CMPE 252
C PROGRAMMING

SPRING 2021

WEEK 2-3

TOP-DOWN DESIGN WITH FUNCTIONS
CHAPTER 3

Problem Solving & Program Design in C

Eighth Edition
Global Edition

Jeri R. Hanly & Elliot B. Koffman

Library Functions

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

C Library Functions

- You need to prepend library name with #include
- Examples:

Function	Standard header file	Purpose	Arguments	Result
abs(x)	<stdlib.h>	Absolute value computation, e.g. abs(-5) is 5	int	int
ceil(x)	<math.h>	Smallest integral value that is not less than x, e.g. ceil(45.23) is 46.0	double	double
log(x)	<math.h>	Natural logarithm of x.	double	double
pow(x,y)	<math.h>	x^y	double, double	double
sin(x)	<math.h>	Sine of angle x, e.g. sin(1.5708) is 1.0	double (radians)	double

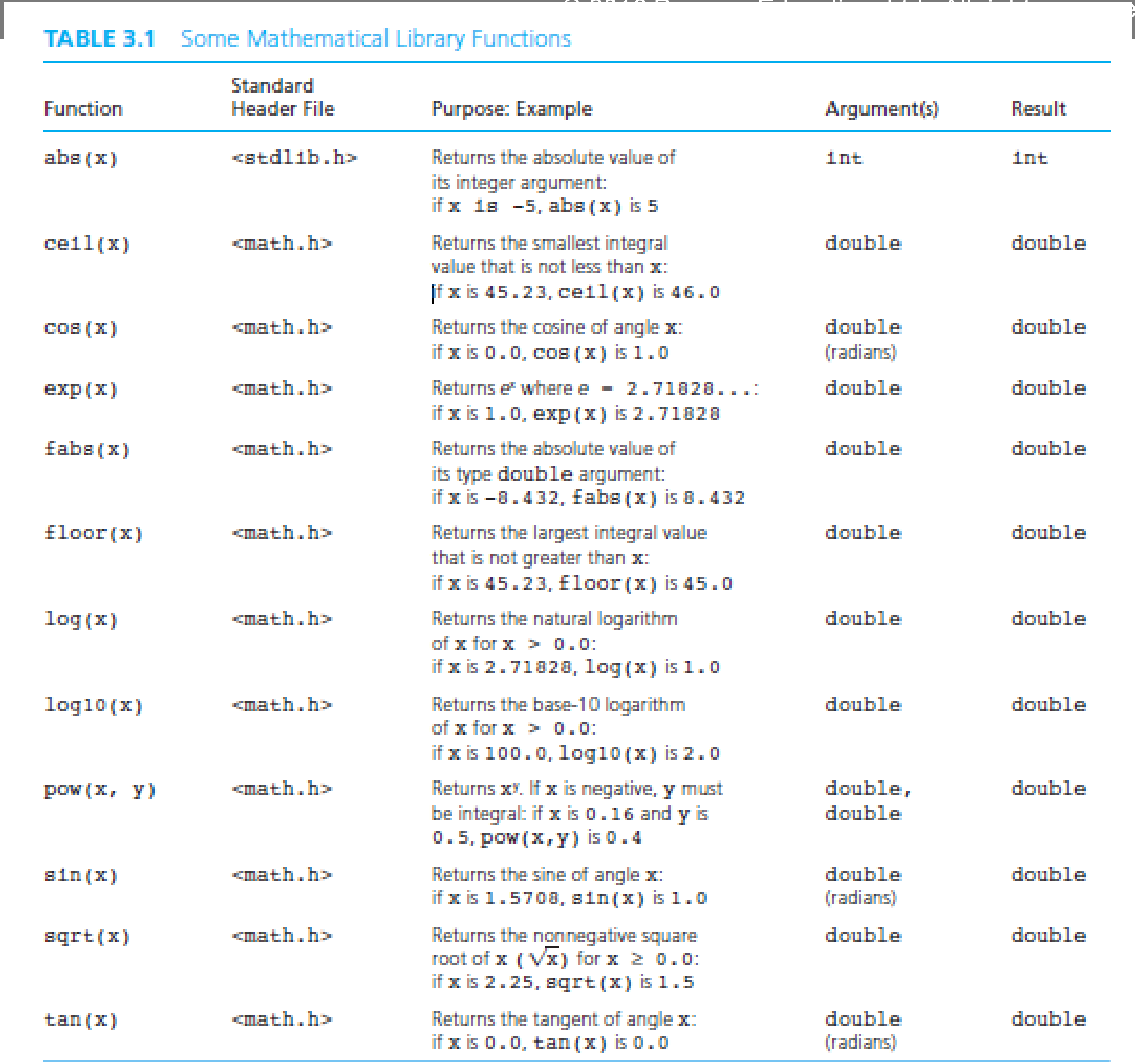


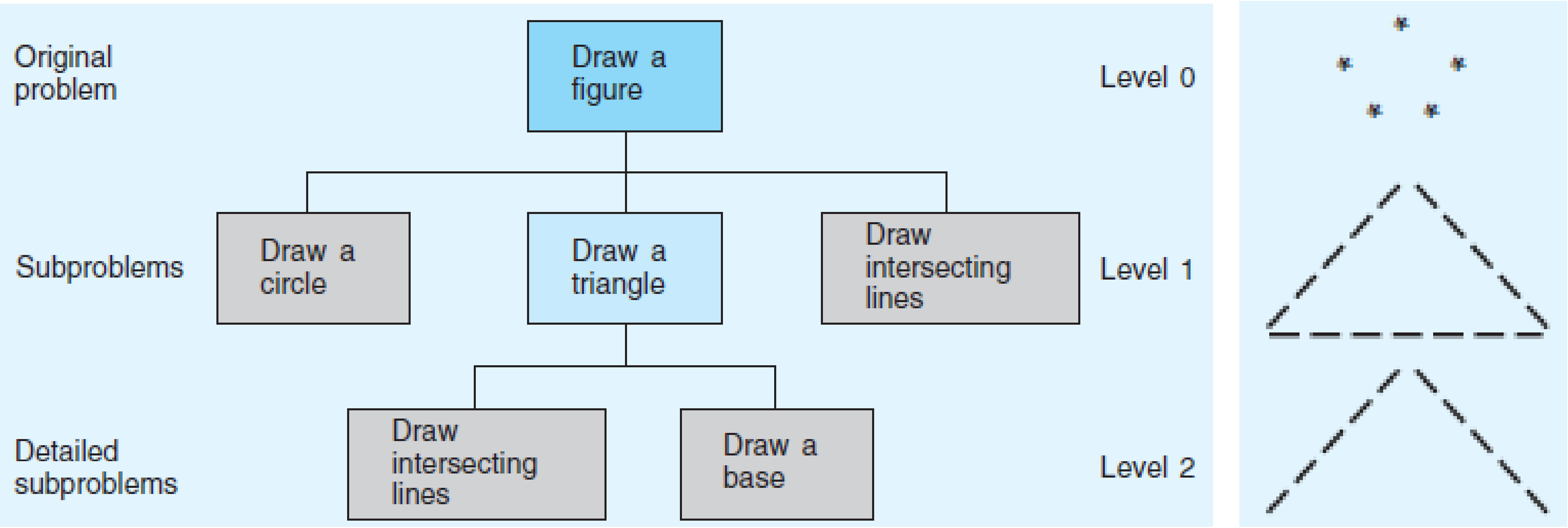
TABLE 3.1 Some Mathematical Library Functions

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 nonnegative square root of `x` ($\sqrt{x}$) for $x \geq 0.0$: 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`

Top-Down Design and Structure Charts

- 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
- structure chart
  - a documentation tool that shows the relationships among the subproblems of a problem

Figure 3.10
Structure Chart for Drawing a Stick Figure



Original problem

Draw a figure

Level 0

Subproblems

Draw a circle

Draw a triangle

Draw intersecting lines

Level 1

Detailed subproblems

Draw intersecting lines

Draw a base

Level 2

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

Function Prototype (Function Without Arguments)

- Syntax

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

- Example:

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

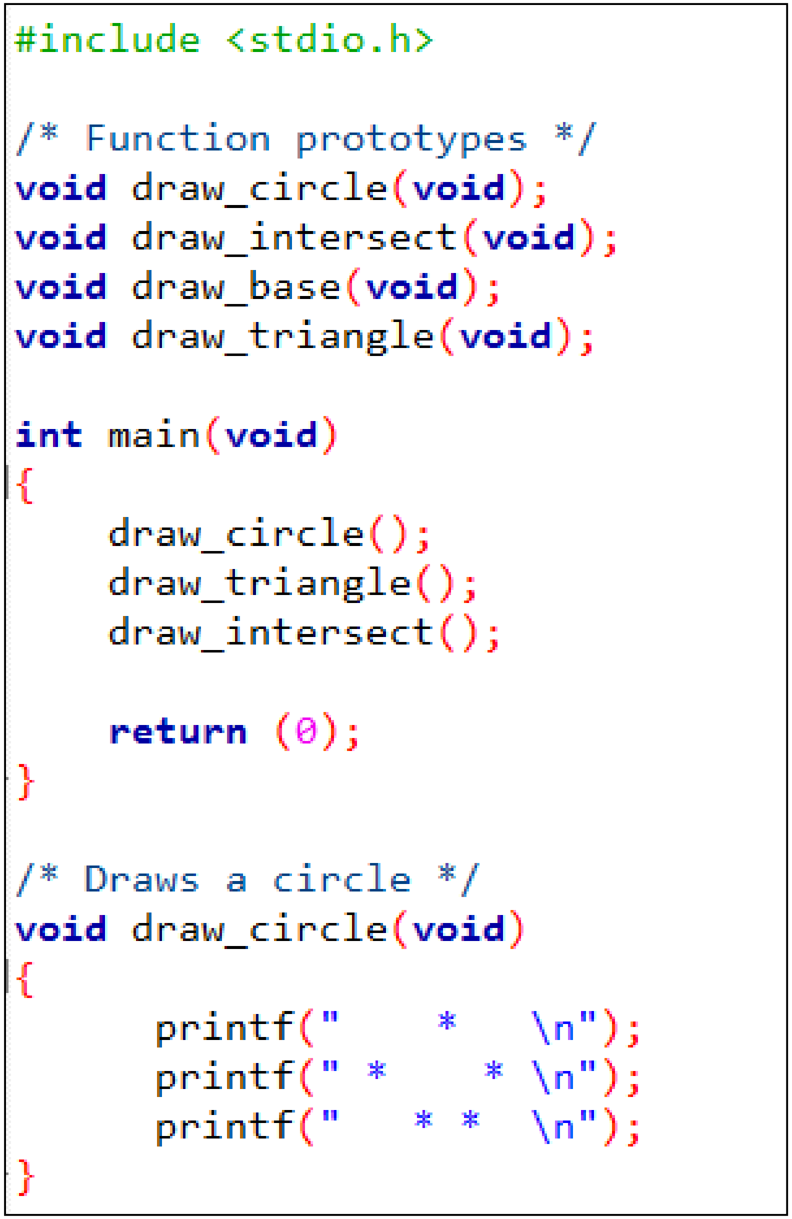
- Interpretation
  - the identifier fname is declared to be the name of a function
  - the identifier ftype specifies the data type of the function result

Function Definitions
(Function Without Arguments)

- Syntax

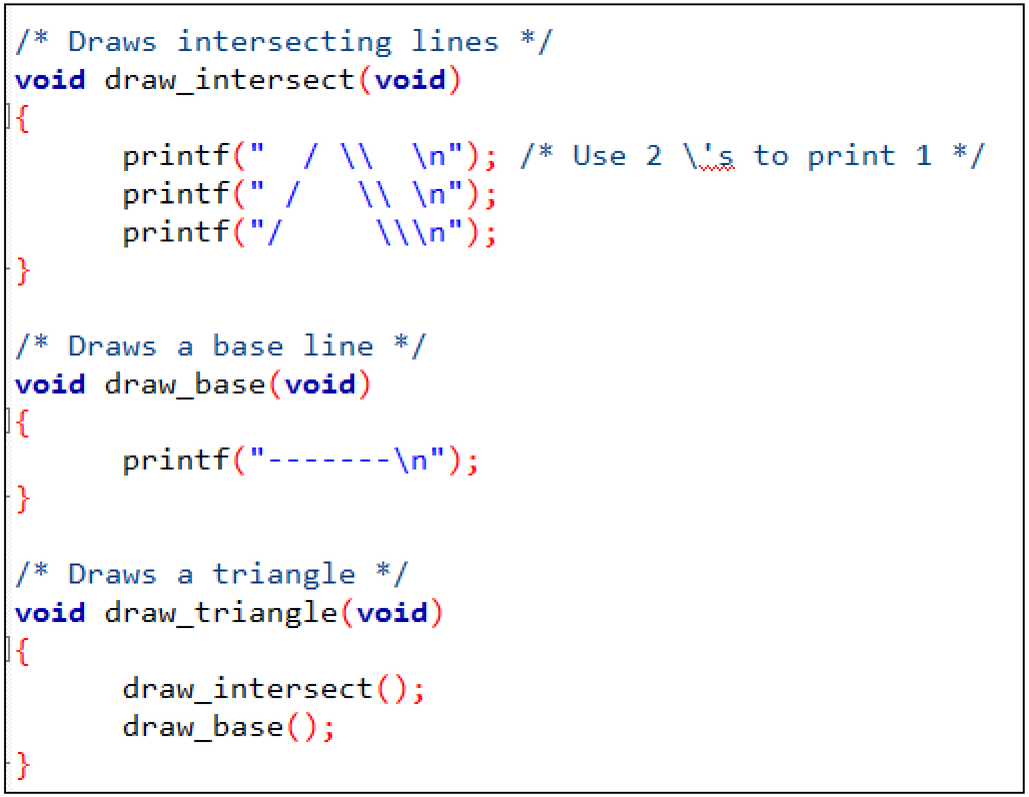
```
        ftype fname(void)
        {
                local declarations
                executable statements
        }
```

Figure 3.14



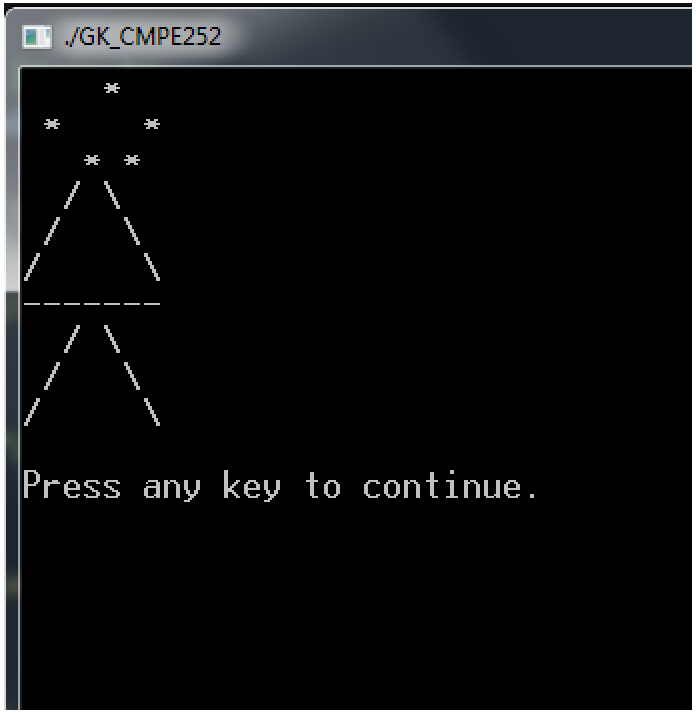
```
#include <stdio.h>

/* Function prototypes */
void draw_circle(void);
void draw_intersect(void);
void draw_base(void);
void draw_triangle(void);

int main(void)
{
    draw_circle();
    draw_triangle();
    draw_intersect();

    return (0);
}

/* Draws a circle */
void draw_circle(void)
{
      printf("    *   \n");
      printf(" *    * \n");
      printf("   * *  \n");
}
```

Figure 3.14 (cont.)



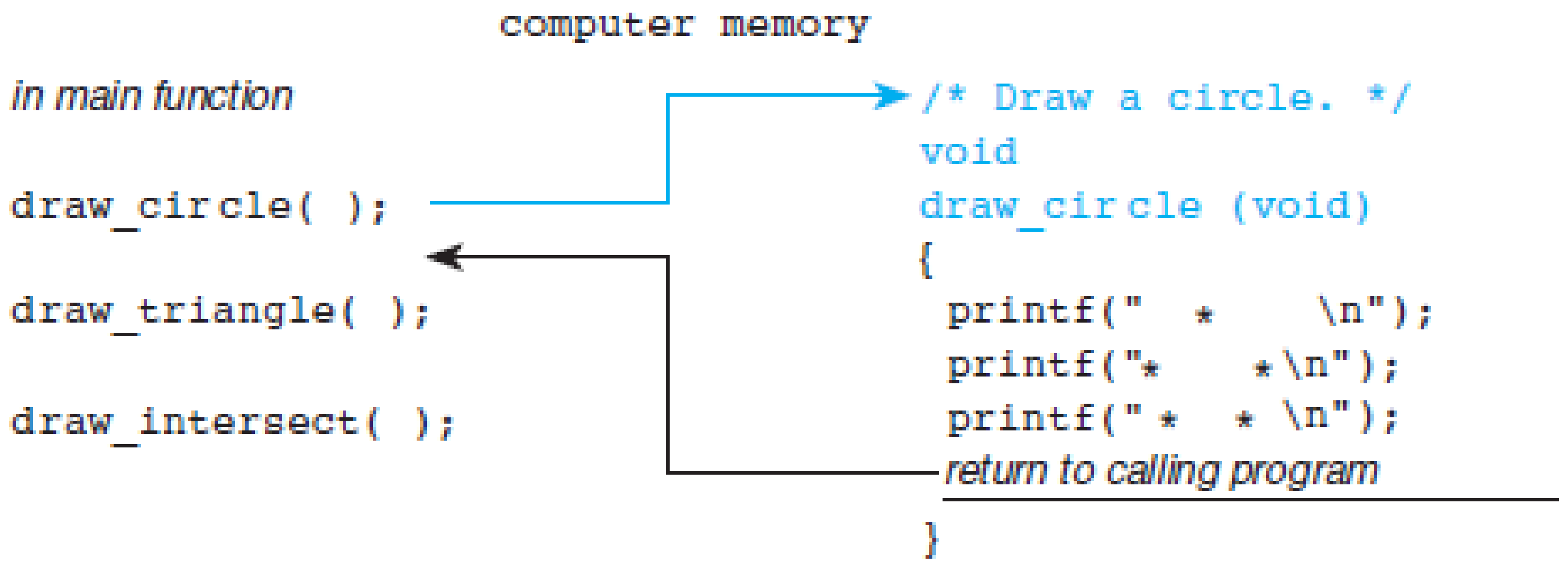
```c
/* Draws intersecting lines */
void draw_intersect(void)
{
      printf("  / \\  \n"); /* Use 2 \'s to print 1 */
      printf(" /   \\ \n");
      printf("/     \\\n");
}

/* Draws a base line */
void draw_base(void)
{
      printf("-------\n");
}

/* Draws a triangle */
void draw_triangle(void)
{
      draw_intersect();
      draw_base();
}
```

Output



```
    *
 *    *
   * *
  / \
 /   \
/     \
-------
  / \
 /   \
/     \

Press any key to continue.
```

Figure 3.15
Flow of Control Between the main Function and a Function Subprogram



computer memory

in main function

```
draw_circle( );

draw_triangle( );

draw_intersect( );
```

```
/* Draw a circle. */
void
draw_circle (void)
{
  printf("   *   \n");
  printf(" *     *\n");
  printf("  *   * \n");
```

return to calling program

```
}
```

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

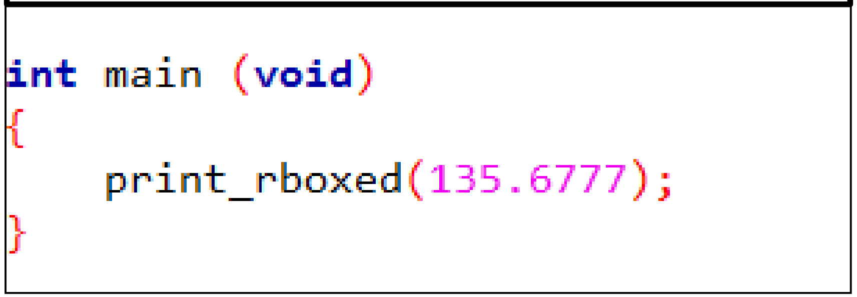
void Functions with Input Arguments

- actual argument
  - an expression used inside the parentheses of a function call
- formal parameter
  - an identifier that represents a corresponding actual argument in a function definition

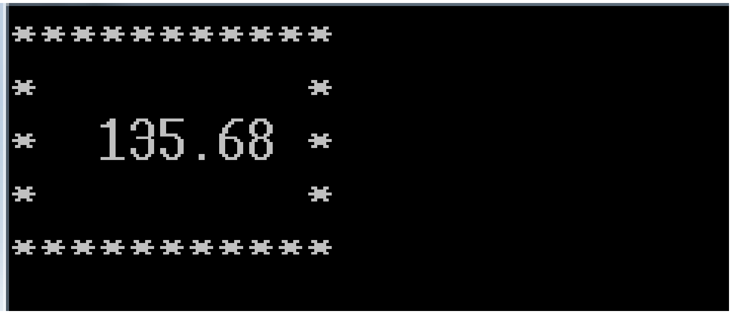
Figure 3.18

Function print_rboxed and Sample Run

Quick check: Write a function called print_rboxed which *gets a double number as parameter* and *returns nothing*. See the main method and the sample run below.



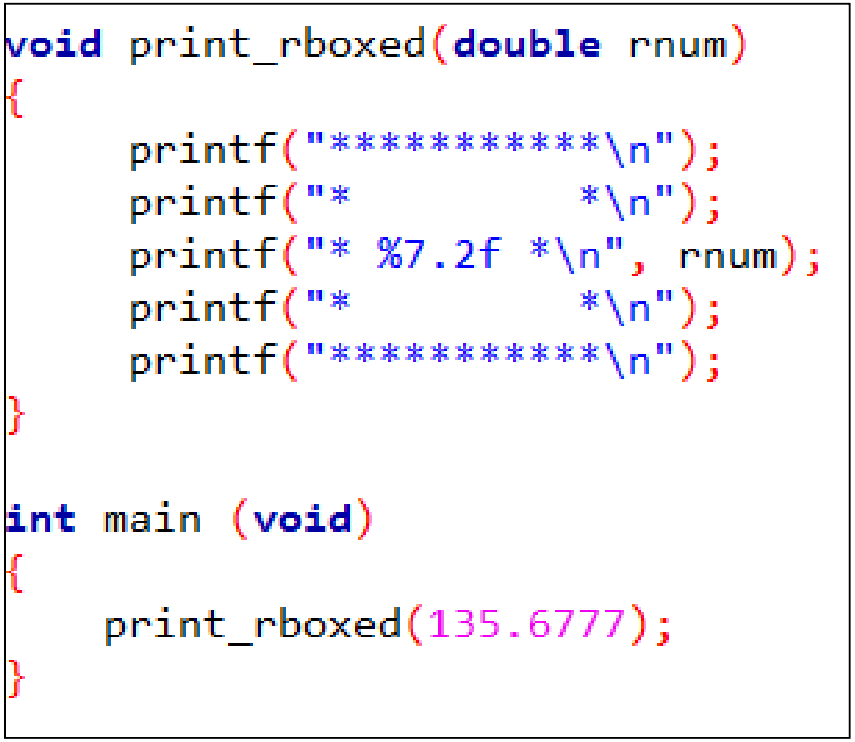
```
int main (void)
{
    print_rboxed(135.6777);
}
```



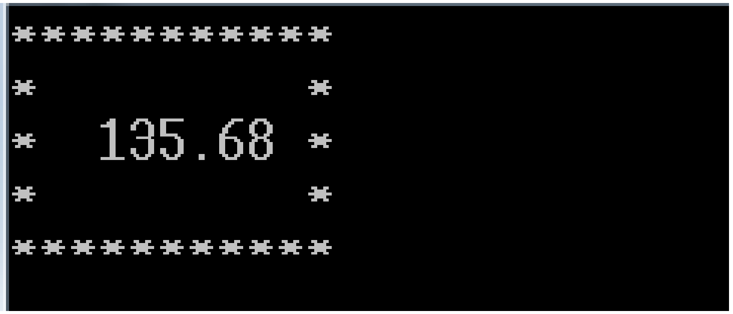
```
***********
*         *
*  135.68 *
*         *
***********
```

Figure 3.18
Function print_rboxed and Sample Run

Quick check: Write a function called print_rboxed which *gets a double number as parameter* and *returns nothing*. See the main method and the sample run below.



```
void print_rboxed(double rnum)
{
     printf("***********\n");
     printf("*         *\n");
     printf("* %7.2f *\n", rnum);
     printf("*         *\n");
     printf("***********\n");
}

int main (void)
{
    print_rboxed(135.6777);
}
```



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

Function Definition
(Input Arguments and a Single Result)

- Syntax

What can it be?

```
function interface comment
ftype  fname(formal parameter declaration list)
{
        local variable declarations
        executable statements
}
```

Function Interface Comment

- precondition
  - a condition assumed to be true before a function call

- postcondition
  - a condition assumed to be true after a function executes

Quick Check

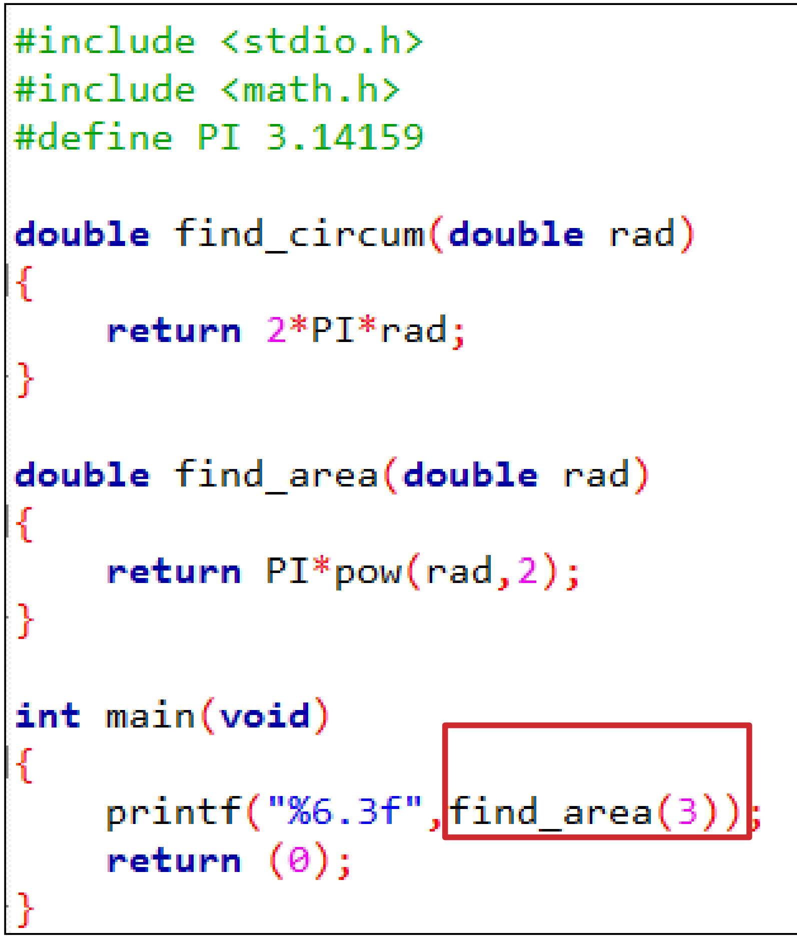
Write 2 functions find_circum and find_area where each one has one formal double parameter: radius and returns circumference or area. Use pow function of Math library.

What are their pre- and post conditions?

Quick Check

Write 2 functions find_circum and find_area where each one has one formal double parameter: radius and returns circumference or area. Use pow function of Math library.

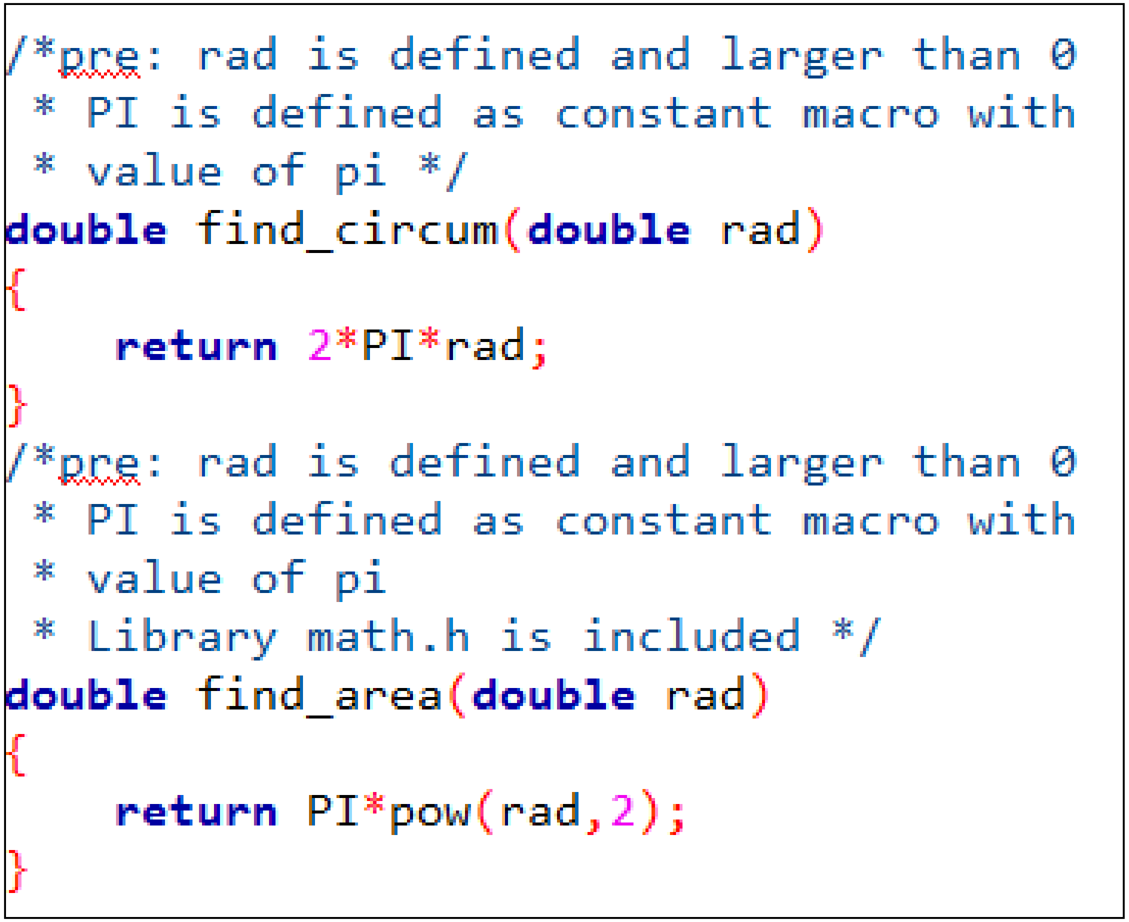
What are their pre- and post conditions?



```c
#include <stdio.h>
#include <math.h>
#define PI 3.14159

double find_circum(double rad)
{
    return 2*PI*rad;
}

double find_area(double rad)
{
    return PI*pow(rad,2);
}

int main(void)
{
    printf("%6.3f",find_area(3));
    return (0);
}
```

Output: 28.274

Quick Check



```
/*pre: rad is defined and larger than 0
 * PI is defined as constant macro with
 * value of pi */
double find_circum(double rad)
{
    return 2*PI*rad;
}
/*pre: rad is defined and larger than 0
 * PI is defined as constant macro with
 * value of pi
 * Library math.h is included */
double find_area(double rad)
{
    return PI*pow(rad,2);
}
```

Functions with Multiple Arguments

Argument List Correspondence

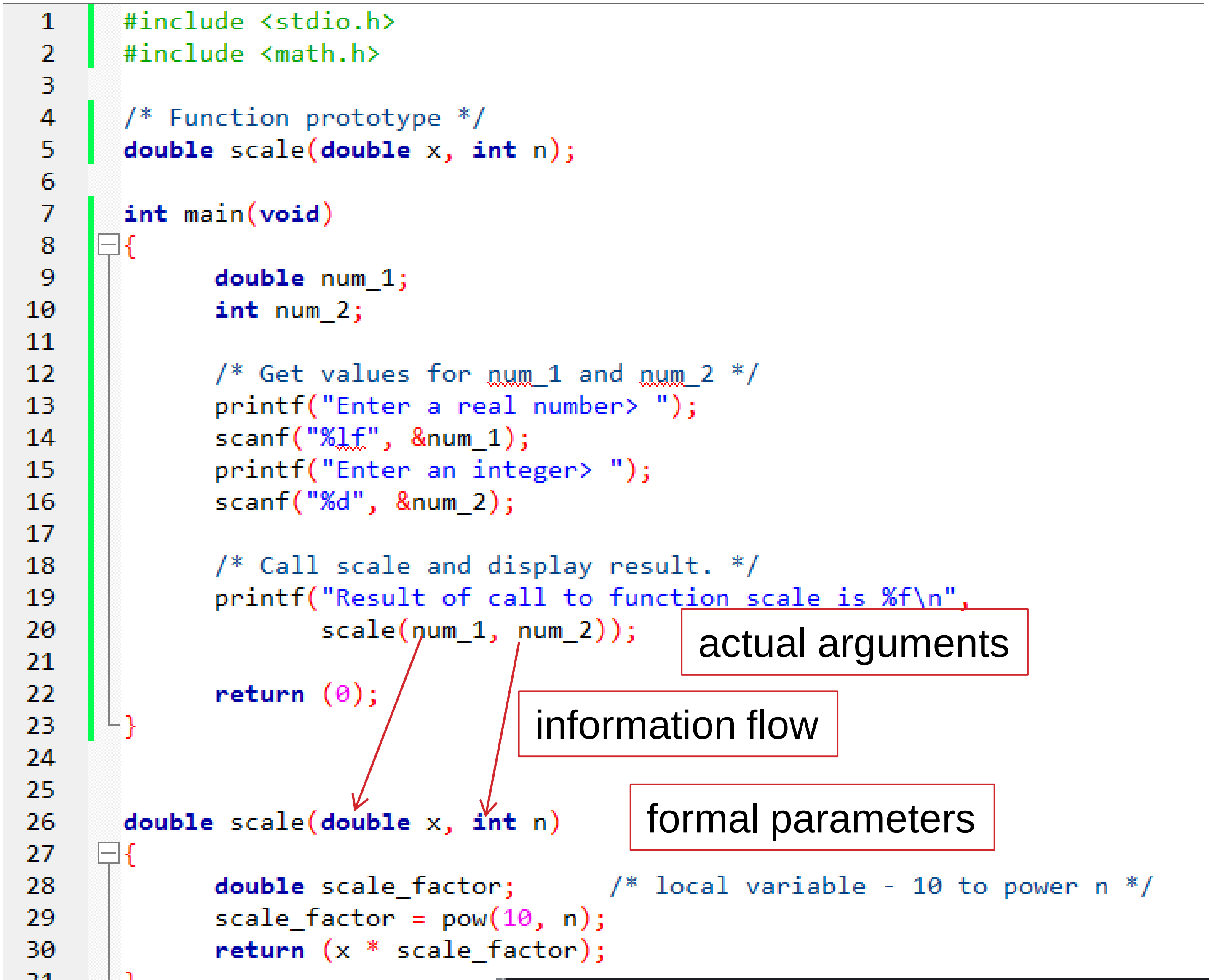
- The number of actual arguments used in a call to a function must be the same as the number of formal parameters listed in the function prototype.

- Each actual argument must be of a data type that can be assigned to the corresponding format 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 form parameter.
  - And so on…

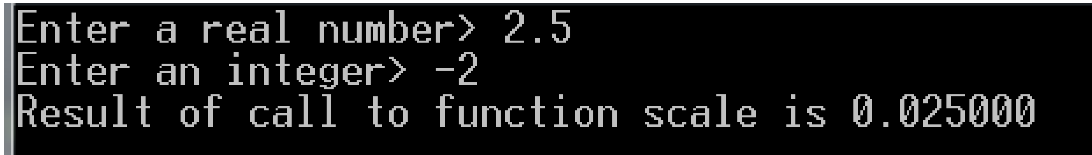


```
#include <stdio.h>
#include <math.h>

/* 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

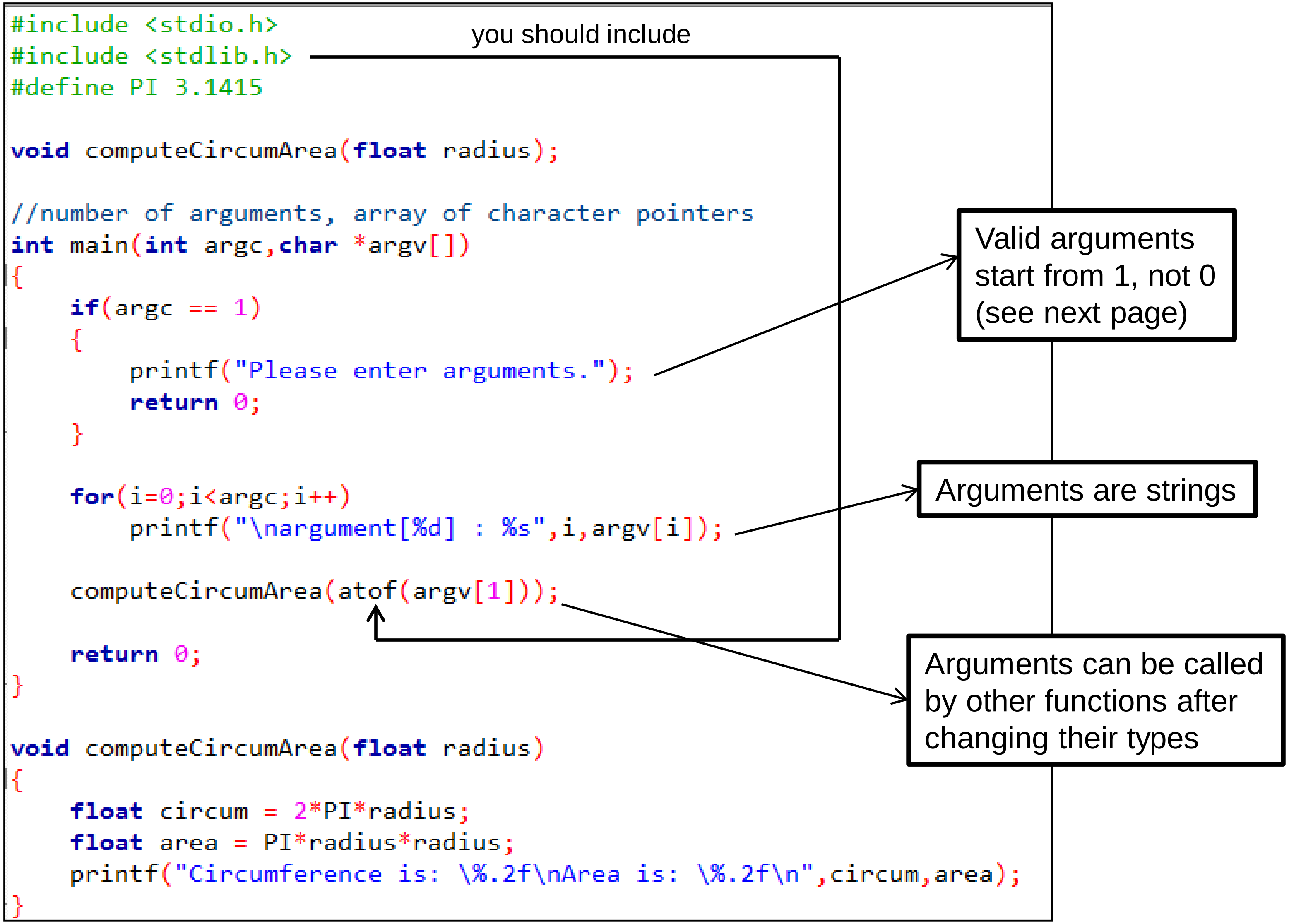
information flow

formal parameters



```
Enter a real number> 2.5
Enter an integer> -2
Result of call to function scale is 0.025000
```

Command Line Arguments



```
#include <stdio.h>
#include <stdlib.h>
#define PI 3.1415

void computeCircumArea(float radius);

//number of arguments, array of character pointers
int main(int argc,char *argv[])
{
    if(argc == 1)
    {
        printf("Please enter arguments.");
        return 0;
    }

    for(i=0;i<argc;i++)
        printf("\nargument[%d] : %s",i,argv[i]);

    computeCircumArea(atof(argv[1]));

    return 0;
}

void computeCircumArea(float radius)
{
    float circum = 2*PI*radius;
    float area = PI*radius*radius;
    printf("Circumference is: \%.2f\nArea is: \%.2f\n",circum,area);
}
```

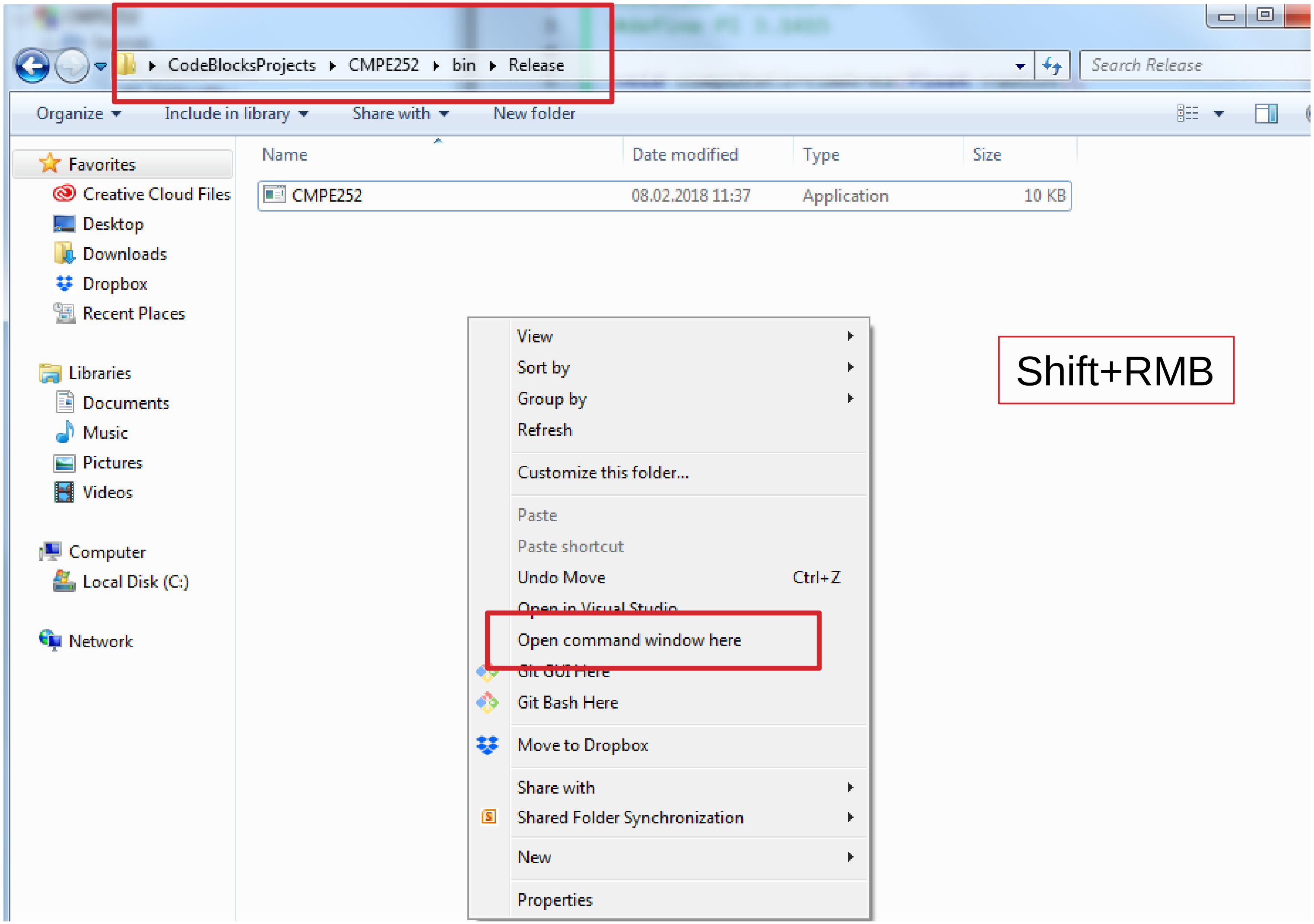
you should include

Valid arguments start from 1, not 0 (see next page)

Arguments are strings

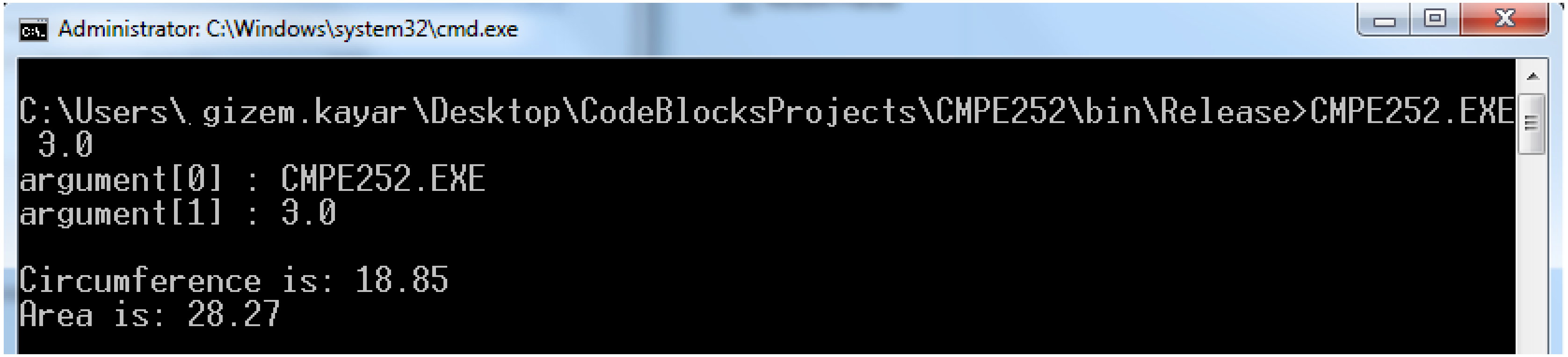
Arguments can be called by other functions after changing their types

From Command Window

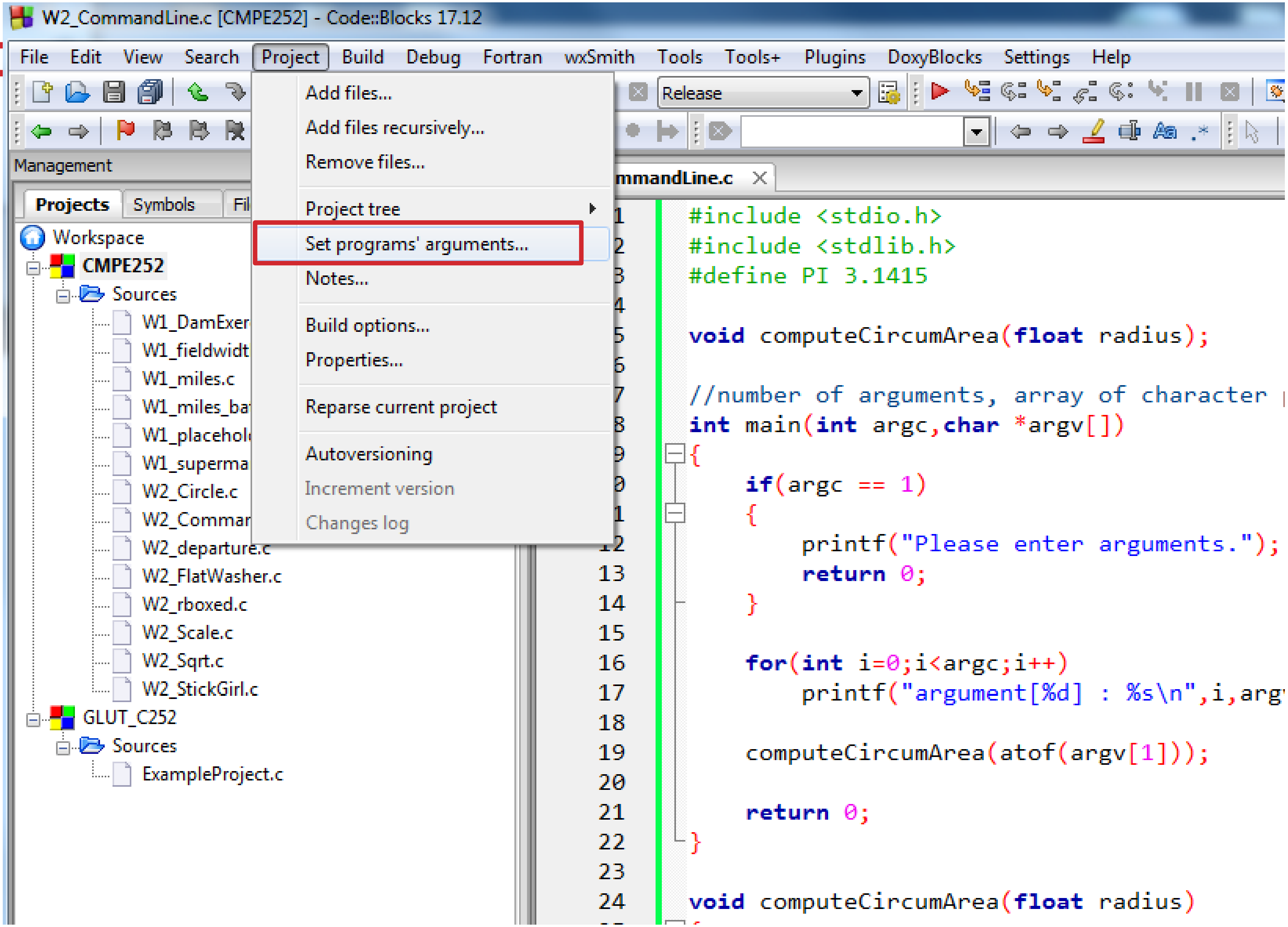


Name	Date modified	Type	Size
CMPE252	08.02.2018 11:37	Application	10 KB

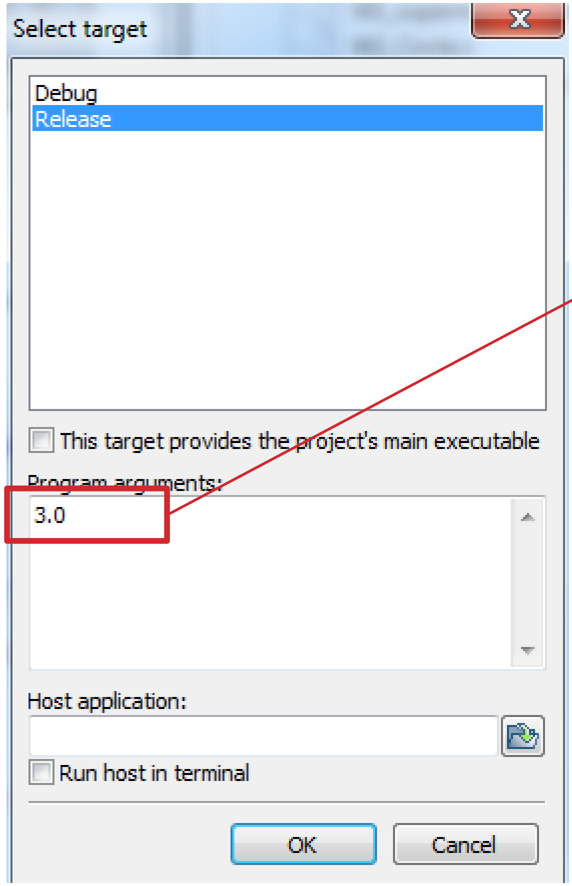
Shift+RMB



```
C:\Users\ gizem.kayar\Desktop\CodeBlocksProjects\CMPE252\bin\Release>CMPE252.EXE
 3.0
argument[0] : CMPE252.EXE
argument[1] : 3.0

Circumference is: 18.85
Area is: 28.27
```



```
#include <stdio.h>
#include <stdlib.h>
#define PI 3.1415

void computeCircumArea(float radius);

//number of arguments, array of character
int main(int argc,char *argv[])
{
    if(argc == 1)
    {
        printf("Please enter arguments.");
        return 0;
    }

    for(int i=0;i<argc;i++)
        printf("argument[%d] : %s\n",i,arg

    computeCircumArea(atof(argv[1]));

    return 0;
}

void computeCircumArea(float radius)
```



Debug

Release

□ This target provides the project's main executable

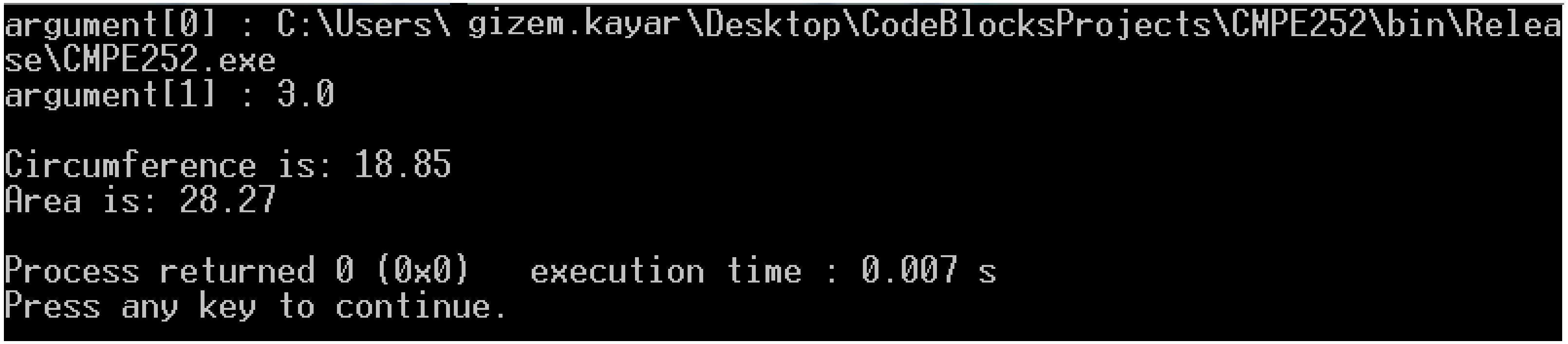
Program arguments:

3.0

Host application:

□ Run host in terminal

Press OK, now build and run your code



```
argument[0] : C:\Users\ gizem.kayar\Desktop\CodeBlocksProjects\CMPE252\bin\Relea
se\CMPE252.exe
argument[1] : 3.0

Circumference is: 18.85
Area is: 28.27

Process returned 0 (0x0)   execution time : 0.007 s
Press any key to continue.
```