*"while (noSuccess) { tryAgain(); if (dead) break; }"*

*- Unknown*

# CSE102
# Computer Programming with C

2017-2018 Spring Semester

Top-Down Design with Functions

Largely adapted from J.R. Hanly, E.B. Koffman, F.E. Sevilgen, and others…

## Function: modules of program

- Programmers use segments of earlier programs to construct new programs
  - Documentation is very important
  - Use of predefined functions
  - Top-down stepwise refinement
    - Major steps = modules of program

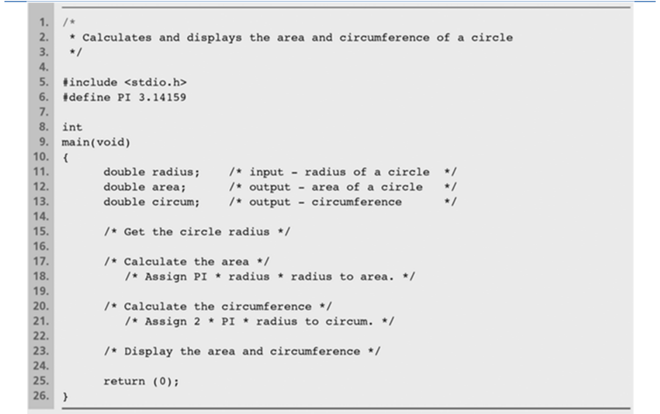
## Case Study: Circle

- Problem: Compute and display the area and the circumference of a circle
- Analysis:
  - Input: radius (double)
  - Outputs: area and circumference (double)
  - Relationship: ???
- Design:
  1. Get the radius
  2. Calculate the area
  3. Calculate the circumference
  4. Display the area and the circumference
  - Some steps requires refinement

## Case Study: Circle

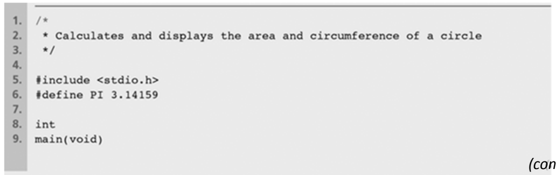
- Implementation:
  - The following slides contains the initial program

## Outline of Program Circle



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

#include <stdio.h>
#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 */

      /* Calculate the area */
         /* Assign PI * radius * radius to area. */

      /* Calculate the circumference */
         /* Assign 2 * PI * radius to circum. */

      /* Display the area and circumference */

      return (0);
}
```

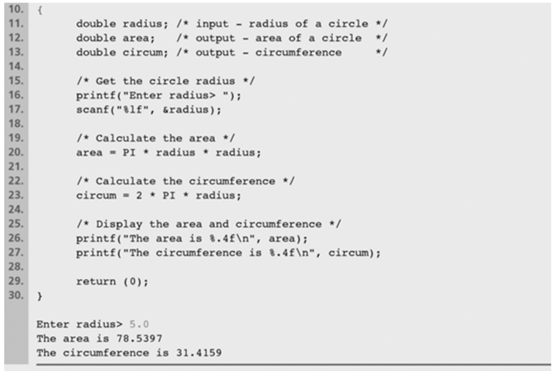
## Program Circle



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

#include <stdio.h>
#define PI 3.14159

int
main(void)
```

*(continued)*

## Outline of Program Circle

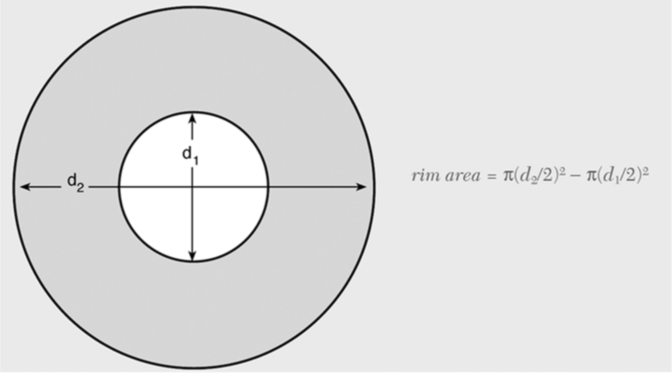


```
{
      double radius; /* input - radius of a circle */
      double area;   /* output - area of a circle  */
      double circum; /* output - circumference     */

      /* 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);
}

Enter radius> 5.0
The area is 78.5397
The circumference is 31.4159
```

## Case Study: Weight of Washers

- Here, we will use the solution of the previous case study
- Problem: Manufacturer of flat washers needs to estimate shipping cost. They need to compute the weight of a specifies quantity of flat washers
- Analysis:
    - Weight is volume times density of the material
    - Volume is the rim area times thickness
    - Rim area is calculated as in the next slide
  - Inputs: diameters, thickness, density, quantity
  - Outputs: weight
  - Relationships: ??

## Computing Area of a Flat Washer

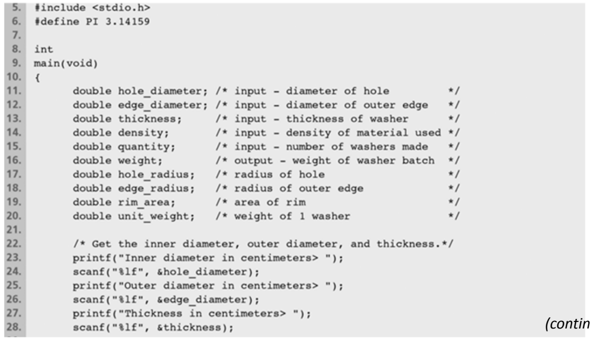


$$rim\ area = \pi(d_2/2)^2 - \pi(d_1/2)^2$$

## Case Study: Weight of Washers

- Design:
  - Initial Algorithm: ??
- Implementation:
  - next

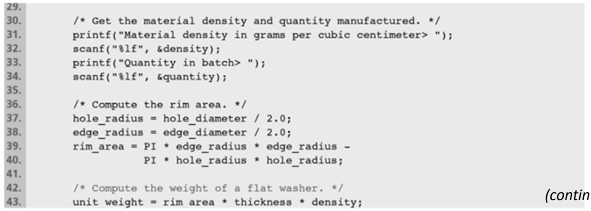
## Program Washer



```
#include <stdio.h>
#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);
```

*(continued)*

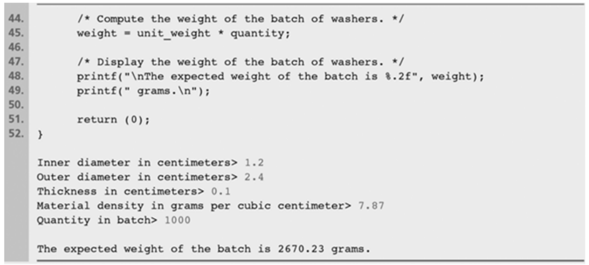
## Program Washer (cont’d)



```

      /* 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;
      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;
```

*(continued)*

## Program Washer (cont'd)



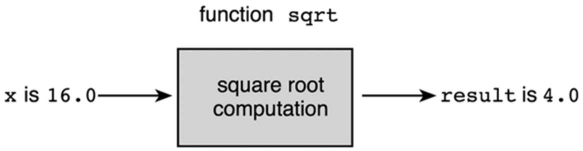
```
        /* Compute the weight of the batch of washers. */
        weight = unit_weight * quantity;

        /* 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.
```

## Library Functions

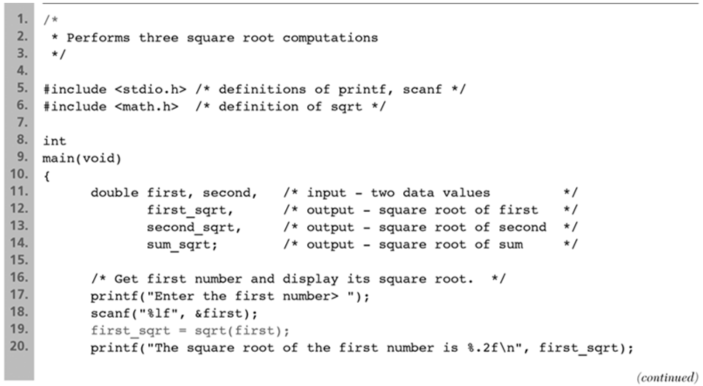
- Software engineering:
  - Goal: writing error-free codes
  - Use well tested existing codes: code reuse
  - Use predefined functions
    - EX: sqrt function in math library
    - Use it as a black box

      y = sqrt(x);
    - EX: printf and scanf in stdio library

## Function sqrt as a "Black Box"



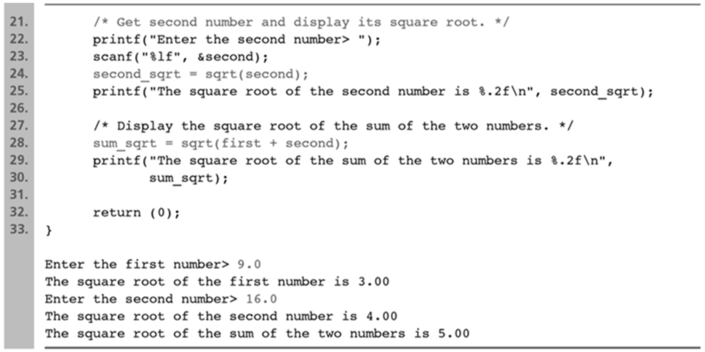
## Square Root Program



```
/*
 * Performs three square root computations
 */

#include <stdio.h> /* definitions of printf, scanf */
#include <math.h>  /* definition of sqrt */

int
main(void)
{
      double first, second,   /* input - two data values         */
             first_sqrt,      /* output - square root of first   */
             second_sqrt,     /* output - square root of second  */
             sum_sqrt;        /* output - square root of sum     */

      /* Get first number and display its square root.  */
      printf("Enter the first number> ");
      scanf("%lf", &first);
      first_sqrt = sqrt(first);
      printf("The square root of the first number is %.2f\n", first_sqrt);
```

*(continued)*

## Square Root Program (cont'd)



```
    /* Get second number and display its square root. */
    printf("Enter the second number> ");
    scanf("%lf", &second);
    second_sqrt = sqrt(second);
    printf("The square root of the second number is %.2f\n", second_sqrt);

    /* Display the square root of the sum of the two numbers. */
    sum_sqrt = sqrt(first + second);
    printf("The square root of the sum of the two numbers is %.2f\n",
           sum_sqrt);

    return (0);
}

Enter the first number> 9.0
The square root of the first number is 3.00
Enter the second number> 16.0
The square root of the second number is 4.00
The square root of the sum of the two numbers is 5.00
```

## Math Library

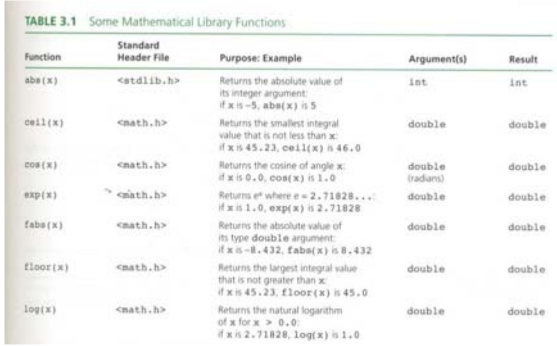
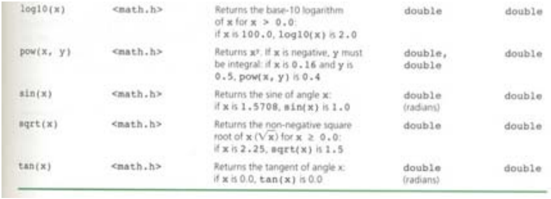


**TABLE 3.1** Some Mathematical Library Functions

| 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` |

## Math Library



| | | | | |
|---|---|---|---|---|
| `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`,<br>`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 non-negative square root of `x` ($\sqrt{x}$) for `x ≥ 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` |

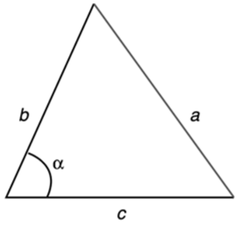
## Library Functions

- Example: Compute the roots of a quadratic equation

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

- Example: Compute the length of the third side of a triangle

$a^2 = b^2 + c^2 - 2bc \cos \alpha$

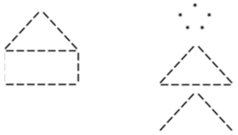


## User defined Functions

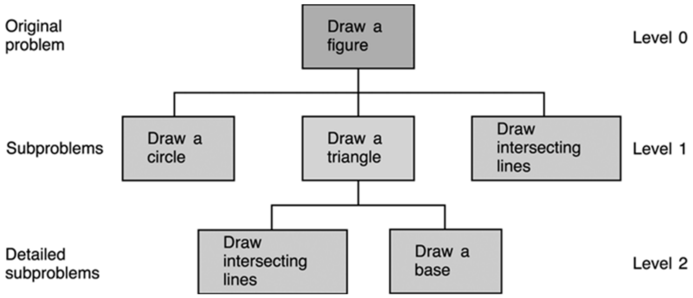
- Example: area of a circle
  area = find_area(radius);
- Example: circumference of a circle
  circum = find_circum(radius);
- Example: rim area calculation
  rim_area = find_area(edge_radius) - find_area(hole_radius);

## Case Study: Simple Diagrams

- Problem: Draw simple diagrams on your screen
  - Ex: house, person
- Analysis: Basic components
  - Circle
  - Parallel lines
  - Base line
  - Intersecting lines
- Design: Divide the problem into three subproblems
  - Draw a circle
  - Draw a triangle
  - Draw intersecting lines
    - Further refinement in triangle – see following structure chart



## 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

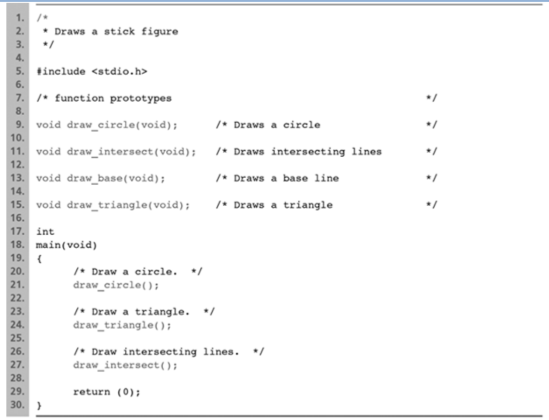
Detailed subproblems

Draw intersecting lines

Draw a base

Level 2

## Function Prototypes and Main Function



```
/*
 * Draws a stick figure
 */

#include <stdio.h>

/* function prototypes                                          */

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

void draw_intersect(void);   /* Draws intersecting lines        */

void draw_base(void);        /* Draws a base line               */

void draw_triangle(void);    /* Draws a triangle                */

int
main(void)
{
      /* Draw a circle.  */
      draw_circle();

      /* Draw a triangle.  */
      draw_triangle();

      /* Draw intersecting lines.  */
      draw_intersect();

      return (0);
}
```

# User Defined Functions

- Function prototype
  - Functions should be defined before they are used
    - Insert the whole function definition
    - Insert the function prototype
  - Defines
    - Data types of the function
    - Function name
    - Arguments and their types

    function_type function_name (argument types);

  - Ex:

    void draw_circle(void);

# User Defined Functions

- Function call
  - Calling a function

    function_name (arguments);

  - Ex:

    draw_circle();

    printf(“%d”, year);

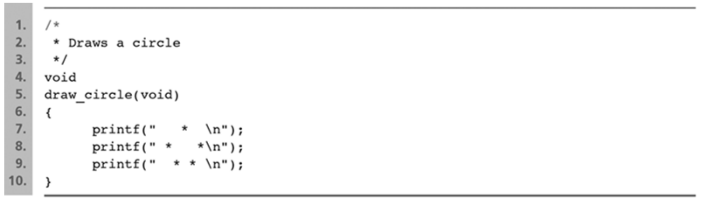
# User Defined Functions

- Function definition
  - Defines the operation of a function
  - Similar to main function

```
function_type function_name (argument list)
{
    local declerations
    executable statements
}
```

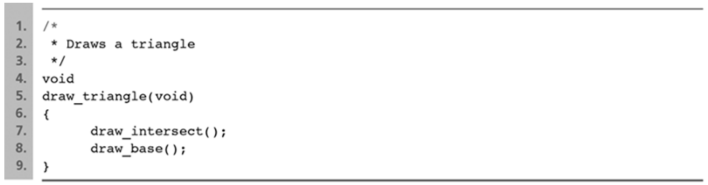
  - Function heading: similar to function prototype
  - Function body: enclosed in braces

# Function draw_circle



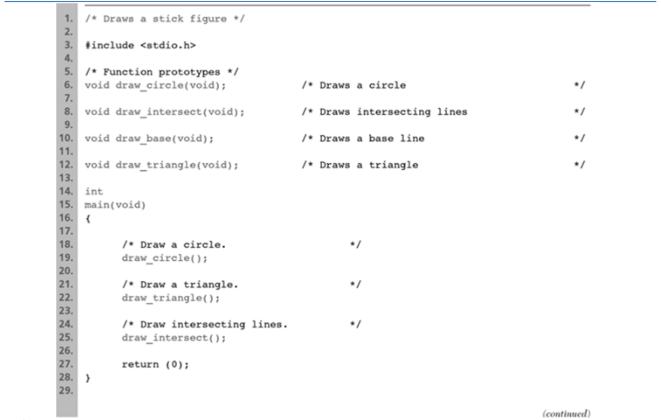
```
/*
 * Draws a circle
 */
void
draw_circle(void)
{
      printf("   *  \n");
      printf(" *   *\n");
      printf("  * * \n");
}
```

# Function draw_triangle



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

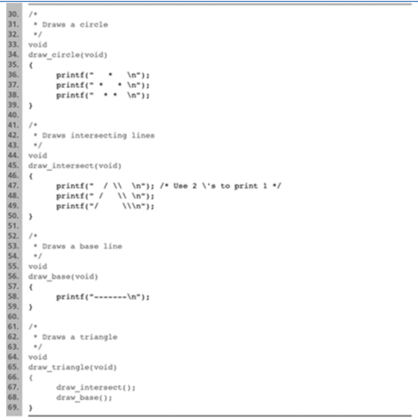
# Program to Draw a Stick Figure



```
/* Draws a stick figure */

#include <stdio.h>

/* Function prototypes */
void draw_circle(void);          /* Draws a circle                  */

void draw_intersect(void);       /* Draws intersecting lines        */

void draw_base(void);            /* Draws a base line               */

void draw_triangle(void);        /* Draws a triangle                */

int
main(void)
{

      /* Draw a circle.                 */
      draw_circle();

      /* Draw a triangle.               */
      draw_triangle();

      /* Draw intersecting lines.       */
      draw_intersect();

      return (0);
}

```

(continued)

# Program to Draw a Stick Figure

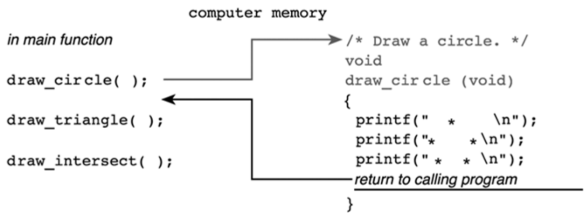


```
/*
 * Draws a circle
 */
void
draw_circle(void)
{
      printf("   *   \n");
      printf(" *   * \n");
      printf("  * *  \n");
}

/*
 * 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();
}
```

# Flow of Control

- Compiling the program:
  - Function prototypes: compiler knows the functions
    - enables compiler to translate function calls
  - Function definition: translates the code of the function
    - Allocates memory needed
  - Function call: Transfers of the control to the function
  - End of the function: Transfer of the control back to the calling statement
    - Releases the local memory

## Flow of Control



computer memory

*in main function*

```
draw_circle( );

draw_triangle( );

draw_intersect( );
```

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

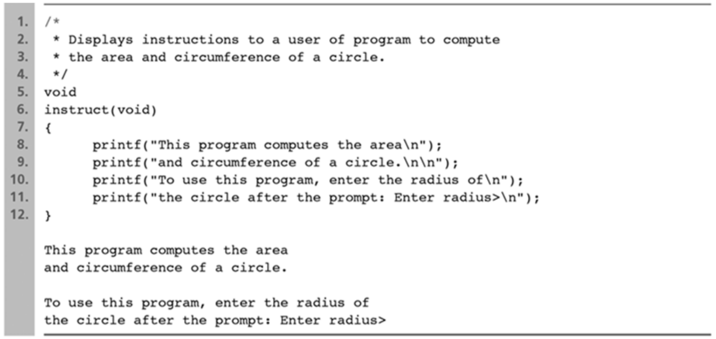
*return to calling program*

```
}
```

## Advantages of Functions

- For team of programmers:
  - Dividing programming tasks to the programmers
- Procedural abstraction
  - Move the details of the operation to the functions
  - Focus on the main operations
- Code reuse
  - In a program
  - In other programs
    - Well tested functions

## Function instruct



```
/*
 * Displays instructions to a user of program to compute
 * the area and circumference of a circle.
 */
void
instruct(void)
{
      printf("This program computes the area\n");
      printf("and circumference of a circle.\n\n");
      printf("To use this program, enter the radius of\n");
      printf("the circle after the prompt: Enter radius>\n");
}
```

```
This program computes the area
and circumference of a circle.

To use this program, enter the radius of
the circle after the prompt: Enter radius>
```

## Functions with Input Arguments

- Functions are building blocks to construct large programs
  - Like Lego blocks

- Arguments:
  - to carry information to functions : input arguments
  - to return multiple results : output arguments
- Arguments makes functions more versatile
  - Manipulate different data at each call

rim_area = find_area(edge_radius) - find_area(hole_radius);

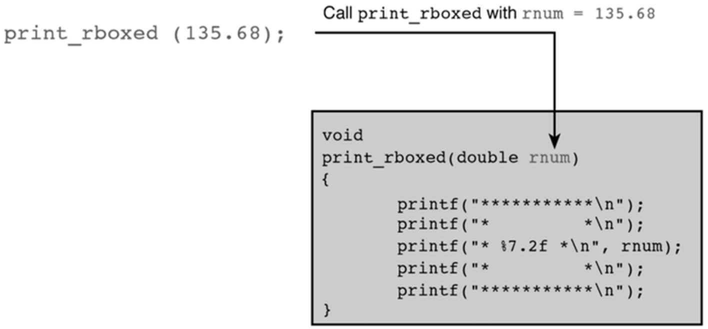
## Function print_rboxed

```
/*
 * Displays a real number in a box.
 */

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

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

## Executing print_rboxed (135.68);

- Actual parameter: 135.68

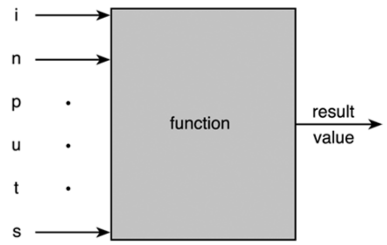


```
print_rboxed (135.68);
```

Call print_rboxed with rnum = 135.68

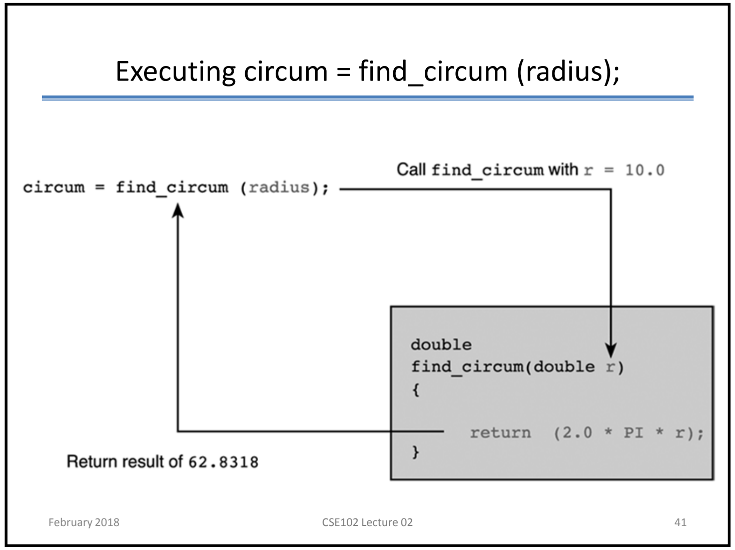
```
void
print_rboxed(double rnum)
{
        printf("***********\n");
        printf("*          *\n");
        printf("* %7.2f *\n", rnum);
        printf("*          *\n");
        printf("***********\n");
}
```

## Function with Input Arguments and Result



## Functions find_circum and find_area

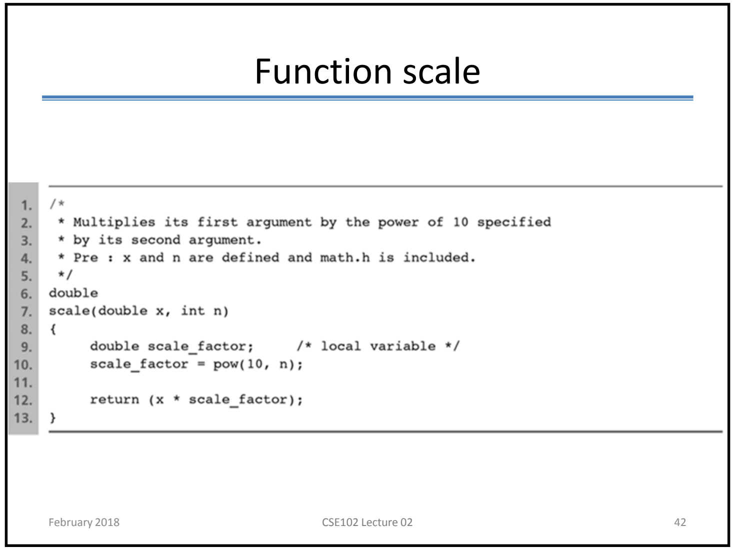
```
/*
 * Computes the circumference of a circle with radius r.
 * Pre:  r is defined and is > 0.
 *       PI is a constant macro representing an approximation of pi.
 */
double
find_circum(double r)
{
      return (2.0 * PI * r);
}

/*
 * Computes the area of a circle with radius r.
 * Pre:  r is defined and is > 0.
 *       PI is a constant macro representing an approximation of pi.
 *       Library math.h is included.
 */
double
find_area(double r)
{
      return (PI * pow(r, 2));
}
```



## Executing circum = find_circum (radius);

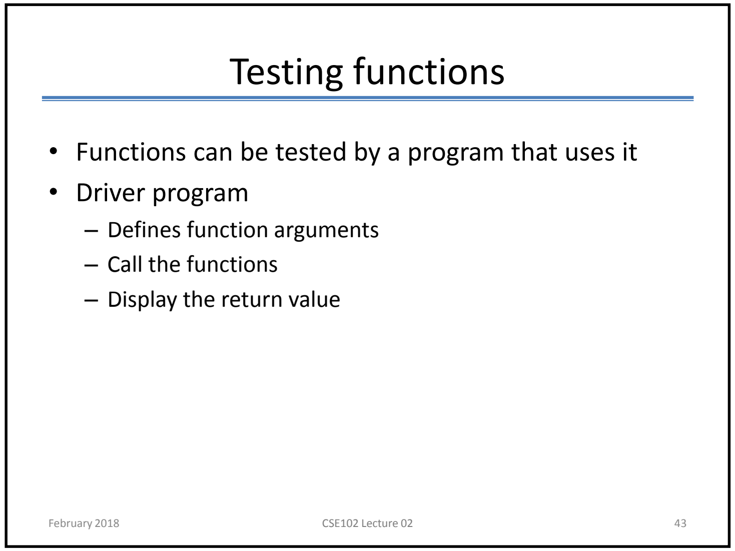
```
circum = find_circum (radius);
```

```
double
find_circum(double r)
{
      return  (2.0 * PI * r);
}
```



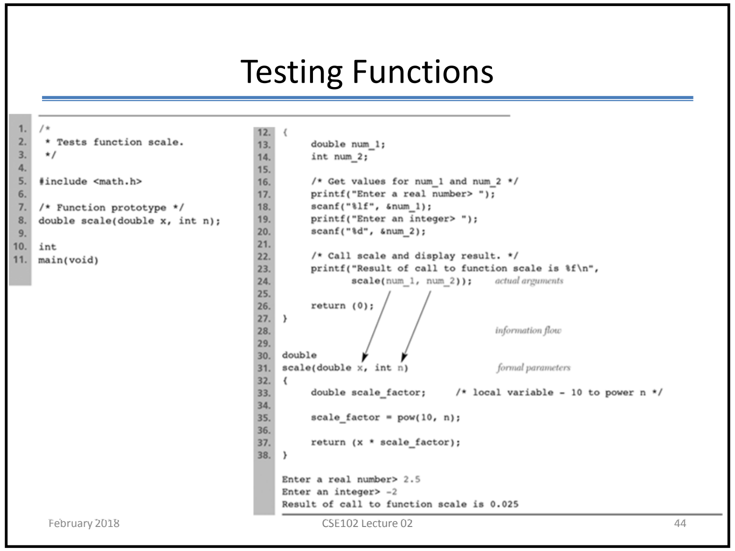
## Function scale

```
/*
 * Multiplies its first argument by the power of 10 specified
 * by its second argument.
 * Pre : x and n are defined and math.h is included.
 */
double
scale(double x, int n)
{
     double scale_factor;     /* local variable */
     scale_factor = pow(10, n);

     return (x * scale_factor);
}
```



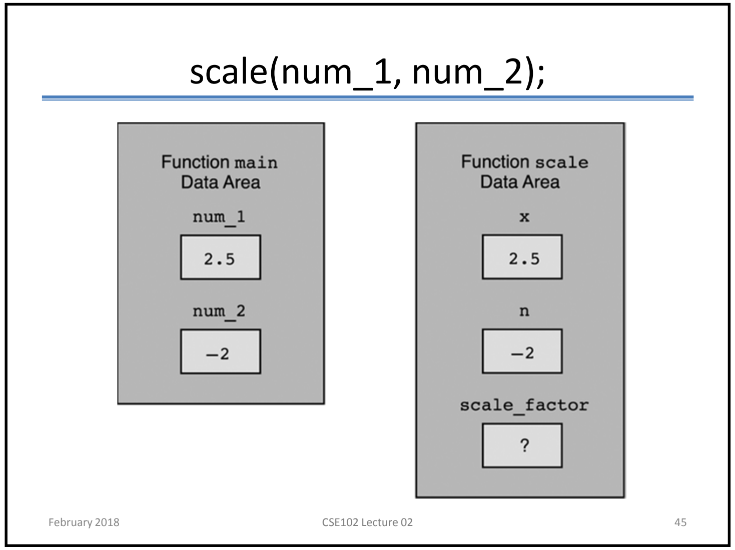
## Testing functions

- Functions can be tested by a program that uses it
- Driver program
  - Defines function arguments
  - Call the functions
  - Display the return value



## Testing Functions

```
/*
 * Tests function scale.
 */

#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);
}

Enter a real number> 2.5
Enter an integer> -2
Result of call to function scale is 0.025
```



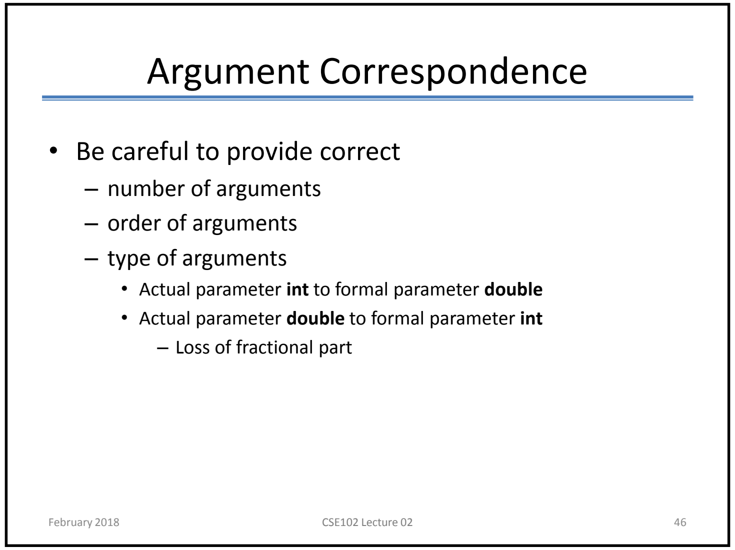
## scale(num_1, num_2);

Function main
Data Area

| num_1 | num_2 |
|---|---|
| 2.5 | −2 |

Function scale
Data Area

| x | n | scale_factor |
|---|---|---|
| 2.5 | −2 | ? |



## Argument Correspondence

- Be careful to provide correct
  - number of arguments
  - order of arguments
  - type of arguments
    - Actual parameter **int** to formal parameter **double**
    - Actual parameter **double** to formal parameter **int**
      - Loss of fractional part