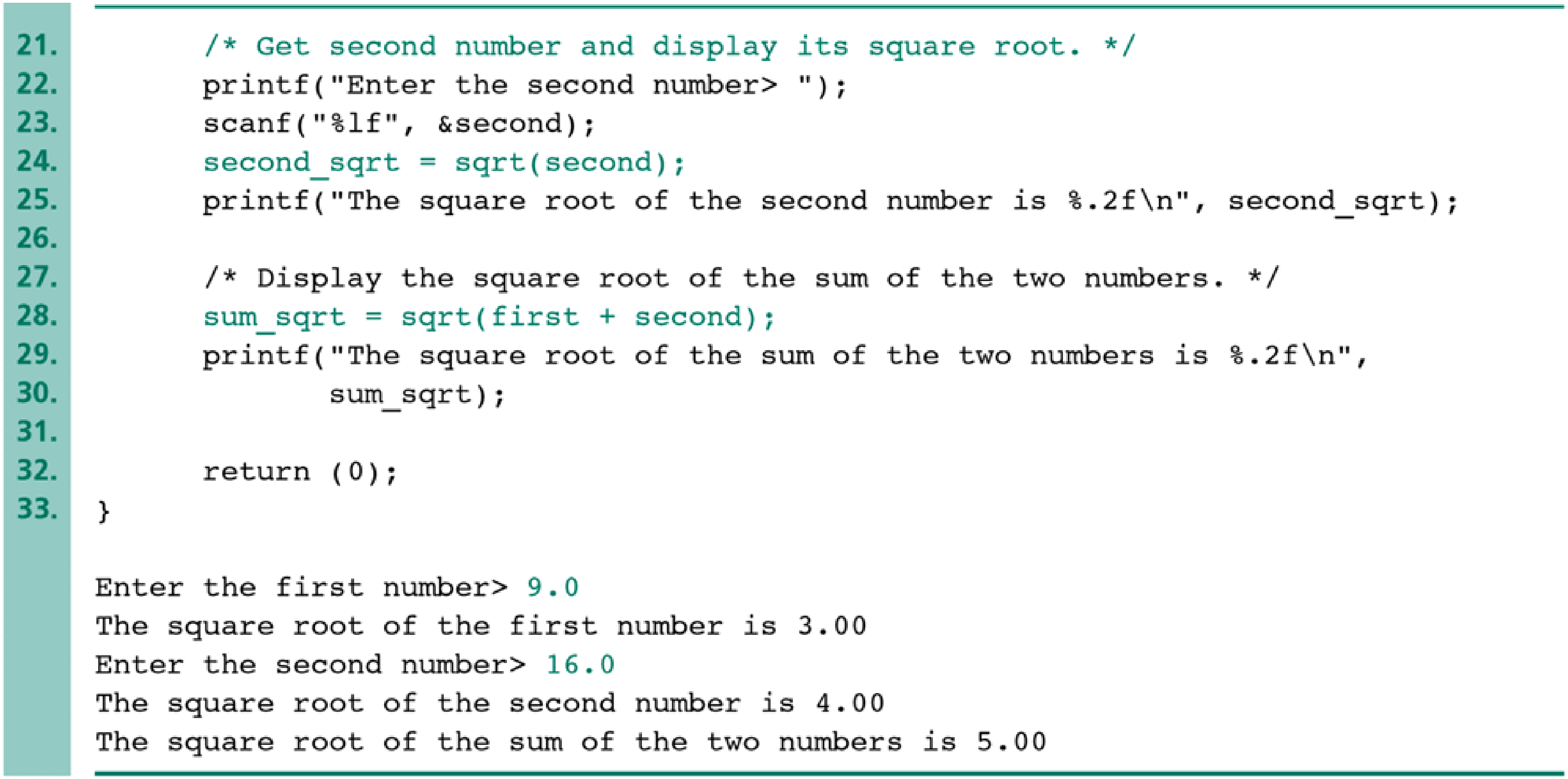
Figure 3.7 Square Root Program



```c
/*
 * Performs three square root computations
 */

#include <stdio.h> /* definitions of printf, scanf */
#include <math.h>  /* definition of sqrt */

int
main(void)
{
      double first, second,   /* input - two data values         */
             first_sqrt,      /* output - square root of first   */
             second_sqrt,     /* output - square root of second  */
             sum_sqrt;        /* output - square root of sum     */

      /* Get first number and display its square root.  */
      printf("Enter the first number> ");
      scanf("%lf", &first);
      first_sqrt = sqrt(first);
      printf("The square root of the first number is %.2f\n", first_sqrt);
```

(continued)

Figure 3.7 Square Root Program (cont’d)



```
      /* Get second number and display its square root. */
      printf("Enter the second number> ");
      scanf("%lf", &second);
      second_sqrt = sqrt(second);
      printf("The square root of the second number is %.2f\n", second_sqrt);

      /* Display the square root of the sum of the two numbers. */
      sum_sqrt = sqrt(first + second);
      printf("The square root of the sum of the two numbers is %.2f\n",
             sum_sqrt);

      return (0);
}
```

```
Enter the first number> 9.0
The square root of the first number is 3.00
Enter the second number> 16.0
The square root of the second number is 4.00
The square root of the sum of the two numbers is 5.00
```

Table 3.1 Some Mathematical Functions(1/3)

Function	Standard Header File	Example	Argument(s)	Result
abs(x)	<stdio.h>	x=-5 abs(x)=5	int	int
ceil(x)	<math.h>	x=45.23 ceil(x)=46	double	double
cos(x)	<math.h>	x=0.0 cos(x)=1.0	double (radians)	double
exp(x)	<math.h>	x=1.0 exp(x)=2.71828	double	double

Table 3.1 Some Mathematical Functions (2/3)

Function	Standard Header File	Example	Argument(s)	Result
fabs(x)	<math.h>	x=-8.432 fab(x)=8.432	double	double
floor(x)	<math.h>	x=45.23 floor(x)=45	double	double
log(x)	<math.h>	x=2.71828 log(x)=1.0	double	double
log10(x)	<math.h>	x=100.0 log10(x)=2.0	double	double

Table 3.1 Some Mathematical Functions (3/3)

Function	Standard Header File	Example	Argument(s)	Result
pow(x,y)	<math.h>	x=0.16 y=0.5 pow(x,y)=0.4	double double	double
sin(x)	<math.h>	x=1.5708 sin(x)=1.0	double (radians)	double
sqrt(x)	<math.h>	x=2.25 sqrt(x)=1.5	double	double
tan(x)	<math.h>	x=0.0 tan(x)=0.0	double (radians)	double

Example

- Example 3.2
  - We can use C funtion *pow* and *sqrt* to compute the roots of a quadratic equation in *x* of the form

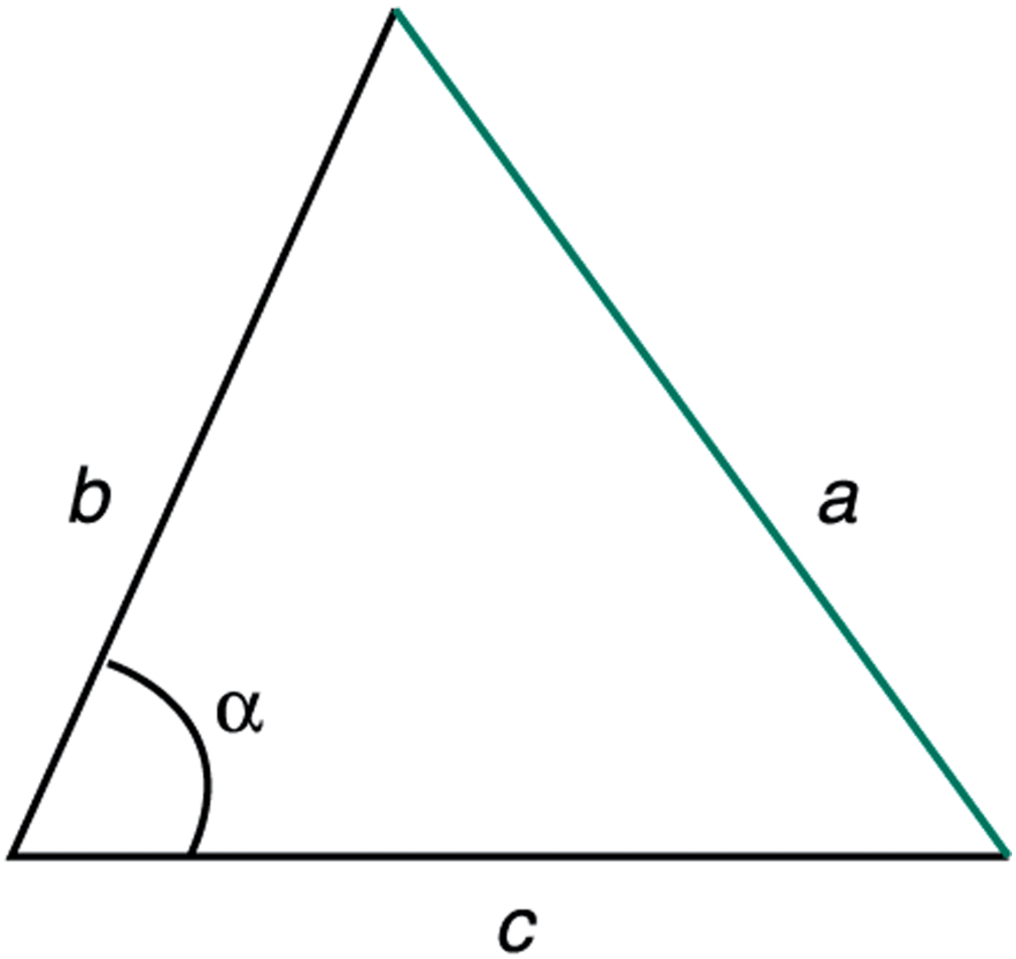
$$ax^2+bx+c=0 \quad \Rightarrow \quad root=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$$

```
/* Compute two roots, for disc > 0.0 */
    disc=pow(b,2)-4*a*c;
    root_1=(-b+sqrt(disc)) / (2*a);
    root_2=(-b-sqrt(disc)) / (2*a);
```

Example

- Example 3.3 (Figure 3.8)
  - If we know the lengths of two sides (b and c) of a triangle and the angle between them in degrees(α), we can compute the length of the third side(a) using the following formula
    - $a^2=b^2+c^2-2bc \cos\alpha$
    - a=sqrt(pow(b,2)+pow(c,2) - 2 * b* c* cos(alpha * PI / 180.0));

Figure 3.8 Triangle with Unknown Side a



3.3 Top-Down Design and Structure Charts

- Top-down design
  - a problem-solving method in which you first break a problem up into its major subproblems and then solve the subproblems to derive the solutions to the original problem
- Structure chart
  - a documentation tool that shows the relationship among the subproblems of a problem

Case Study:
Drawing Simple Diagrams(1/2)

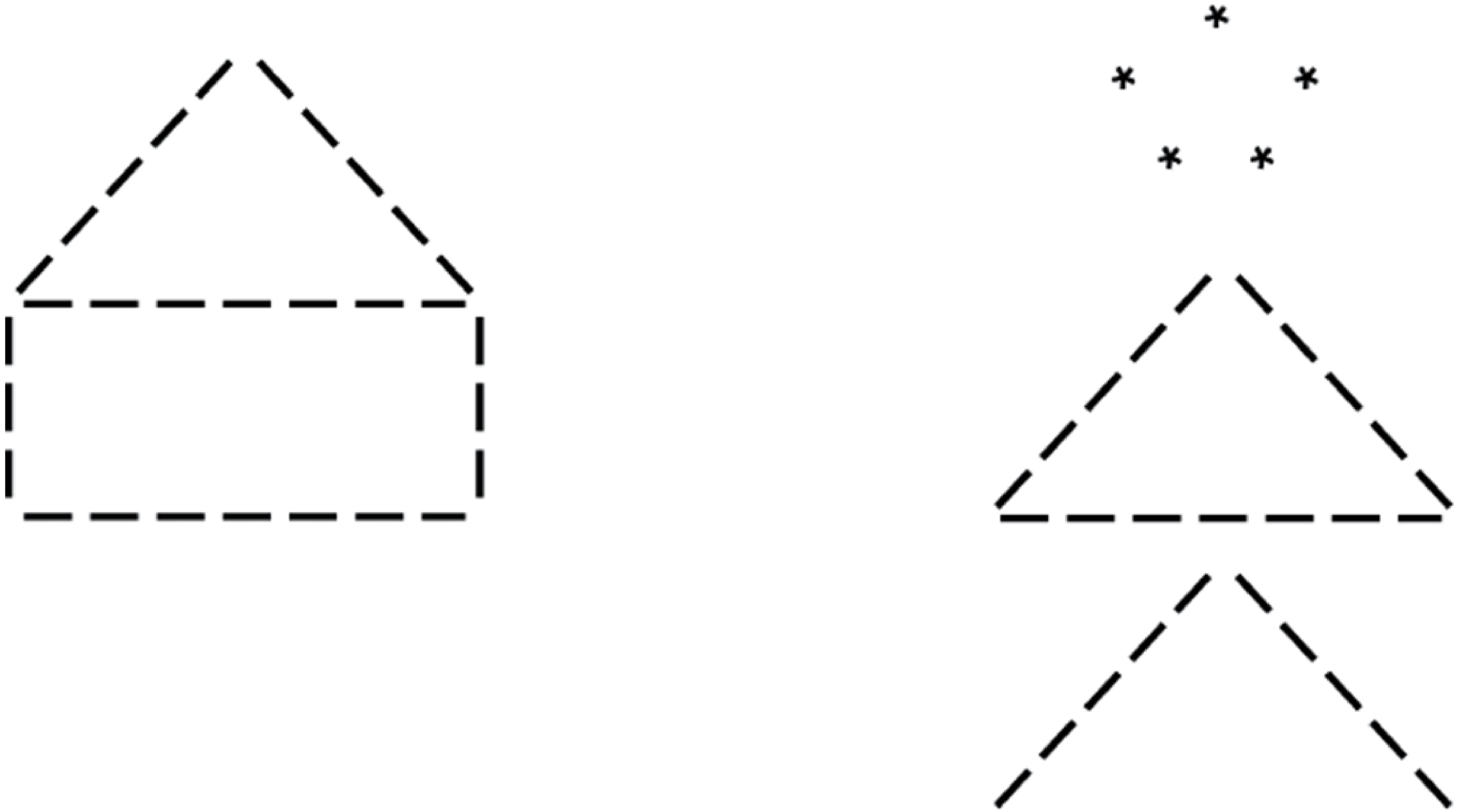
Step 1: Problem

- You want to draw some simple diagrams on your printer or screen. Two examples are the house and female stick figure in Fig.3.9.

Step 2: Analysis

- four basic components
  - a circle
  - a base line
  - parallel lines
  - Intersecting lines

Figure 3.9 House and Stick Figure

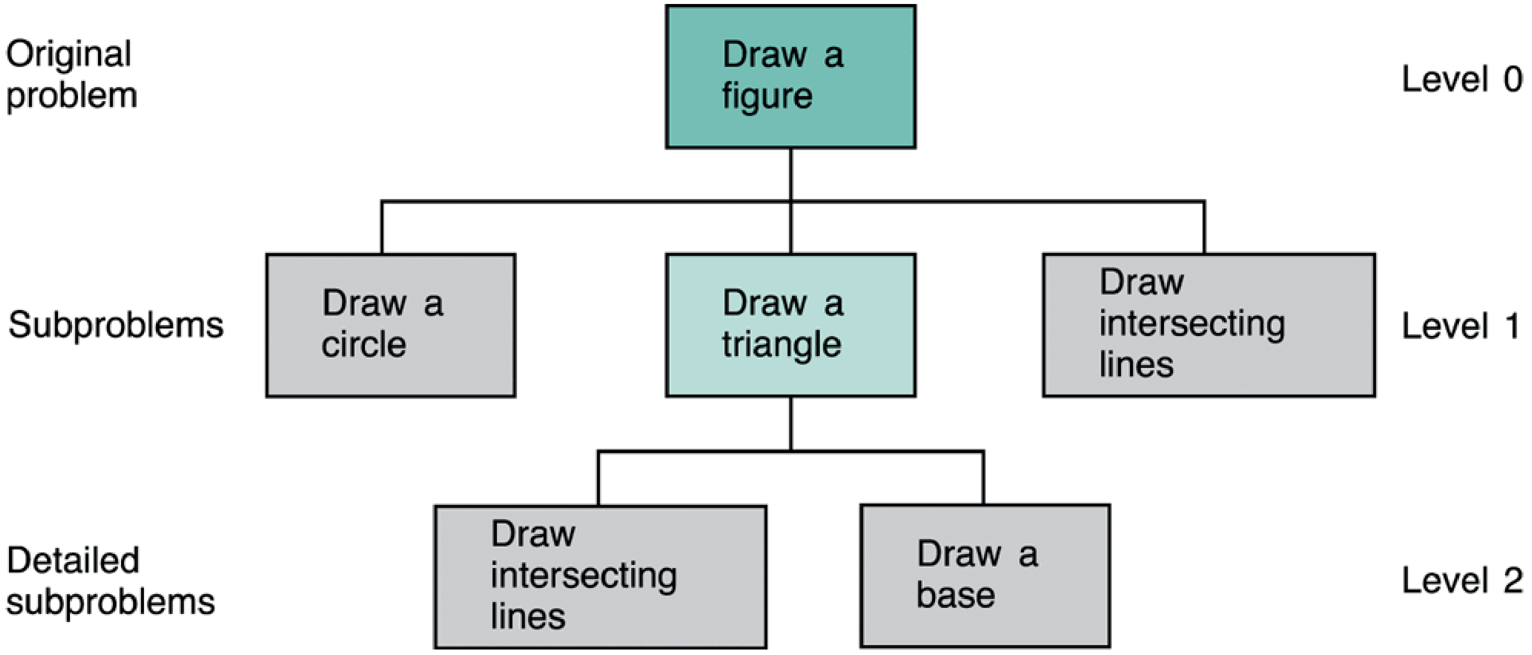


Case Study:
Drawing Simple Diagrams(2/2)

Step 3: Design (Figure 3.10)

- Initial algorithm
  - 1. Draw a circle.
  - 2. Draw a triangle.
  - 3. Draw intersecting lines.
- Refinements
  - 2.1 Draw intersecting lines.
  - 2.2 Draw a base.

Figure 3.10 Structure Chart for Drawing a Stick Figure



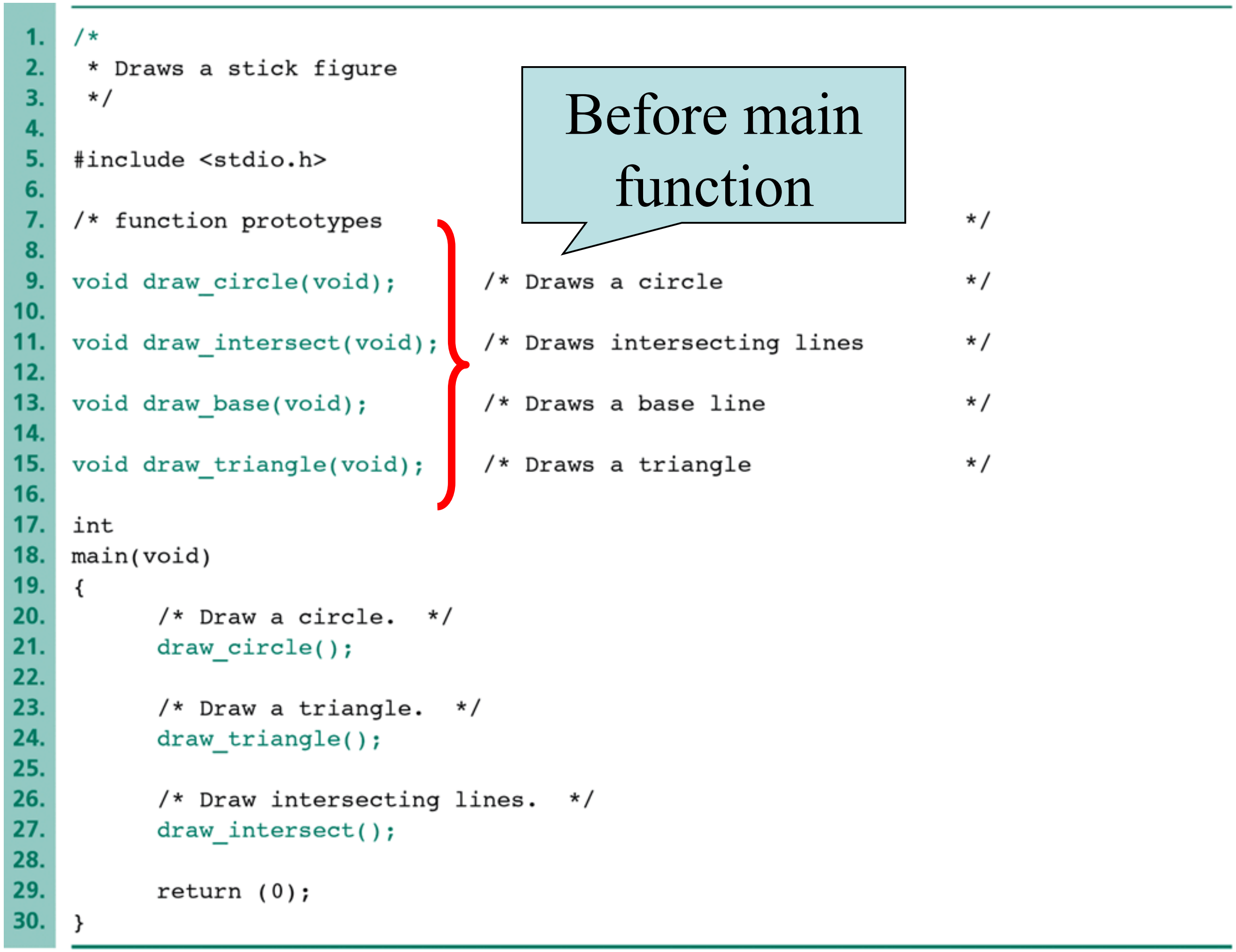
3.4 Function Without Arguments(1/2)

- Function call statement (without arguments)
  - Syntax
    - fname();
  - Example
    - draw_circle();
- Function prototype (without arguments)
  - Form
    - ftype fname(void);
  - Example
    - void draw_circle(void);

Function Without Arguments(2/2)

- ftype：void
  - function does not return a value
- argument list： (void)
  - function has no arguments
- Function prototype must appear before the first call to the function. (Figure 3.11)

Figure 3.11 Function Prototypes and Main Function for Stick Figure



```
/*
 * Draws a stick figure
 */

#include <stdio.h>

/* function prototypes                                     */

void draw_circle(void);     /* Draws a circle              */

void draw_intersect(void);  /* Draws intersecting lines    */

void draw_base(void);       /* Draws a base line           */

void draw_triangle(void);   /* Draws a triangle            */

int
main(void)
{
      /* Draw a circle.  */
      draw_circle();

      /* Draw a triangle.  */
      draw_triangle();

      /* Draw intersecting lines.  */
      draw_intersect();

      return (0);
}
```

Before main function

Function Definitions

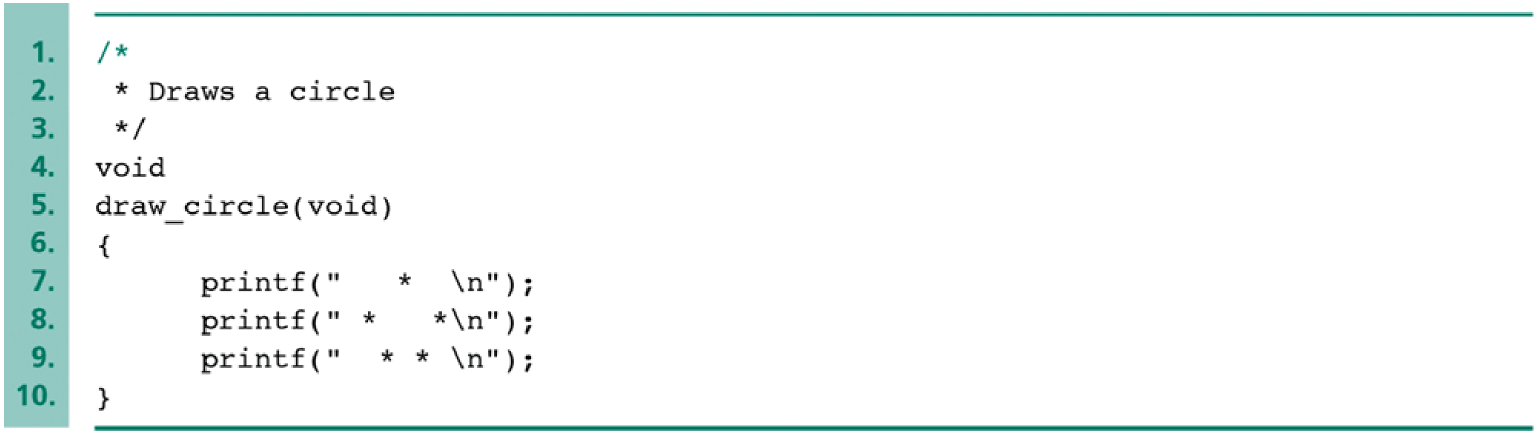
- Syntax :

```
ftype
fname(void)
{
    local declarations
    executable
statements
}
```

Example

```
/*
 *  Display a block-letter H
 */
void
printf_h(void)
{
   printf(“** **\n”)
   printf(“** **\n”)
   printf(“*****\n”)
   printf(“**  **\n”)
   printf(“**  **\n”)
}
```

Figure 3.12 Function draw_circle

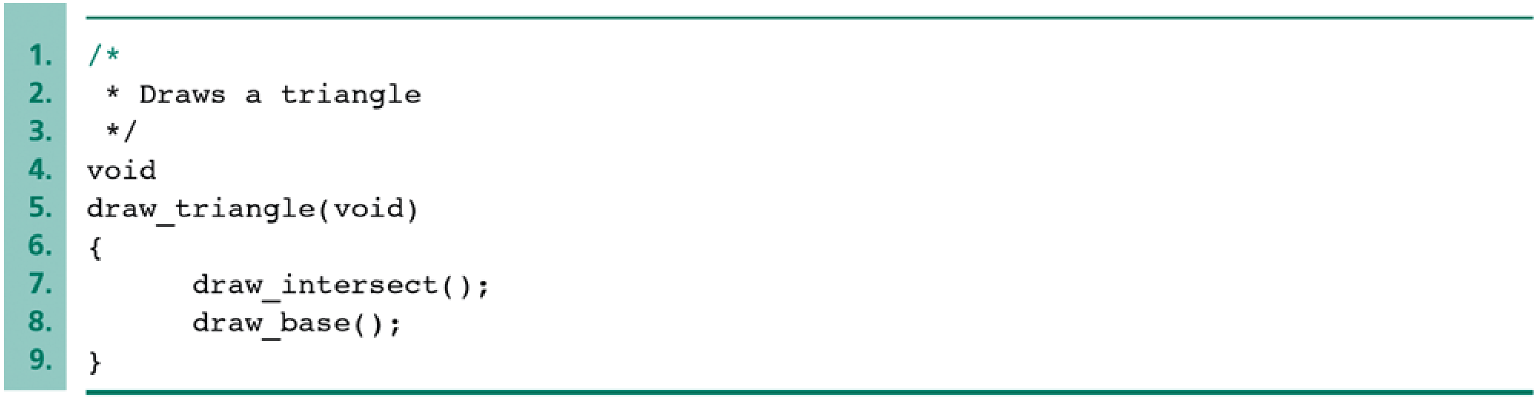


```
/*
 * Draws a circle
 */
void
draw_circle(void)
{
      printf("   *  \n");
      printf(" *   *\n");
      printf("  * * \n");
}
```

Example

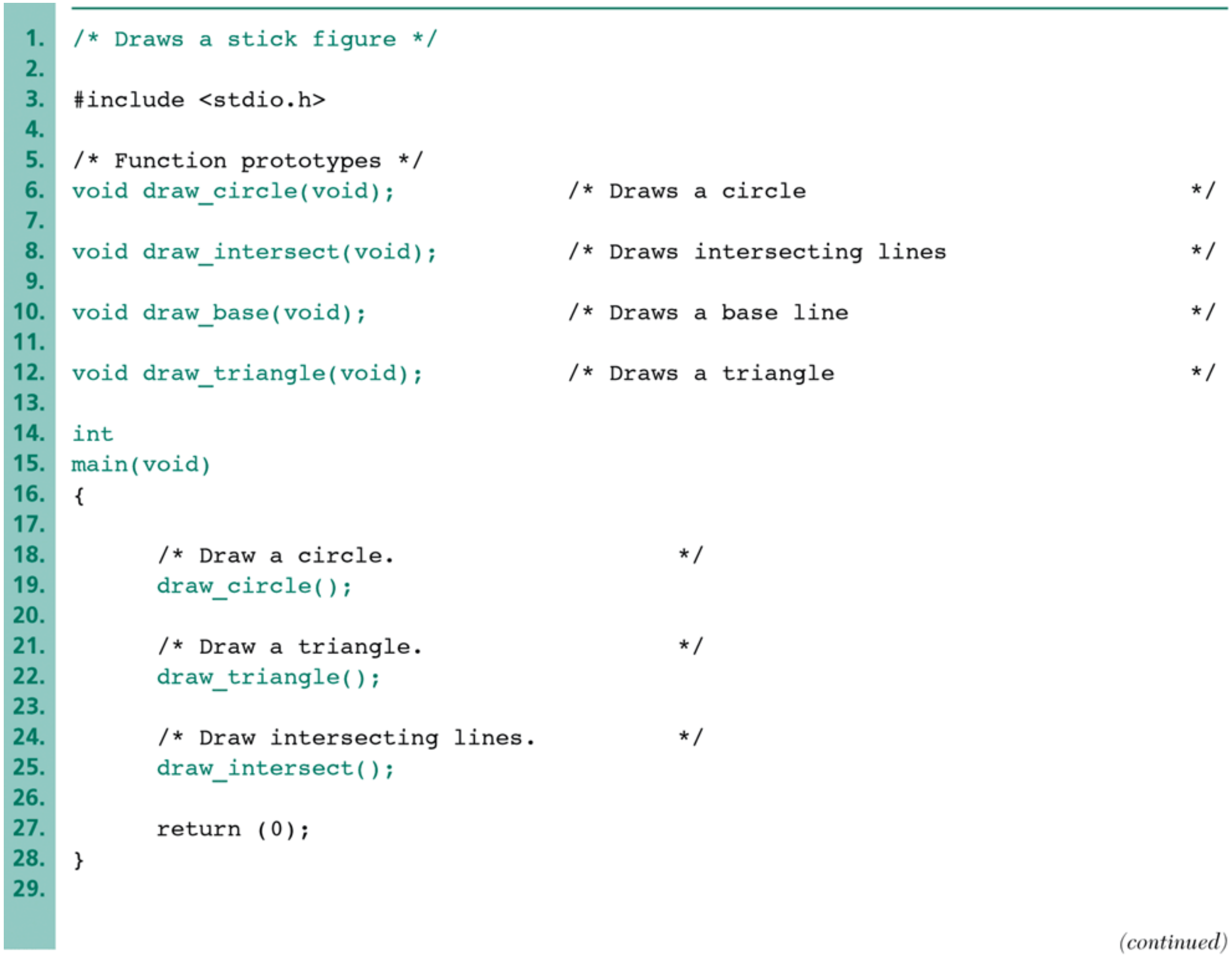
- Figure 3.13 shows how to use top-down design to code function.
  - the body function calls other functions instead of using printf statements
- Figure 3.14 shows the complete program with function subprograms.

Figure 3.13 Function draw_triangle



```
/*
 * Draws a triangle
 */
void
draw_triangle(void)
{
      draw_intersect();
      draw_base();
}
```

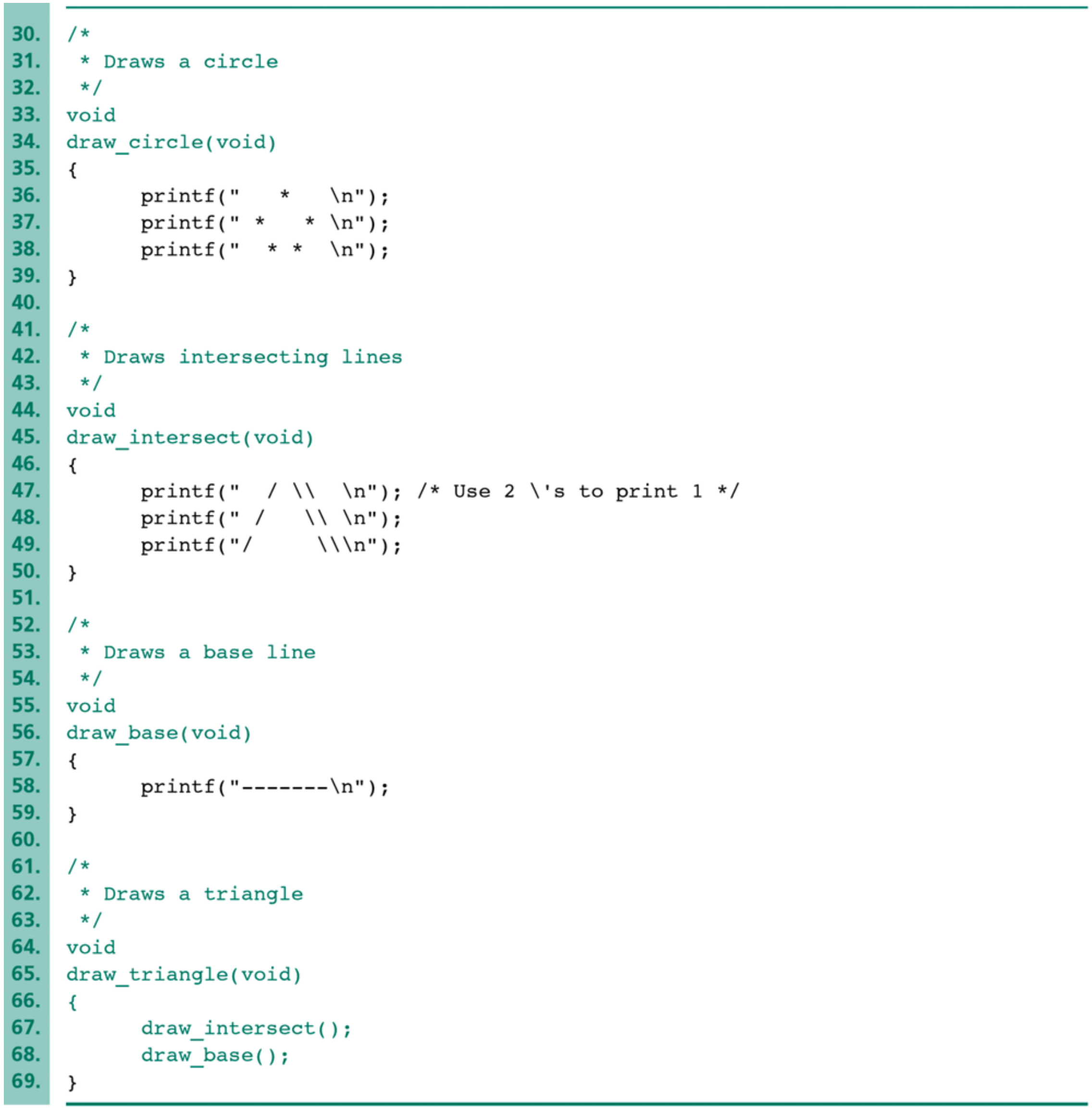
Figure 3.14 Program to Draw a Stick Figure



```
/* Draws a stick figure */

#include <stdio.h>

/* Function prototypes */
void draw_circle(void);            /* Draws a circle                          */

void draw_intersect(void);         /* Draws intersecting lines                */

void draw_base(void);              /* Draws a base line                       */

void draw_triangle(void);          /* Draws a triangle                        */

int
main(void)
{

      /* Draw a circle.                    */
      draw_circle();

      /* Draw a triangle.                  */
      draw_triangle();

      /* Draw intersecting lines.          */
      draw_intersect();

      return (0);
}

```

(continued)

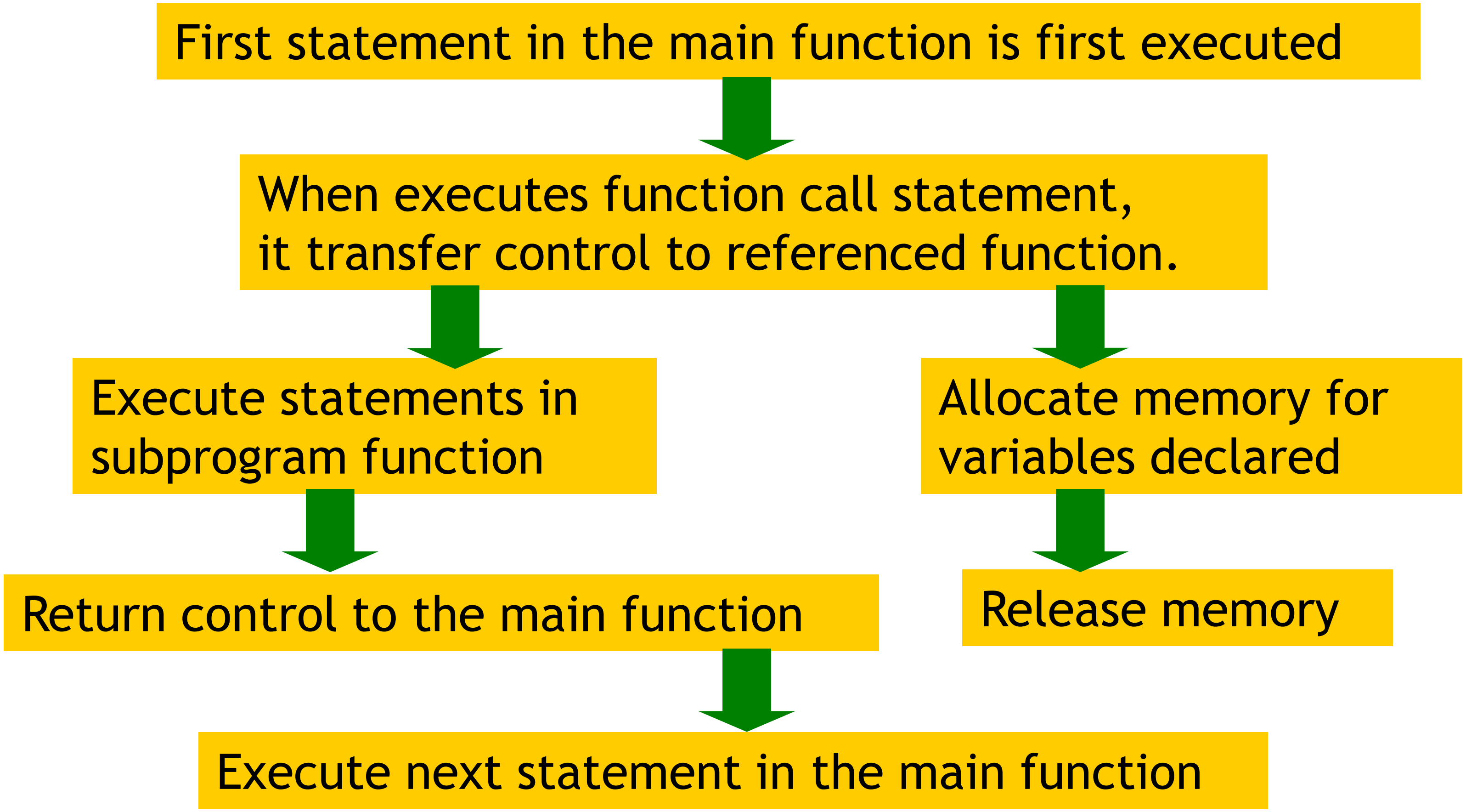
Figure 3.14 Program to Draw a Stick Figure (cont’d)



```
/*
 * Draws a circle
 */
void
draw_circle(void)
{
      printf("   *   \n");
      printf(" *   * \n");
      printf("  * *  \n");
}

/*
 * Draws intersecting lines
 */
void
draw_intersect(void)
{
      printf("  / \\  \n"); /* Use 2 \'s to print 1 */
      printf(" /   \\ \n");
      printf("/     \\\n");
}

/*
 * Draws a base line
 */
void
draw_base(void)
{
      printf("-------\n");
}

/*
 * Draws a triangle
 */
void
draw_triangle(void)
{
      draw_intersect();
      draw_base();
}
```

Order of Execution of Function Subprograms and Main Function

- Running a program steps (Figure 3.15)



First statement in the main function is first executed

When executes function call statement,
it transfer control to referenced function.

Execute statements in subprogram function

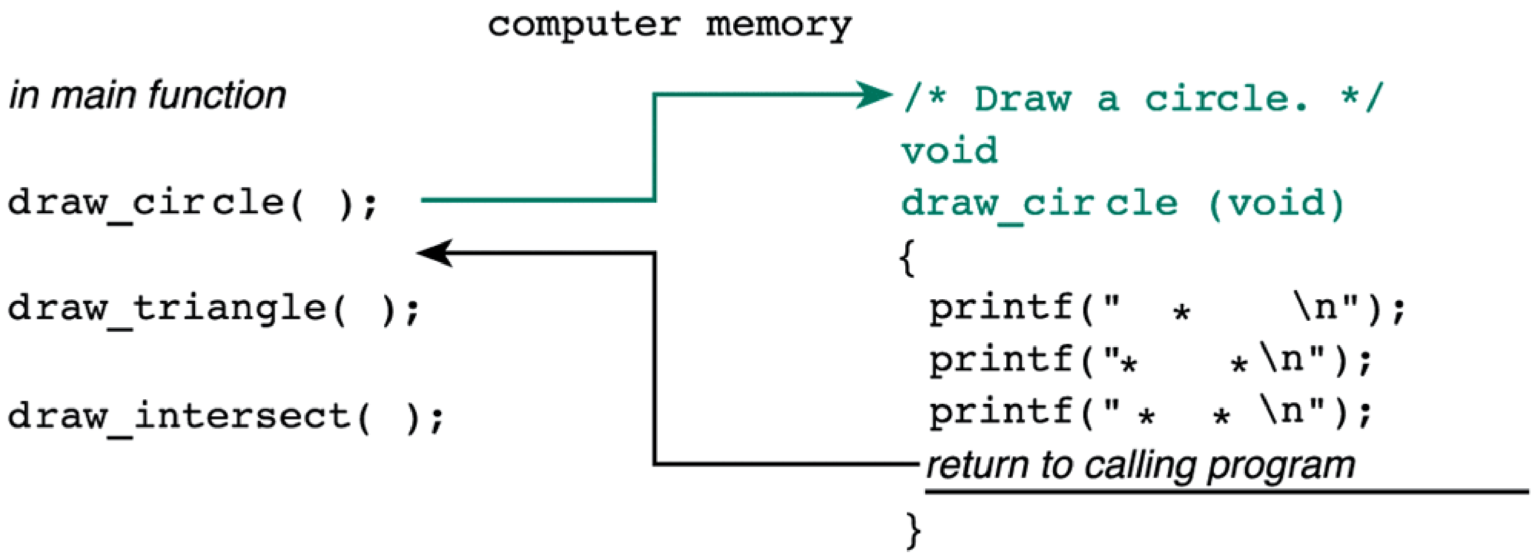
Allocate memory for variables declared

Return control to the main function

Release memory

Execute next statement in the main function

Figure 3.15 Flow of Control Between the main Function and a Function Subprogram



computer memory

in main function

```
draw_circle( );

draw_triangle( );

draw_intersect( );
```

```
/* Draw a circle. */
void
draw_circle (void)
{
 printf("   *    \n");
 printf("*     *\n");
 printf(" *   * \n");
```

return to calling program

```
}
```

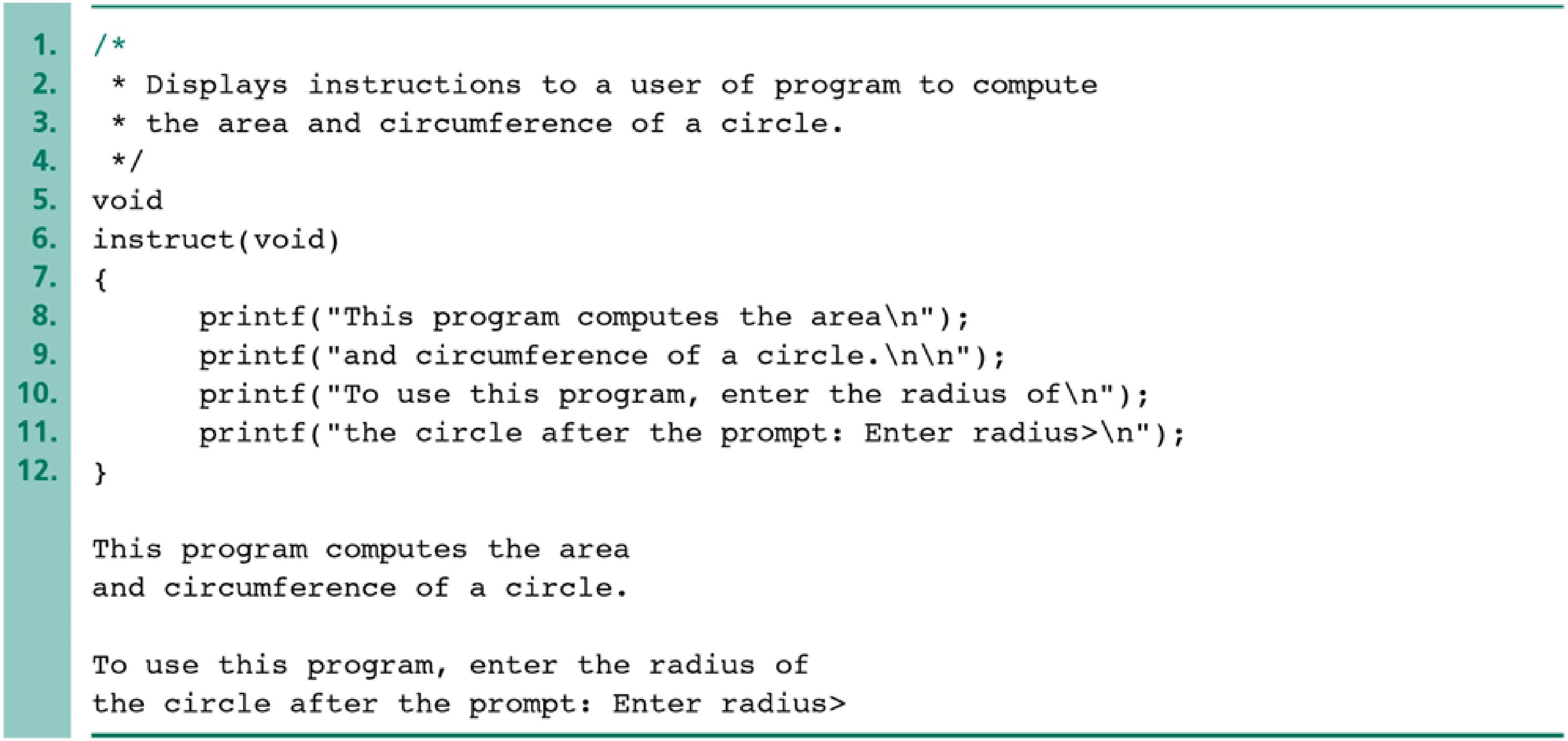
Advantages of Using Function Subprograms

- Procedure abstraction
  - a programming technique in which a main function consists of a sequence of function calls and each function is implemented separately
- Reuse of function subprograms
  - functions can be executed more than once

Display User Instructions

- We can use functions only to display multiple lines of program output.
- (Figure 3.16) Displays instructions to a user.

Figure 3.16 Function instruct and the Output Produced by a Call



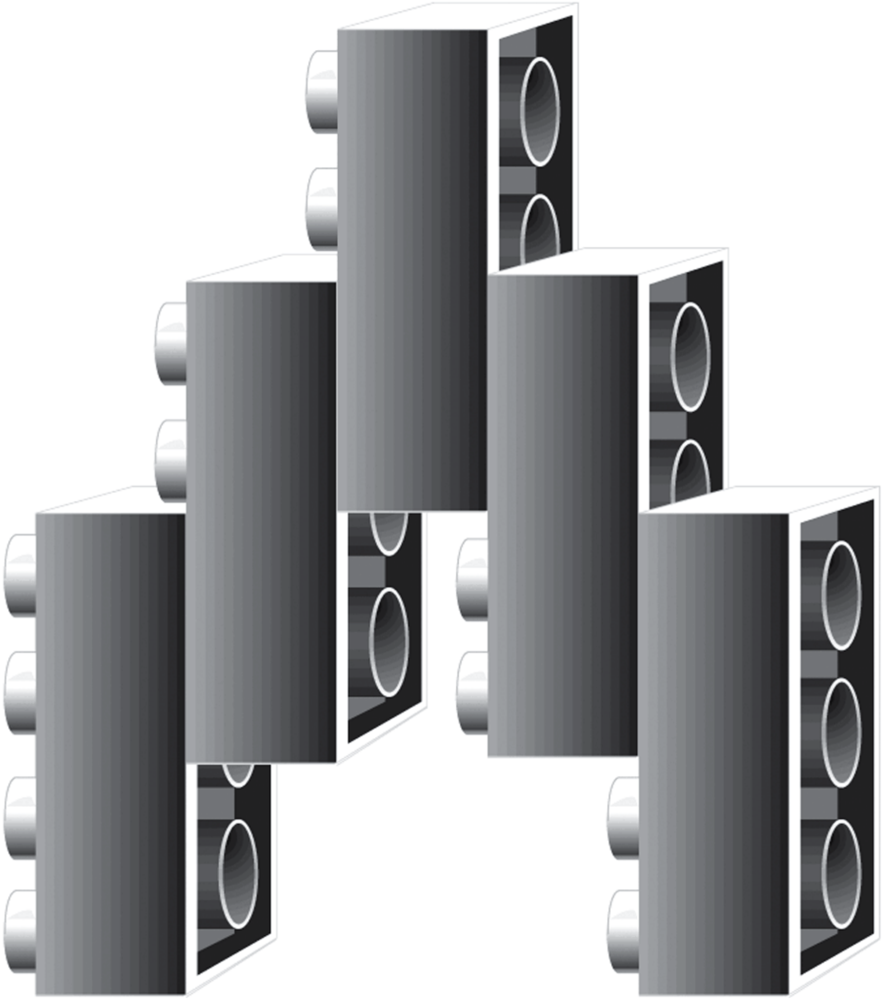
```
/*
 * Displays instructions to a user of program to compute
 * the area and circumference of a circle.
 */
void
instruct(void)
{
      printf("This program computes the area\n");
      printf("and circumference of a circle.\n\n");
      printf("To use this program, enter the radius of\n");
      printf("the circle after the prompt: Enter radius>\n");
}
```

```
This program computes the area
and circumference of a circle.

To use this program, enter the radius of
the circle after the prompt: Enter radius>
```

3.5 Functions with Input Arguments

- To be able to construct more interesting programs, we must provide functions with "protrusions" and "cups" so they can easily interconnected. (Figure 3.17)
- Input arguments
  - Arguments used to pass information into a function subprogram
- Output arguments
  - Arguments used to return results to the calling function

Figure 3.17 Lego® Blocks



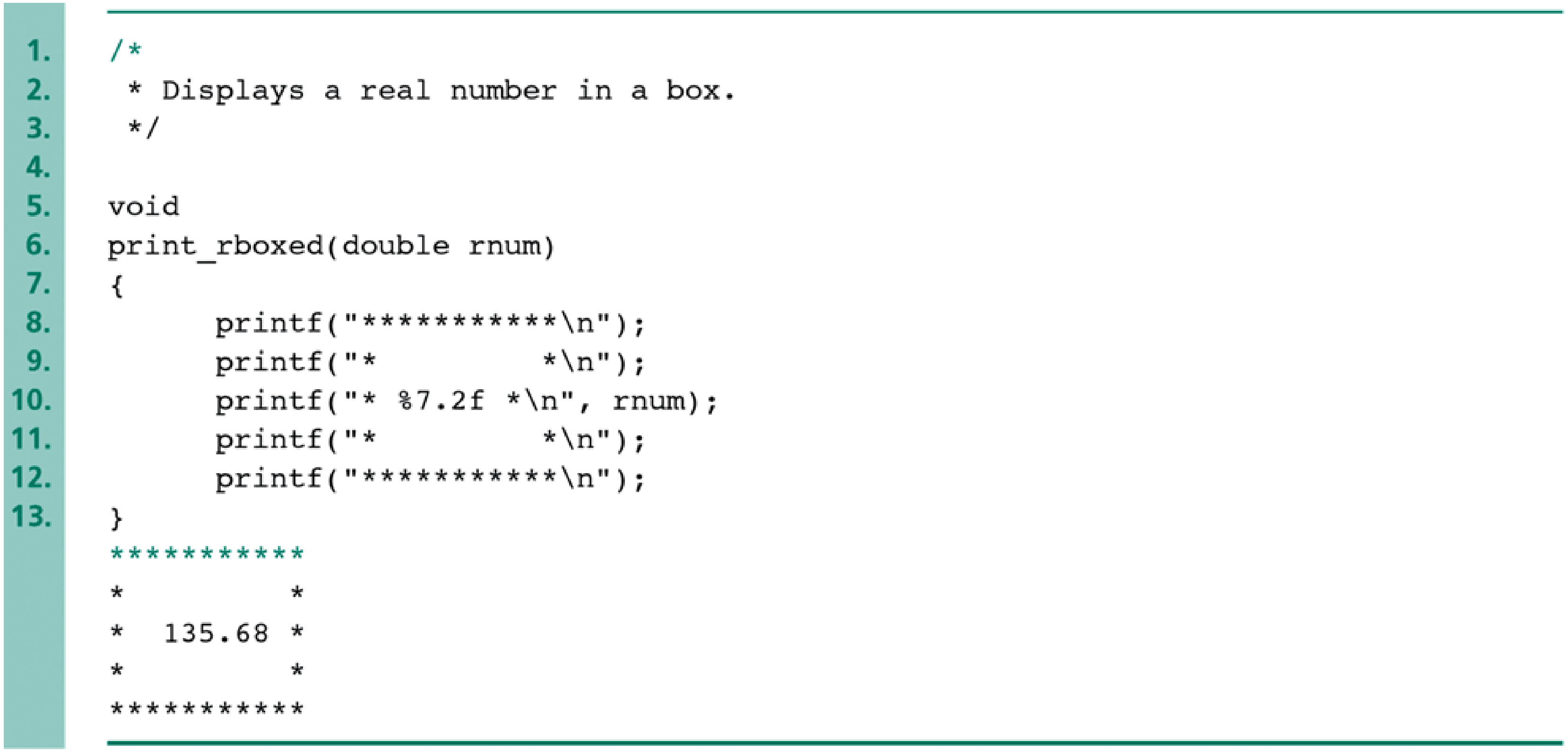
Void Functions with Input Arguments

- void function does not return a result
- Actual argument
  - an expression used inside the parentheses of a function call
- Formal parameter
  - an identifier that represents a corresponding actual argument in a function definition

3.5 (cont) Example 3.5

- Function *print_rboxed* display the value of its argument. (Figure 3.18)
- Actual argument (135.68)
- Formal parameter (rnum)
- (Figure 3.19) Shows the effect of the function call.

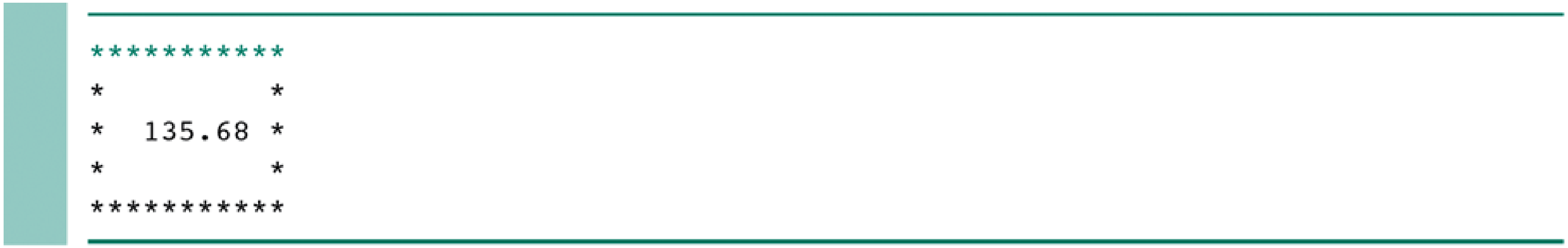
Figure 3.18 Function print_rboxed and Sample Run



```c
/*
 * Displays a real number in a box.
 */

void
print_rboxed(double rnum)
{
      printf("***********\n");
      printf("*         *\n");
      printf("* %7.2f *\n", rnum);
      printf("*         *\n");
      printf("***********\n");
}
```

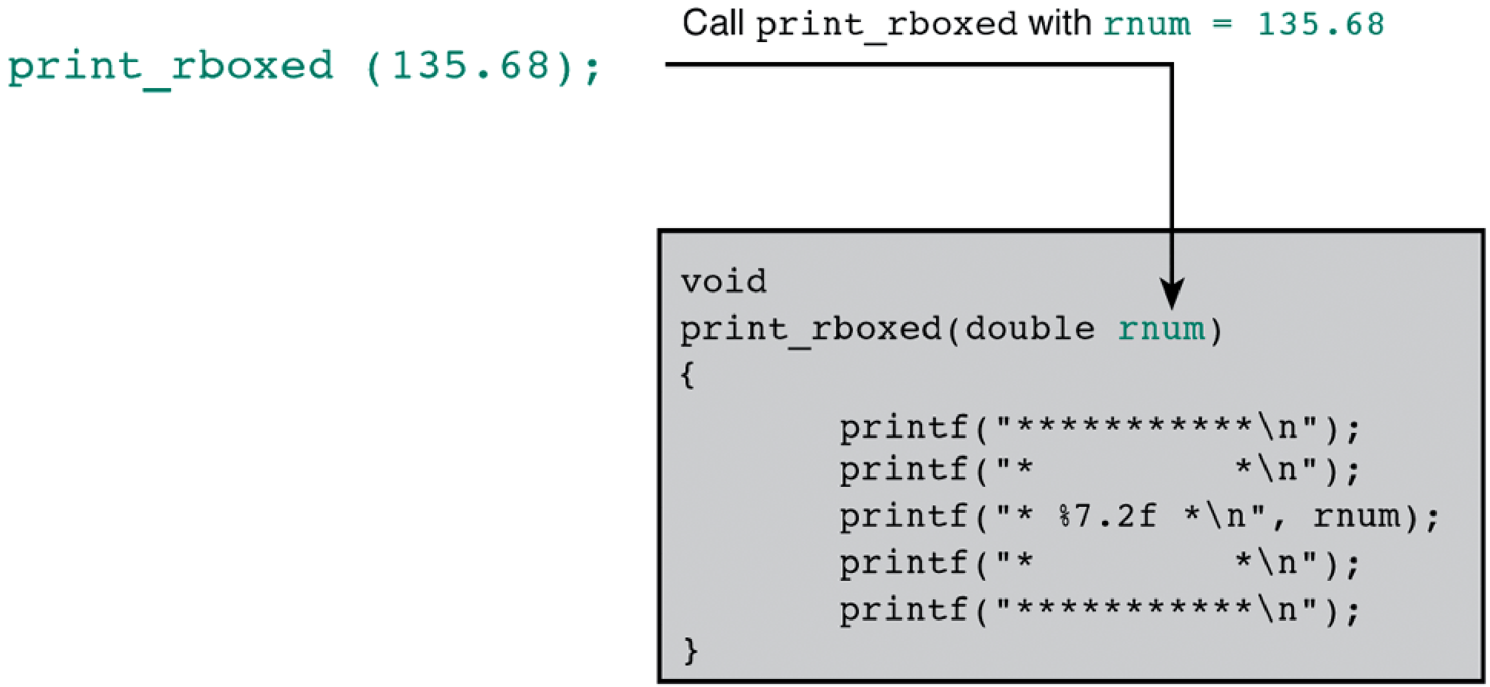
```
***********
*         *
*  135.68 *
*         *
***********
```

Figure 3.18 Function print_rboxed and Sample Run (cont'd)



```
***********
*         *
*  135.68 *
*         *
***********
```

Figure 3.19 Effect of Executing print_rboxed (135.68);



```
print_rboxed (135.68);
```

Call `print_rboxed` with `rnum = 135.68`

```
void
print_rboxed(double rnum)
{
        printf("***********\n");
        printf("*         *\n");
        printf("* %7.2f *\n", rnum);
        printf("*         *\n");
        printf("***********\n");
}
```

Function with Input Argument and a Single Result

- (Figure 3.20) & (Figure 3.21) & (Figure 3.22)
  - shows how to write functions with input arguments that return a single results.

Figure 3.20 Function with Input Arguments and One Result

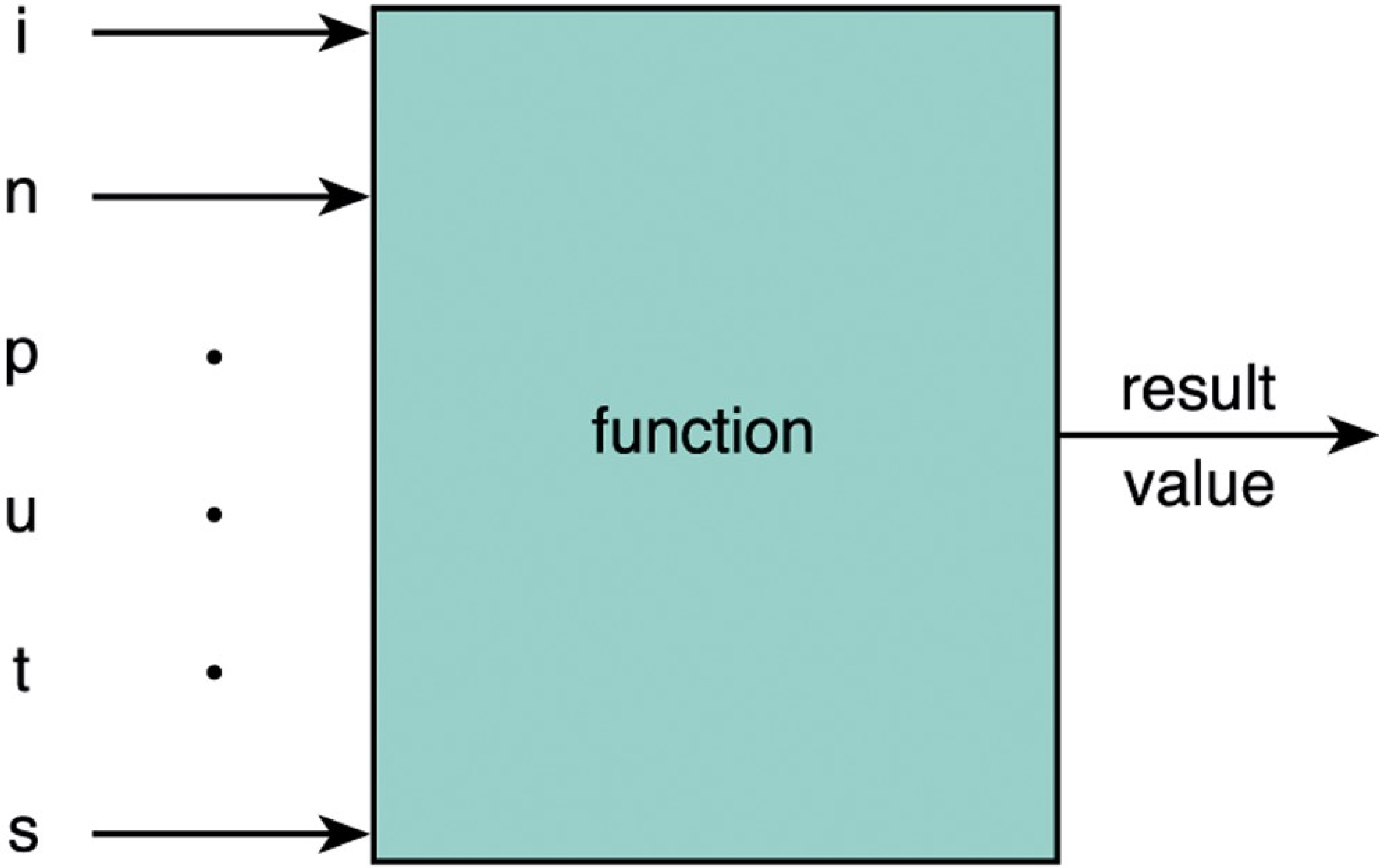
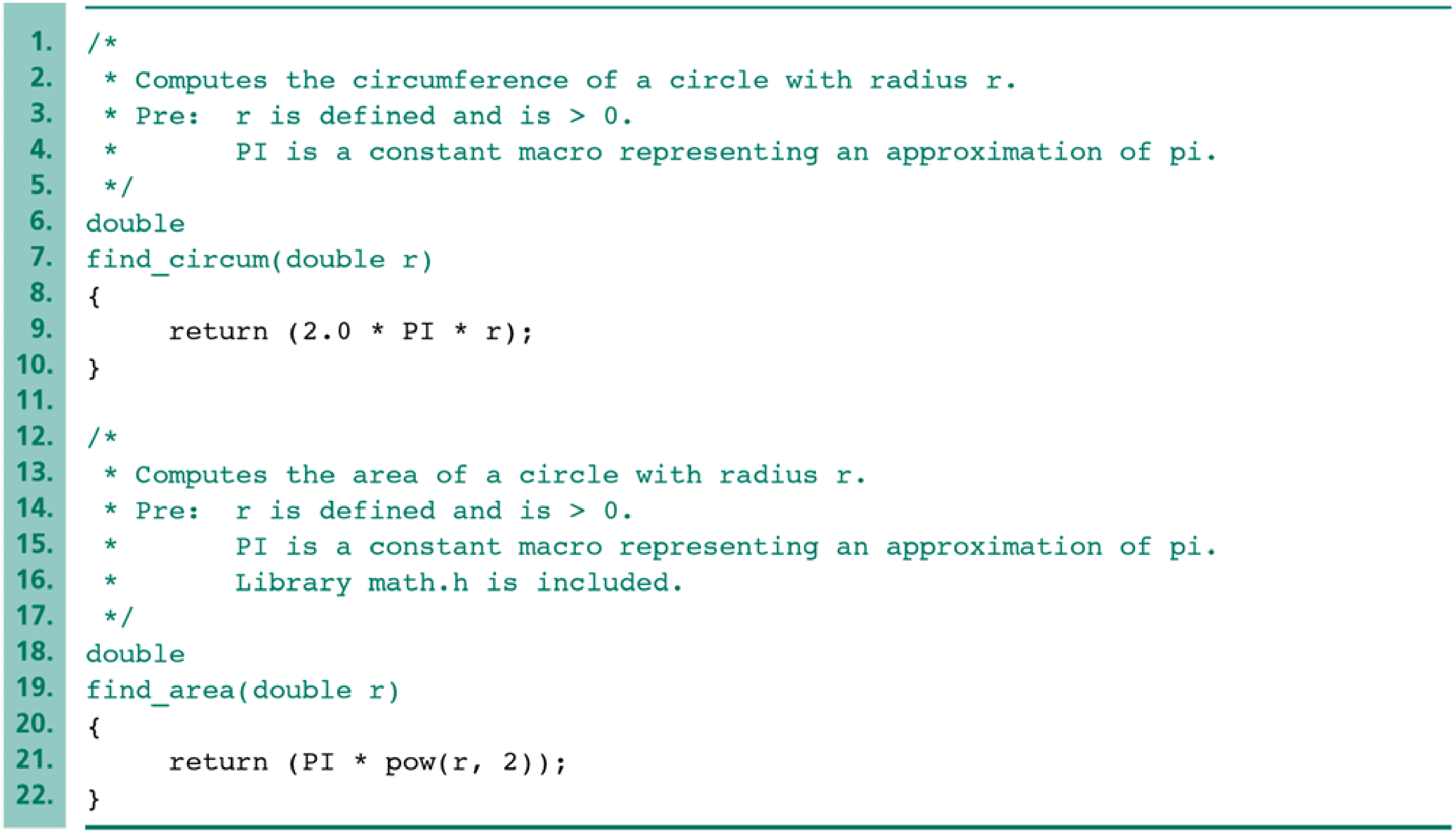
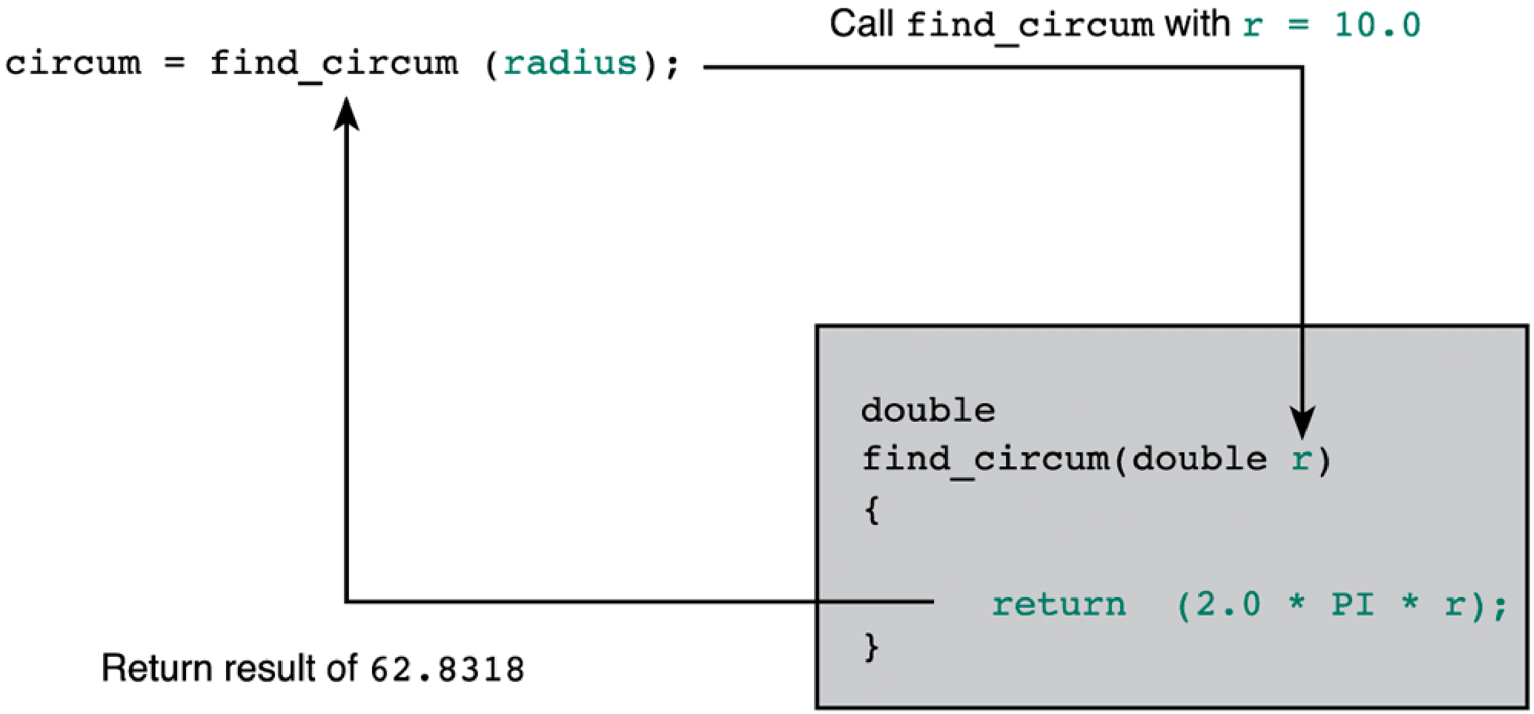


Figure 3.21 Functions find_circum and find_area



```c
/*
 * Computes the circumference of a circle with radius r.
 * Pre:  r is defined and is > 0.
 *       PI is a constant macro representing an approximation of pi.
 */
double
find_circum(double r)
{
     return (2.0 * PI * r);
}

/*
 * Computes the area of a circle with radius r.
 * Pre:  r is defined and is > 0.
 *       PI is a constant macro representing an approximation of pi.
 *       Library math.h is included.
 */
double
find_area(double r)
{
     return (PI * pow(r, 2));
}
```

Figure 3.22 Effect of Executing circum = find_circum (radius);



```
circum = find_circum (radius);
```

Call find_circum with r = 10.0

```
double
find_circum(double r)
{

     return  (2.0 * PI * r);
}
```

Return result of 62.8318

Function Definition
(Input Argument and Single Result)

- Syntax：

```
function interface comment
ftype
fname (formal parameter declaration list)
{
        local variable declarations
        executable statements
}
```

Example

- Example ：

```
/*
 * Finds the cube of its argument.
 * Pre: n is defined.
 */
        int
        cube(int n)
        {
                return(n * n * n) ;
        }
```

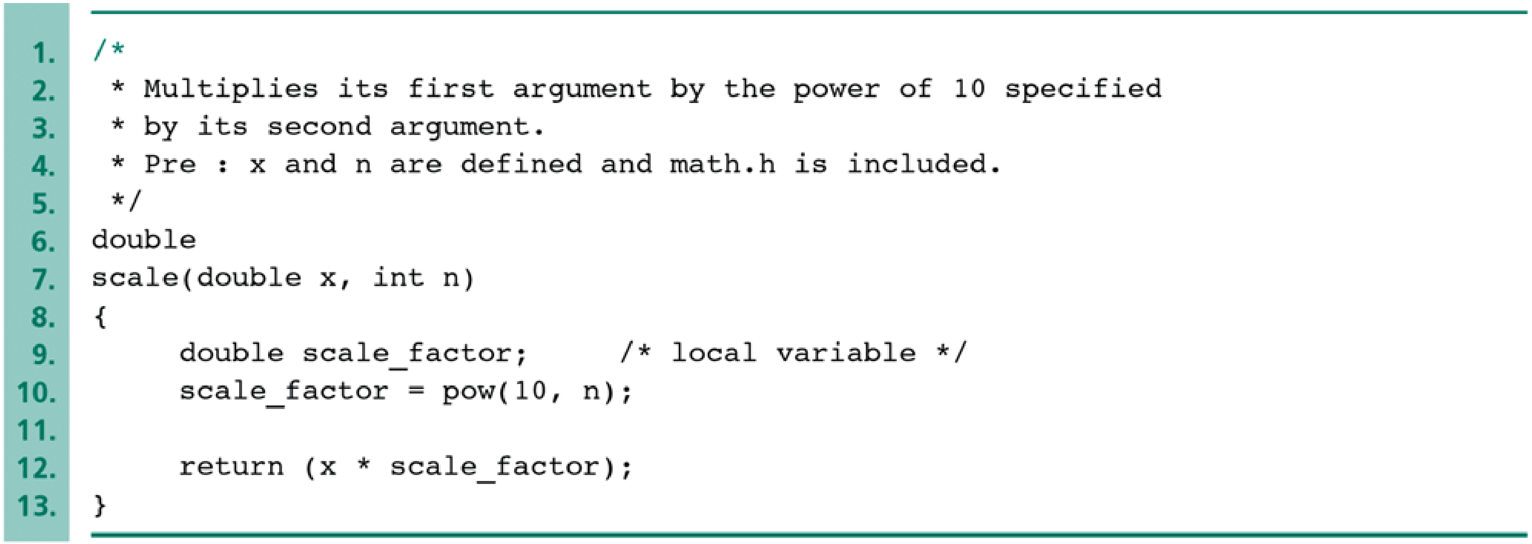
Function Interface Comment

- Precondition
  - a condition assumed to be true before a function call
- Postcondition
  - A condition assumed to be true after a function executes

Functions with Multiple Arguments

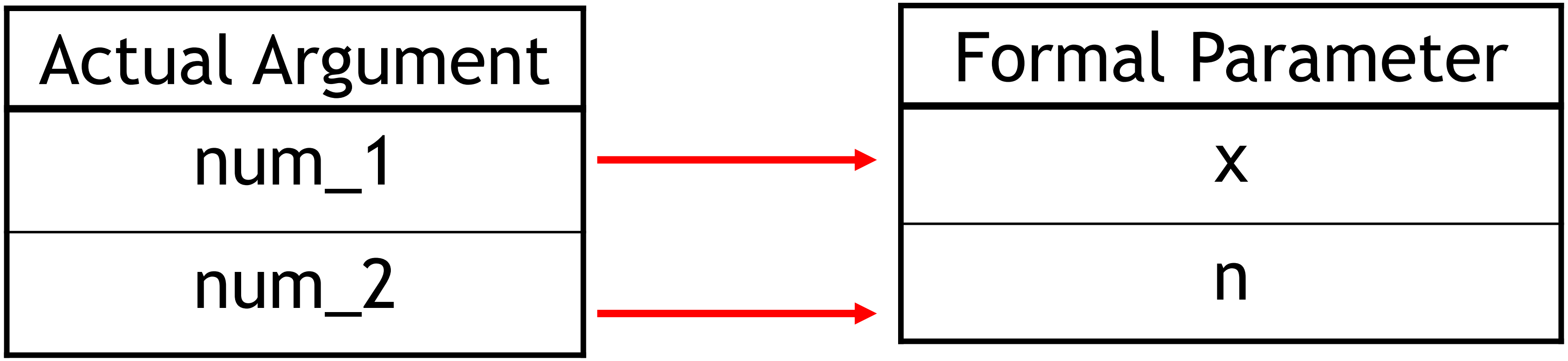
- Example 3.6
  - Function scale multiplies its first argument by 10 raised to the power indicated by its second argument. (Figure 3.23)

Figure 3.23 Function scale



```
/*
 * Multiplies its first argument by the power of 10 specified
 * by its second argument.
 * Pre : x and n are defined and math.h is included.
 */
double
scale(double x, int n)
{
     double scale_factor;     /* local variable */
     scale_factor = pow(10, n);

     return (x * scale_factor);
}
```

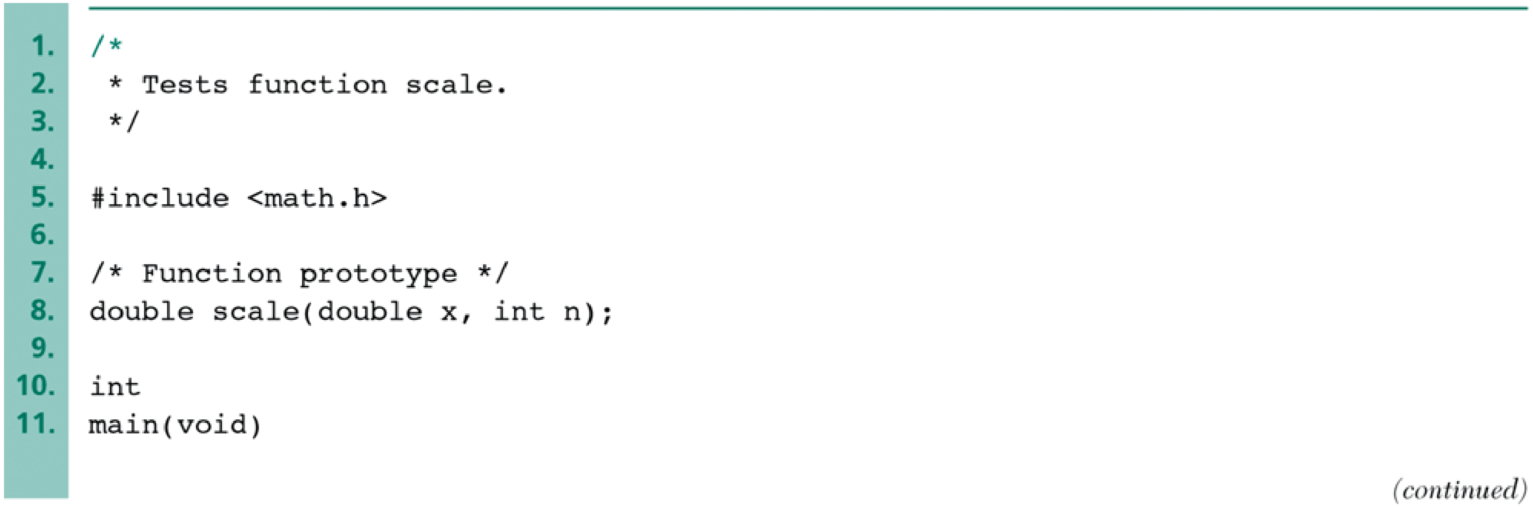
Example: Testing Function Scale



Actual Argument
num_1
num_2

Formal Parameter
x
n

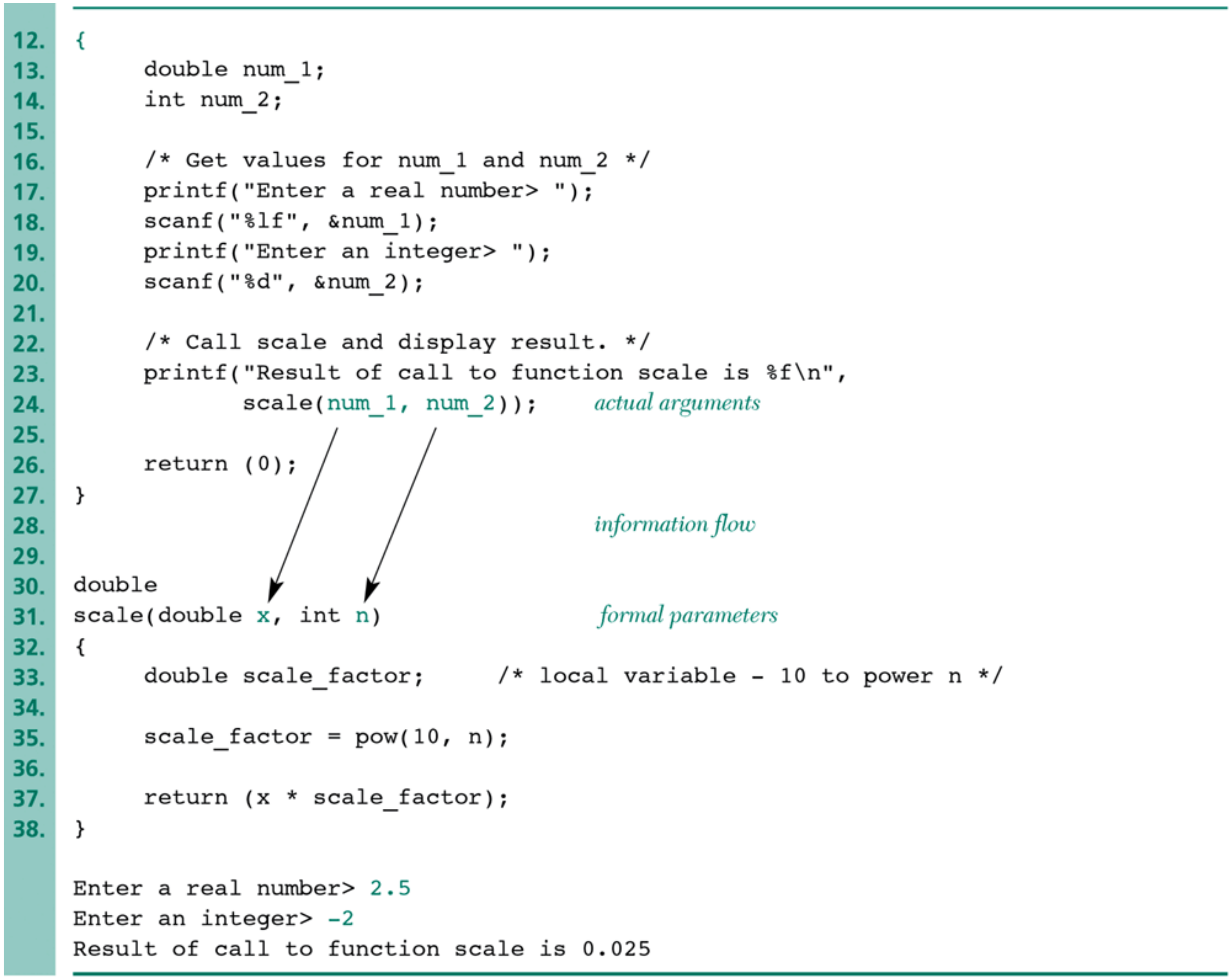
Figure 3.24 Testing Function scale



```
/*
 * Tests function scale.
 */

#include <math.h>

/* Function prototype */
double scale(double x, int n);

int
main(void)
```

(continued)

Figure 3.24 Testing Function scale (cont'd)



```
{
      double num_1;
      int num_2;

      /* Get values for num_1 and num_2 */
      printf("Enter a real number> ");
      scanf("%lf", &num_1);
      printf("Enter an integer> ");
      scanf("%d", &num_2);

      /* Call scale and display result. */
      printf("Result of call to function scale is %f\n",
               scale(num_1, num_2));

      return (0);
}


double
scale(double x, int n)
{
      double scale_factor;      /* local variable - 10 to power n */

      scale_factor = pow(10, n);

      return (x * scale_factor);
}
```

```
Enter a real number> 2.5
Enter an integer> -2
Result of call to function scale is 0.025
```

actual arguments

information flow

formal parameters

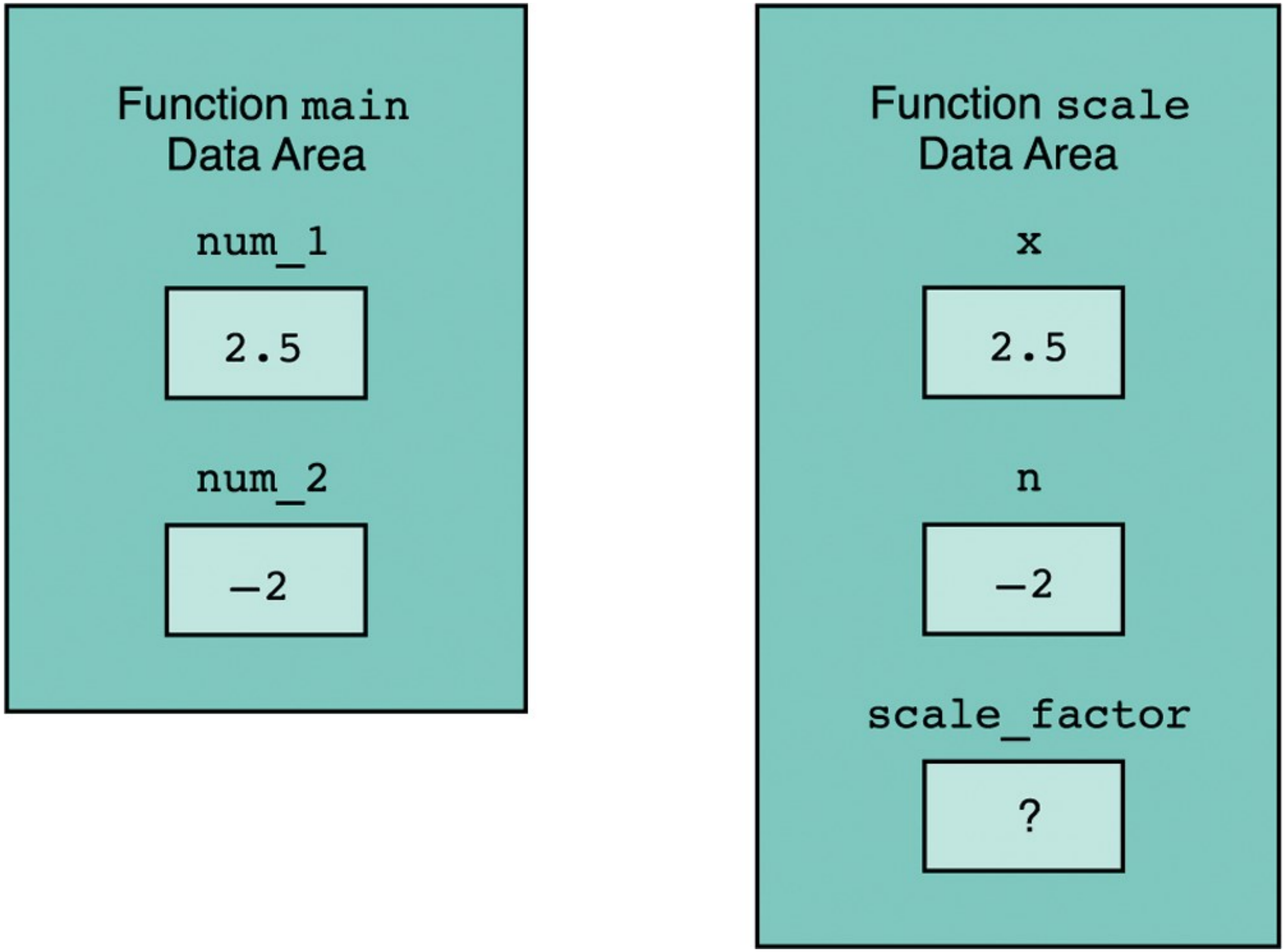
Argument List Correspondence (The not rules)

- The number of actual arguments used in a call to a function must be the same as the number of formal parameters listed in the function prototype.
- The order of arguments in the lists determines correspondence.
- Each actual argument must be of a data type that can be assigned to the corresponding formal parameter.

The Function Data Area

- Function call：
  - an area of memory is allocated for storage of function data
- Function terminates：
  - function data are lost
- Local variable：
  - initially undefined
  - the execution of the function body changes the value of this variable

Figure 3.25 Data Areas After Call scale(num_1, num_2);



Introduction to Computer Graphics

- Text mode: a display mode in which a C program display only characters

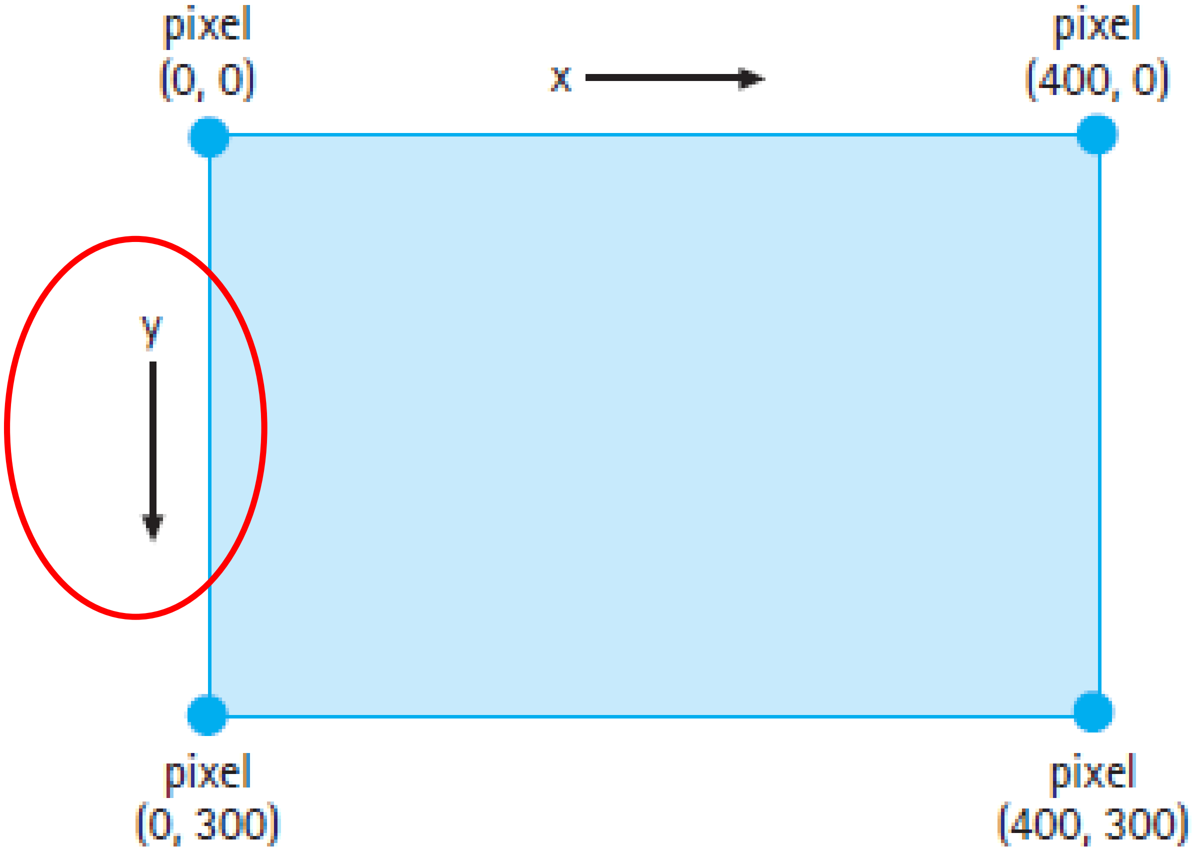
- Graphic mode: a display mode in which a C program draws graphics patterns and shapes in an output window

Composition of a Window

- In graphics programming, you control the location of each line or sharp that you draw in a window.

- You must know your window size and how to reference the individual picture element in a window.
  - Pixel: a picture element on a computer screen

Figure 3.26 Referencing pixels in a window

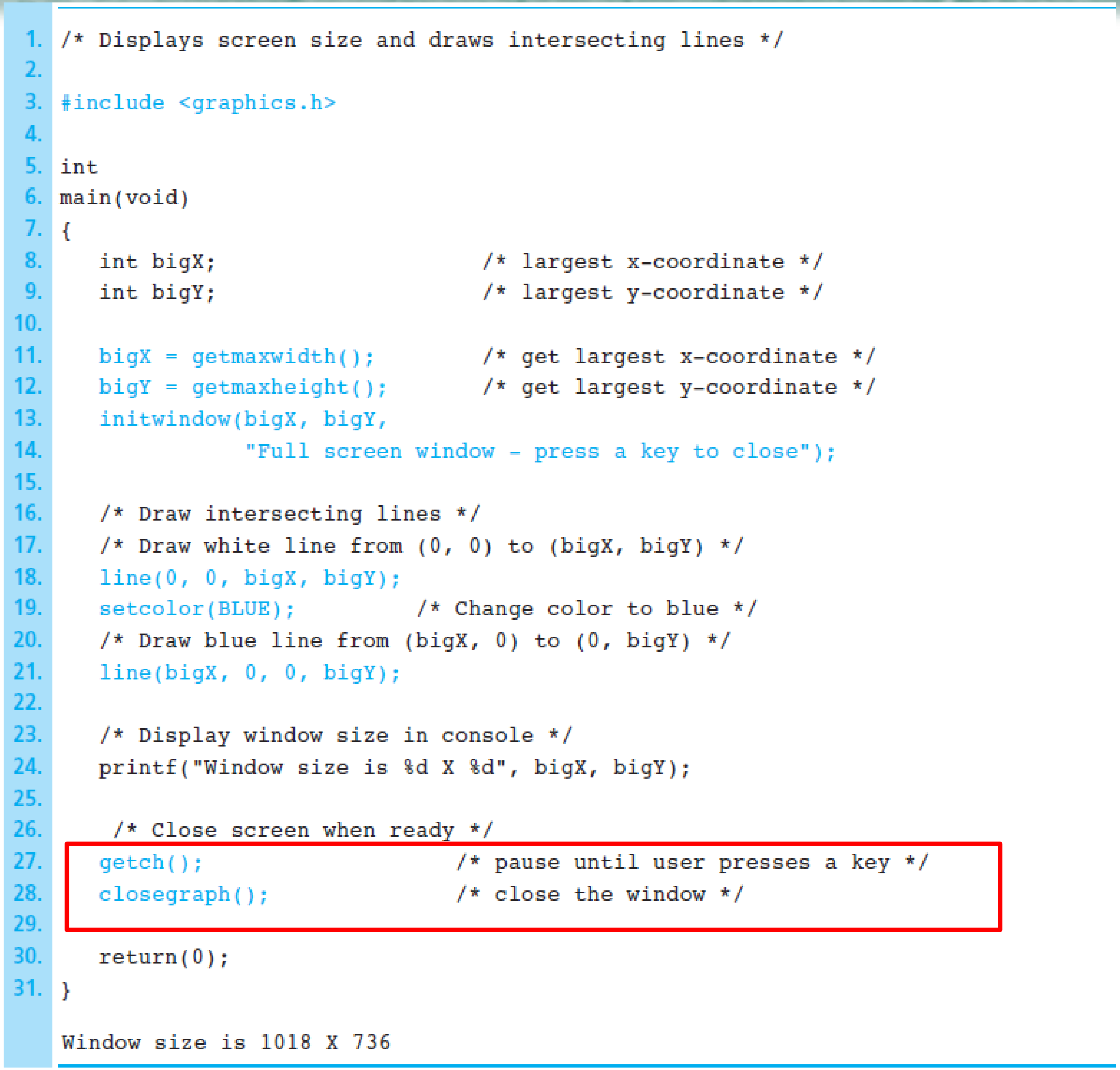


Some Common Graphics Function

- A graphic program is a sequence of statements that call graphic functions to do the work.
- bigx = getmaxwidth()   /* get largest x-coordinate*/
- bigy = getmaxheight() /* get largest y-coordinate*/
- Initwindow(bigx, bigy,

 "Full screen window – press a character to close window"); /* pop up a window with size (bigx,bigy) */

Figure 3.27 Drawing intersecting lines

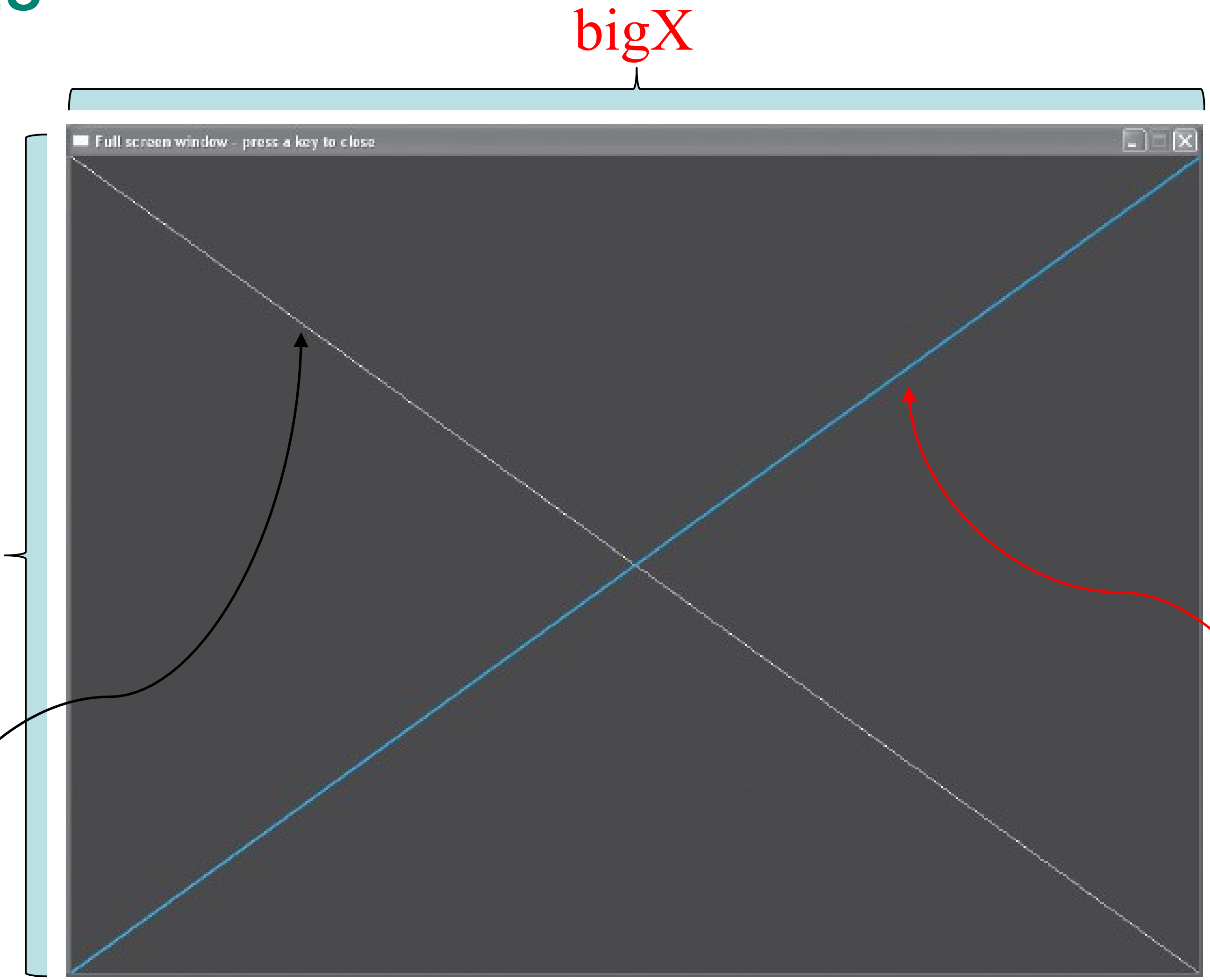


```
/* Displays screen size and draws intersecting lines */

#include <graphics.h>

int
main(void)
{
   int bigX;                    /* largest x-coordinate */
   int bigY;                    /* largest y-coordinate */

   bigX = getmaxwidth();        /* get largest x-coordinate */
   bigY = getmaxheight();       /* get largest y-coordinate */
   initwindow(bigX, bigY,
              "Full screen window - press a key to close");

   /* Draw intersecting lines */
   /* Draw white line from (0, 0) to (bigX, bigY) */
   line(0, 0, bigX, bigY);
   setcolor(BLUE);         /* Change color to blue */
   /* Draw blue line from (bigX, 0) to (0, bigY) */
   line(bigX, 0, 0, bigY);

   /* Display window size in console */
   printf("Window size is %d X %d", bigX, bigY);

    /* Close screen when ready */
   getch();                   /* pause until user presses a key */
   closegraph();              /* close the window */

   return(0);
}
```

```
Window size is 1018 X 736
```

Figure 3.28 Window drawn by lines.c

Window size is 1018 x 736



```
initwindow(bigX, bigY,"Full screen window - press a key to close");
line(0, 0, bigX, bigY); /* Draw white line from (0, 0) to (bigX, bigY) */
setcolor(BLUE);         /* Change color to blue */
line(bigX, 0, 0, bigY);  /* Draw blue line from (bigX, 0) to (0, bigY) */
```

Background Color and Foreground Color

- setbkcolor(GREEN)
  - GREEN is the background color
- setcolor(RED)
  - RED is the foreground color

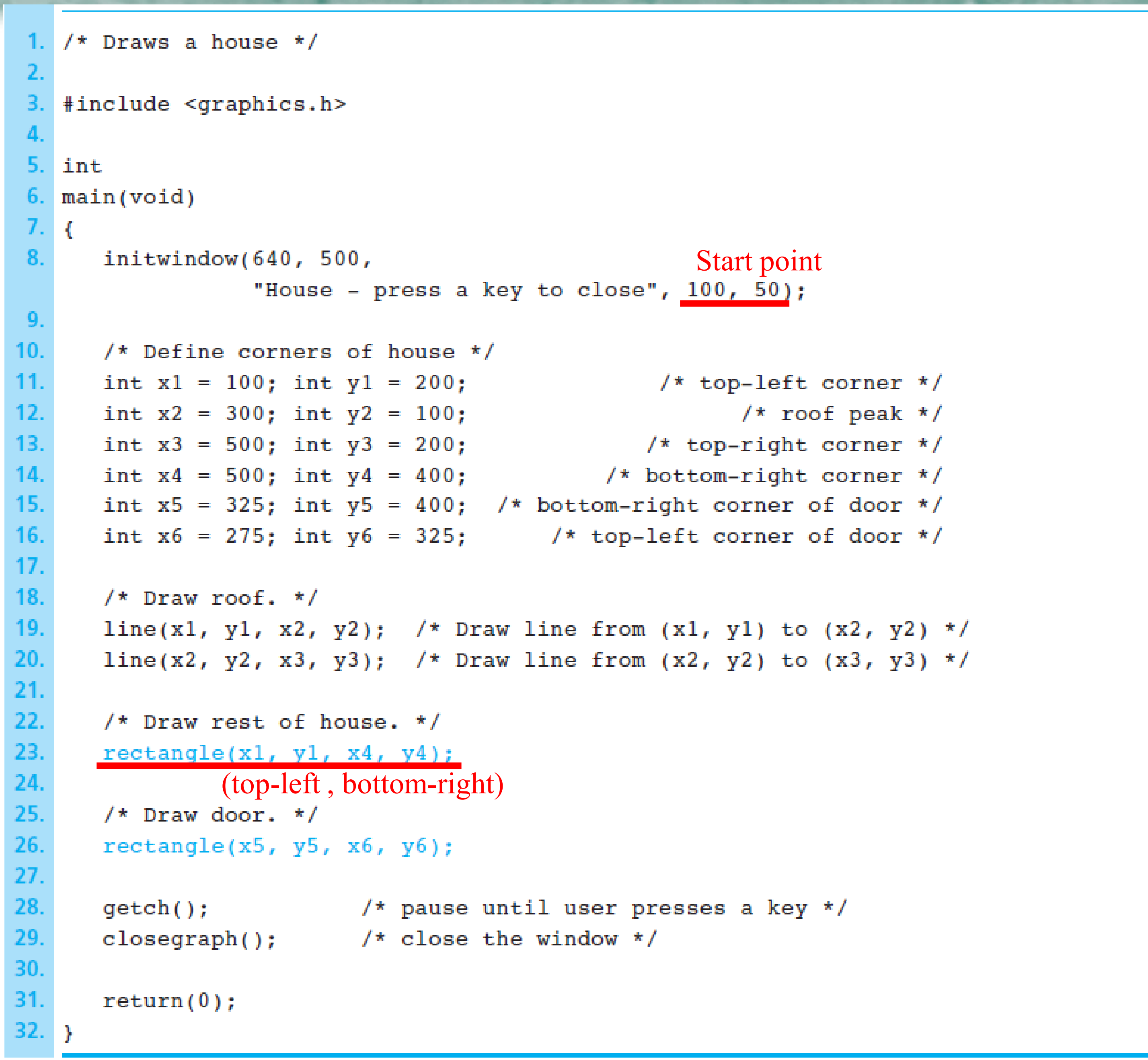
TABLE 3.2 Color Constants

Constant	Value	Constant	Value
BLACK	0	DARKGRAY	8
BLUE	1	LIGHTBLUE	9
GREEN	2	LIGHTGREEN	10
CYAN	3	LIGHTCYAN	11
RED	4	LIGHTRED	12
MAGENTA	5	LIGHTMAGENTA	13
BROWN	6	YELLOW	14
LIGHTGRAY	7	WHITE	15

Drawing Rectangles

- rectangel(x1, y1, x2, y2)
  - Draws a rectangle that has one diagonal with end point (x1,y1) and (x2, y2)

Figure 3.29 Drawing a house

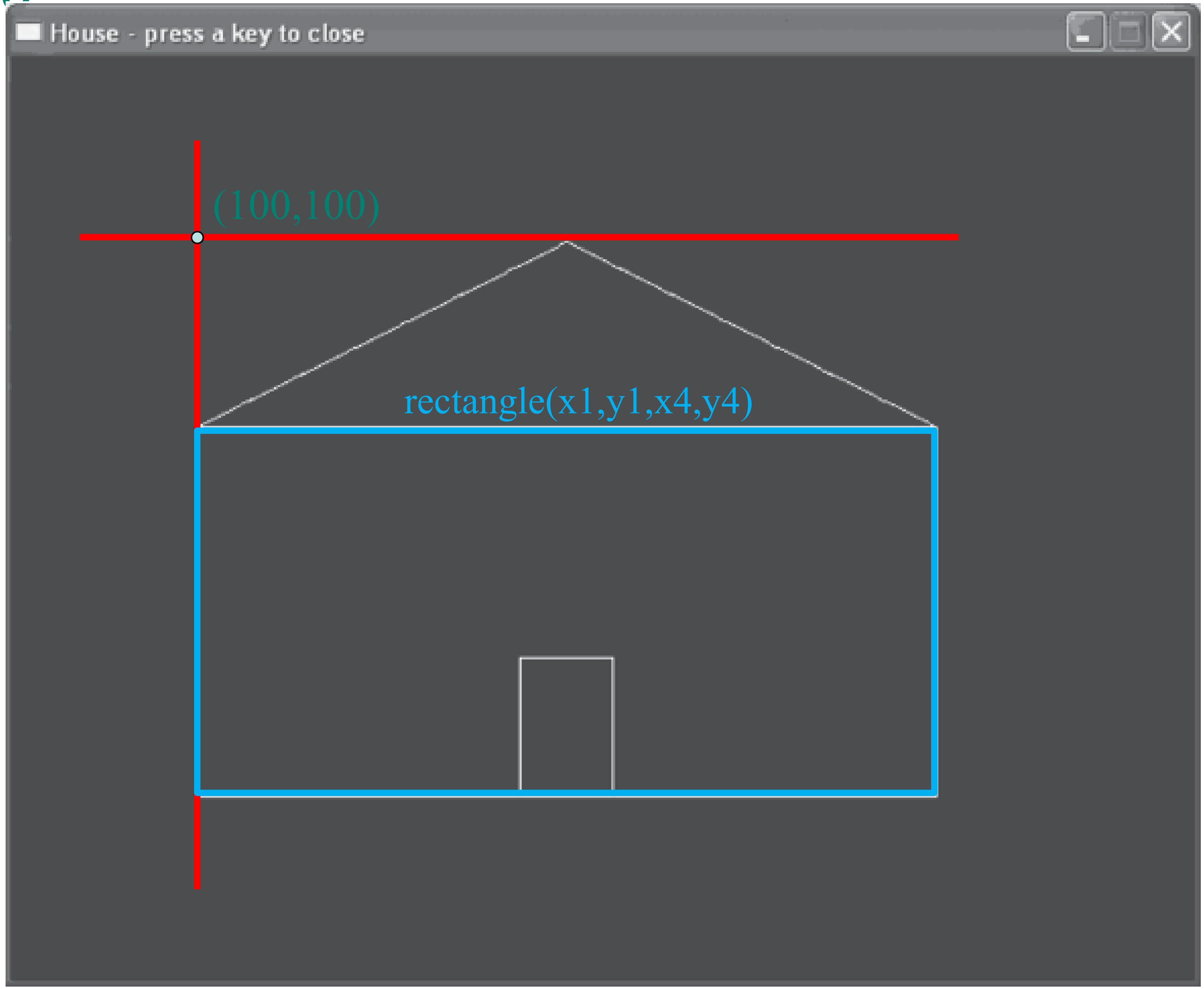


```c
/* Draws a house */

#include <graphics.h>

int
main(void)
{
   initwindow(640, 500,
              "House - press a key to close", 100, 50);

   /* Define corners of house */
   int x1 = 100; int y1 = 200;              /* top-left corner */
   int x2 = 300; int y2 = 100;                    /* roof peak */
   int x3 = 500; int y3 = 200;             /* top-right corner */
   int x4 = 500; int y4 = 400;          /* bottom-right corner */
   int x5 = 325; int y5 = 400;  /* bottom-right corner of door */
   int x6 = 275; int y6 = 325;      /* top-left corner of door */

   /* Draw roof. */
   line(x1, y1, x2, y2);  /* Draw line from (x1, y1) to (x2, y2) */
   line(x2, y2, x3, y3);  /* Draw line from (x2, y2) to (x3, y3) */

   /* Draw rest of house. */
   rectangle(x1, y1, x4, y4);

   /* Draw door. */
   rectangle(x5, y5, x6, y6);

   getch();           /* pause until user presses a key */
   closegraph();      /* close the window */

   return(0);
}
```

Start point

(top-left , bottom-right)

Figure 3.30 House drawn by house.c

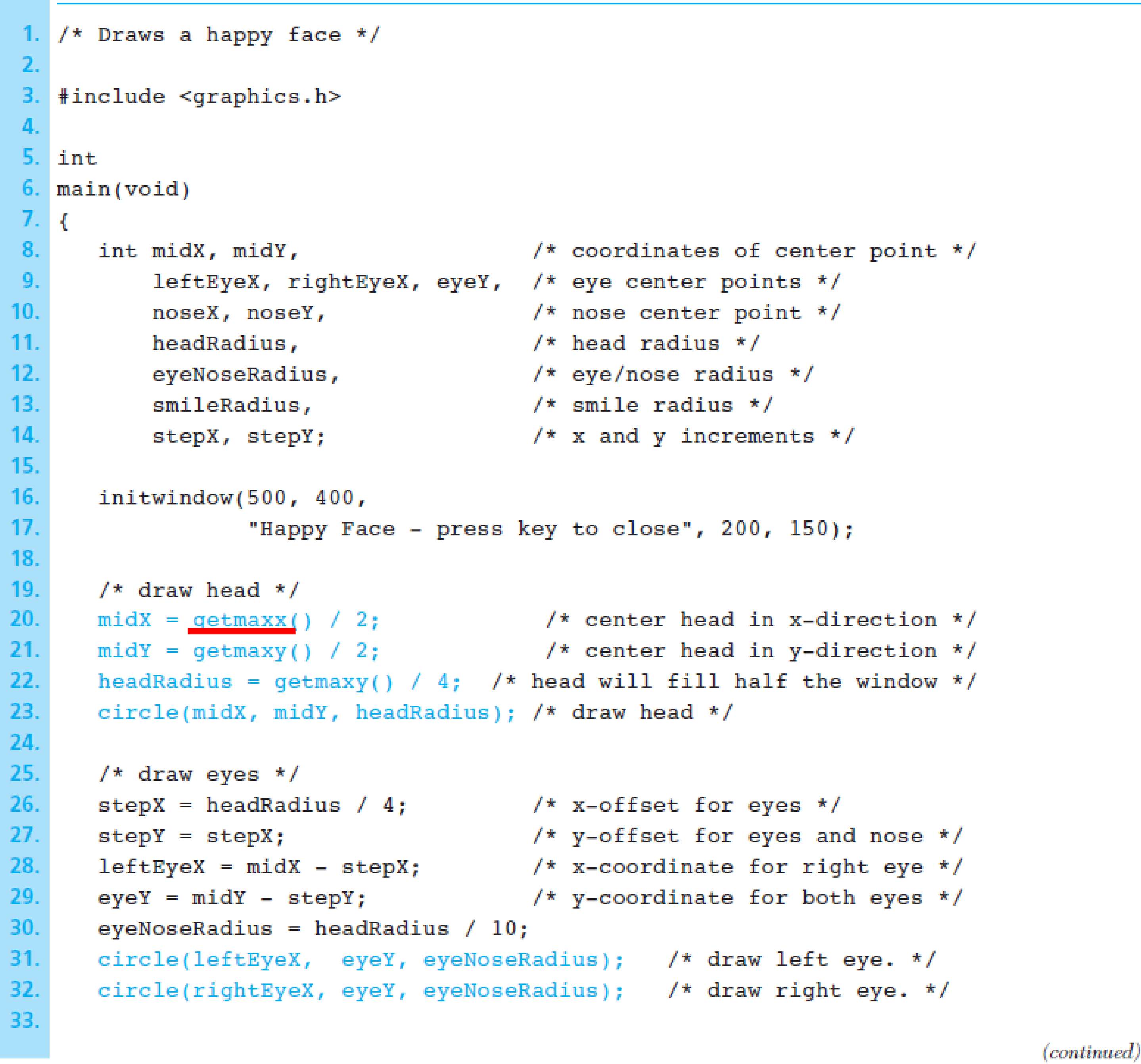


Drawing Circle, Ellipses, and Arcs

- Circle
  - circle(x, y, radius) /* center at (x, y) */
- Arc(x, y, 0, 180, radius)
    - 0 degree- 3 o’clock, 30 degree- 2 o’clock.
- Ellipses
  - ellipse(x, y, 0, 360, radius, 2 * radius)

Figure 3.31 Program to draw a happy face

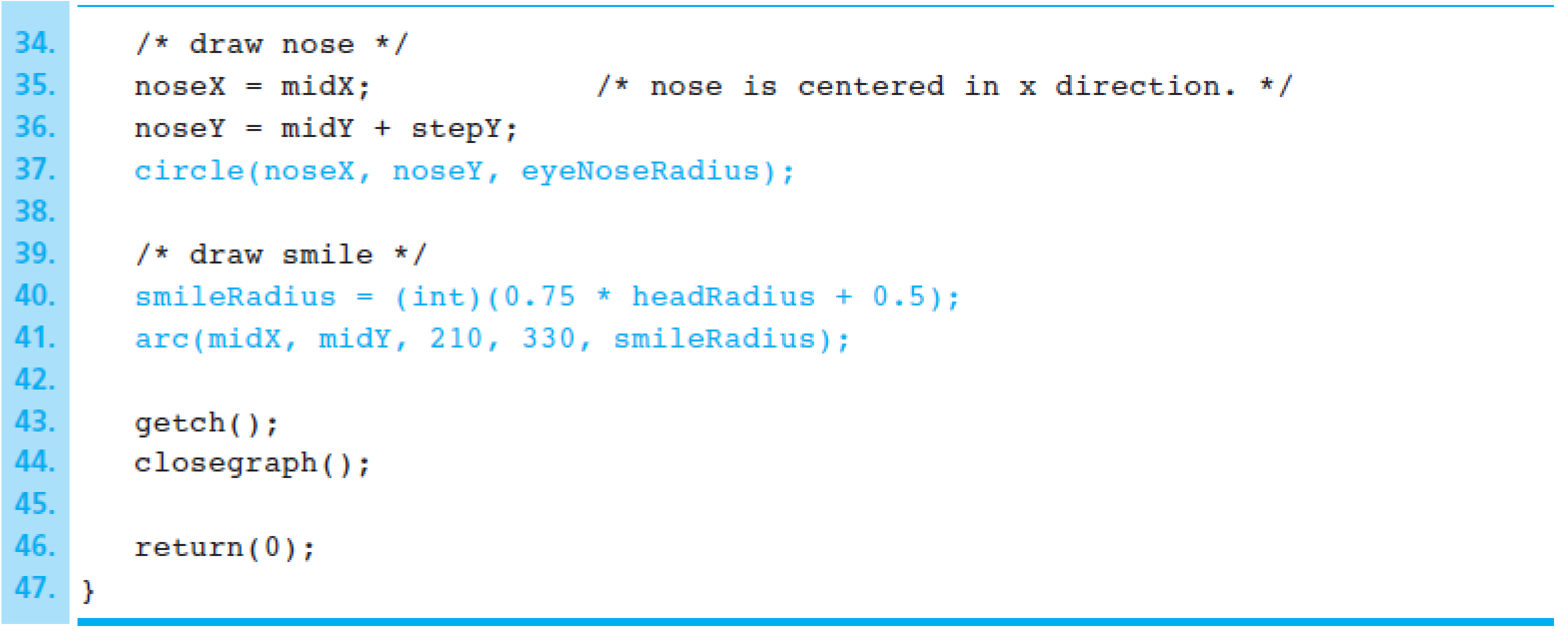
Note that the function getmaxx() and getmaxy() to determine the width and heigh of drawing window, use these function to find the coordinates of the center of the window.



```
/* Draws a happy face */

#include <graphics.h>

int
main(void)
{
   int midX, midY,                  /* coordinates of center point */
       leftEyeX, rightEyeX, eyeY,   /* eye center points */
       noseX, noseY,                /* nose center point */
       headRadius,                  /* head radius */
       eyeNoseRadius,               /* eye/nose radius */
       smileRadius,                 /* smile radius */
       stepX, stepY;                /* x and y increments */

   initwindow(500, 400,
              "Happy Face - press key to close", 200, 150);

   /* draw head */
   midX = getmaxx() / 2;             /* center head in x-direction */
   midY = getmaxy() / 2;             /* center head in y-direction */
   headRadius = getmaxy() / 4;   /* head will fill half the window */
   circle(midX, midY, headRadius); /* draw head */

   /* draw eyes */
   stepX = headRadius / 4;          /* x-offset for eyes */
   stepY = stepX;                   /* y-offset for eyes and nose */
   leftEyeX = midX - stepX;         /* x-coordinate for right eye */
   eyeY = midY - stepY;             /* y-coordinate for both eyes */
   eyeNoseRadius = headRadius / 10;
   circle(leftEyeX,  eyeY, eyeNoseRadius);   /* draw left eye. */
   circle(rightEyeX, eyeY, eyeNoseRadius);   /* draw right eye. */

```

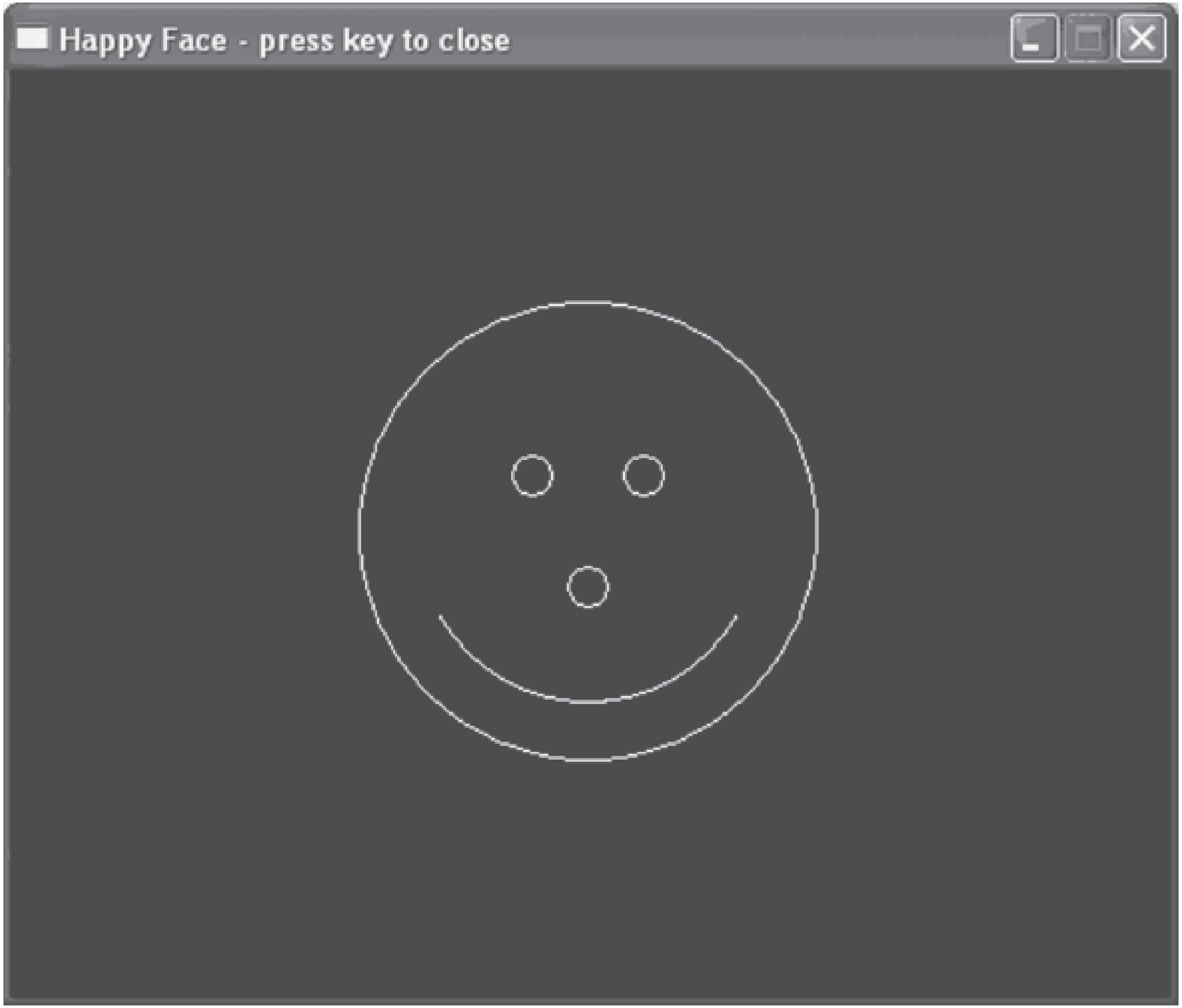
(continued)

Figure 3.31 Program to draw a happy face (cont’d)



```
   /* draw nose */
   noseX = midX;            /* nose is centered in x direction. */
   noseY = midY + stepY;
   circle(noseX, noseY, eyeNoseRadius);

   /* draw smile */
   smileRadius = (int)(0.75 * headRadius + 0.5);
   arc(midX, midY, 210, 330, smileRadius);

   getch();
   closegraph();

   return(0);
}
```

Figure 3.32 Program to draw a happy face (cont'd)

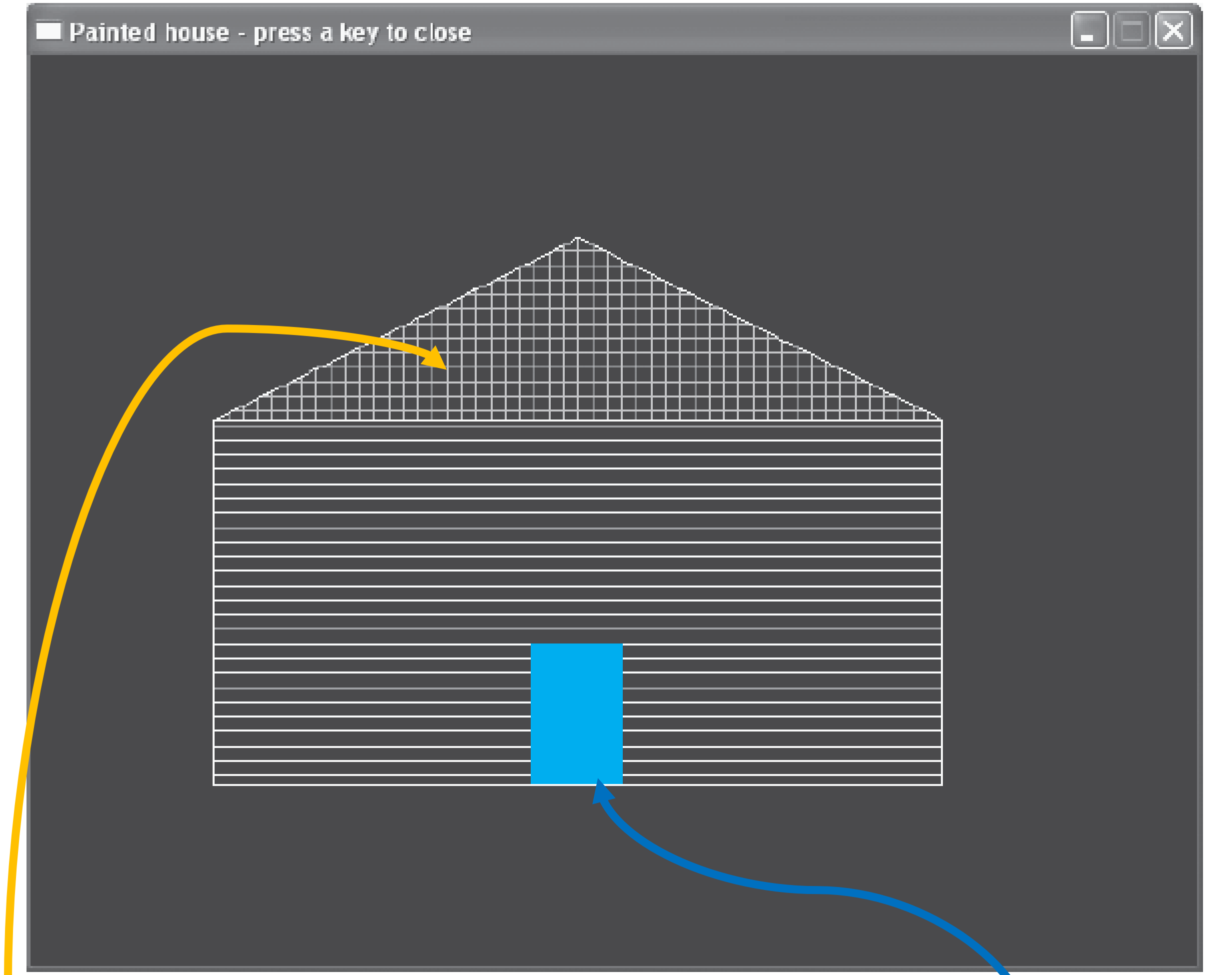


Program Style

- Drawing Filled Figures
  - setfillstyle(SLASH_FILL, RED)
  - floodfill(x, y, WHITE)
  - bar(x1, y1, x2, y2)

Constant	Value	Fill Pattern	Constant	Value	Fill Pattern
EMPTY_FILL	0	Background color	LTBKSLASH_FILL	6	\\\(light)
SOLID_FILL	1	Solid color	HATCH_FILL	7	Hatch(light)
LINK_FILL	2	---	XHATCH_FILL	8	Crosshatch
LTSLASH_FILL	3	///(light)	INTERLEAVE_FILL	9	Interleaving line
SLASH_FILL	4	///(heavy)	WIDE_DOT_FILL	10	Dots(light)
BKSLASH_FILL	5	\\\(heavy)	CLOSE_DOT_FILL	11	Dots(heavy)

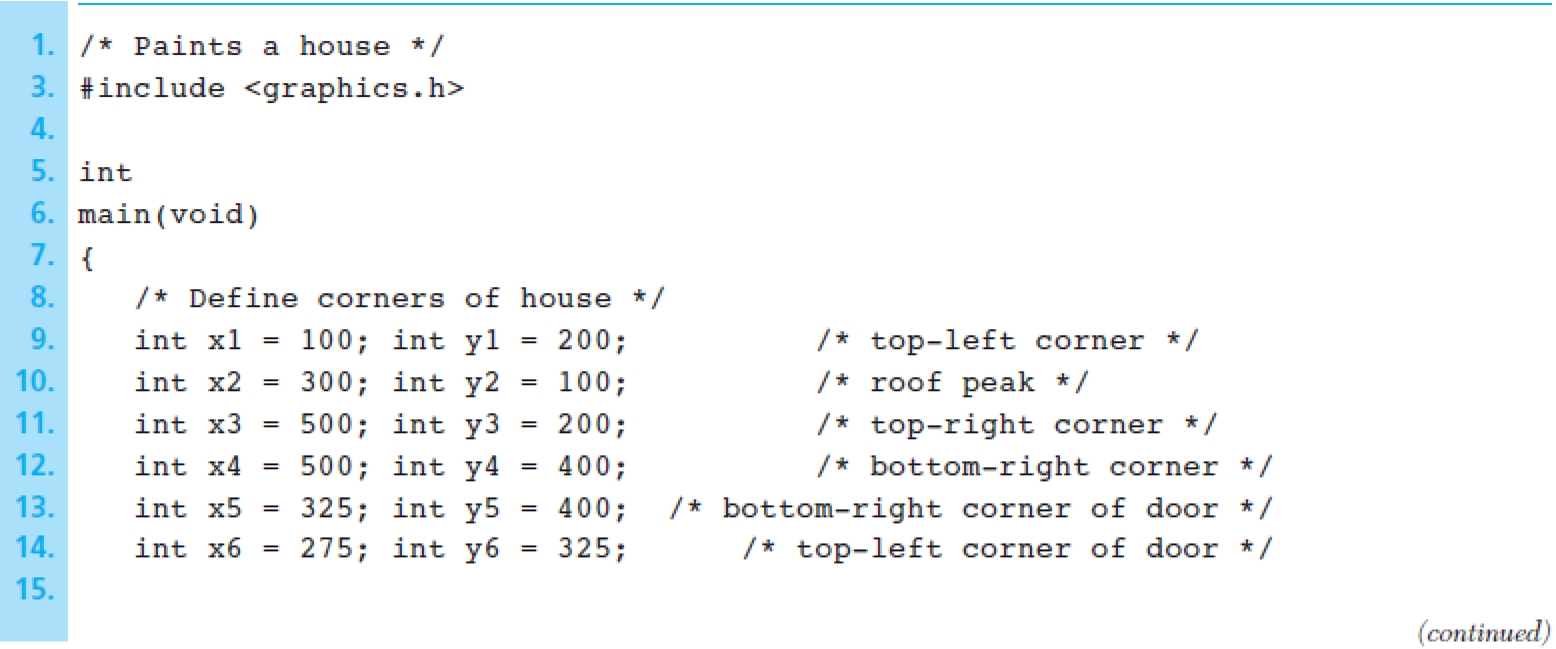
Figure 3.33 Painted house drawn by paintedHouse.c



```
setfillstyle(HATCH_FILL, LIGHTGRAY)
floodfill(x2, y2 + 10, WHITE) /* Paint the roof */
```

```
setfillstyle(SOLID_FILE, BLUE)
bar(x5, y5, x6, y6) /* Draw blue door */
```

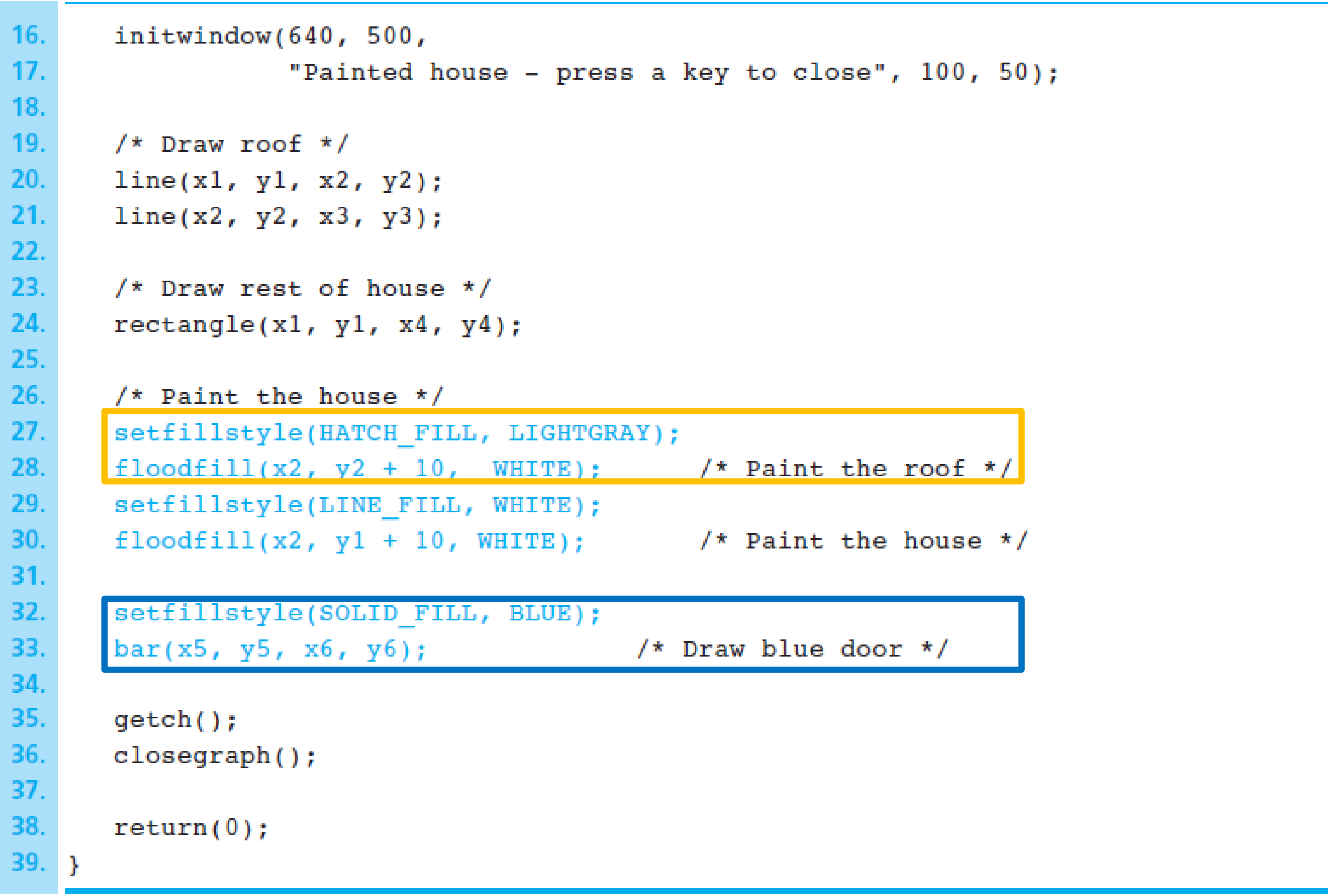
Figure 3.34 Program to paint a house



```
/* Paints a house */
#include <graphics.h>

int
main(void)
{
   /* Define corners of house */
   int x1 = 100; int y1 = 200;          /* top-left corner */
   int x2 = 300; int y2 = 100;          /* roof peak */
   int x3 = 500; int y3 = 200;          /* top-right corner */
   int x4 = 500; int y4 = 400;          /* bottom-right corner */
   int x5 = 325; int y5 = 400;  /* bottom-right corner of door */
   int x6 = 275; int y6 = 325;      /* top-left corner of door */

```

(continued)

Figure 3.34 Program to paint a house (cont’d)

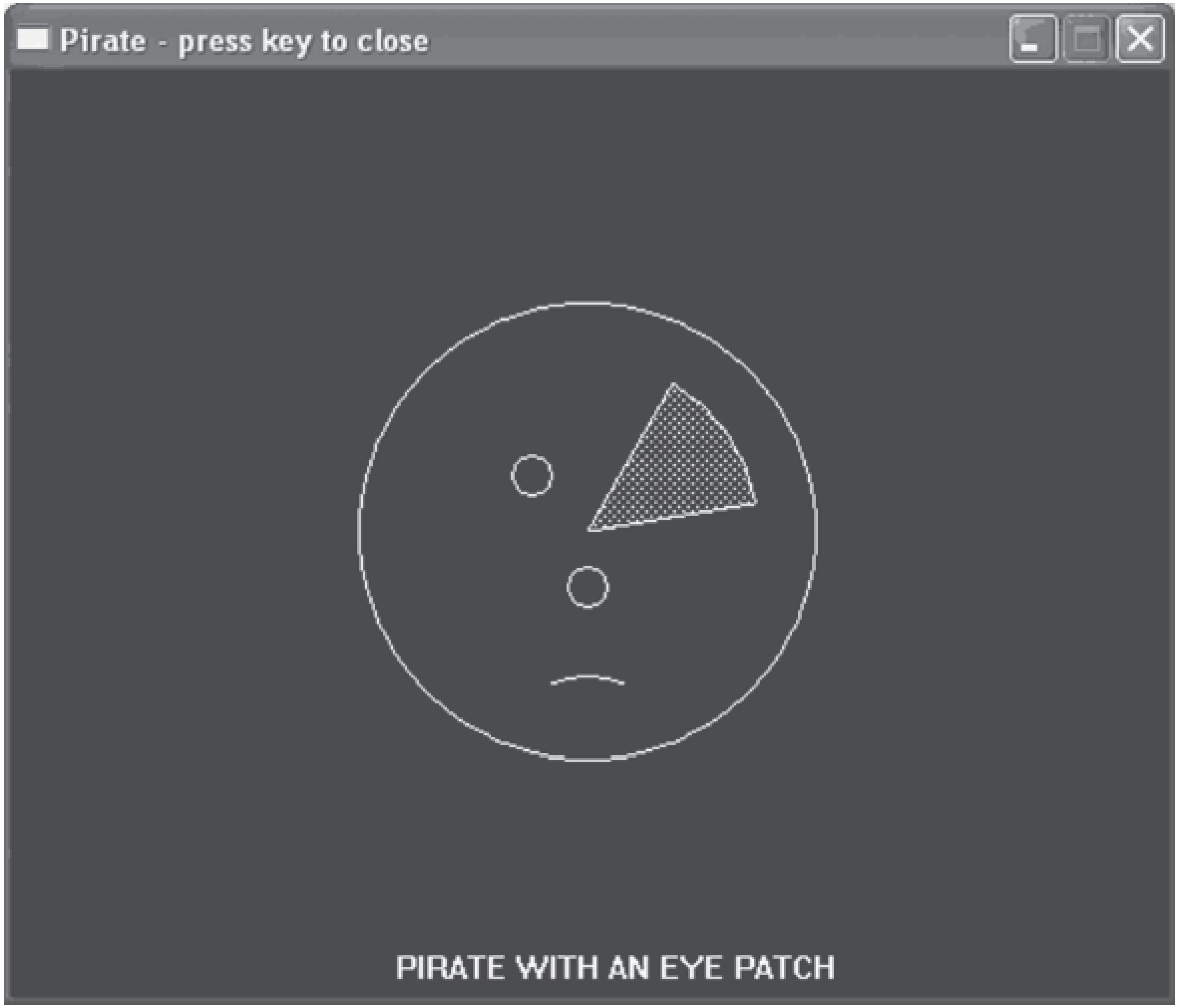


```
   initwindow(640, 500,
              "Painted house - press a key to close", 100, 50);

   /* Draw roof */
   line(x1, y1, x2, y2);
   line(x2, y2, x3, y3);

   /* Draw rest of house */
   rectangle(x1, y1, x4, y4);

   /* Paint the house */
   setfillstyle(HATCH_FILL, LIGHTGRAY);
   floodfill(x2, y2 + 10,  WHITE);     /* Paint the roof */
   setfillstyle(LINE_FILL, WHITE);
   floodfill(x2, y1 + 10, WHITE);      /* Paint the house */

   setfillstyle(SOLID_FILL, BLUE);
   bar(x5, y5, x6, y6);            /* Draw blue door */

   getch();
   closegraph();

   return(0);
}
```

Pie Slices and Filled Ellipses

- pieslice
  - Draws a filled pie slice (section of circle)
- fillellipse
  - Draws a filled ellipses or circle

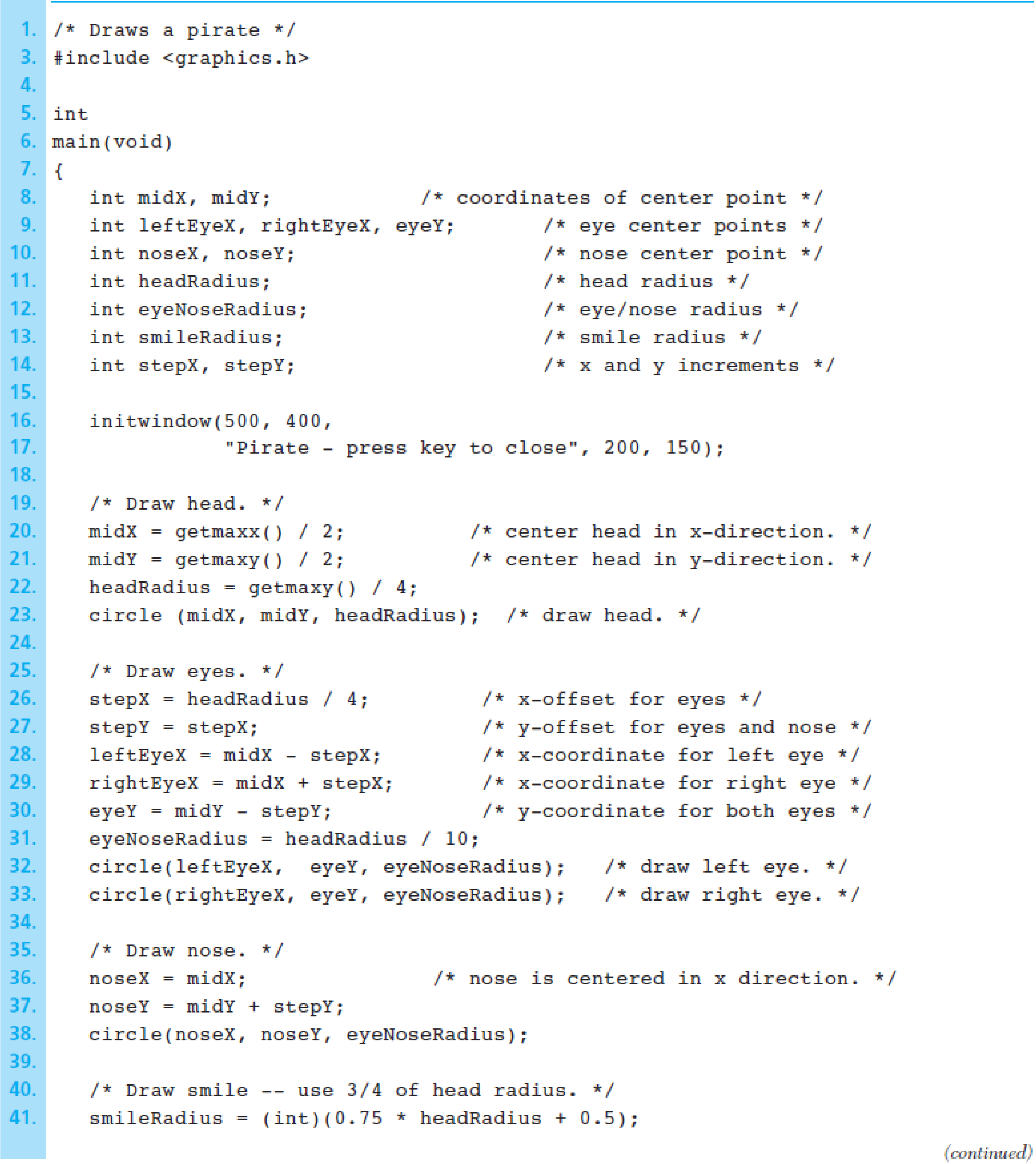
Figure 3.35 Pirate drawn by pirate.c



Pirate - press key to close

PIRATE WITH AN EYE PATCH

Figure 3.36 Program to draw a pirate



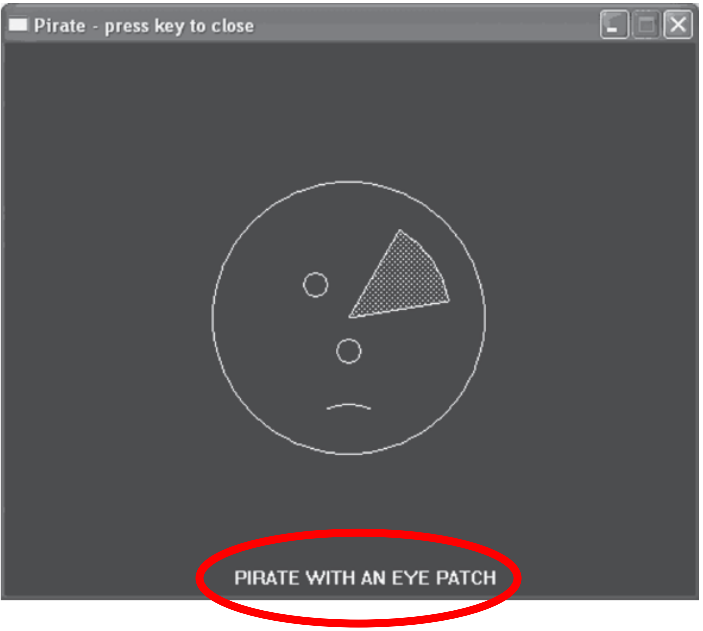
```
/* Draws a pirate */
#include <graphics.h>

int
main(void)
{
   int midX, midY;            /* coordinates of center point */
   int leftEyeX, rightEyeX, eyeY;       /* eye center points */
   int noseX, noseY;                    /* nose center point */
   int headRadius;                      /* head radius */
   int eyeNoseRadius;                   /* eye/nose radius */
   int smileRadius;                     /* smile radius */
   int stepX, stepY;                    /* x and y increments */

   initwindow(500, 400,
              "Pirate - press key to close", 200, 150);

   /* Draw head. */
   midX = getmaxx() / 2;          /* center head in x-direction. */
   midY = getmaxy() / 2;          /* center head in y-direction. */
   headRadius = getmaxy() / 4;
   circle (midX, midY, headRadius);  /* draw head. */

   /* Draw eyes. */
   stepX = headRadius / 4;         /* x-offset for eyes */
   stepY = stepX;                  /* y-offset for eyes and nose */
   leftEyeX = midX - stepX;        /* x-coordinate for left eye */
   rightEyeX = midX + stepX;       /* x-coordinate for right eye */
   eyeY = midY - stepY;            /* y-coordinate for both eyes */
   eyeNoseRadius = headRadius / 10;
   circle(leftEyeX,  eyeY, eyeNoseRadius);   /* draw left eye. */
   circle(rightEyeX, eyeY, eyeNoseRadius);   /* draw right eye. */

   /* Draw nose. */
   noseX = midX;               /* nose is centered in x direction. */
   noseY = midY + stepY;
   circle(noseX, noseY, eyeNoseRadius);

   /* Draw smile -- use 3/4 of head radius. */
   smileRadius = (int)(0.75 * headRadius + 0.5);
```

(continued)

Figure 3.36 Program to draw a pirate (cont'd)

```
   /* Draw frown */
   arc(midX, midY + headRadius, 65, 115, smileRadius / 2);

   setfillstyle(CLOSE_DOT_FILL, WHITE);
   pieslice(midX, midY, 10, 60, smileRadius); /* Draw eye patch */

   outtextxy(getmaxx() / 3, getmaxy() - 20,
             "PIRATE WITH AN EYE PATCH");

   getch();
   closegraph();

   return(0);
}
```

Adding Text to Drawing



```
PIRATE WITH AN EYE PATCH
```

The function is like a printf in c

Functions in Graphics Library

Function	Effect
arc(x, y, stAng, endAng, r)	draw an arc from angle stAng to endAng with center at (x, y) and radius r
bar(x1, y1, x2, y2)	draw a filled rectangle with a diagonal through points (x1, y1) and (x2, y2)
circle(x, y, r)	draw a circle with center at (x, y) and radius r
closegraph()	Closes graphics mode
ellipse(x, y, stAng, endAng, xRad, yRad)	Draws an ellipse with center at (x, y) from stAng to endAng with xRad as horizontal radius and yRad as vertical radius
fillellipse(x, y, xRad, yRad)	Draws a filled ellipse with center at (x, y) with xRad as horizontal radius and yRad as vertical radius
floodfill(x, y, border)	fills with the current fill pattern the figure containing the point (x, y) and bounded by lines with color border
getch()	pauses the program until the user enters a character
getmaxheight()	return the position of the last pixel in the y-direction in the screen
getmaxweight()	return the position of the last pixel in the x-direction in the screen
getmaxx()	return the window width in pixels

Functions in Graphics Library

Function	Effect
getmaxy()	return the window height in pixels
initgraph(x, y, label)	displays a window x pixels wide and y pixels high with the given label and top-left corner at (0, 0)
initgraph(x, y, label, x0, y0)	displays a window x pixels wide and y pixels high with the given label and top-left corner at (x0, y0)
line(x1, y1, x2, y2)	draws a line with end points (x1, y1) and (x2, y2)
outtextxy(x, y, textString)	draws the characters for textString starting at point (x, y)
pieslice(x, y, stAng, endAng, r)	draws a filled pie slice with center at (x, y) frome angle stAng to endAng with radius r
rectangle(x1, y1, x2, y2)	draws a rectangle with a diagonal through point (x1, y1) and (x2, y2)
setbkcolor(backColor)	sets the background color to backColor
setbcolor(foreColor)	sets the foreground color to foreColor
setfillstyle(filPat, filCol)	sets the fill pattern to filPat and the fill color to filcol

Testing Functions Using Drivers

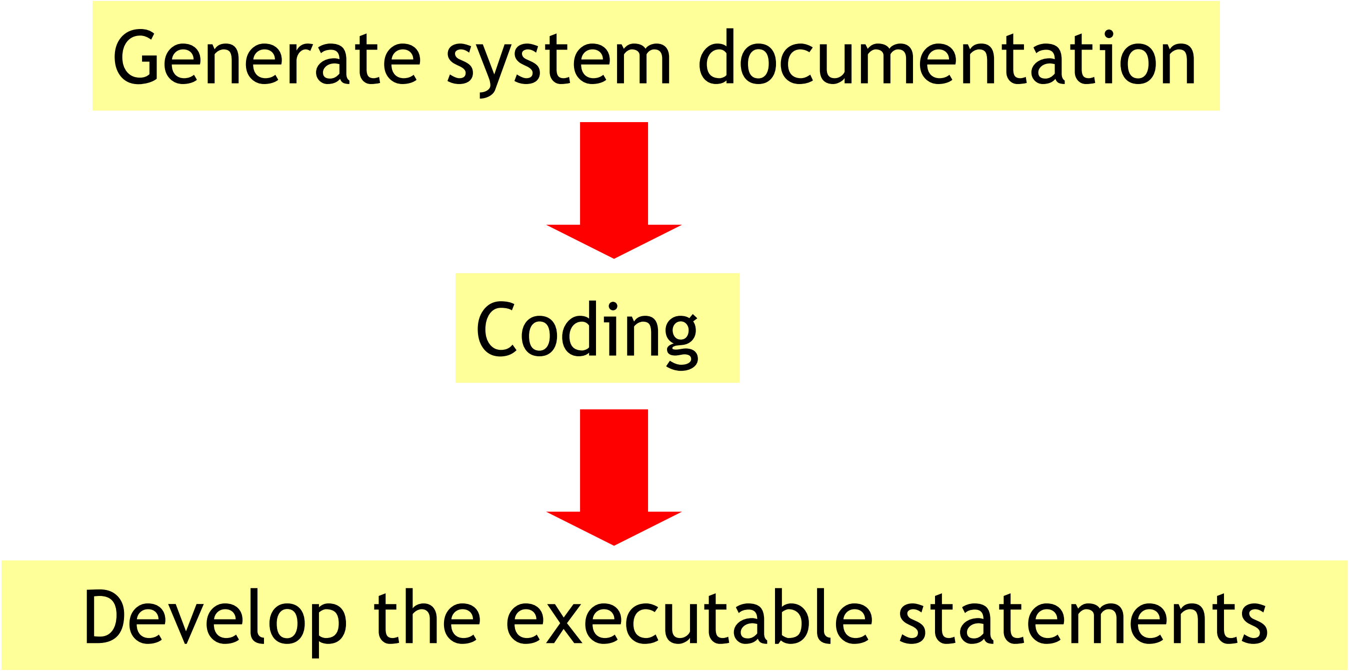
- Driver
  - A short function written to test another function by defining its arguments, calling it, and displaying its result

3.7 Common Programming Errors

- Using functions you must
  - Place prototypes for your own function subprograms in the source file preceding the main function
  - Place the actual function definitions after the main function
- Syntax or run-time errors may occur when you use functions.

Chapter Review (1)

- Code a program steps



Generate system documentation

Coding

Develop the executable statements

Chapter Review (2)

- C library functions provide predefined functions to promote code reuse.
- Top-down design proceeds from the original problem at the top level to the subprograms at each lower level.
- Use a structure chart to show subordinate relationships between subproblems.

Chapter Review (3)

- Utilize modular programming by writing subprogram functions.
- Write functions that have input arguments and that return a single result.
  - Actual argument value is assigned to corresponding formal parameters
- Take care of syntax errors or run-time errors when writing functions

Question?

- A good question deserves a good grade…