Computer Programming

Top-Down Design with Functions

Function: modules of program

- Programmers use segments of earlier programs to construct new programs
  - Documentation is very important
  - Use of predefined functions
  - Top-down stepwise refinement
    - Major steps = modules of program

Case Study: Circle

- Problem: Compute and display the area and the circumference of a circle
- Analysis:
  - Input: radius (double)
  - Outputs: area and circumference (double)
  - Relationship: ???
- Design:
  1. Get the radius
  2. Calculate the area
  3. Calculate the circumference
  4. Display the area and the circumference
  - Some steps requires refinement

Case Study: Circle

- Implementation:
  - The following slide contains the initial program

Outline of Program Circle

```
1.  /*
2.   * Calculates and displays the area and circumference of a circle
3.   */
4.
5.  #include <stdio.h>
6.  #define PI 3.14159
7.
8.  int
9.  main(void)
10. {
11.       double radius;     /* input - radius of a circle  */
12.       double area;       /* output - area of a circle   */
13.       double circum;     /* output - circumference      */
14.
15.       /* Get the circle radius */
16.
17.       /* Calculate the area */
18.          /* Assign PI * radius * radius to area. */
19.
20.       /* Calculate the circumference */
21.          /* Assign 2 * PI * radius to circum. */
22.
23.       /* Display the area and circumference */
24.
25.       return (0);
26. }
```

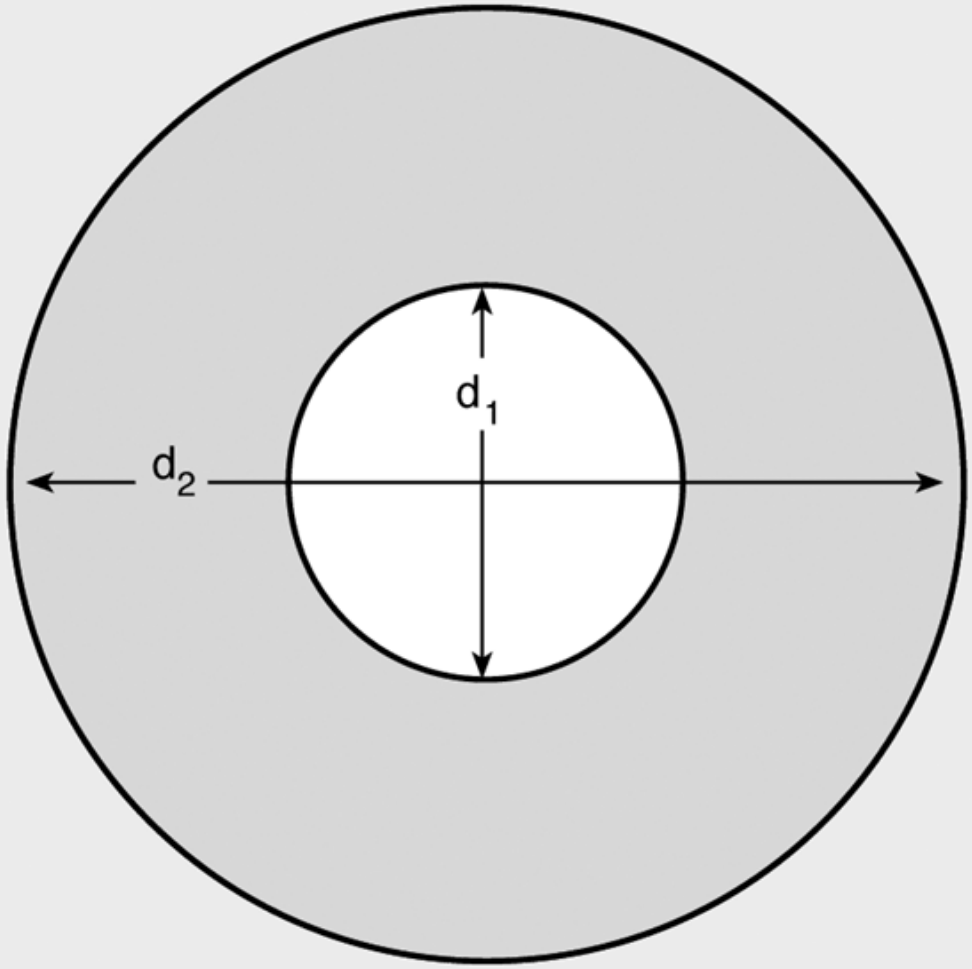
```c
/*
 * Calculates and displays the area and circumference of a circle
 */

#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 */
	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: Weight of Washers

- Here, we will use the solution of the previous case study
- Problem: Manufacturer of flat washers needs to estimate shipping cost. They need to compute the weight of a specifies quantity of flat washers
- Analysis:
  - Weight is volume times density of the material
  - Volume is the rim area times thickness
  - Rim area is calculated as in the next slide
  - Inputs: diameters, thickness, density, quantity
  - Outputs: weight
  - Relationships: ??

Computing Area of a Flat Washer



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

Case Study: Weight of Washers

- Design:
  - Initial Algorithm: ??
- Implementation:
  - next

```
5.  #include <stdio.h>
6.  #define PI 3.14159
7.
8.  int
9.  main(void)
10. {
11.       double hole_diameter; /* input - diameter of hole         */
12.       double edge_diameter; /* input - diameter of outer edge   */
13.       double thickness;     /* input - thickness of washer      */
14.       double density;       /* input - density of material used */
15.       double quantity;      /* input - number of washers made   */
16.       double weight;        /* output - weight of washer batch  */
17.       double hole_radius;   /* radius of hole                   */
18.       double edge_radius;   /* radius of outer edge             */
19.       double rim_area;      /* area of rim                      */
20.       double unit_weight;   /* weight of 1 washer               */
21.
22.       /* Get the inner diameter, outer diameter, and thickness.*/
23.       printf("Inner diameter in centimeters> ");
24.       scanf("%lf", &hole_diameter);
25.       printf("Outer diameter in centimeters> ");
26.       scanf("%lf", &edge_diameter);
27.       printf("Thickness in centimeters> ");
28.       scanf("%lf", &thickness);
29.
30.       /* Get the material density and quantity manufactured. */
31.       printf("Material density in grams per cubic centimeter> ");
32.       scanf("%lf", &density);
33.       printf("Quantity in batch> ");
34.       scanf("%lf", &quantity);
35.
36.       /* Compute the rim area. */
37.       hole_radius = hole_diameter / 2.0;
38.       edge_radius = edge_diameter / 2.0;
39.       rim_area = PI * edge_radius * edge_radius -
40.                  PI * hole_radius * hole_radius;
41.
42.       /* Compute the weight of a flat washer. */
43.       unit_weight = rim_area * thickness * density;
```

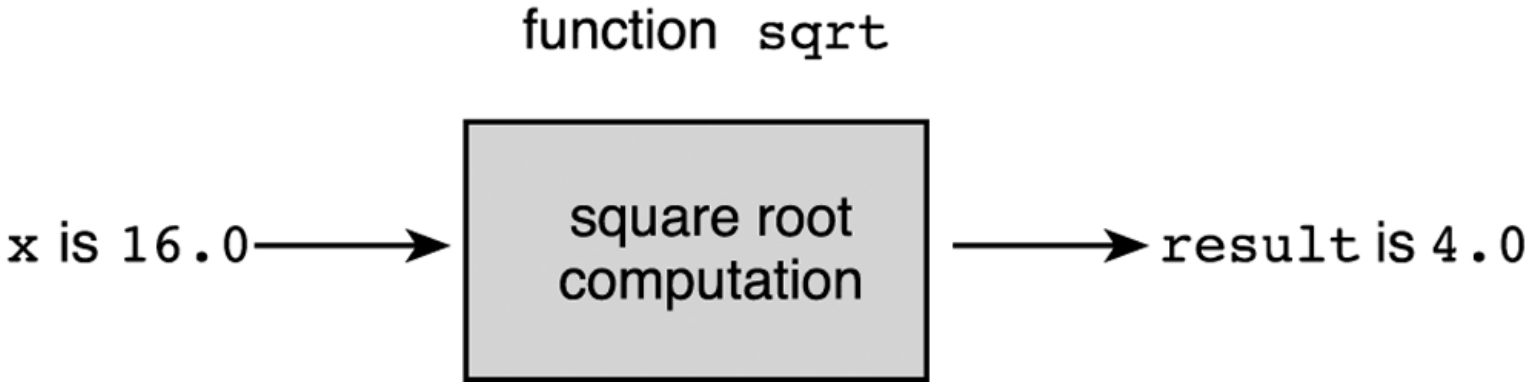
```
44.       /* Compute the weight of the batch of washers. */
45.       weight = unit_weight * quantity;
46.
47.       /* Display the weight of the batch of washers. */
48.       printf("\nThe expected weight of the batch is %.2f", weight);
49.       printf(" grams.\n");
50.
51.       return (0);
52. }

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.
```

Library Functions

- Software engineering:
  - Goal: writing error-free codes
  - Use well tested existing codes: code reuse
  - Use predefined functions
    - EX: sqrt function in math library
    - Use it as a black box

      y = sqrt(x);
    - EX: printf and scanf in stdio library

Function sqrt as a “Black Box”



Square Root Program

```
1.  /*
2.   * Performs three square root computations
3.   */
4.
5.  #include <stdio.h> /* definitions of printf, scanf */
6.  #include <math.h>  /* definition of sqrt */
7.
8.  int
9.  main(void)
10. {
11.       double first, second,    /* input - two data values          */
12.              first_sqrt,       /* output - square root of first    */
13.              second_sqrt,      /* output - square root of second   */
14.              sum_sqrt;         /* output - square root of sum      */
15.
16.       /* Get first number and display its square root.  */
17.       printf("Enter the first number> ");
18.       scanf("%lf", &first);
19.       first_sqrt = sqrt(first);
20.       printf("The square root of the first number is %.2f\n", first_sqrt);
```

(continued)

Square Root Program (cont'd)

```
21.         /* Get second number and display its square root. */
22.         printf("Enter the second number> ");
23.         scanf("%lf", &second);
24.         second_sqrt = sqrt(second);
25.         printf("The square root of the second number is %.2f\n", second_sqrt);
26.
27.         /* Display the square root of the sum of the two numbers. */
28.         sum_sqrt = sqrt(first + second);
29.         printf("The square root of the sum of the two numbers is %.2f\n",
30.                 sum_sqrt);
31.
32.         return (0);
33.   }

      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 Library Functions

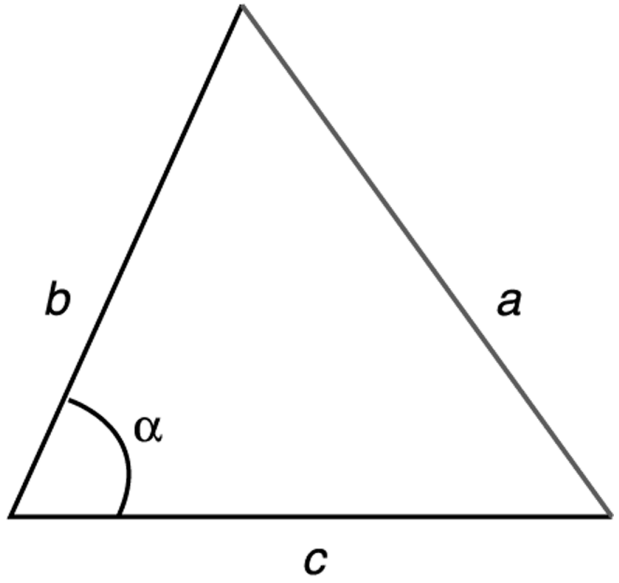
Function	Standard Header File	Purpose: Example	Argument(s)	Result
`abs(x)`	`<stdlib.h>`	Returns the absolute value of its integer argument: if `x` is `-5`, `abs(x)` is `5`	`int`	`int`
`ceil(x)`	`<math.h>`	Returns the smallest integral value that is not less than `x`: if `x` is `45.23`, `ceil(x)` is `46.0`	`double`	`double`
`cos(x)`	`<math.h>`	Returns the cosine of angle `x`: if `x` is `0.0`, `cos(x)` is `1.0`	`double` (radians)	`double`
`exp(x)`	`<math.h>`	Returns e^x where $e = 2.71828...$: if `x` is `1.0`, `exp(x)` is `2.71828`	`double`	`double`
`fabs(x)`	`<math.h>`	Returns the absolute value of its type `double` argument: if `x` is `-8.432`, `fabs(x)` is `8.432`	`double`	`double`
`floor(x)`	`<math.h>`	Returns the largest integral value that is not greater than `x`: if `x` is `45.23`, `floor(x)` is `45.0`	`double`	`double`
`log(x)`	`<math.h>`	Returns the natural logarithm of `x` for `x > 0.0`: if `x` is `2.71828`, `log(x)` is `1.0`	`double`	`double`
`log10(x)`	`<math.h>`	Returns the base-10 logarithm of `x` for `x > 0.0`: if `x` is `100.0`, `log10(x)` is `2.0`	`double`	`double`
`pow(x, y)`	`<math.h>`	Returns x^y. If `x` is negative, `y` must be integral: if `x` is `0.16` and `y` is `0.5`, `pow(x, y)` is `0.4`	`double`, `double`	`double`
`sin(x)`	`<math.h>`	Returns the sine of angle `x`: if `x` is `1.5708`, `sin(x)` is `1.0`	`double` (radians)	`double`
`sqrt(x)`	`<math.h>`	Returns the non-negative square root of `x` ($\sqrt{x}$) for `x >= 0.0`: if `x` is `2.25`, `sqrt(x)` is `1.5`	`double`	`double`
`tan(x)`	`<math.h>`	Returns the tangent of angle x: if `x` is 0.0, `tan(x)` is 0.0	`double` (radians)	`double`

Library Functions

- Example: Compute the roots of a quadratic equation

- Example: Compute the length of the third side of a triangle

$$a^2 = b^2 + c^2 - 2bc \cos \alpha$$



User defined Functions

- Example: area of a circle

  area = find_area(radius);

- Example: circumference of a circle

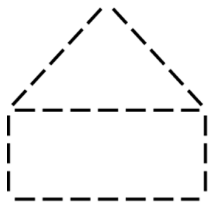
  circum = find_circum(radius);

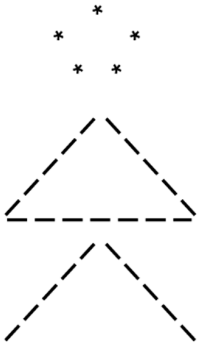
- Example: rim area calculation

  rim_area = find_area(edge_radius) - find_area(hole_radius);

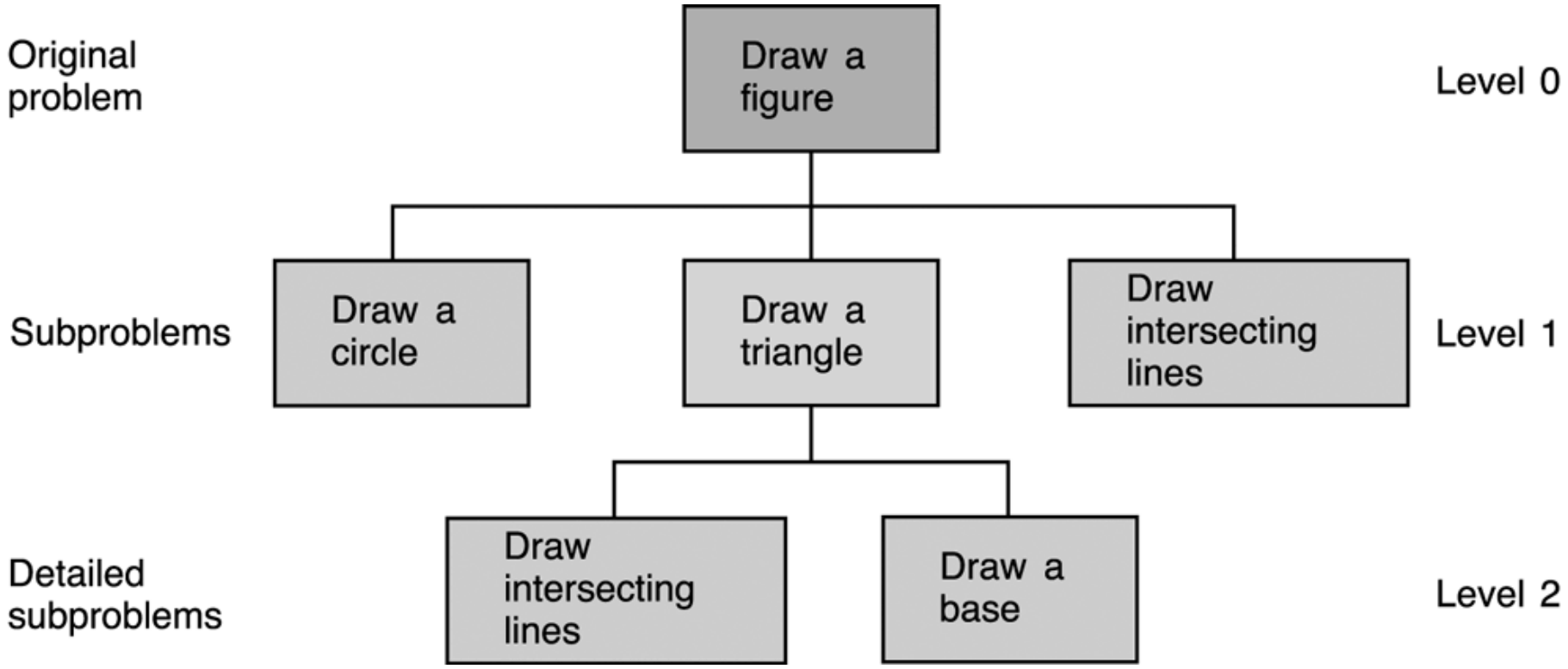
Case Study: Simple Diagrams

- Problem: Draw simple diagrams on your screen
  - Ex: house, person
- Analysis: Basic components
  - Circle
  - Parallel lines
  - Base line
  - Intersecting lines
- Design: Divide the problem into three subproblems
  - Draw a circle
  - Draw a triangle
  - Draw intersecting lines
    - Further refinement in triangle – see following structure chart





Structure Chart for Drawing a Stick Figure



Function Prototypes and Main Function

```
1.  /*
2.   * Draws a stick figure
3.   */
4.
5.  #include <stdio.h>
6.
7.  /* function prototypes                                          */
8.
9.  void draw_circle(void);       /* Draws a circle                 */
10.
11. void draw_intersect(void);    /* Draws intersecting lines       */
12.
13. void draw_base(void);         /* Draws a base line              */
14.
15. void draw_triangle(void);     /* Draws a triangle               */
16.
17. int
18. main(void)
19. {
20.       /* Draw a circle.  */
21.       draw_circle();
22.
23.       /* Draw a triangle.  */
24.       draw_triangle();
25.
26.       /* Draw intersecting lines.  */
27.       draw_intersect();
28.
29.       return (0);
30. }
```

User Defined Functions

- Function prototype
  - Functions should be defined before they are used
    - Insert the whole function definition
    - Insert the function prototype
  - Defines
    - Data types of the function
    - Function name
    - Arguments and their types

    function_type function_name (argument types);

  - Ex:

    void draw_circle(void);

User Defined Functions

- Function call
  - Calling a function

```
function_name (arguments);
```

  - Ex:

```
draw_circle();
printf("%d", year);
```

User Defined Functions

- Function definition
  - Defines the operation of a function
  - Similar to main function

```
function_type function_name (argument list)
{
    local declerations
    executable statements
}
```

  - Function heading: similar to function prototype
  - Function body: enclosed in braces

Function draw_circle

```
1.  /*
2.   * Draws a circle
3.   */
4.  void
5.  draw_circle(void)
6.  {
7.        printf("   *  \n");
8.        printf(" *   *\n");
9.        printf("  * * \n");
10. }
```

Function draw_triangle

```
1.  /*
2.   * Draws a triangle
3.   */
4.  void
5.  draw_triangle(void)
6.  {
7.        draw_intersect();
8.        draw_base();
9.  }
```

Program to Draw a Stick Figure

```
1.  /* Draws a stick figure */
2.
3.  #include <stdio.h>
4.
5.  /* Function prototypes */
6.  void draw_circle(void);            /* Draws a circle                    */
7.
8.  void draw_intersect(void);         /* Draws intersecting lines          */
9.
10. void draw_base(void);              /* Draws a base line                 */
11.
12. void draw_triangle(void);          /* Draws a triangle                  */
13.
14. int
15. main(void)
16. {
17.
18.       /* Draw a circle.                    */
19.       draw_circle();
20.
21.       /* Draw a triangle.                  */
22.       draw_triangle();
23.
24.       /* Draw intersecting lines.          */
25.       draw_intersect();
26.
27.       return (0);
28. }
29.
```

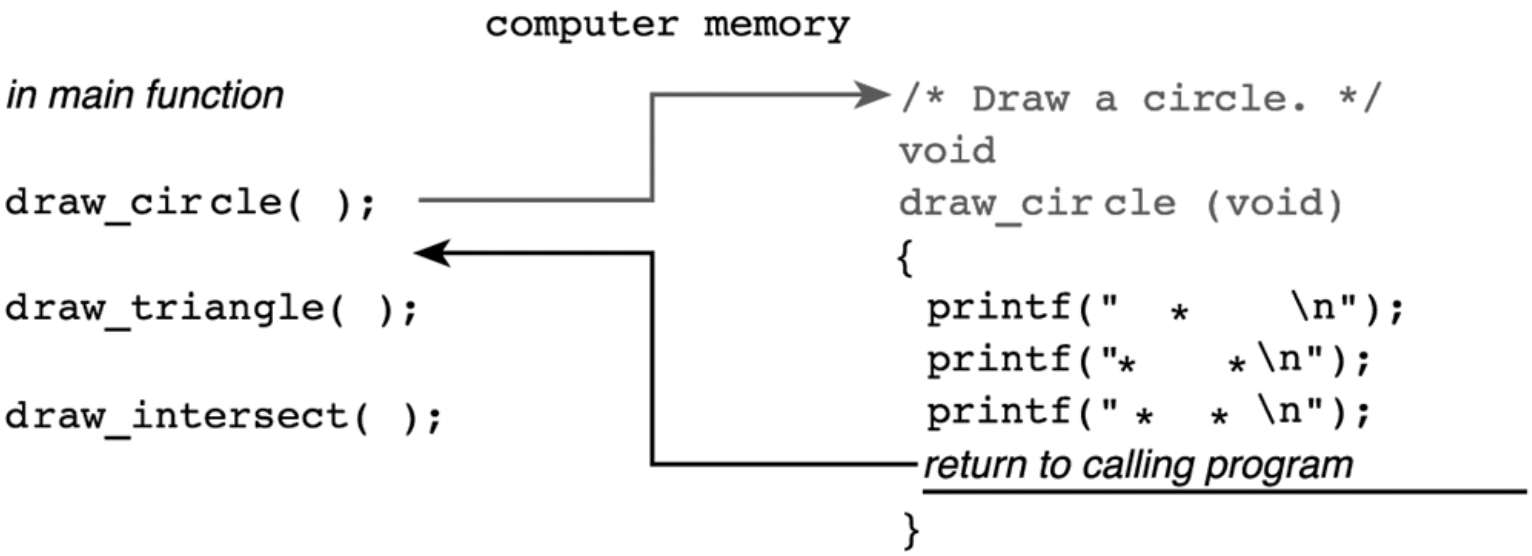
(continued)

```
30. /*
31.  * Draws a circle
32.  */
33. void
34. draw_circle(void)
35. {
36.       printf("   *   \n");
37.       printf(" *   * \n");
38.       printf("  * *  \n");
39. }
40.
41. /*
42.  * Draws intersecting lines
43.  */
44. void
45. draw_intersect(void)
46. {
47.       printf("  / \\  \n"); /* Use 2 \'s to print 1 */
48.       printf(" /   \\ \n");
49.       printf("/     \\\n");
50. }
51.
52. /*
53.  * Draws a base line
54.  */
55. void
56. draw_base(void)
57. {
58.       printf("-------\n");
59. }
60.
61. /*
62.  * Draws a triangle
63.  */
64. void
65. draw_triangle(void)
66. {
67.       draw_intersect();
68.       draw_base();
69. }
```

Flow of Control

- Compiling the program:
  - Function prototypes: compiler knows the functions
    - enables compiler to translate function calls
  - Function call: Transfers of the control to the function
  - Function definition: translates the code of the function
    - Allocates memory needed
  - End of the function: Transfer of the control back to the calling statement
    - Releases the local memory

Flow of Control



Advantages of Functions

- For team of programmers:
  - Dividing programming tasks to the programmers
- Procedural abstraction
  - Move the details of the operation to the functions
  - Focus on the main operations
- Code reuse
  - In a program
  - In other programs
    - Well tested functions

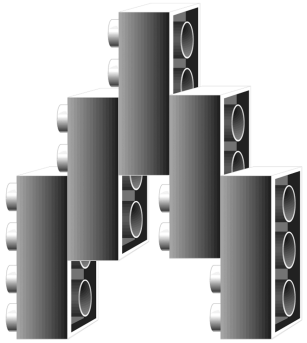
Function instruct

```
1.  /*
2.   * Displays instructions to a user of program to compute
3.   * the area and circumference of a circle.
4.   */
5.  void
6.  instruct(void)
7.  {
8.        printf("This program computes the area\n");
9.        printf("and circumference of a circle.\n\n");
10.       printf("To use this program, enter the radius of\n");
11.       printf("the circle after the prompt: Enter radius>\n");
12. }
```

```
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>
```

Functions with Input Arguments

- Functions are building blocks to construct large programs
  - Like Lego blocks



- Arguments:
  - to carry information to functions : input arguments
  - to return multiple results : output arguments
- Arguments makes functions more versatile
  - Manipulate different data at each call

```
rim_area = find_area(edge_radius) - find_area(hole_radius);
```

Function print_rboxed

```
1.  /*
2.   * Displays a real number in a box.
3.   */
4.
5.  void
6.  print_rboxed(double rnum)
7.  {
8.        printf("***********\n");
9.        printf("*         *\n");
10.       printf("* %7.2f *\n", rnum);
11.       printf("*         *\n");
12.       printf("***********\n");
13. }
```

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

Executing print_rboxed (135.68);

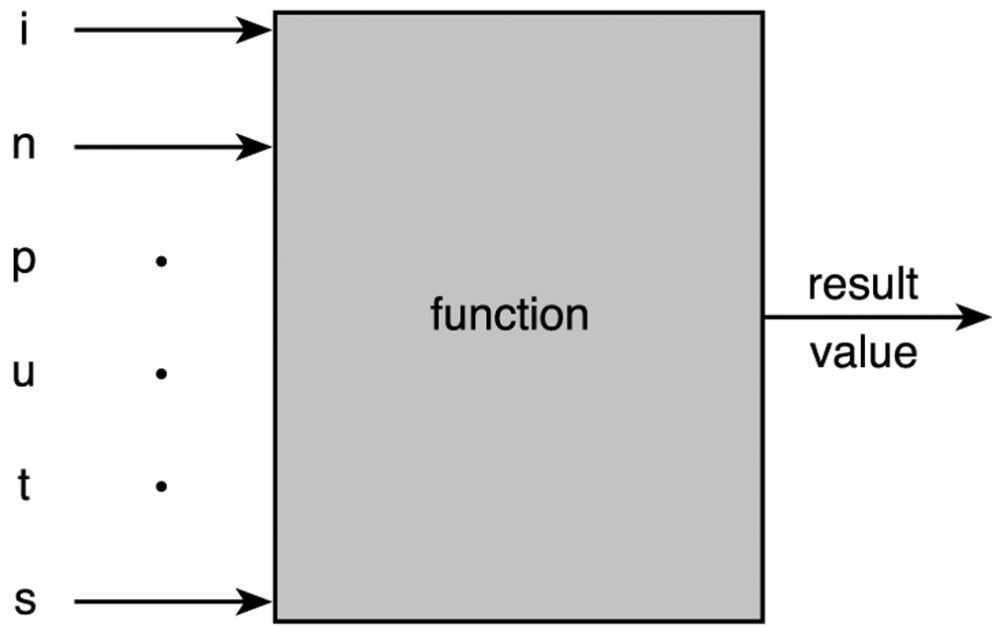
- Actual parameter: 135.68
- Formal parameter: rnum

```
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 Arguments and Result



Functions find_circum and find_area

```
1.  /*
2.   * Computes the circumference of a circle with radius r.
3.   * Pre:  r is defined and is > 0.
4.   *       PI is a constant macro representing an approximation of pi.
5.   */
6.  double
7.  find_circum(double r)
8.  {
9.       return (2.0 * PI * r);
10. }
11.
12. /*
13.  * Computes the area of a circle with radius r.
14.  * Pre:  r is defined and is > 0.
15.  *       PI is a constant macro representing an approximation of pi.
16.  *       Library math.h is included.
17.  */
18. double
19. find_area(double r)
20. {
21.      return (PI * pow(r, 2));
22. }
```

Executing circum = find_circum (radius);

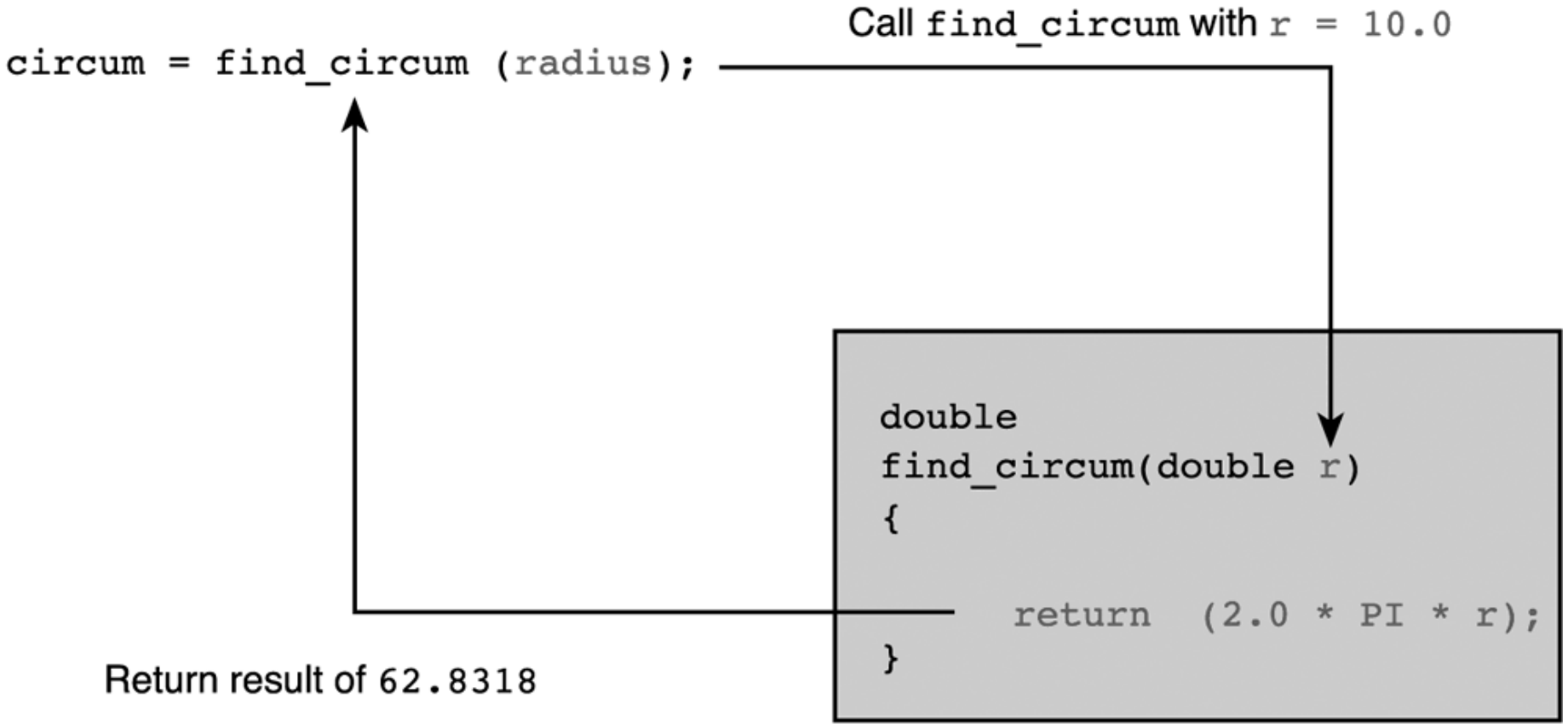


Figure 3.23 Function scale

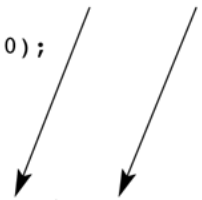
```
1.  /*
2.   * Multiplies its first argument by the power of 10 specified
3.   * by its second argument.
4.   * Pre : x and n are defined and math.h is included.
5.   */
6.  double
7.  scale(double x, int n)
8.  {
9.       double scale_factor;      /* local variable */
10.      scale_factor = pow(10, n);
11.
12.      return (x * scale_factor);
13. }
```

Testing functions

- Functions can be tested by a program that uses it.
- Driver program
  - Defines function arguments
  - Call the functions
  - Display the return value

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

#include <math.h>

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

int
main(void)
{
     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);
}
```

actual arguments

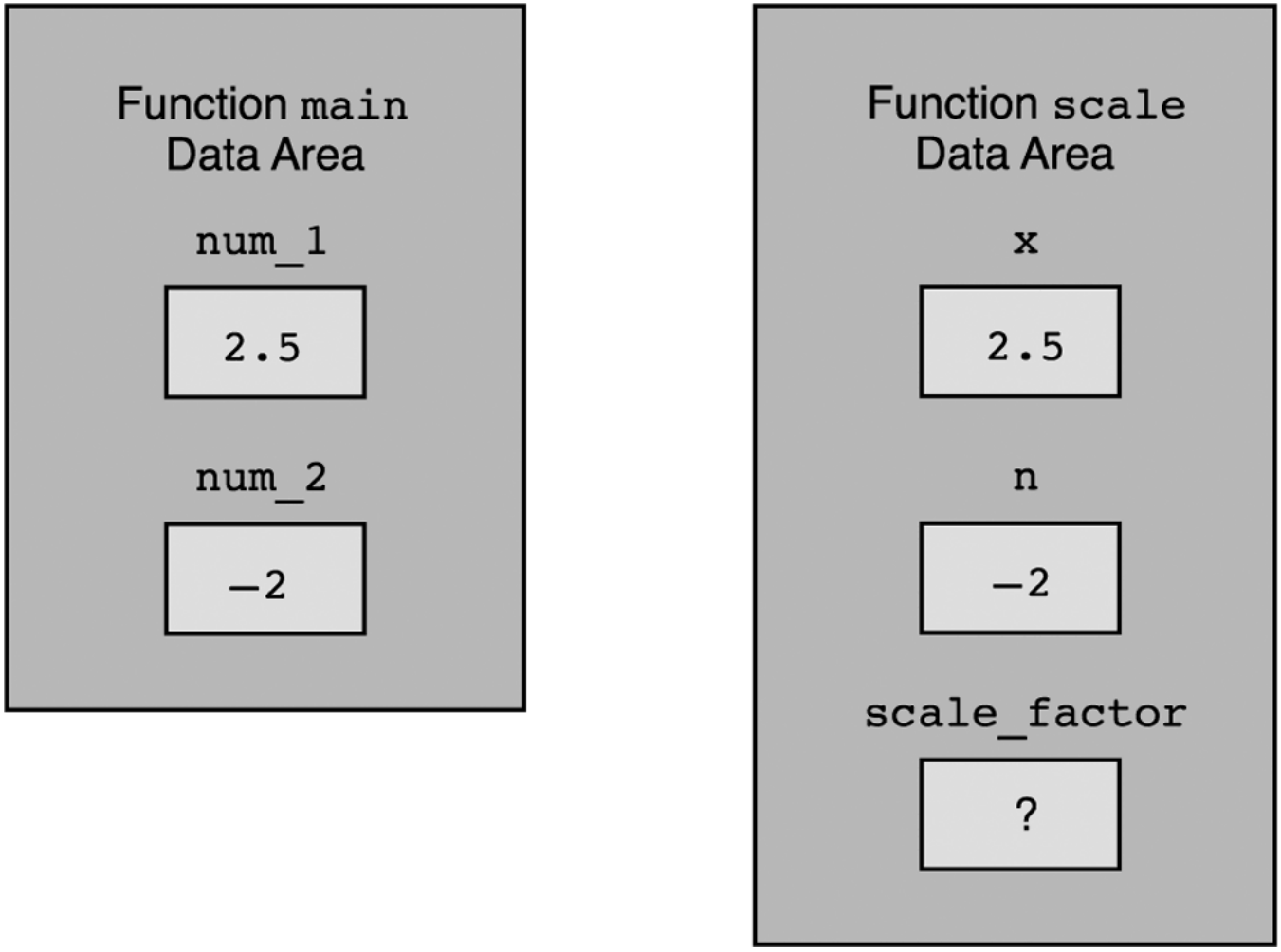


information flow

formal parameters

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

scale(num_1, num_2);



Argument Correspondence

- Be careful to provide correct
  - number of arguments
  - order of arguments
  - type of arguments
    - Actual parameter **int** to formal parameter **double**
    - Actual parameter **double** to formal parameter **int**
      - Loss of fractional part