
C Programming for Engineers

Functions



UNIVERSITY
AT ALBANY

State University of New York

ICEN 360– Spring 2017

Prof. Dola Saha



UNIVERSITY AT ALBANY
State University of New York

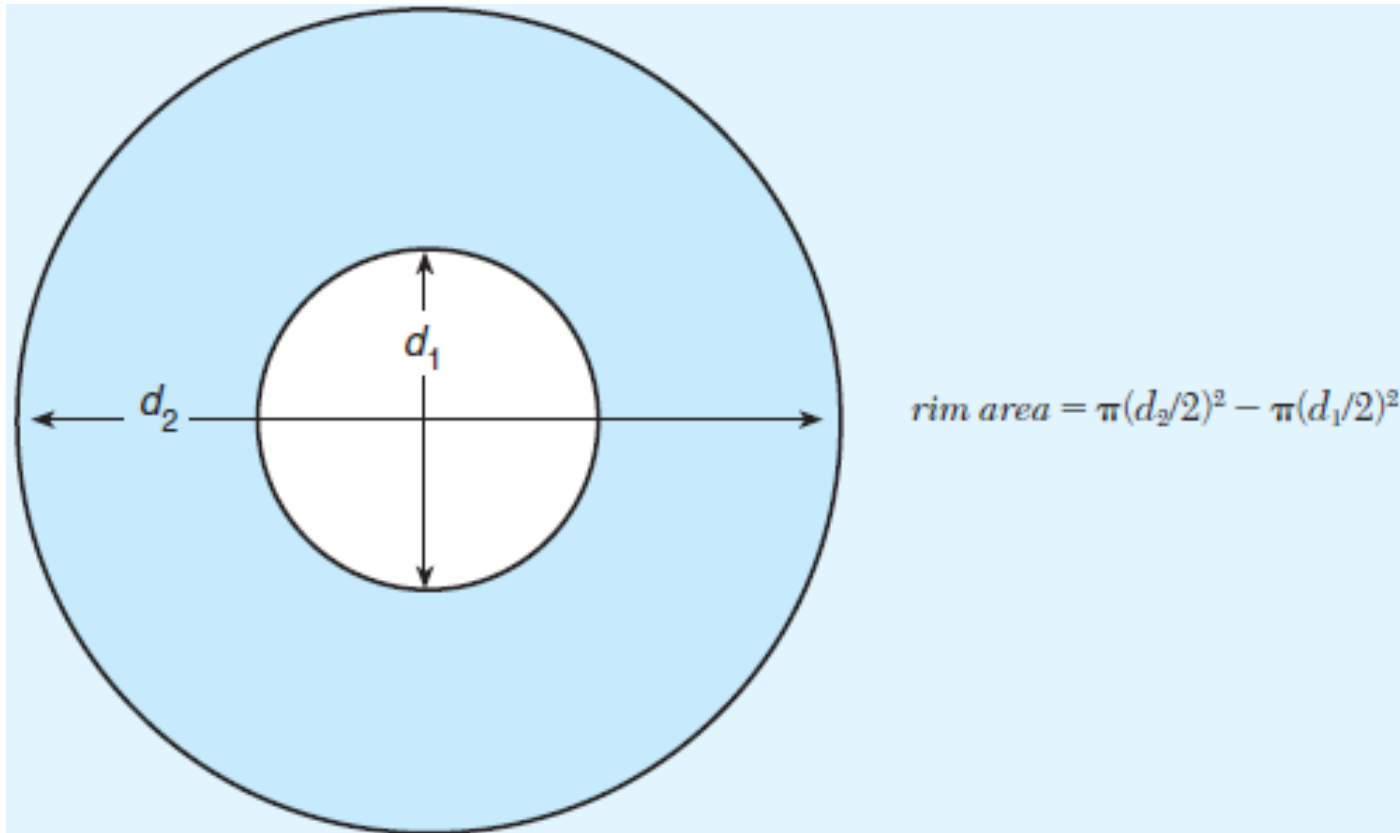
Introduction

- Real world problems are larger, more complex
- Top down approach
- Modularize – divide and control
- Easier to track smaller problems / modules
- Repeated set of statements

Example: Area and circumference of a circle

```
1. /*
2.  * Calculates and displays the area and circumference of a circle
3.  */
4.
5. #include <stdio.h> /* printf, scanf definitions */
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.     printf("Enter radius> ");
17.     scanf("%lf", &radius);
18.
19.     /* Calculate the area */
20.     area = PI * radius * radius;
21.
22.     /* Calculate the circumference */
23.     circum = 2 * PI * radius;
24.
25.     /* Display the area and circumference */
26.     printf("The area is %.4f\n", area);
27.     printf("The circumference is %.4f\n", circum);
28.
29.     return (0);
30. }
```

Computing Rim Area of a Flat Washer



C Code (1)

```
1.  /*
2.   * Computes the weight of a batch of flat washers.
3.   */
4.
5.  #include <stdio.h> /* printf, scanf definitions */
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;
```

C Code (2)

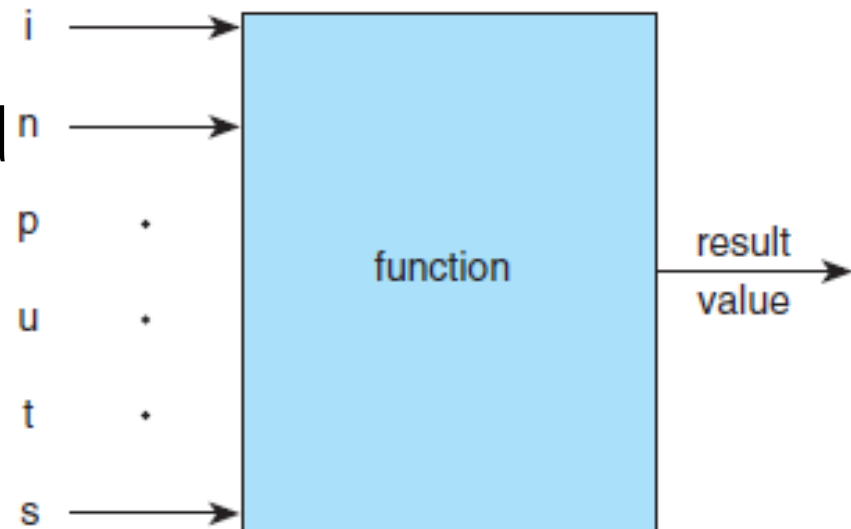
```
39.     rim_area = PI * edge_radius * edge_radius -
40.           PI * hole_radius * hole_radius;
41.
42.     /* Compute the weight of a flat washer. */
43.     unit_weight = rim_area * thickness * density;
44.     /* Compute the weight of the batch of washers. */
45.     weight = unit_weight * quantity;
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.
```

Functions

- Functions allow us to
 - modularize a program
 - reuse the code
- Two types:
 - Programmer/user write, called *programmer-defined* functions
 - *prepackaged* functions available in the C standard library.
- Input Variables
- Output value, which is returned
- Function body

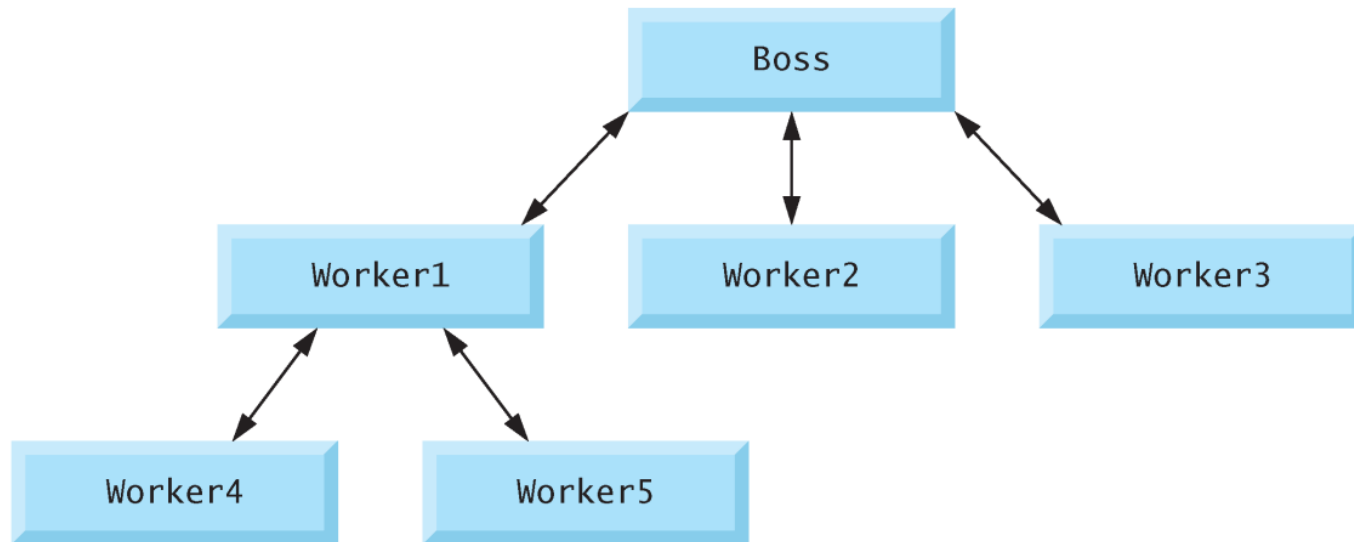


Function

- The statements defining the function are written only once, and the statements are hidden from other functions.
- Functions are **invoked** by a **function call**, which specifies the function name and provides information (as **arguments**) that the called function needs to perform its designated task.

Modularizing Program

- Analogy : Hierarchical management
 - A boss (the **calling function** or **caller**) asks a worker (the **called function**) to perform a task and report back when the task is done
-



Function

- All variables defined in function definitions are **local variables**—they can be accessed *only* in the function in which they're defined.
- Most functions have a list of **parameters** that provide the means for communicating information between functions.
- A function's parameters are also local variables of that function.
- The format of a function definition is

```
return-value-type function-name(parameter-list)  
{  
    definitions  
    statements  
}
```

Example of User-defined Function

```
1 // Fig. 5.3: fig05_03.c
2 // Creating and using a programmer-defined function.
3 #include <stdio.h>
4
5 int square(int y); // function prototype
6
7 int main(void)
8 {
9     // loop 10 times and calculate and output square of x each time
10    for (int x = 1; x <= 10; ++x) {
11        printf("%d ", square(x)); // function call
12    }
13
14    puts("");
15 }
16
17 // square function definition returns the square of its parameter
18 int square(int y) // y is a copy of the argument to the function
19 {
20     return y * y; // returns the square of y as an int
21 }
```

1 4 9 16 25 36 49 64 81 100

Function Definition

- Function square is **invoked** or **called** in main within the printf statement

```
printf("%d ", square(x)); // function call
```

- Function square receives a *copy* of the value of x in the parameter y.
- Then square calculates $y * y$.
- The result is passed back returned to function printf in main where square was invoked, and printf displays the result.
- This process is repeated 10 times using the for statement.

Function Definition... cont.

- The definition of function `square` shows that `square` expects an integer parameter `y`.
- The keyword `int` preceding the function name indicates that `square` *returns* an integer result.
- The `return statement` in `square` passes the value of the expression `y * y` (that is, the result of the calculation) back to the calling function.
- `int square(int y); // function prototype`
 - The `int` in parentheses informs the compiler that `square` expects to *receive* an integer value from the caller.
 - The `int` to the left of the function name `square` informs the compiler that `square` returns an integer result to the caller.

Function Definition... cont.

- The compiler refers to the function prototype to check that any calls to `square` contain
 - the *correct return type*
 - the *correct number of arguments*
 - the *correct argument types*
 - the *arguments are in the correct order*
- The *function-name* is any valid identifier.
- The *return-value-type* is the data type of the result returned to the caller.
- The *return-value-type* `void` indicates that a function does not return a value.
- Together, the *return-value-type*, *function-name* and *parameter-list* are sometimes referred to as the function *header*.

Function Definition... cont.

- The *parameter-list* is a comma-separated list that specifies the parameters received by the function when it's called.
- If a function does not receive any values, *parameter-list* is void.
- A type must be listed explicitly for each parameter.
- The *definitions* and *statements* within braces form the *function body*, which is also referred to as a *block*.
- Variables can be declared in any block, and blocks can be nested.

Return Control

- Returns control to calling function after function execution
 - the function does *not* return a result, control returns immediately after the execution of function body
 - Returns after executing the statement `return;`
 - Returns the value of the expression to the caller by the statement - `return expression;`

main() 's Return Type

- `main` has an `int` return type.
- The return value of `main` is used to indicate whether the program executed correctly.
- In earlier versions of C, we had to explicitly place
`return 0;`
- at the end of `main`—0 indicates that a program ran successfully.
- `main` implicitly returns 0 if we omit the return statement.
- We can explicitly return non-zero values from `main` to indicate that a problem occurred during your program's execution.

Function Example: maximum()

```
1 // Fig. 5.4: fig05_04.c
2 // Finding the maximum of three integers.
3 #include <stdio.h>
4
5 int maximum(int x, int y, int z); // function prototype
6
7 int main(void)
8 {
9     int number1; // first integer entered by the user
10    int number2; // second integer entered by the user
11    int number3; // third integer entered by the user
12
13    printf("%s", "Enter three integers: ");
14    scanf("%d%d%d", &number1, &number2, &number3);
15
16    // number1, number2 and number3 are arguments
17    // to the maximum function call
18    printf("Maximum is: %d\n", maximum(number1, number2, number3));
19 }
20
```



Function Example: maximum()

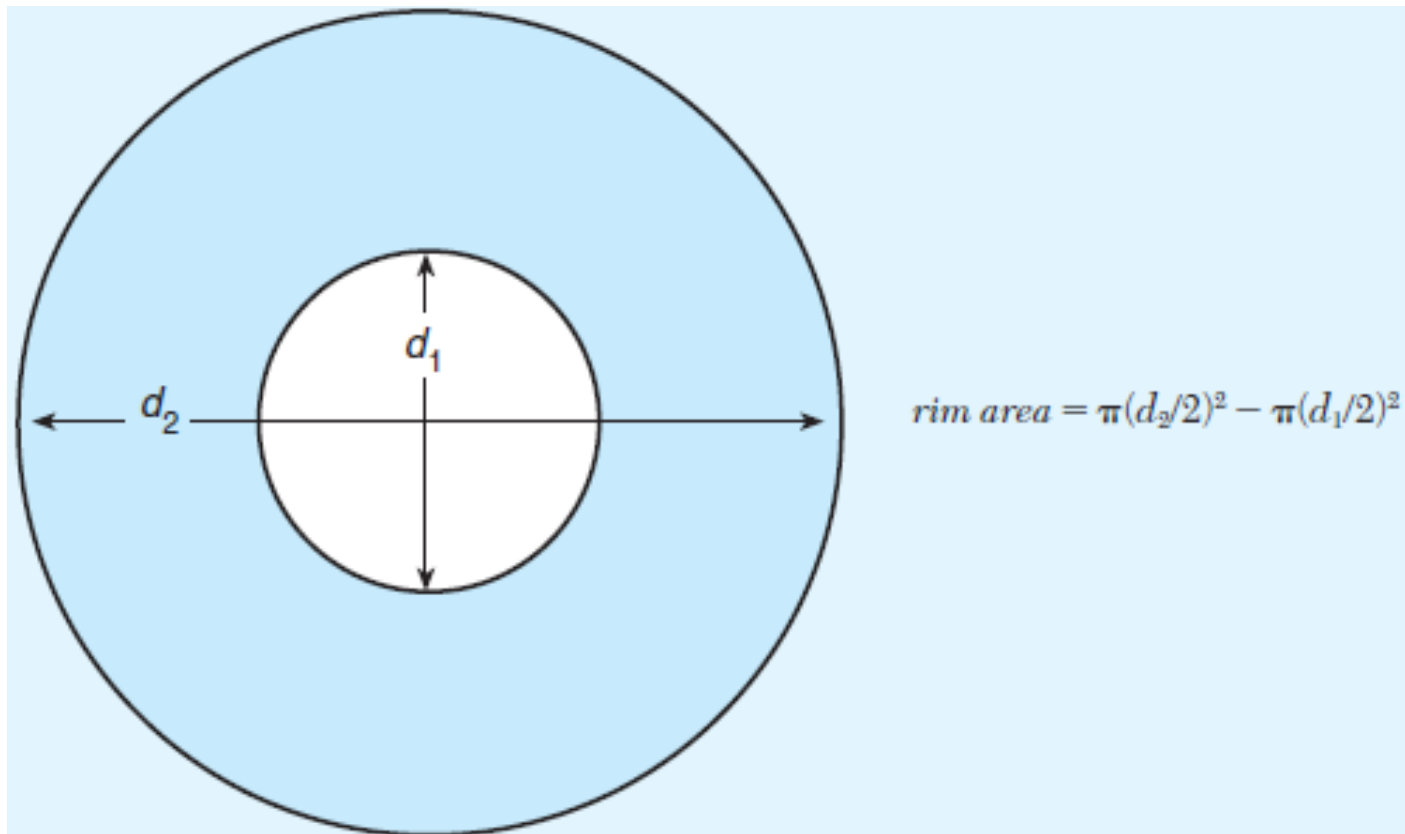
```
21 // Function maximum definition
22 // x, y and z are parameters
23 int maximum(int x, int y, int z)
24 {
25     int max = x; // assume x is largest
26
27     if (y > max) { // if y is larger than max,
28         max = y; // assign y to max
29     }
30
31     if (z > max) { // if z is larger than max,
32         max = z; // assign z to max
33     }
34
35     return max; // max is largest value
36 }
```

Enter three integers: 22 85 17
Maximum is: 85



Classwork Assignment

- Compute rim area of a flat washer using function to calculate the area.



Math Library Functions

➤ Performs common mathematical calculations.

Function	Description	Example
<code>sqrt(x)</code>	square root of x	<code>sqrt(900.0)</code> is 30.0 <code>sqrt(9.0)</code> is 3.0
<code>cbrt(x)</code>	cube root of x (C99 and C11 only)	<code>cbrt(27.0)</code> is 3.0 <code>cbrt(-8.0)</code> is -2.0
<code>exp(x)</code>	exponential function e^x	<code>exp(1.0)</code> is 2.718282 <code>exp(2.0)</code> is 7.389056
<code>log(x)</code>	natural logarithm of x (base e)	<code>log(2.718282)</code> is 1.0 <code>log(7.389056)</code> is 2.0
<code>log10(x)</code>	logarithm of x (base 10)	<code>log10(1.0)</code> is 0.0 <code>log10(10.0)</code> is 1.0 <code>log10(100.0)</code> is 2.0
<code>fabs(x)</code>	absolute value of x as a floating-point number	<code>fabs(13.5)</code> is 13.5 <code>fabs(0.0)</code> is 0.0 <code>fabs(-13.5)</code> is 13.5
<code>ceil(x)</code>	rounds x to the smallest integer not less than x	<code>ceil(9.2)</code> is 10.0 <code>ceil(-9.8)</code> is -9.0

More Math Library Functions

➤ `#include <math.h>`

Function	Description	Example
<code>floor(x)</code>	rounds x to the largest integer not greater than x	<code>floor(9.2)</code> is 9.0 <code>floor(-9.8)</code> is -10.0
<code>pow(x, y)</code>	x raised to power y (x^y)	<code>pow(2, 7)</code> is 128.0 <code>pow(9, .5)</code> is 3.0
<code>fmod(x, y)</code>	remainder of x/y as a floating-point number	<code>fmod(13.657, 2.333)</code> is 1.992
<code>sin(x)</code>	trigonometric sine of x (x in radians)	<code>sin(0.0)</code> is 0.0
<code>cos(x)</code>	trigonometric cosine of x (x in radians)	<code>cos(0.0)</code> is 1.0
<code>tan(x)</code>	trigonometric tangent of x (x in radians)	<code>tan(0.0)</code> is 0.0

Self Review Assignment

- Modify the program to compute the rim area of a flat washer using two functions
 - To calculate the area
 - To compute the square

