Programming for Engineers

Functions

UNIVERSITY AT ALBANY
State University of New York

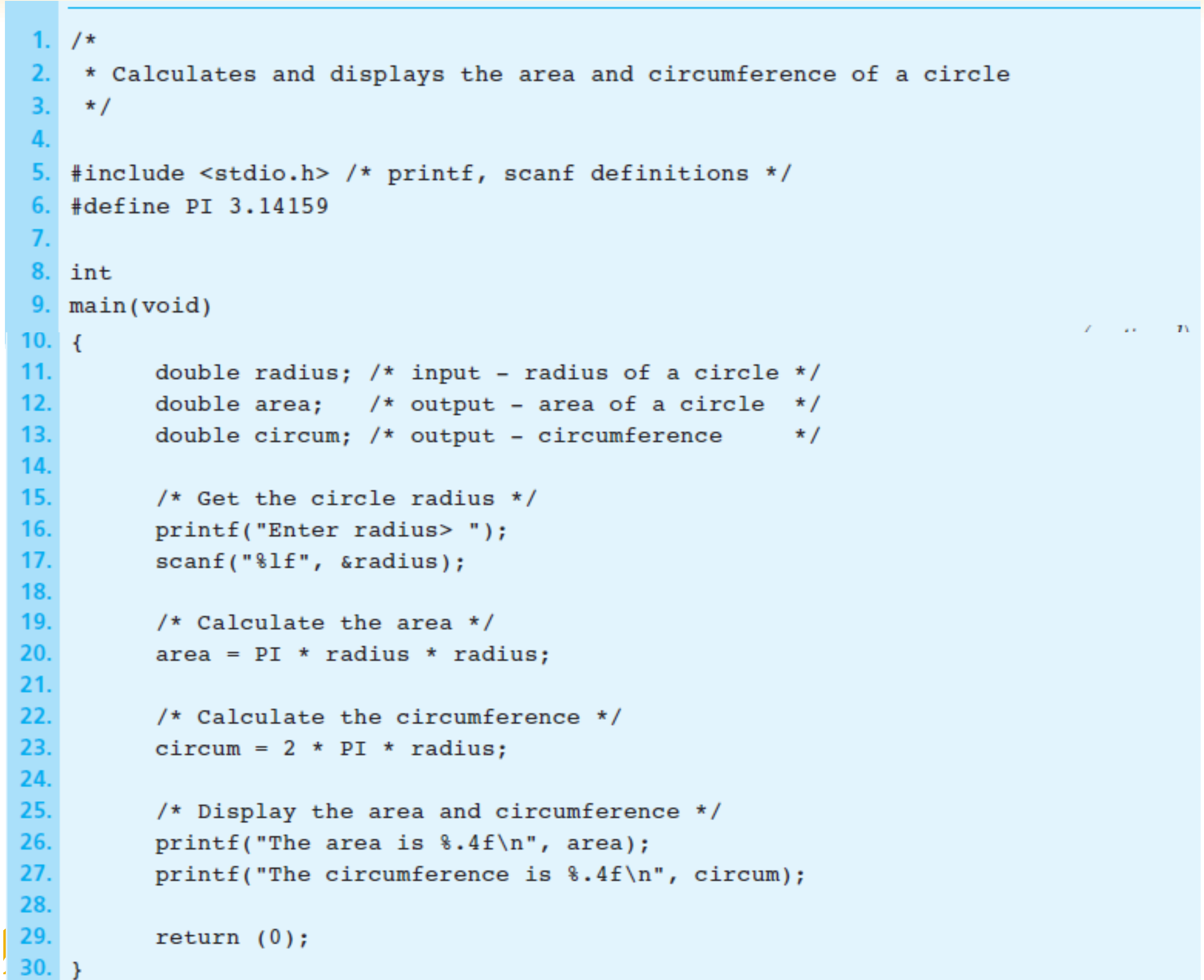
ICEN 200– Spring 2018

Prof. Dola Saha

Introduction

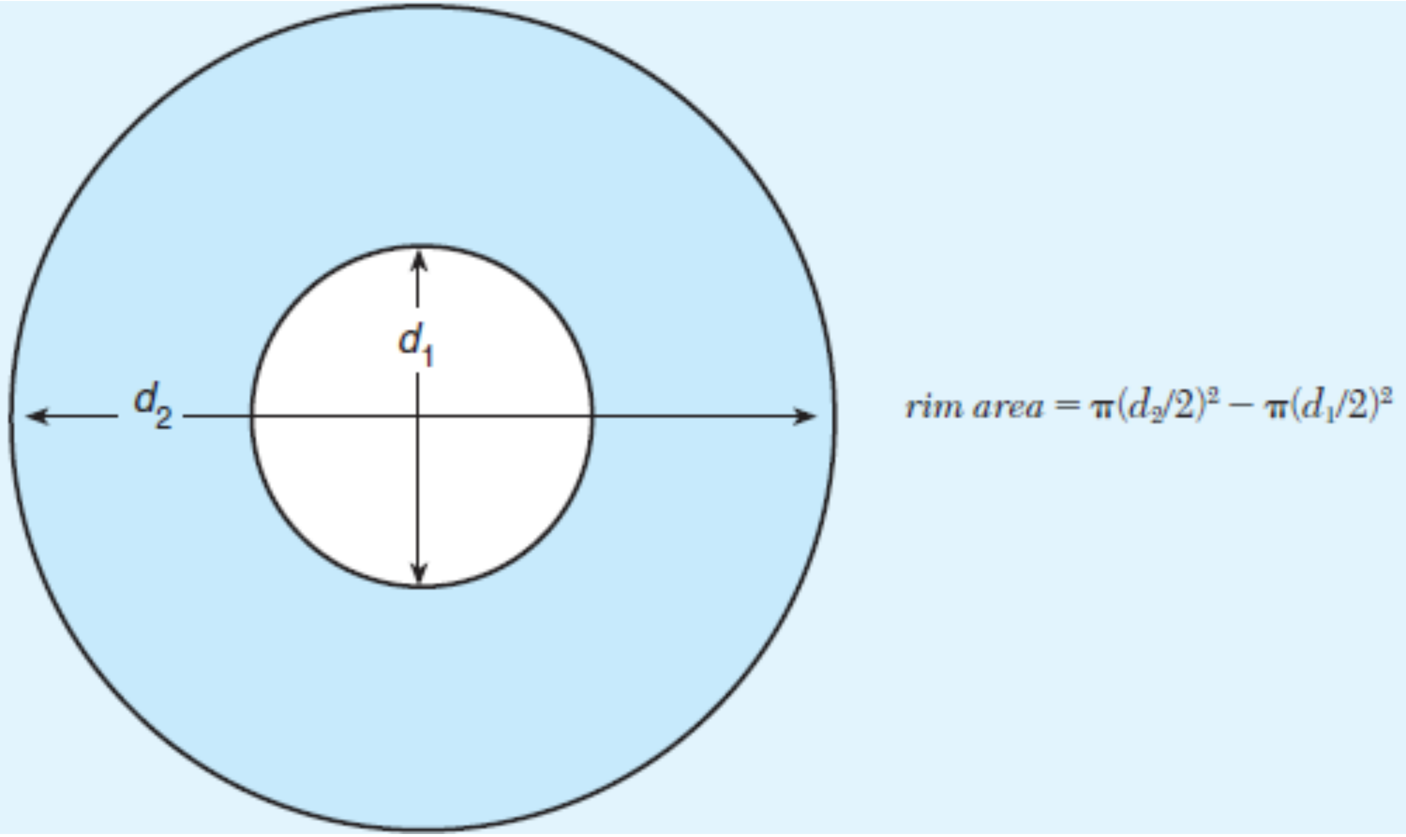
- Real world problems are larger, more complex
- Top down approach
- Modularize – divide and control
- Easier to track smaller problems / modules
- Repeated set of statements

Example: Area and circumference of a circle



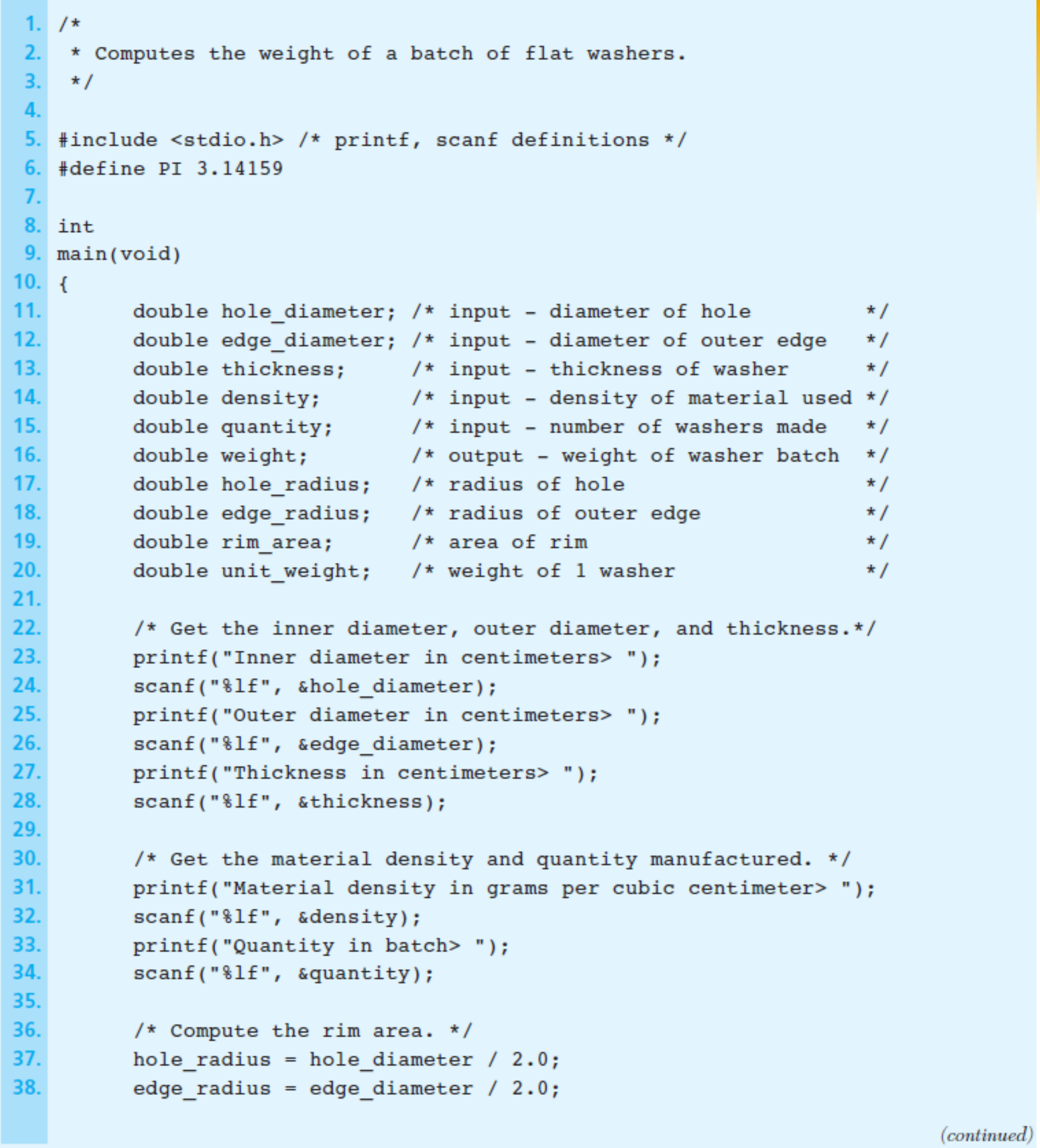
```c
/*
 * Calculates and displays the area and circumference of a circle
 */

#include <stdio.h> /* printf, scanf definitions */
#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);
}
```

Computing Rim Area of a Flat Washer



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

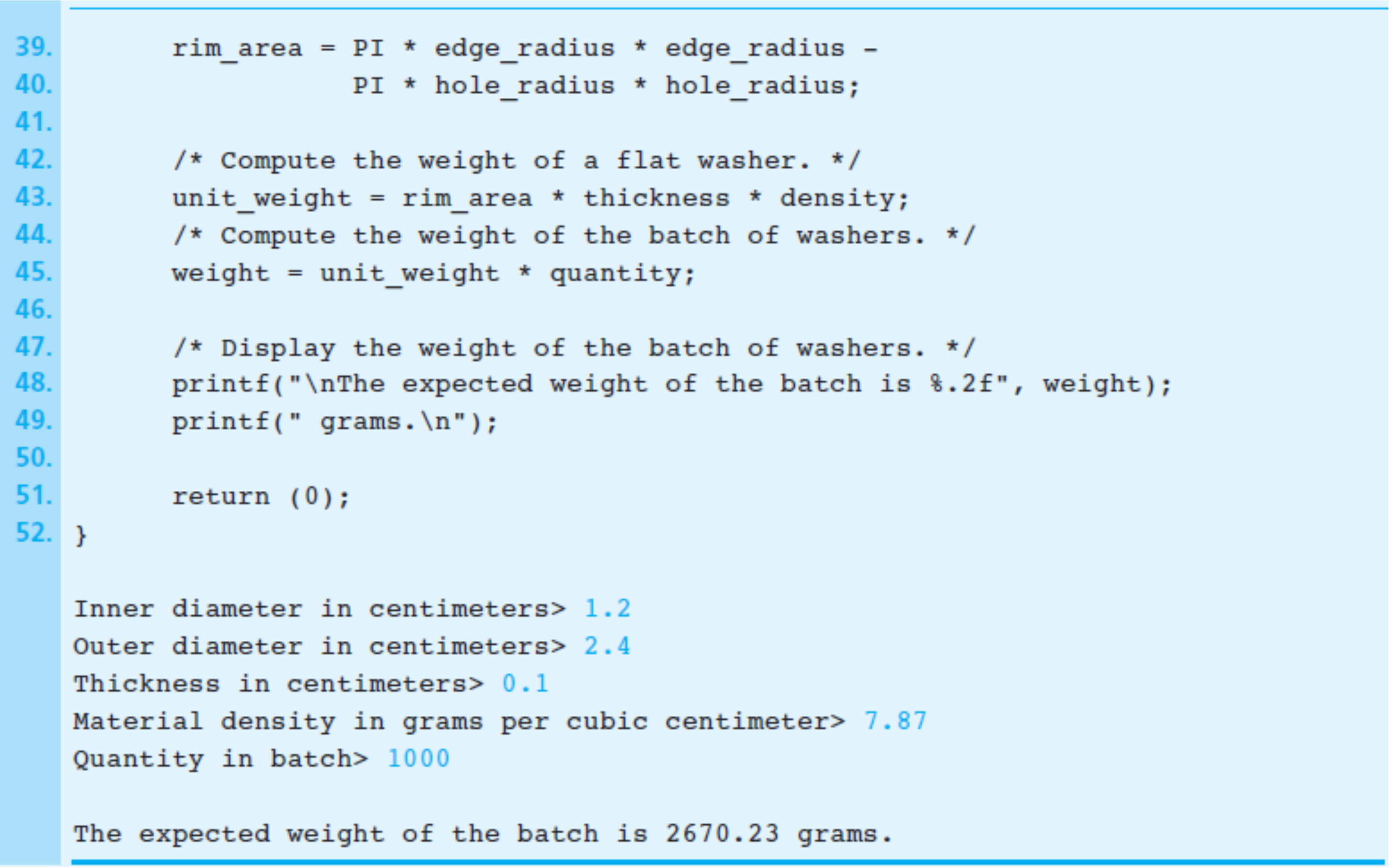
C Code (1)



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

#include <stdio.h> /* printf, scanf definitions */
#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;
```

(continued)

C Code (2)

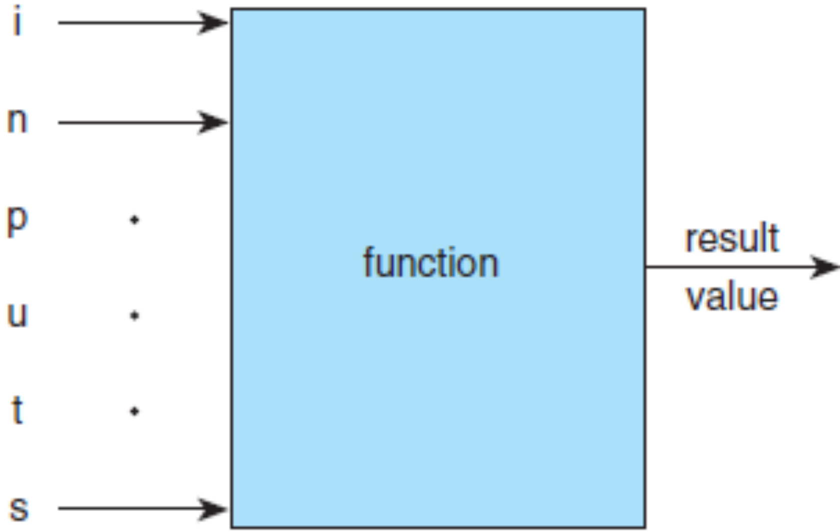


```
      rim_area = PI * edge_radius * edge_radius -
                 PI * hole_radius * hole_radius;

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

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

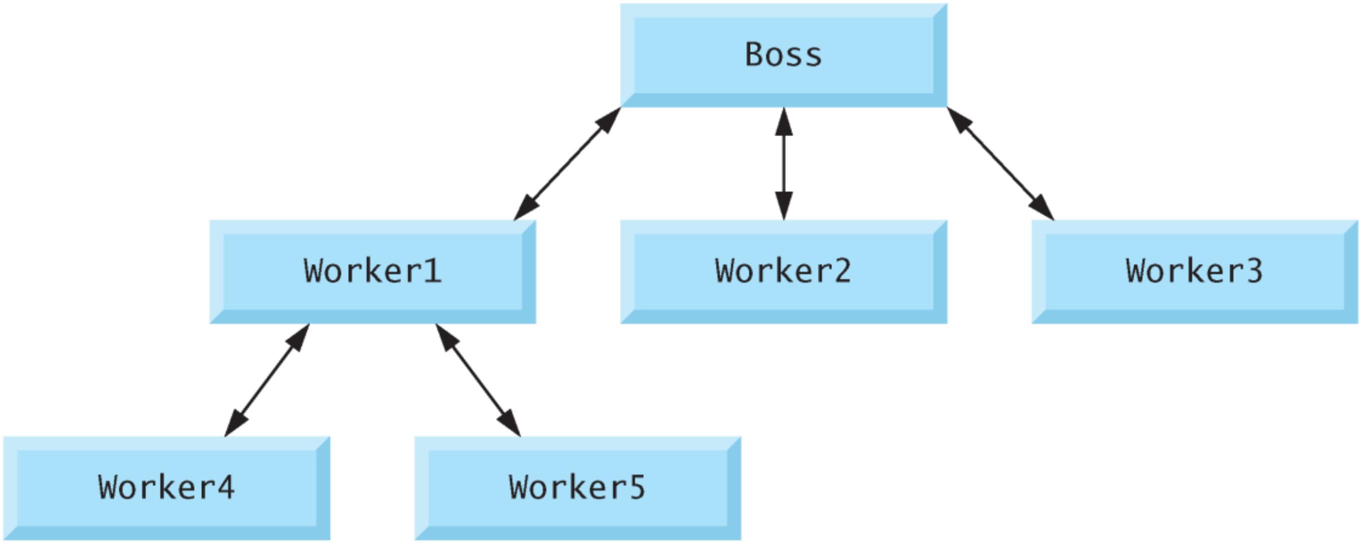


Function

- 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

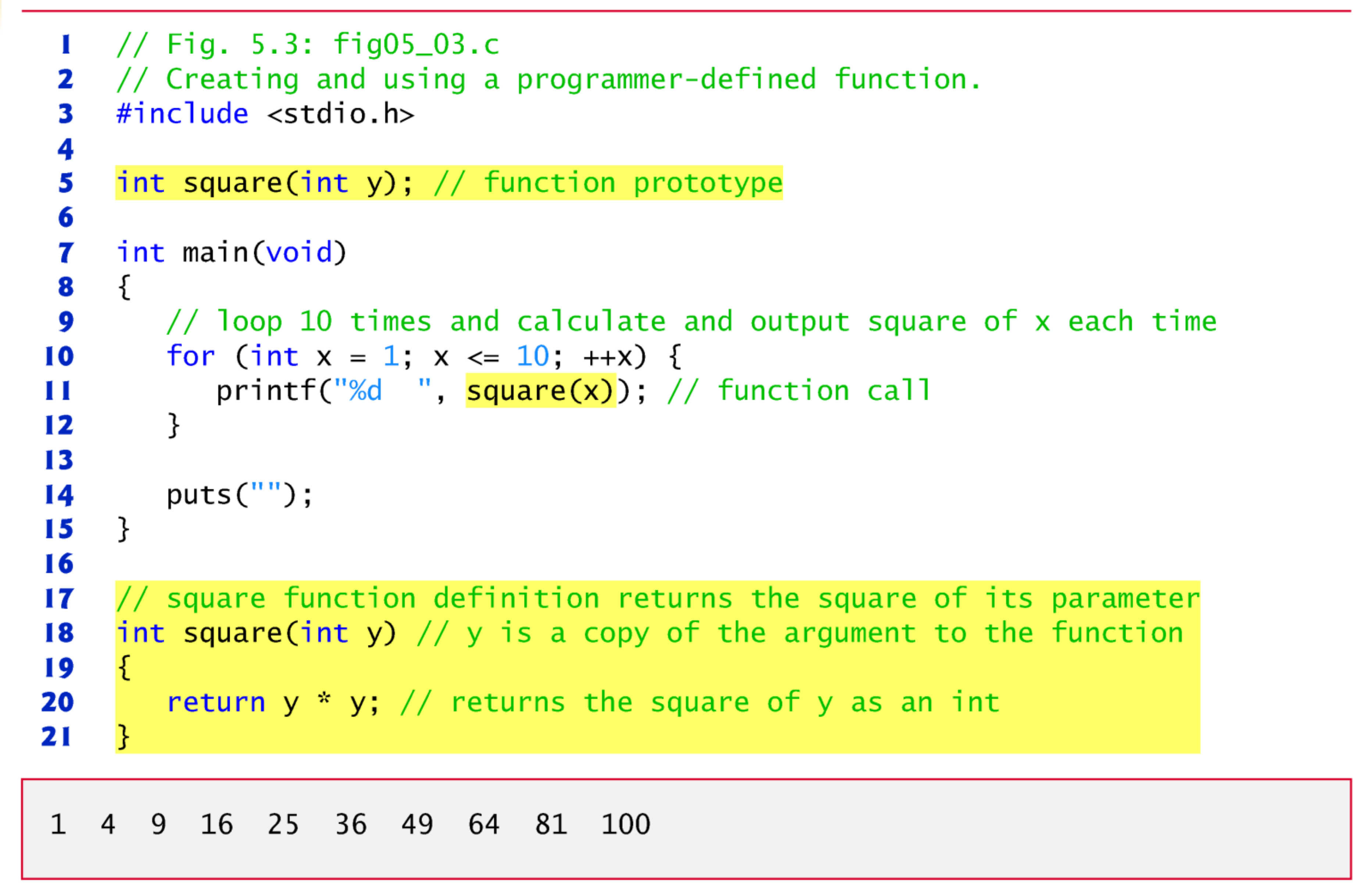


Function

- 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



```c
// Fig. 5.3: fig05_03.c
// Creating and using a programmer-defined function.
#include <stdio.h>

int square(int y); // function prototype

int main(void)
{
   // loop 10 times and calculate and output square of x each time
   for (int x = 1; x <= 10; ++x) {
      printf("%d  ", square(x)); // function call
   }

   puts("");
}

// square function definition returns the square of its parameter
int square(int y) // y is a copy of the argument to the function
{
   return y * y; // returns the square of y as an int
}
```

```
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;*

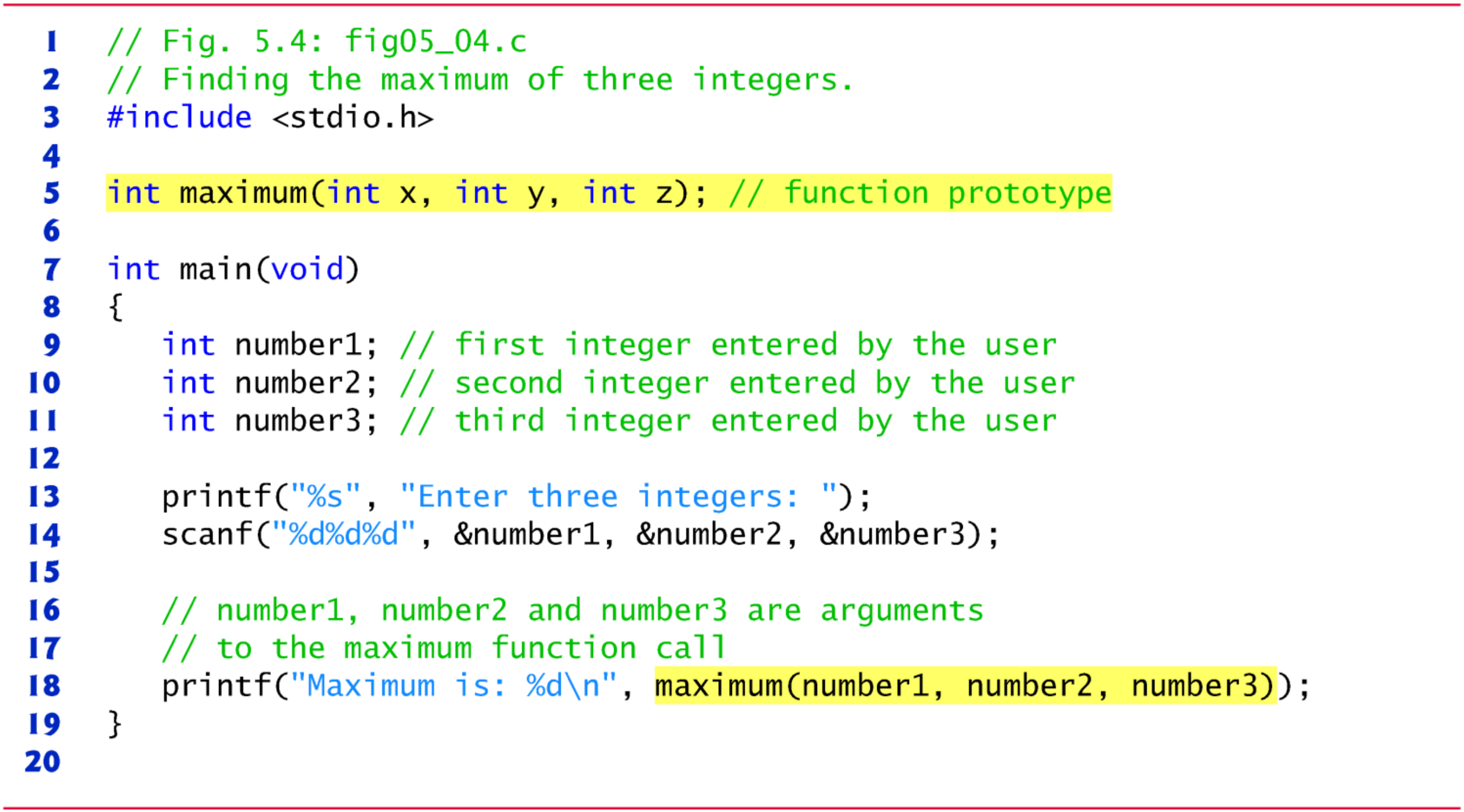
`main()`’s Return Type

- `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()



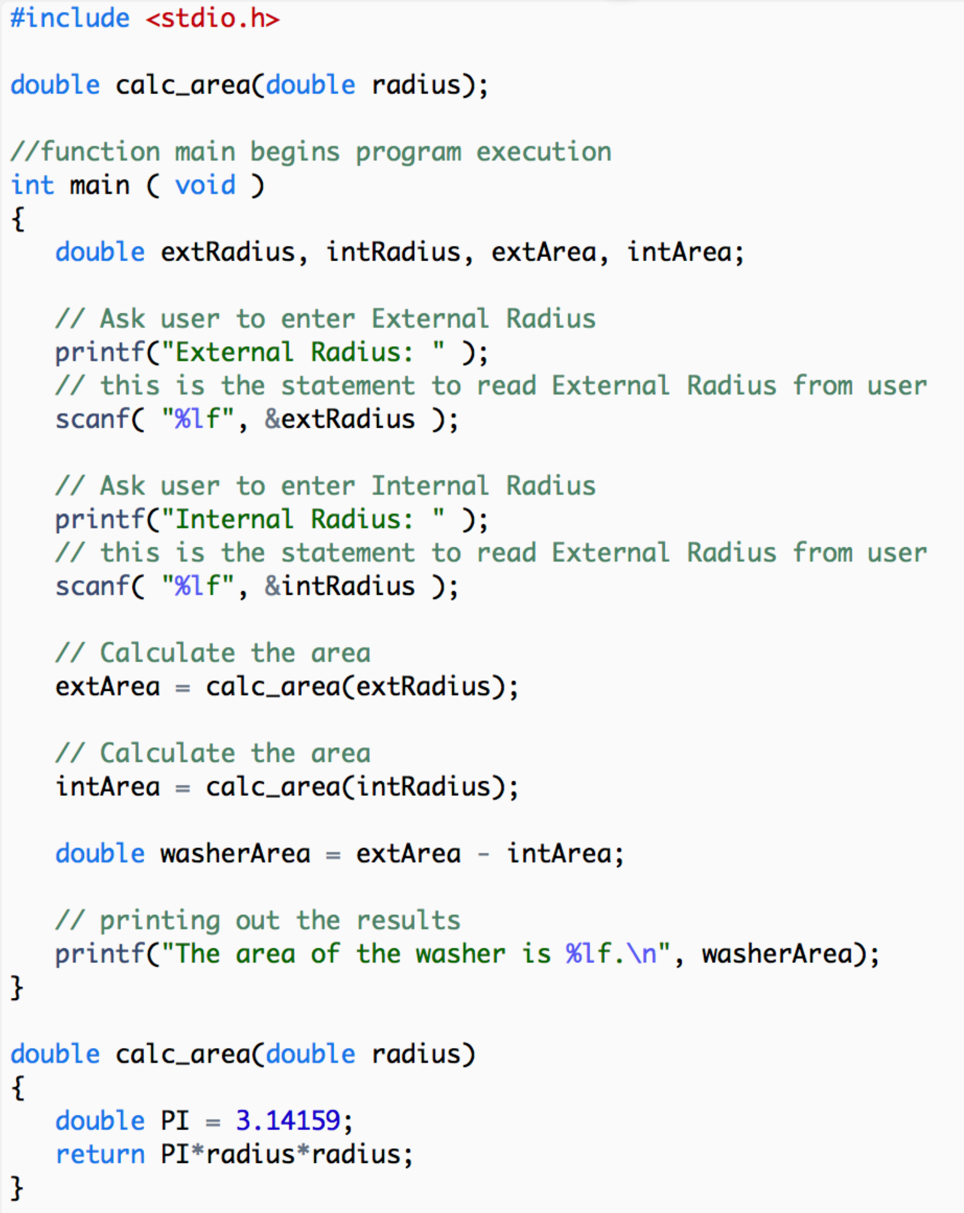
```c
// Fig. 5.4: fig05_04.c
// Finding the maximum of three integers.
#include <stdio.h>

int maximum(int x, int y, int z); // function prototype

int main(void)
{
   int number1; // first integer entered by the user
   int number2; // second integer entered by the user
   int number3; // third integer entered by the user

   printf("%s", "Enter three integers: ");
   scanf("%d%d%d", &number1, &number2, &number3);

   // number1, number2 and number3 are arguments
   // to the maximum function call
   printf("Maximum is: %d\n", maximum(number1, number2, number3));
}

```

Function Example: maximum()

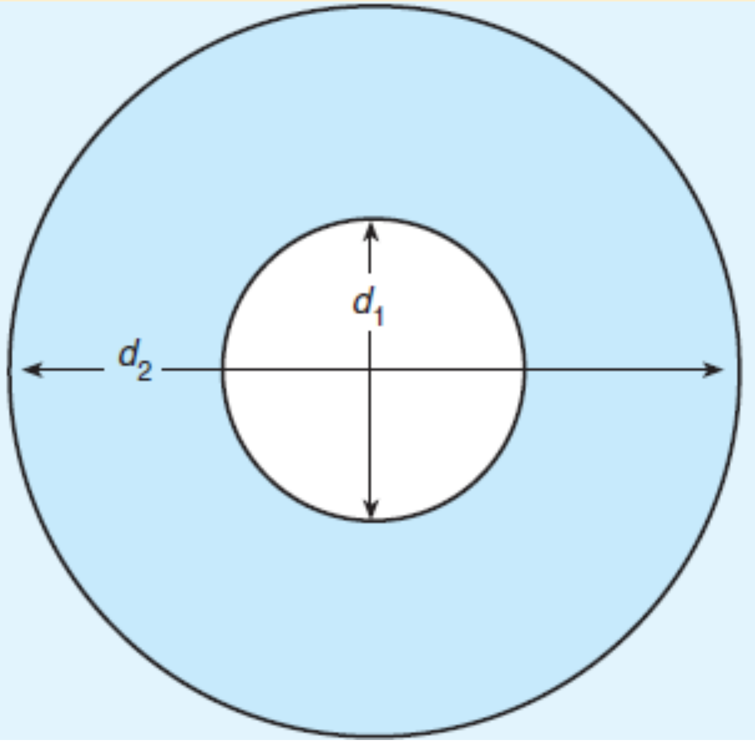
```
// Function maximum definition
// x, y and z are parameters
int maximum(int x, int y, int z)
{
   int max = x; // assume x is largest

   if (y > max) { // if y is larger than max,
      max = y; // assign y to max
   }

   if (z > max) { // if z is larger than max,
      max = z; // assign z to max
   }

   return max; // max is largest value
}
```

```
Enter three integers: 22 85 17
Maximum is: 85
```

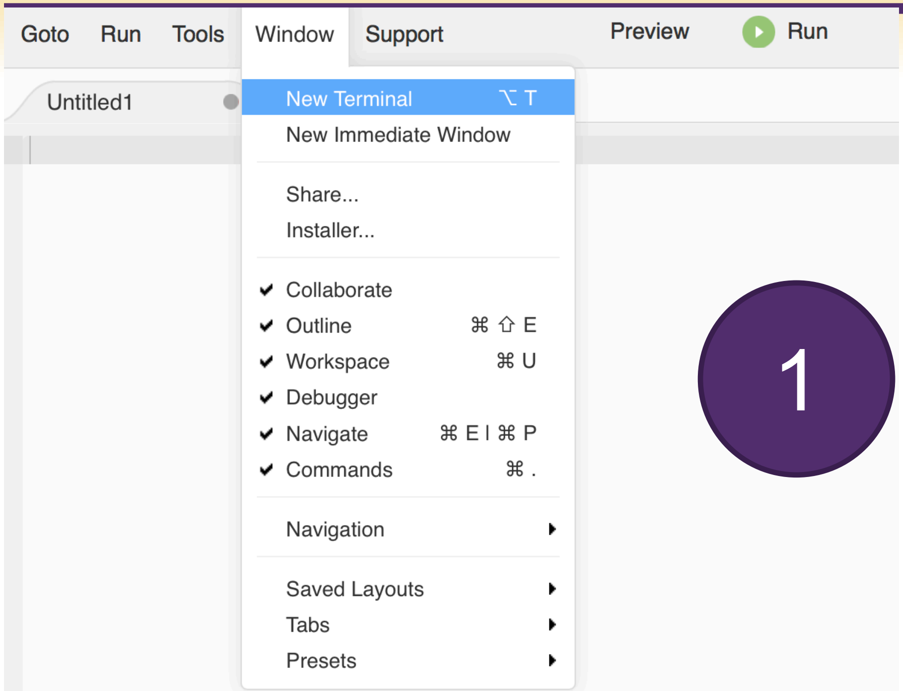
Write a function to calculate area of a washer

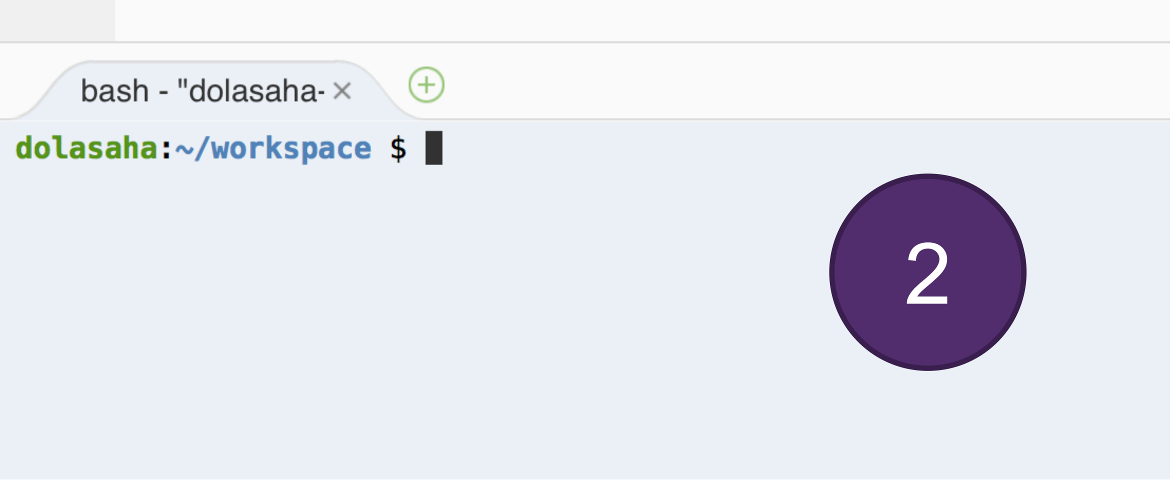


```c
#include <stdio.h>

double calc_area(double radius);

//function main begins program execution
int main ( void )
{
   double extRadius, intRadius, extArea, intArea;

   // Ask user to enter External Radius
   printf("External Radius: " );
   // this is the statement to read External Radius from user
   scanf( "%lf", &extRadius );

   // Ask user to enter Internal Radius
   printf("Internal Radius: " );
   // this is the statement to read External Radius from user
   scanf( "%lf", &intRadius );

   // Calculate the area
   extArea = calc_area(extRadius);

   // Calculate the area
   intArea = calc_area(intRadius);

   double washerArea = extArea - intArea;

   // printing out the results
   printf("The area of the washer is %lf.\n", washerArea);
}

double calc_area(double radius)
{
   double PI = 3.14159;
   return PI*radius*radius;
}
```



Compiling your own code

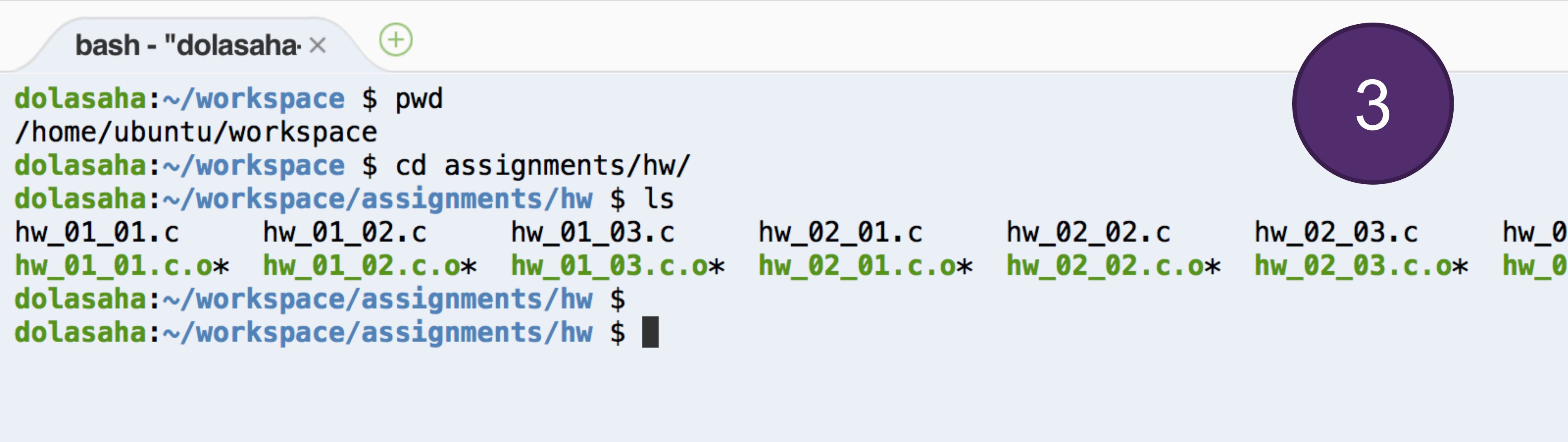




```
dolasaha:~/workspace $ █
```

Compiling your own code

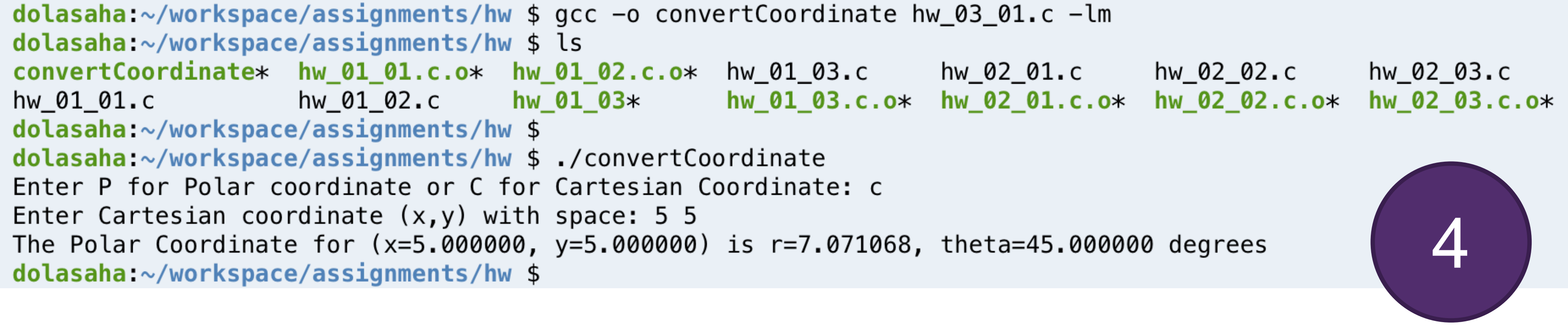
- pwd – print work directory
- cd directory_name – change directory
- ls – list the content of current directory



```
dolasaha:~/workspace $ pwd
/home/ubuntu/workspace
dolasaha:~/workspace $ cd assignments/hw/
dolasaha:~/workspace/assignments/hw $ ls
hw_01_01.c      hw_01_02.c      hw_01_03.c      hw_02_01.c      hw_02_02.c      hw_02_03.c      hw_0
hw_01_01.c.o*   hw_01_02.c.o*   hw_01_03.c.o*   hw_02_01.c.o*   hw_02_02.c.o*   hw_02_03.c.o*   hw_0
dolasaha:~/workspace/assignments/hw $
dolasaha:~/workspace/assignments/hw $ █
```

Linking with Math Library

- ➢ `gcc -o object_filename c_file.c -lm`
  - `-l` link to the library
  - `-lm` is specific for math
- ➢ Run the object file
  - `./object_filename`

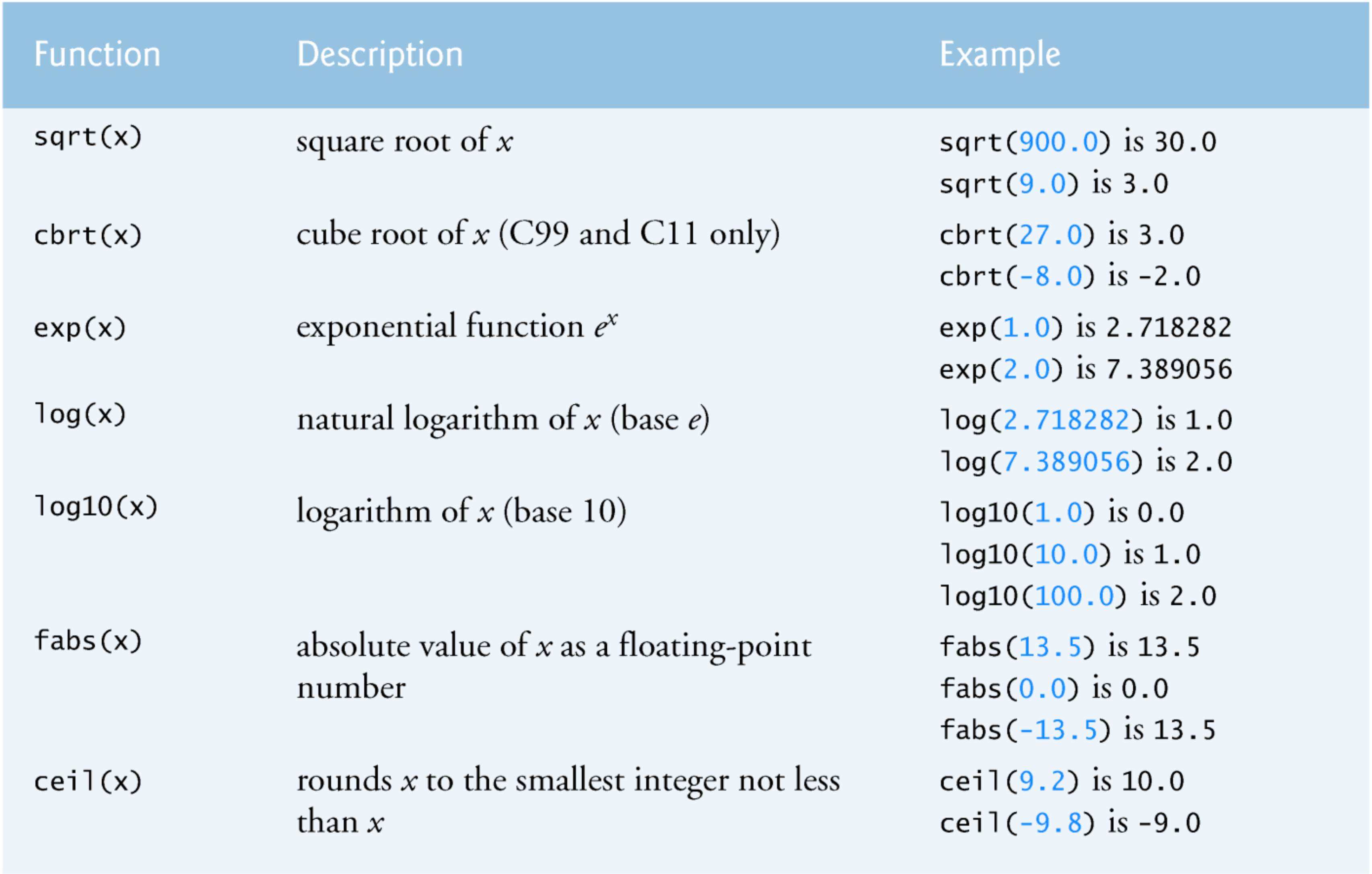


```
dolasaha:~/workspace/assignments/hw $ gcc -o convertCoordinate hw_03_01.c -lm
dolasaha:~/workspace/assignments/hw $ ls
convertCoordinate*  hw_01_01.c.o*  hw_01_02.c.o*  hw_01_03.c     hw_02_01.c     hw_02_02.c     hw_02_03.c
hw_01_01.c          hw_01_02.c     hw_01_03*      hw_01_03.c.o*  hw_02_01.c.o*  hw_02_02.c.o*  hw_02_03.c.o*
dolasaha:~/workspace/assignments/hw $
dolasaha:~/workspace/assignments/hw $ ./convertCoordinate
Enter P for Polar coordinate or C for Cartesian Coordinate: c
Enter Cartesian coordinate (x,y) with space: 5 5
The Polar Coordinate for (x=5.000000, y=5.000000) is r=7.071068, theta=45.000000 degrees
dolasaha:~/workspace/assignments/hw $
```

4

Math Library Functions

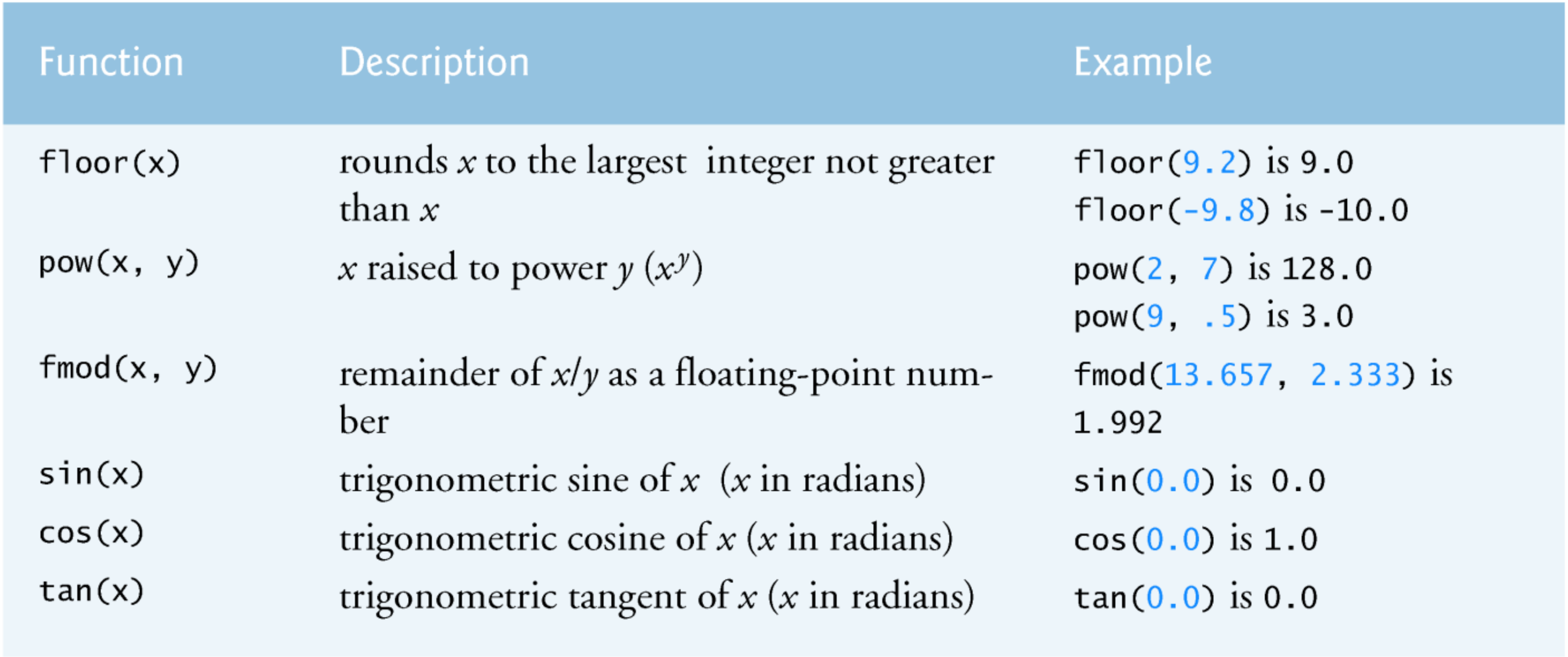
- Performs common mathematical calculations.



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

More Math Library Functions

- `#include <math.h>`



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

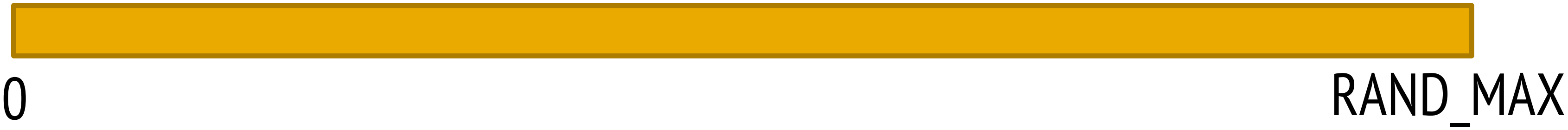
Random Number Generation

- Why?
  - For example, a program that simulates coin tossing might require only 0 for "heads" and 1 for "tails."
  - A dice-rolling program that simulates a six-sided die would require random integers from 1 to 6.

- The `rand` function generates an integer between `0` and `RAND_MAX` (a symbolic constant defined in the `<stdlib.h>` header).
  - `i = rand();`
- To get a range of values, use remainder operation.
  - `i = rand()%N; // random values in {0 to N-1}`

Scaling and Shifting

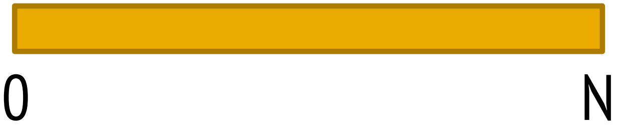
- Generate Random Number
  - `r_num = rand();`



0

RAND_MAX

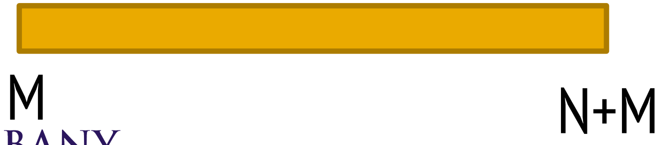
- Scale
  - `r_scaled = r_num()%N;`



0

N

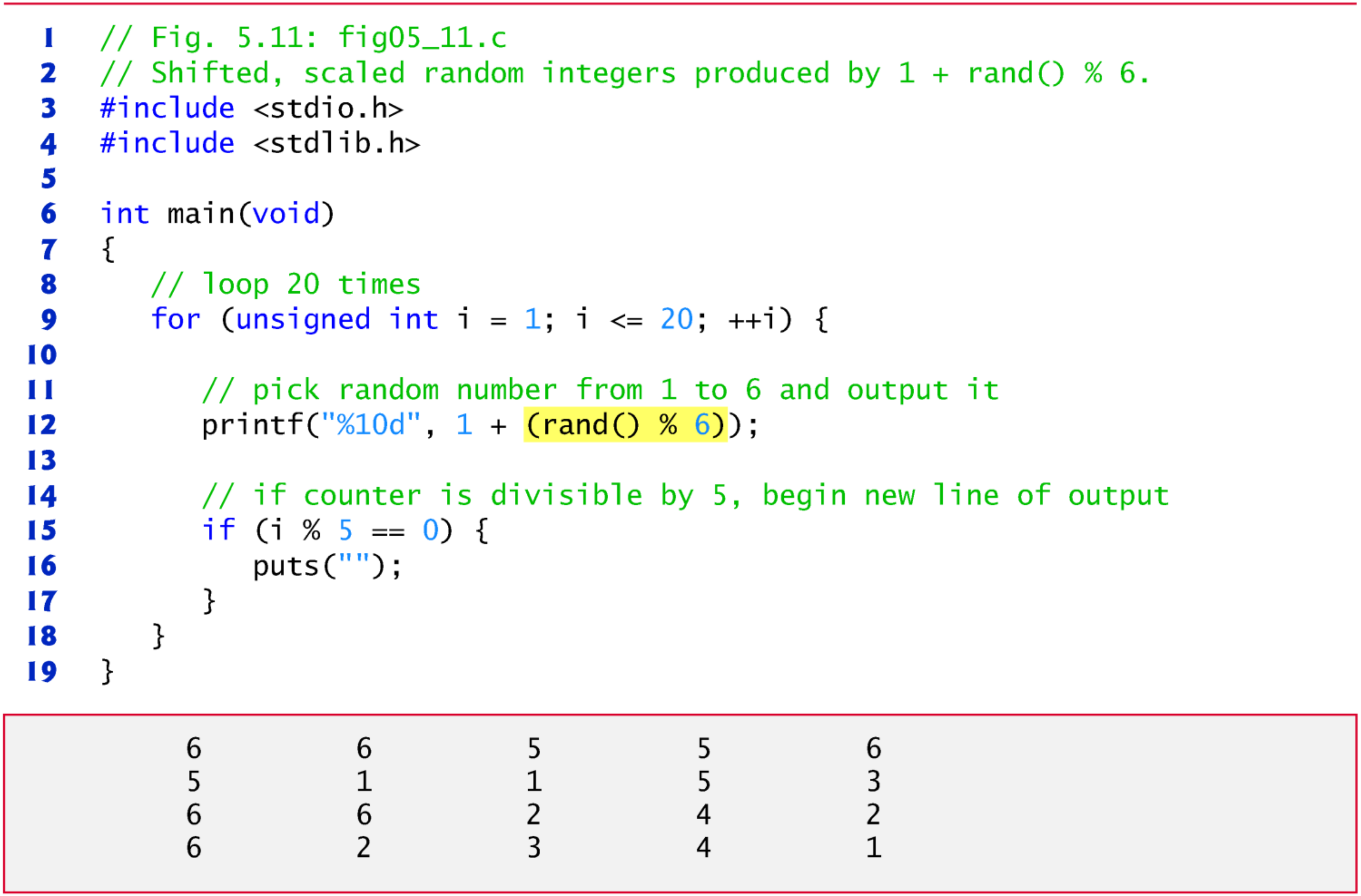
- Shift
  - `r_shifted = r_scaled+M;`



M

N+M

Random Number Generation Code



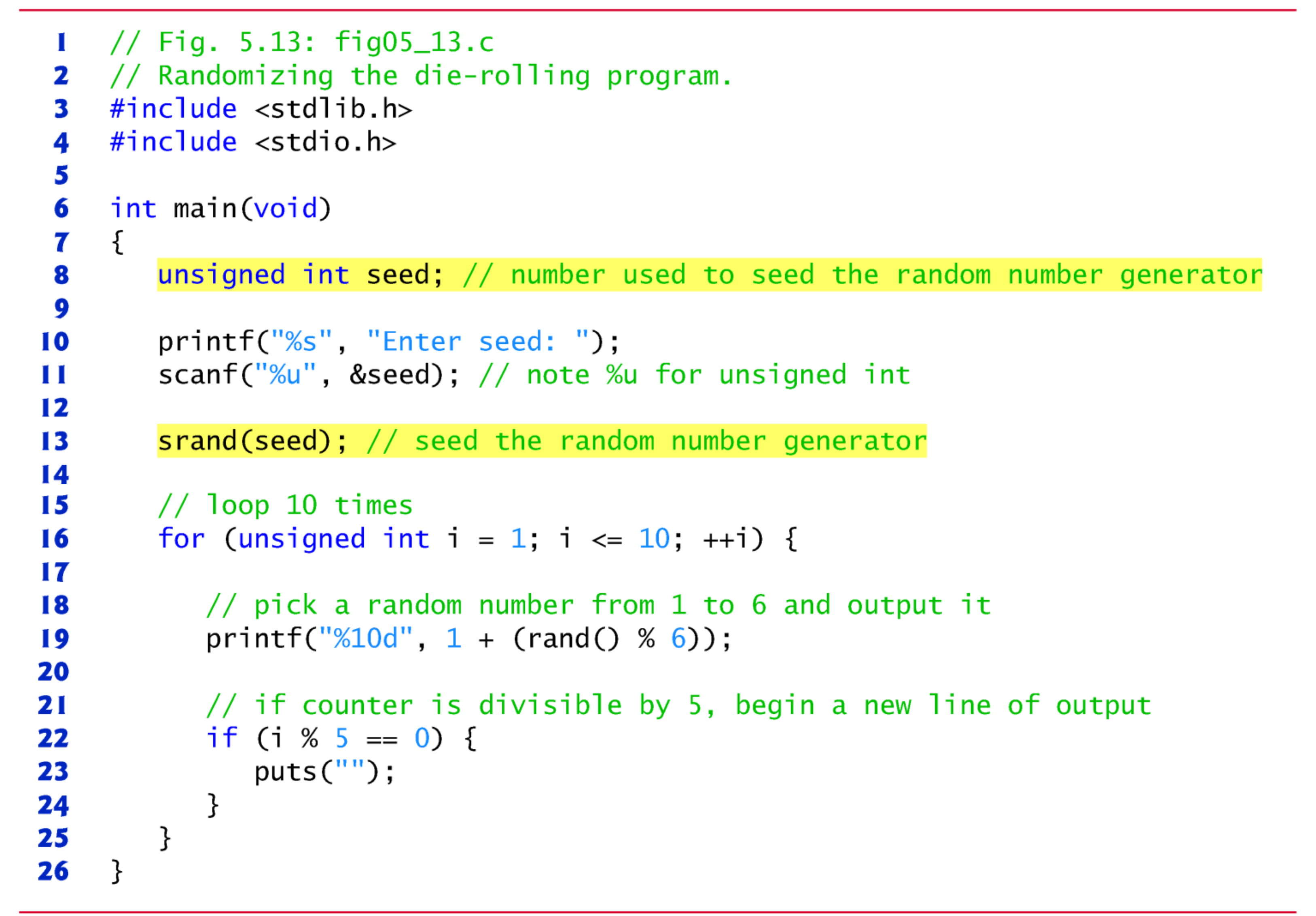
```c
// Fig. 5.11: fig05_11.c
// Shifted, scaled random integers produced by 1 + rand() % 6.
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
   // loop 20 times
   for (unsigned int i = 1; i <= 20; ++i) {

      // pick random number from 1 to 6 and output it
      printf("%10d", 1 + (rand() % 6));

      // if counter is divisible by 5, begin new line of output
      if (i % 5 == 0) {
         puts("");
      }
   }
}
```

```
         6         6         5         5         6
         5         1         1         5         3
         6         6         2         4         2
         6         2         3         4         1
```

Pseudorandom numbers

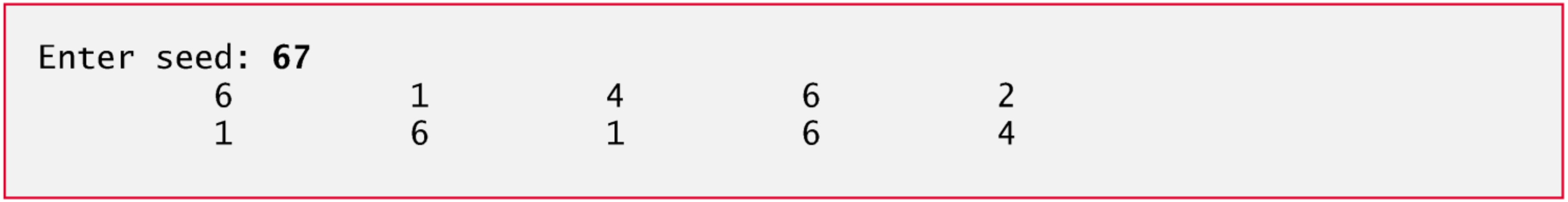
- Function `rand()` generates pseudorandom numbers.
- Calling `rand()` repeatedly produces a sequence of numbers that appears to be random.
- Randomizing
  - A program conditioned to produce a different sequence of random numbers for each execution
  - Accomplished with the standard library function `srand()`.
- Function `srand()` takes an unsigned integer argument and seeds function `rand()` to produce a different sequence of random numbers for each execution of the program.

Randomizing with a seed

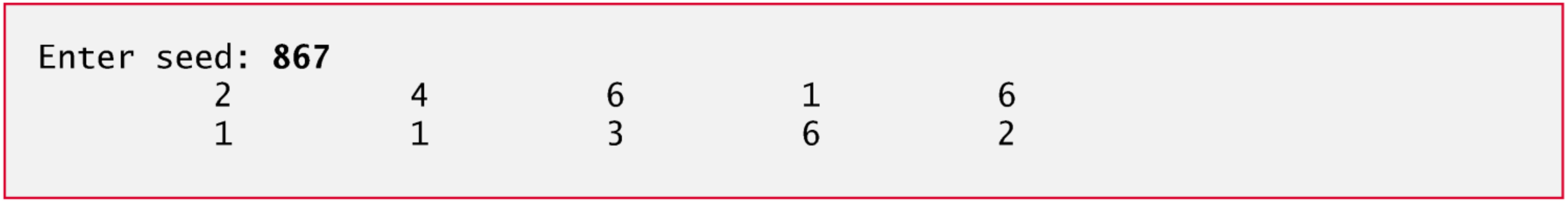


```c
// Fig. 5.13: fig05_13.c
// Randomizing the die-rolling program.
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
   unsigned int seed; // number used to seed the random number generator

   printf("%s", "Enter seed: ");
   scanf("%u", &seed); // note %u for unsigned int

   srand(seed); // seed the random number generator

   // loop 10 times
   for (unsigned int i = 1; i <= 10; ++i) {

      // pick a random number from 1 to 6 and output it
      printf("%10d", 1 + (rand() % 6));

      // if counter is divisible by 5, begin a new line of output
      if (i % 5 == 0) {
         puts("");
      }
   }
}
```

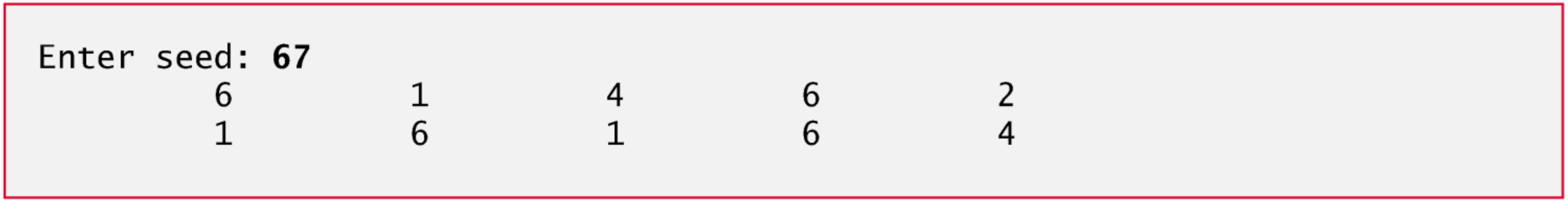
Output



```
Enter seed: 67
         6         1         4         6         2
         1         6         1         6         4
```



```
Enter seed: 867
         2         4         6         1         6
         1         1         3         6         2
```

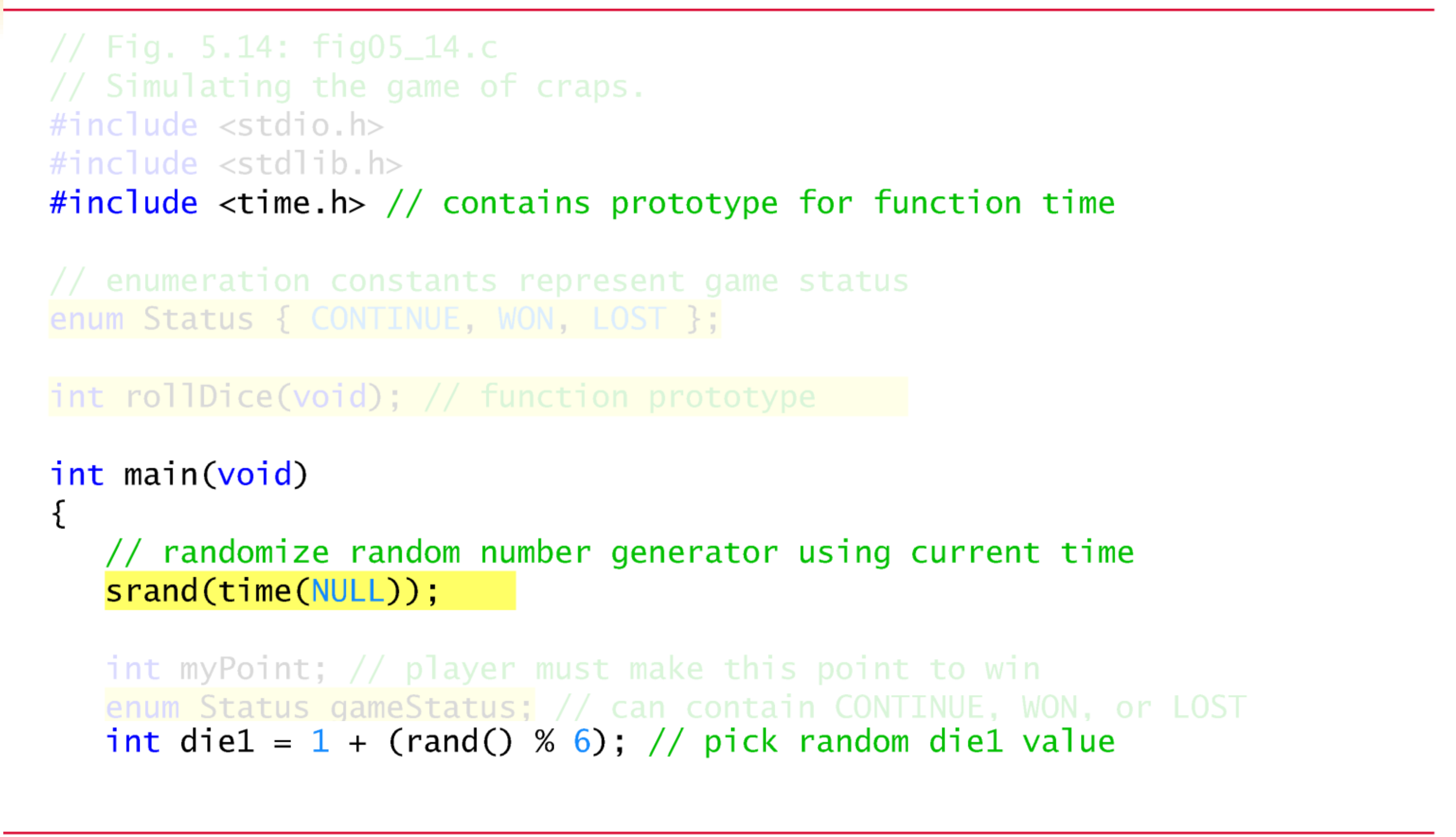


```
Enter seed: 67
         6         1         4         6         2
         1         6         1         6         4
```

Randomize without providing a seed

- To randomize without entering a seed each time, use a statement like `srand(time(NULL));`
- The function prototype for time is in `<time.h>`.
- Function `time` returns the number of seconds that have passed since midnight on January 1, 1970.
- This value is converted to an unsigned integer and used as the seed to the random number generator.

Randomize with time

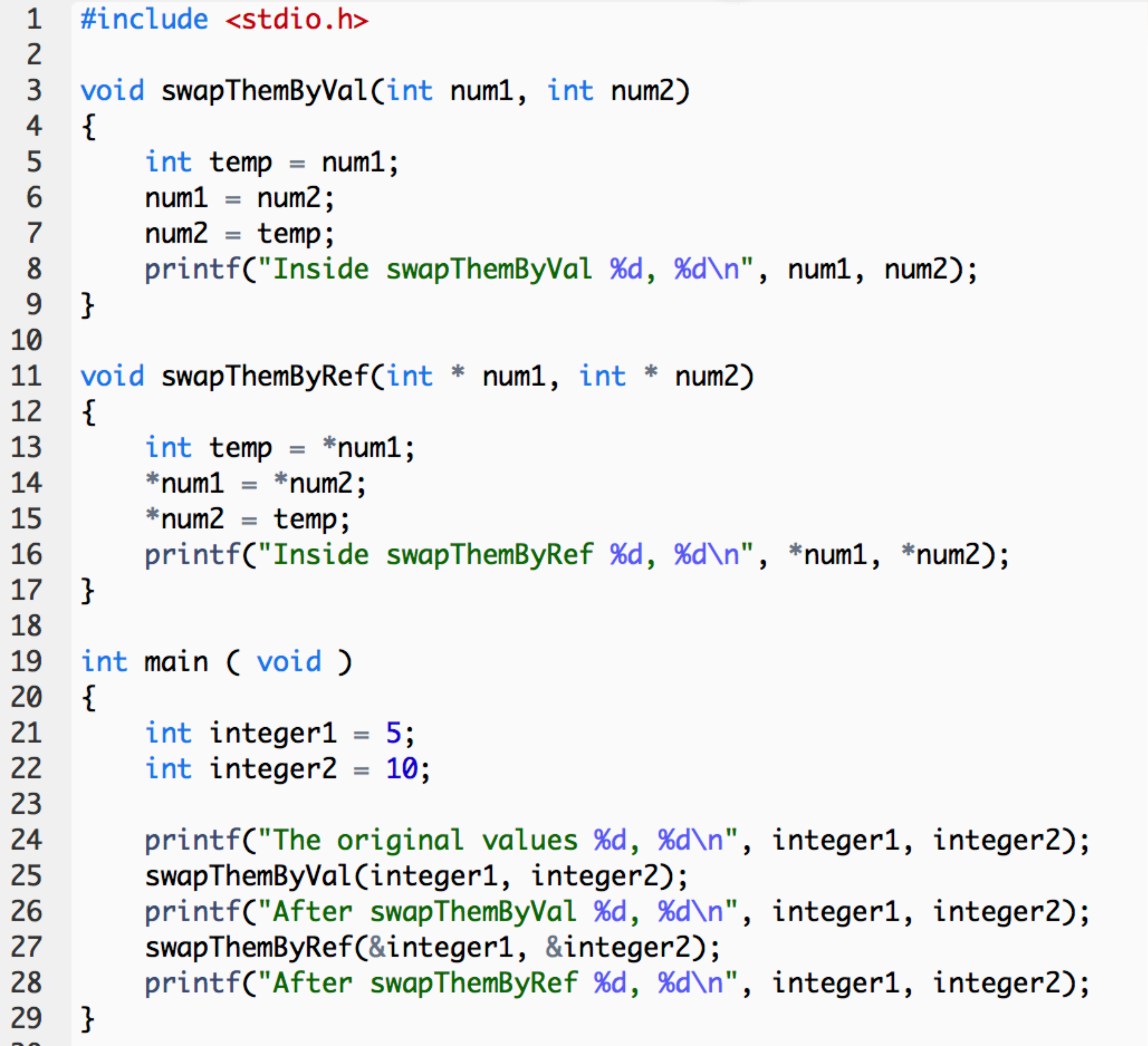


```
// Fig. 5.14: fig05_14.c
// Simulating the game of craps.
#include <stdio.h>
#include <stdlib.h>
#include <time.h> // contains prototype for function time

// enumeration constants represent game status
enum Status { CONTINUE, WON, LOST };

int rollDice(void); // function prototype

int main(void)
{
   // randomize random number generator using current time
   srand(time(NULL));

   int myPoint; // player must make this point to win
   enum Status gameStatus; // can contain CONTINUE, WON, or LOST
   int die1 = 1 + (rand() % 6); // pick random die1 value
```

Passing argument by value & by pointer

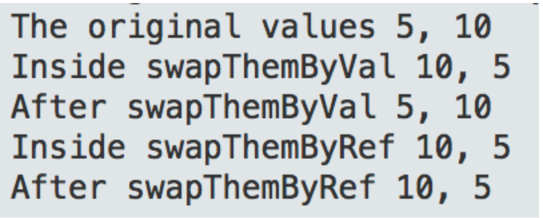
Pass by Value	Pass by Pointer
A copy of argument’s value is made and passed to the function	An address to the argument is passed to the function
Changes to copy do not change the original value	Changes to the value of the address does change the original value
Most commonly used	Should be used by trusted functions only

Example Pass-by-value & Pass-by-reference



```c
#include <stdio.h>

void swapThemByVal(int num1, int num2)
{
    int temp = num1;
    num1 = num2;
    num2 = temp;
    printf("Inside swapThemByVal %d, %d\n", num1, num2);
}

void swapThemByRef(int * num1, int * num2)
{
    int temp = *num1;
    *num1 = *num2;
    *num2 = temp;
    printf("Inside swapThemByRef %d, %d\n", *num1, *num2);
}

int main ( void )
{
    int integer1 = 5;
    int integer2 = 10;

    printf("The original values %d, %d\n", integer1, integer2);
    swapThemByVal(integer1, integer2);
    printf("After swapThemByVal %d, %d\n", integer1, integer2);
    swapThemByRef(&integer1, &integer2);
    printf("After swapThemByRef %d, %d\n", integer1, integer2);
}
```

Output



```
The original values 5, 10
Inside swapThemByVal 10, 5
After swapThemByVal 5, 10
Inside swapThemByRef 10, 5
After swapThemByRef 10, 5
```