Introduction to Computers & Programming

Comp 1330/ First Semester 2024/2025

Instructor: Saif Harbia

Faculty of Engineering and Technology
Department of Computer Science

Chapter 03

Top-Down Design with Functions

3.1 Building Programs from Existing Information

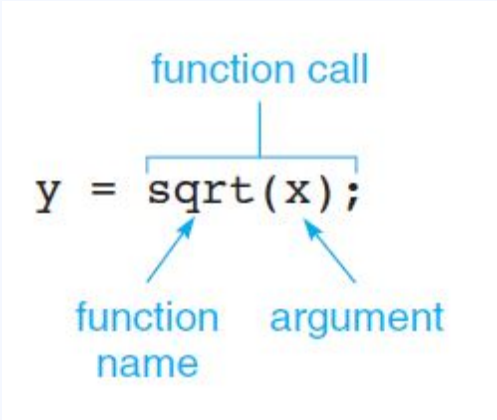
1. *Programmer can use existing information to solve problems by following software development methods.*

2. Another way in which programmers use existing information is by extending the solution for one problem to solve another

3.2 LIBRARY FUNCTIONS

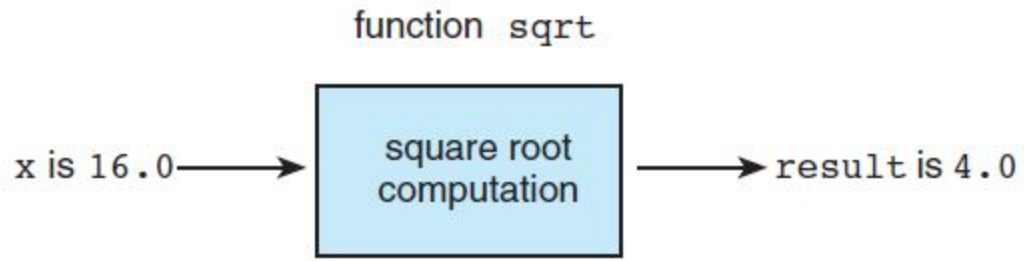
- Predefined Functions and Code Reuse

- C promotes reuse by providing many predefined functions that can be used to perform mathematical computations
- i.e. C's standard math library defines a function named ***sqrt*** that performs the square root computation.



1. `x` is `16.0`, so function `sqrt` computes the $\sqrt{16.0}$ or `4.0`.
2. The function result, `4.0`, is assigned to `y`.

Figure 3.6



If w is 9.0 , the assignment statement

z = 5.7 + sqrt(w);

is evaluated as follows:

1. w is 9.0 , so function ***sqrt*** computes the square root of 9.0 , or 3.0 .
2. The values 5.7 and 3.0 are added together.
3. The sum, 8.7 , is stored in **z** .

EXAMPLE 3.1 P.118

The program in Fig. 3.7, pg. 119 displays 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 ***#include the math library*** *and call the C function* ***sqrt*** *three times.*

- first_sqrt = sqrt(first);
- second_sqrt = sqrt(second);
- sum_sqrt = sqrt(first + second);

- you see that each statement contains a call to a library function **(printf , scanf , sqrt)—**we have used C's predefined functions as building blocks to construct a new program.

C LIBRARY FUNCTIONS RULES

- If one of the functions in **Table 3.1, pg.121** is called with a numeric argument that is not of the argument type listed, the argument value is converted to the required type before it is used.

- Conversions of type **int** to type **double** cause no problems, but a conversion of type **double** to type **int** leads to the loss of any fractional part, just as in a mixed-type assignment.

- The arguments for **log** and **log10** must be positive; the argument for sqrt cannot be negative.

- The arguments for **sin , cos ,** and **tan** must be expressed in radians, not in degrees.

- Example 3.2 p. 121

RE-USING OUR OWN FUNCTIONS

- C also allows us to write our own functions.
- Let's assume that we have already written functions **find_area** and **find_circum** :
  - **Function find_area(r)** returns the area of a circle with radius r .
  - **Function find_circum(r)** returns the circumference of a circle with radius r .
- We can reuse these functions in other programs:

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

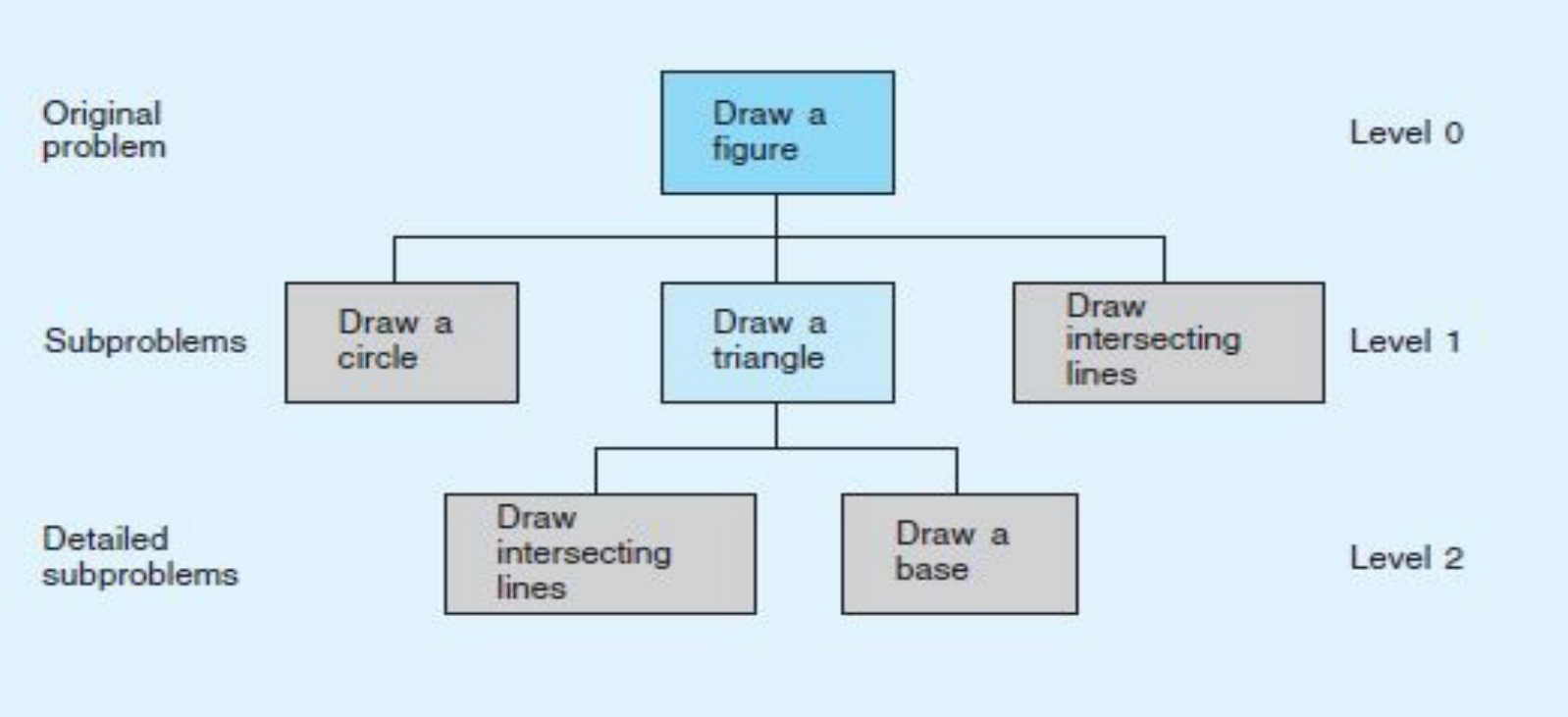
3.3 TOP-DOWN DESIGN AND STRUCTURE

- **Top-down design:**

- In attempting to solve a **subproblem** at one level, we introduce new **subproblems** at lower levels.

- This process proceeds from the original problem at the top level to the **subproblems** at each lower level

CASE STUDY P.124 Drawing Simple Diagrams



3.4 FUNCTIONS WITHOUT ARGUMENTS

- One way that programmers implement top-down design in their programs is by defining their own functions.

- Often, a programmer will write one function **subprogram** for each **subproblem** in the structure chart.

- Use the main function in **Fig. 3.11, pg.127** to draw the stick figure of a person.
- In **Fig. 3.11** , the three algorithm steps are coded as calls to three function subprograms.
- For example, the statement: **draw_circle**(); calls a function (**draw_circle**) that implements the algorithm step Draw a circle

Function Prototypes

- One way to declare a function is to insert a function prototype before the main function.
- A function prototype tells the C compiler the data type of the function, the function name, and information about the arguments that the function expects.
- The functions declared in Fig. 3.11 are void functions (that is, their type is void) because they do not return a value.
- In the function prototype

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

the second void indicates that draw_circle expects no arguments.

Function Definitions

- To specify the function operation, you need to provide a definition for each function subprogram similar to the definition of the main function.

- Any identifiers that are declared in the optional local declarations are defined only during the execution of the function and can be referenced only within the function.

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. }
```

We omit the return statement because draw_circle does not return a result.
Control returns to the main function after the circle shape is displayed.
You can omit the void and write the argument list as () .

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. }
```

Instead of using printf statements to display a triangular pattern, the body of function draw_triangle calls functions draw_intersect and draw_base to draw a triangle.

Placement of Functions in a Program

- Figure 3.14 , pg.130 shows the complete program with function subprograms

- **The subprogram prototypes precede the main function (after any #include or #define directives).**

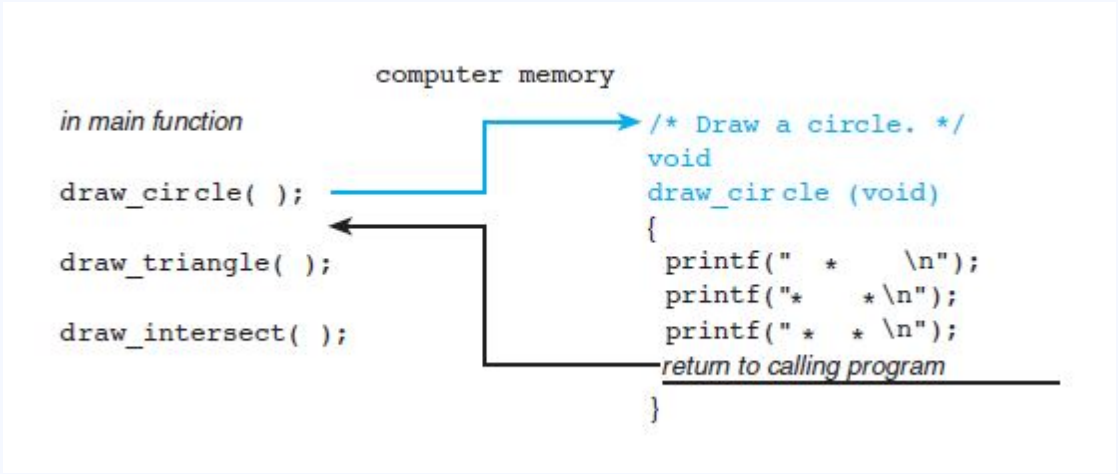
- **The subprogram definitions follow the main function.**

- **The relative order of the function definitions does not affect their order of execution; that is determined by the order of execution of the function call statements.**

Order of Execution of Function Subprograms and Main Function

- The compiler processes the function prototypes before it translates the main function.
- The information in each prototype enables the compiler to correctly translate a call to that function.
- The compiler translates a function call statement as a transfer of control to the function.
- After compiling the main function, the compiler translates each function subprogram.
- During translation, when the compiler reaches the end of a function body, it inserts a machine language statement that causes a transfer of control back from the function to the calling statement

Figure 3.15



- **When we run the program, the first statement in the main function is the first statement executed (the call to draw_circle in Fig. 3.15).**
- **When the computer executes a function call statement, it transfers control to the function that is referenced (indicated by the colored line in Fig. 3.15).**

Advantages of Using Function Subprograms

- Procedural Abstraction

- Function subprograms allow us to remove from the main function the code that provides the detailed solution to a subproblem

- So, we can write the main function as a sequence of function call statements as soon as we have specified the initial algorithm and before we refine any of the steps.

- We should delay writing the function for an algorithm step until we have finished refining that step

Advantages of Using Function Subprograms

- Reuse of Function Subprograms

- Once you have written and tested a function, you can use it in other programs or functions.

- For example, function draw_intersect is called twice in Fig. 3.14 (once by draw_triangle and once by the main function)

3.5 FUNCTIONS WITH INPUT ARGUMENTS

- The arguments of a function are used to carry information into the function subprogram from the main function (or from another function subprogram) or to return multiple results computed by a function subprogram.
- Arguments that carry information into the function subprogram are called **input arguments** ; arguments that return results are called **output arguments** (CH06)**.**
- We can also return a single result from a function by executing a return statement in the function body.
- Arguments make function subprograms more versatile because they enable a function to manipulate different data each time it is called

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

.

void Functions with Input Arguments

- **void** functions do not return a result

FIGURE 3.18 Function print_rboxed and Sample Run

```
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. }
```

FIGURE 3.18

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

void Functions with Input Arguments

- the effect of the function call **print_rboxed(135.68);**

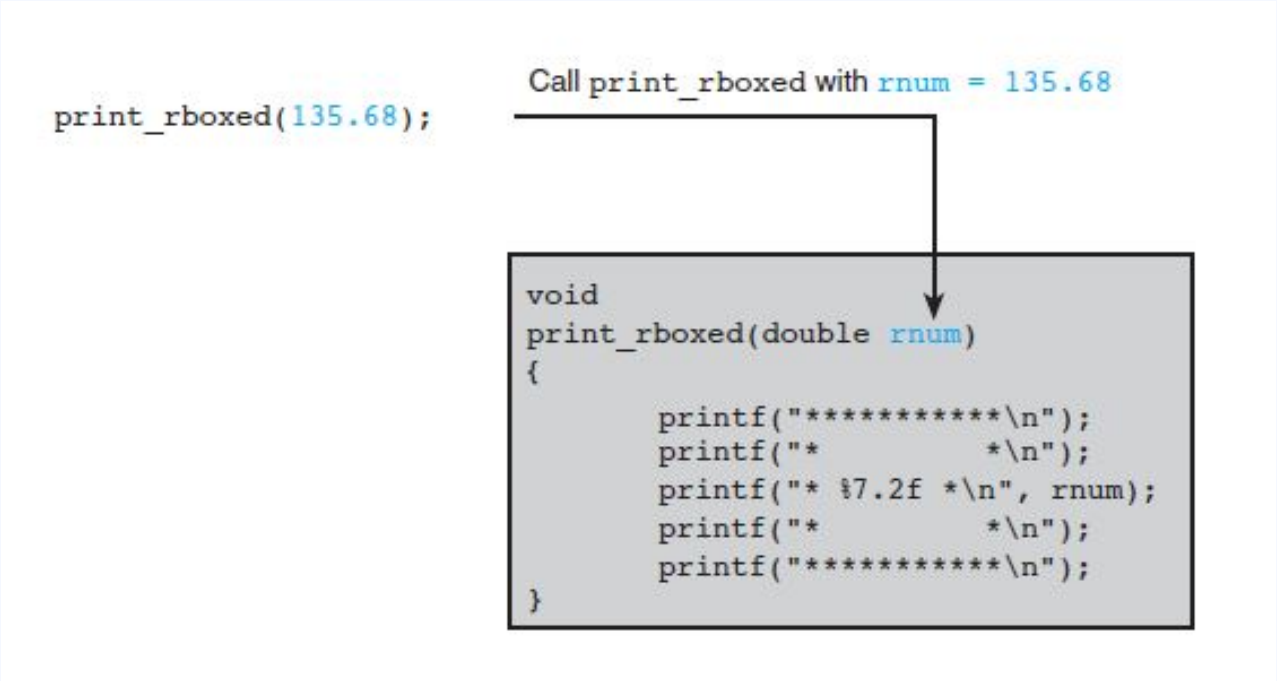


Figure 3.19

Functions with Input Arguments and a Single Result

If function f that has 2 type double inputs we can reference it in an expression such as **w + f(2.5, 4.0)**

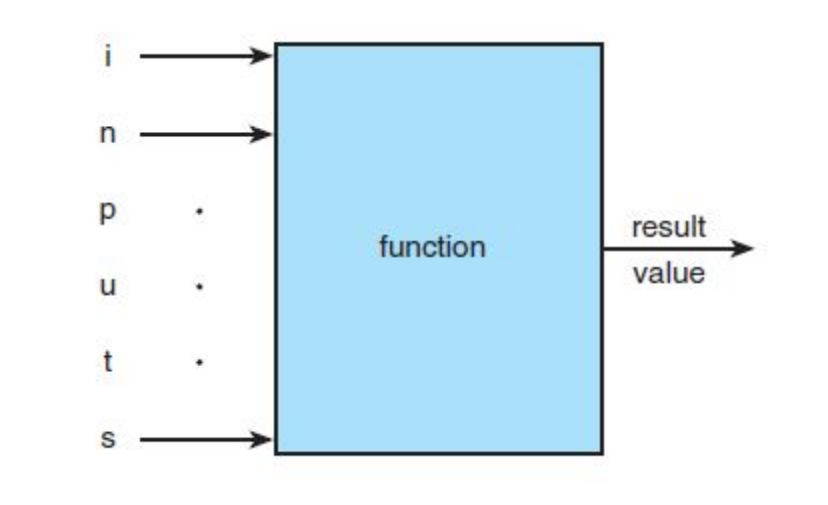


Figure 3.20

Functions with Input Arguments and a Single Result

```
radius = 10.0;
circum = find_circum(radius);
```

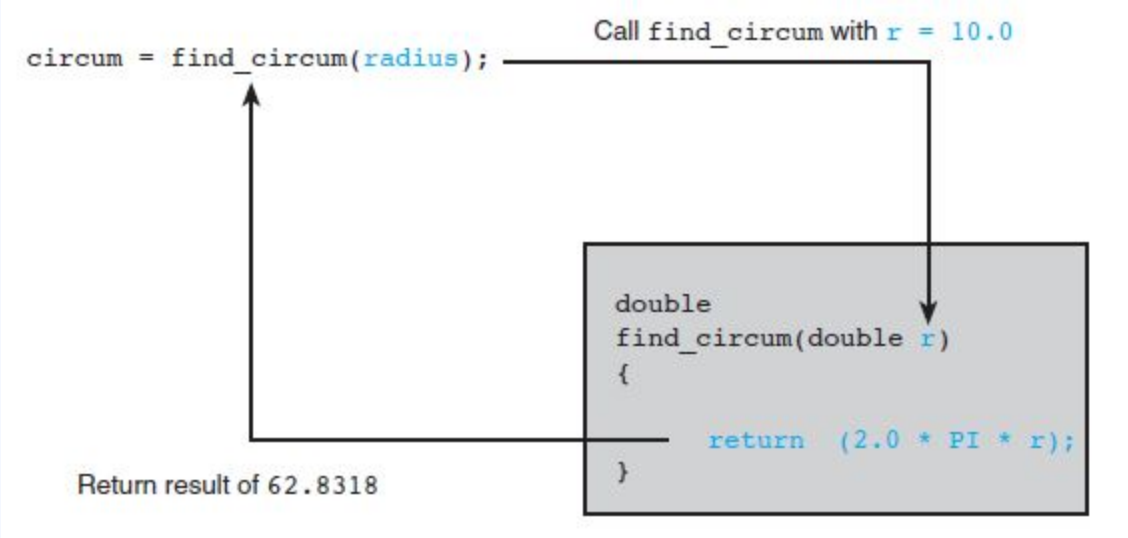


Figure 3.22

Functions with Multiple Arguments

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. }
```

Argument List Correspondence (number,order,and type(not))

- 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 determine correspondence. The first actual argument corresponds to the first formal parameter, the second actual argument corresponds to the second formal parameter, and so on.
- Each actual argument must be of a data type that can be assigned to the corresponding formal parameter with no unexpected loss of information.

The Function Data Area

- Each time a function call is executed, an area of memory is allocated for storage of that function's data.

- Included in the function data area are storage cells for its formal parameters and any local variables that may be declared in the function.

- The function data area is always lost when the function terminates; it is recreated empty (all values undefined) when the function is called again.

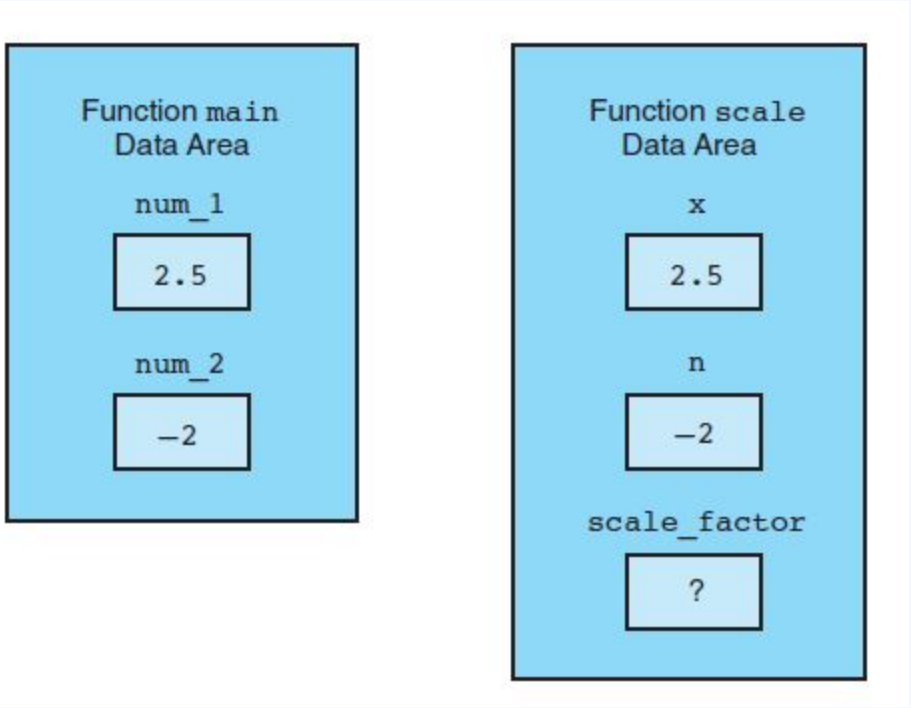


Figure 3.25

Refernces

Problem Solving and Program Design in C, 7th Ed., by Jeri R. Hanly and Elliot B. Koffman