

Top-Down Design with Functions

Chapter 3

Problem Solving & Program Design in C

Eighth Edition

Jeri R. Hanly & Elliot B. Koffman

Chapter Objectives

- To learn about functions and how to use them to write programs with separate modules
- To understand the capabilities of some standard functions in C
- To understand how control flows between function main and other functions
- To learn how to pass information to functions using input arguments
- To learn how to return a value from a function

Top-Down Design

- top-down design
 - a problem solving method
 - first, break a problem up into its major subproblems
 - solve the subproblems to derive the solution to the original problem

Figure 3.9

House and Stick Figure

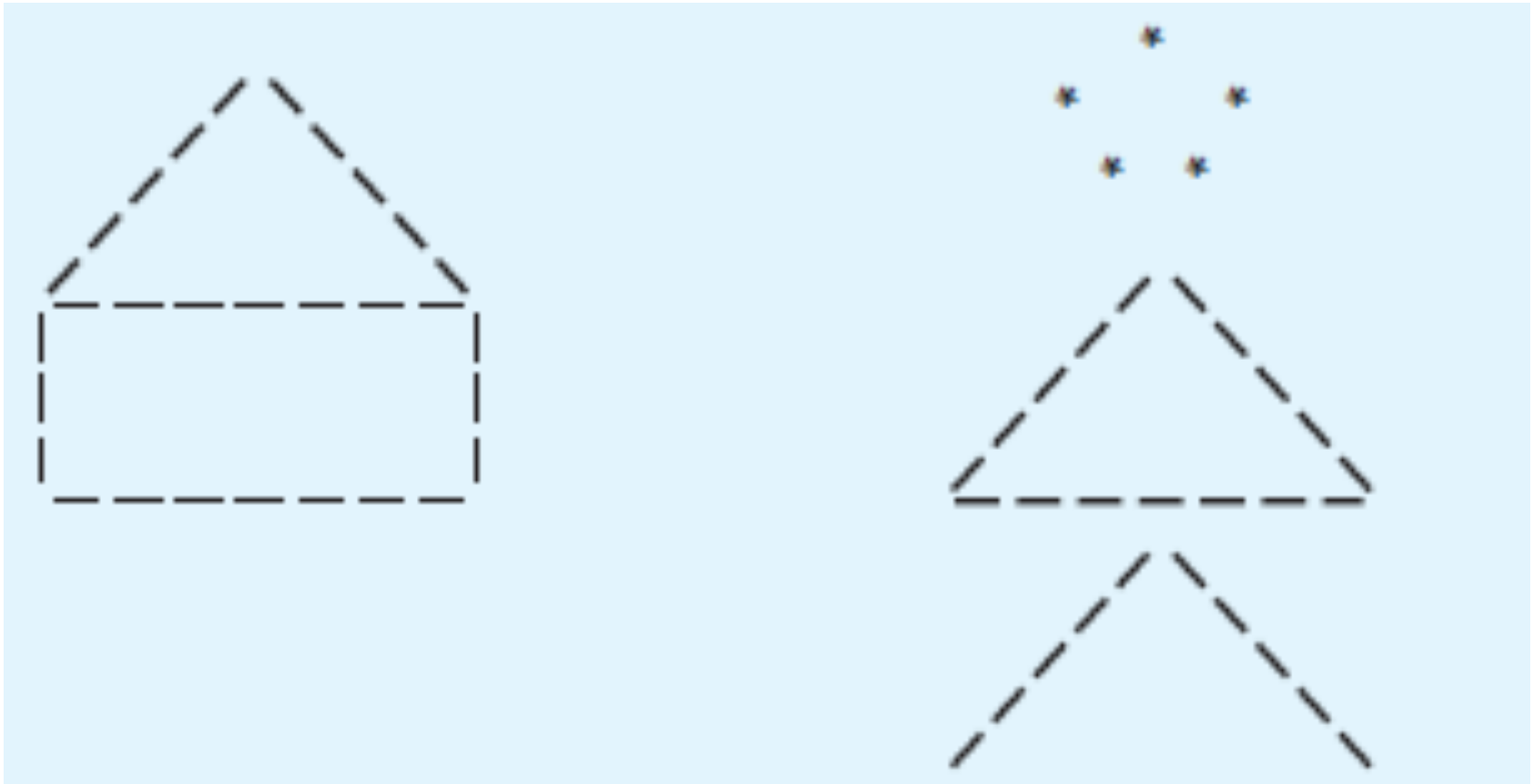
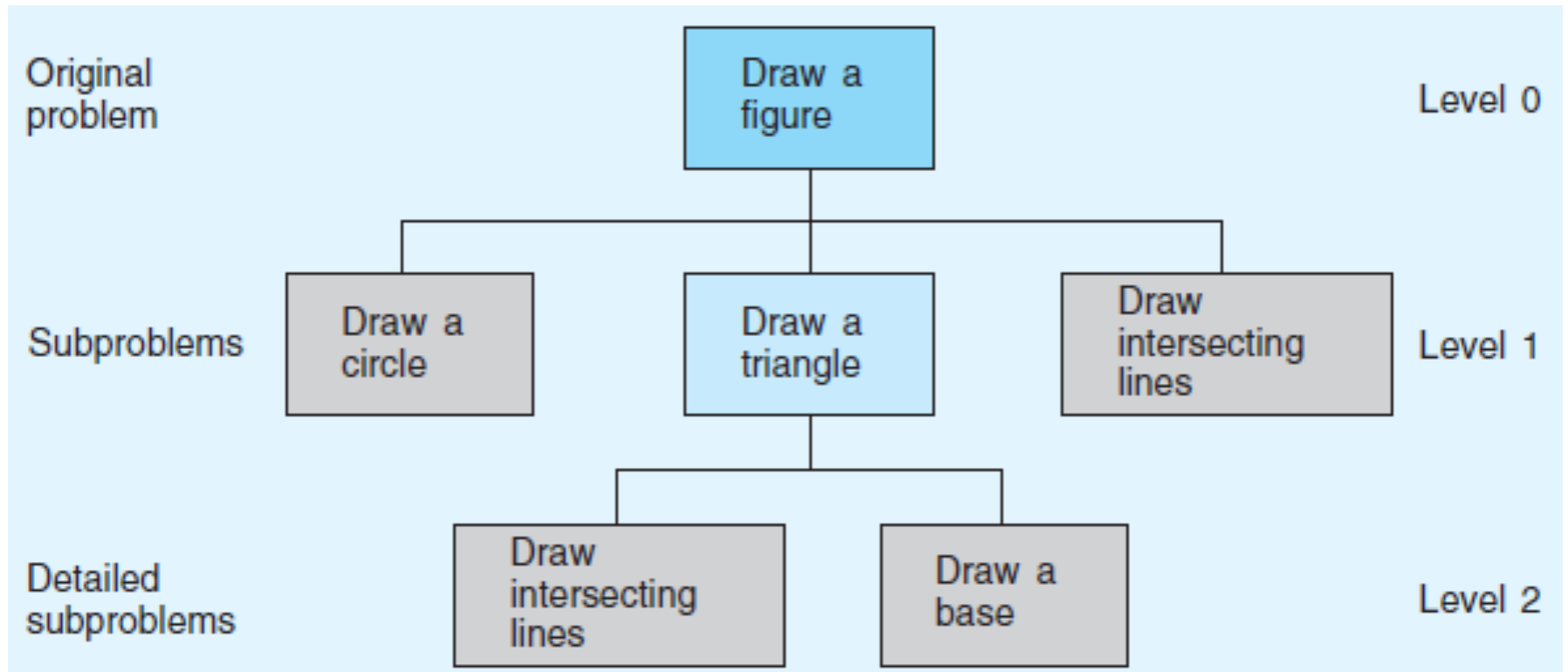


Figure 3.10

Structure Chart for Drawing a Stick Figure



Functions Call Statement (Function Without Arguments)

- Syntax

```
fname();
```

- Example:

```
draw_circle();
```

- Interpretation

- the function fname is called
- after fname has finished execution, the program statement that follows the function call will be executed

Figure 3.11

Function Prototypes and Main Function for Stick Figure

```
1. /*
2.  * Draws a stick figure
3.  */
4.
5. #include <stdio.h>          /* printf definition */
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. }
```

Function Prototype

(Function Without Arguments)

- Syntax

```
ftype fname(void);
```

- Example:

```
void draw_circle(void);
```

- Interpretation

- the identifier `fname` is declared to be the name of a function
- the identifier `ftype` specifies the data type of the function result

Figure 3.12

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 Definitions

(Function Without Arguments)

- Syntax

f_{type}

f_{name}(void)

{

local declarations

executable statements

}

Figure 3.13

Function draw_triangle

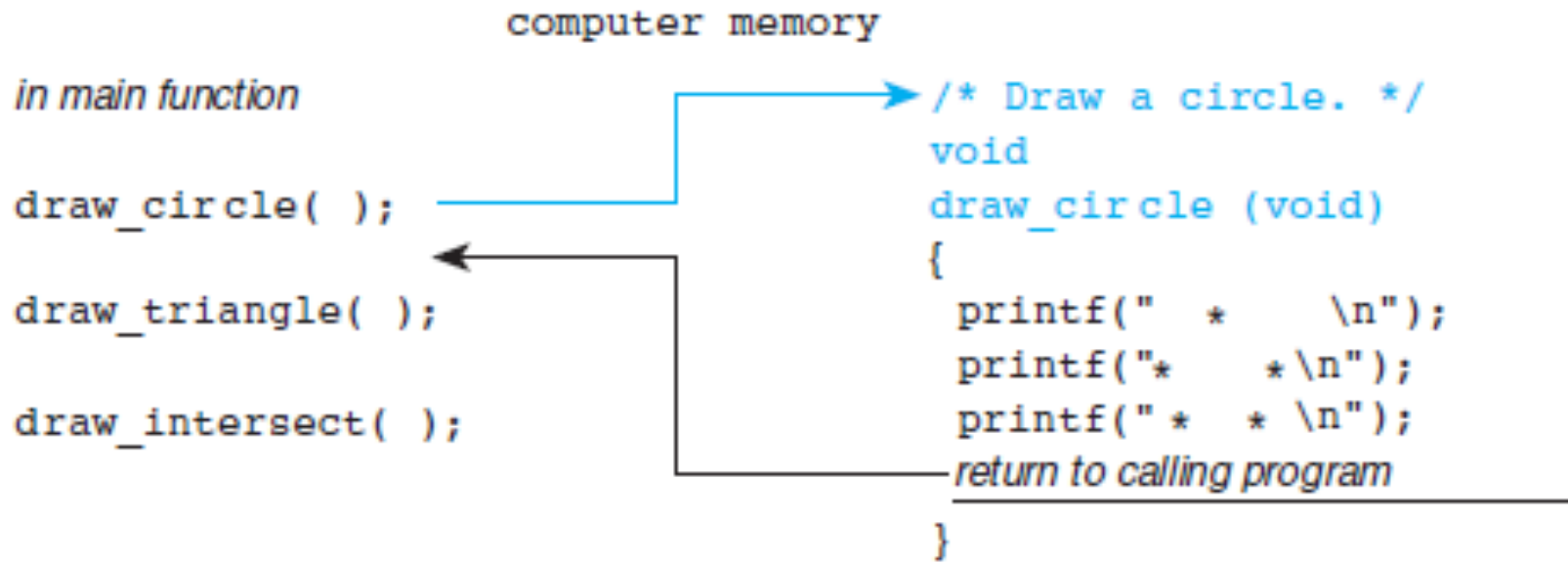
```
1. /*
2.  * Draws a triangle
3.  */
4. void
5. draw_triangle(void)
6. {
7.     draw_intersect();
8.     draw_base();
9. }
```

Advantages of Using Function Subprograms

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

Figure 3.15

Flow of Control Between the main Function and a Function Subprogram



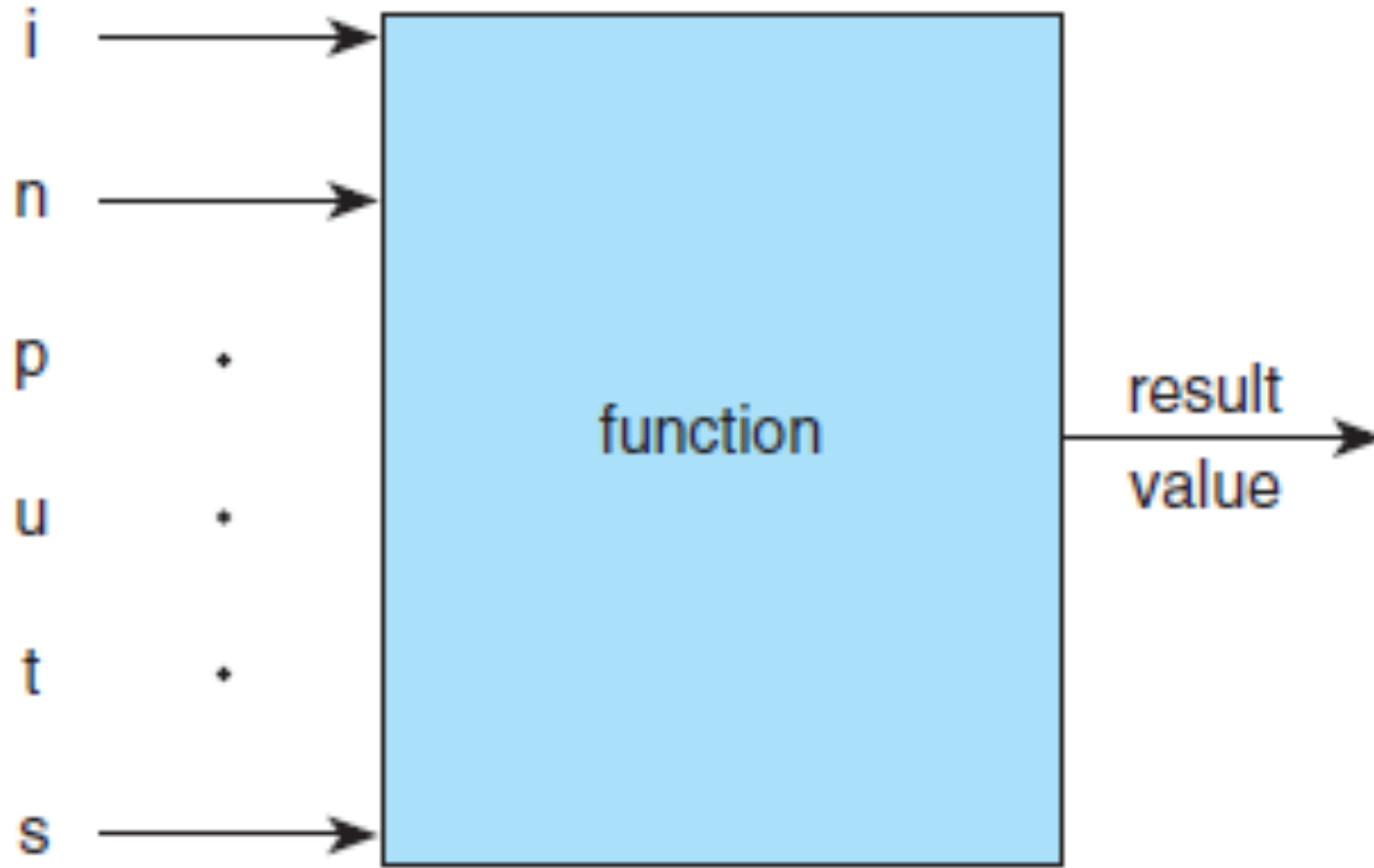
Functions with Input Arguments

- input argument
 - arguments used to pass information into a function subprogram

- output argument
 - arguments used to return results to the calling function

Figure 3.18

Function with Input Arguments and One Result



Functions with Multiple Arguments

Argument List Correspondence

- 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.
- Each actual argument must be of a data type that can be assigned to the corresponding formal parameter with no unexpected loss of information.

Functions with Multiple Arguments

Argument List Correspondence

- The order of arguments in the lists determines correspondence.
 - The first actual argument corresponds to the first formal parameter.
 - The second actual argument corresponds to the second formal parameter.
 - And so on...

Library Functions

- code reuse
 - reusing program fragments that have already been written and tested
- C standard libraries
 - many predefined functions can be found here

stdio.h
math.h

Note: must use `-lm` flag to compile when using math library
For example, `gcc -o exe -Wall my_c_program.c -lm`

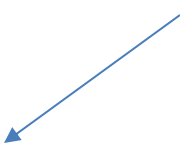
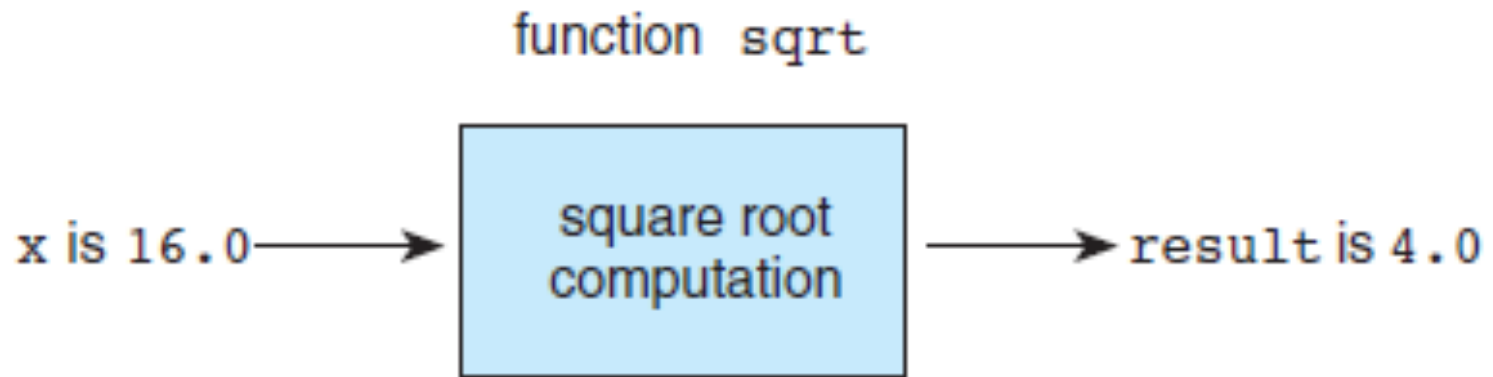


Figure 3.6

Function `sqrt` as a “Black Box”



C Math Library Functions

- Examples
 - `abs(x)`
 - `ceil(x)`
 - `log(x)`
 - `sin(x)`
 - `sqrt(x)`

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. }
```

Wrap Up

- Code reuse is good.
- When possible, develop your solution from existing information.
- Use C's library functions to simplify mathematical computations.
- You can write functions with none, one, or multiple input arguments.
- Functions can only return one value.