Introduction to Computers II
Lecture 4

Dr Ali Ziya Alkar

Dr Mehmet Demirer

- Contents:
  - Utilizing the existing information
  - Top-down design
    - Start with the broadest statement of the problem
    - Works down to more detailed sub-problems.
  - Modular programming

Existing Information

- Programmers seldom start from scratch when writing a program.
- Typically, you will reuse work that has been done by yourself or others
  - For example, using `printf` and `scanf`
- You start with your algorithm, and then implement it piece by piece
  - When implementing these pieces, you can save effort by reusing functionality.

Utilizing existing information

- Generated system documents
  - Problem description (data requirement)
  - Solution algorithm
- Strategy
  - Editing the data requirements to conform constant and variable definitions
  - Using initial algorithm and its refinements (formulas) as the program comments.

Case Study

- Problem:

  get the radius of a circle, compute and display the circle's area and circumference.

Analysis

- Data requirements:
  - Constant

    PI = 3.14159
  - Input

    radius
  - Output

    area

    circumference
  - Relevant formulas

    area of a circle = PI * $radius^2$

    circumference = 2 * PI * radius

Design

- Algorithm
  - Get the circle radius
  - Calculate the area and circumference
  - Display the results
- Refinements:
  - Assign PI * radius * radius to area
  - Assign 2 * PI * radius to circumference

```
/*
 * Calculate and display the area and circumference of a circle
 */
#include <stdio.h>
#define PI 3.14159    /* constant PI */

int main(void)
{
 double radius;                    /* input – radius of a circle */
 double area;                      /* output – area of a circle */
 double circum;                    /* output – circumference */

 /* Get the radius */

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

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

 /* Display the area and circumference */

 return (0);
}
```

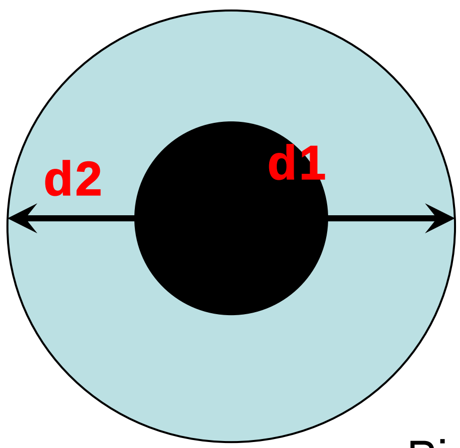
```c
/*
 * Calculate and display the area and circumference of a circle
 */
#include <stdio.h>
#define PI 3.14159        /* constant PI */

int main(void)
{
 double radius;                       /* input – radius of a circle */
 double area;                         /* output – area of a circle */
 double circum;                       /* output – circumference */

 /* Get the 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);
}
```

Solution reuse

- Use existing information (the solution for one problem) to solve another.

Case Study

- Problem: computes the weight of a specified quantity of flat washers.



Rim area = PI(d2/2)^2 - PI(d1/2)^2

Data requirement

- Problem constant

  PI 3.14159

- Problem input

  double hole_diameter, edge_diameter

  double thickness, density, quanlity

- Problem output

  double weight

Data requirement (Cont.)

- Program variables

  double hole_radius, edge_radius

  double rim_area, unit_weight

- Relevant formulas

  area of a circle = PI * radius^2

  radius of a circle = diameter / 2

  rim area = area(outer) – area(inner)

  unit weight = rim area * thickness * density

Design

1. Get the diameters and thickness, density, quantity
2. Compute the rim area
3. Compute the weight of one flat washer
4. Compute the weight of the batch of washers
5. Display the weight of the batch of washers

Refinement

- 3.1 compute radius
- 3.2 rim_area is PI * edge_radius * edge_radius – PI * hole_radius * hole_radius
- 4.1 unit_weight is rim_area * thickness * density

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

#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);

      /* 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)

```
    /* 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

- Predefined Functions and Code Reuse
- C Library Functions
- A Look at Where We Are Heading

Library functions

- Code-reuse

  benefits: avoid redevelopment.

  avoid errors.
- C providing many predefined functions that can be used to perform certain tasks.
- For example, mathematic computations.

  sqrt(x)

Example

- Display the square root of two numbers provided as the input data (first and second) and the square root of their sum.

```
/*
 * Perform three square root computation
 */
#include <stdio.h>
#include <math.h>

int main(void)
{
 double first, second,    /* input – two data value */
 double first_sqrt;                        /* output – square root of first */
 double second_sqrt;     /* output – square root of second */
 double 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);

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

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

 return (0);
}
```

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 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`

Standard math functions in C

- Comments:
  - Type conversion

    int → double, no problem

    double → int, lost fractional part
  - Other restrictions

    arguments for log and log10 must be positive

    arguments for sqrt can not be negative

Example

- Using pow and sqrt functions to compute the roots of equation: $ax^2 + bx + c = 0$
- disc= pow(b,2) – 4 * a * c

  root_1 = (-b + sqrt(disc)) / (2 * a)

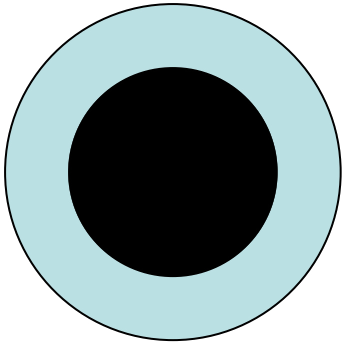
  root_2 = (-b - sqrt(disc)) / (2 * a)

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

Using your own functions

- find_area(r) returns the area
  find_circum(r) returns the circumference

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



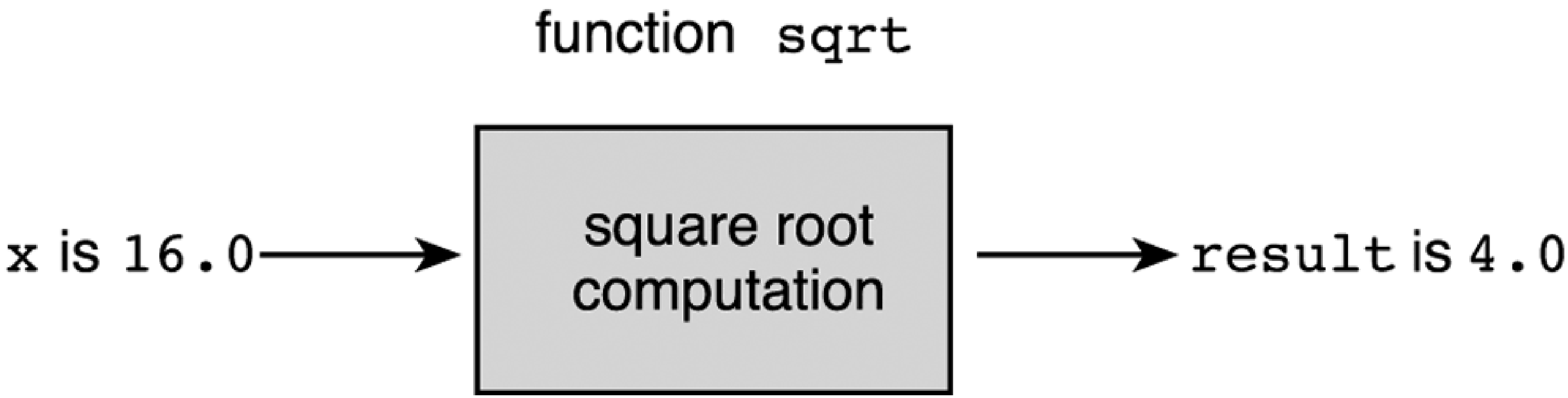
Predefined Functions and Code Reuse

- The primary goal of software engineering is to write error-free code.
- Reusing code that has already been written & tested is one way to achieve this.
  - ”Why reinvent the wheel?”
- C promotes reuse by providing many predefined functions. e.g.
  - Mathematical computations.
  - Input/Output: e.g. `printf`, `scanf`
- C’s standard math library defines a function named `sqrt` that performs the square root computation. It is called like:

```
y = sqrt(x)
```

- This passes the argument `x` to the function `sqrt`. After the function executes, the result is assigned to the left hand side variable `y`.

Function sqrt.



Function sqrt as a black box.

```
first_sqrt = sqrt(25.0);

second_sqrt = sqrt(second);

third_sqrt = sqrt(first+second);

Z = 5.7 + sqrt(num);
```

C Library Functions

- The next slide lists some commonly used mathematical functions (Table 3.1 in the text)
- In order to use them you must use `#include` with the appropriate library.
  - Example, to use function `sqrt` you must include `math.h`.
- If one of the functions in the next slide 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.
  - Conversion of type `int` to type `double` cause no problems
  - Conversion of type `double` to type `int` leads to the loss of any fractional part.
- Make sure you look at documentation for the function so you use it correctly.

Some Mathematical Library Functions

Function	Header File	Purpose	Arguments	Result
abs(x)	<stdlib.h>	Returns the absolute value of its integer argument x.	int	int
sin(x),cos(x), tan(x)	<math.h>	Returns the sine, cosine, or tangent of angle x.	double (in radians)	double
log(x)	<math.h>	Returns the natural log of x.	double (must be positive)	double
pow(x,y)	<math.h>	Returns x^y	double, double	double
sqrt(x)	<math.h>	$\sqrt{x}$	double (must be positive)	double

Function we have seen so far

- We’ve seen a few other I/O library functions
  - printf, scanf
  - fprintf, fscanf
  - fopen, fclose
  - To use them, have to use `#include <stdio.h>`
- Mathematical Functions
  - sqrt, pow, sin, cos etc.
  - To use them, have to use `#include <math.h>`
- We use C’s predefined functions as building blocks to construct a new program.

Where We are Heading?

- C also allows us to write **our own functions**.
- We could write our own functions to find area and find circumference of a circle.
  - 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.
  - The following statements can be used to find these values.

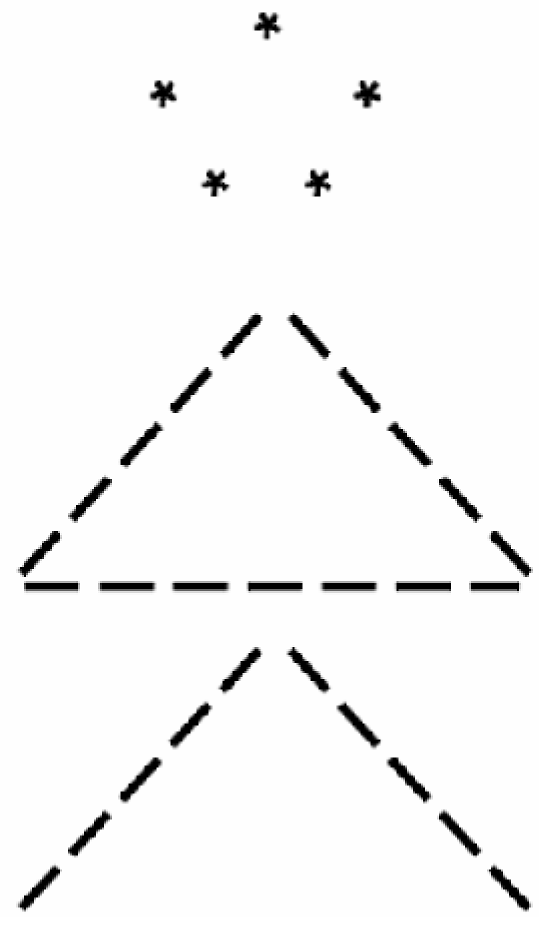
```
area = find_area(r);
circum = find_circum(r);
```

Top Down Design

- Use the top-down approach for analyzing all complex problems.
- The solution to any complex problem is conceptually simpler if viewed hierarchically as a tree of subproblems.
- It is more convenient to design your solution first with rough blocks, and then refine them gradually.
- You first break a problem up into its major subproblems and then solve those subproblems to derive the solution to the original problem.

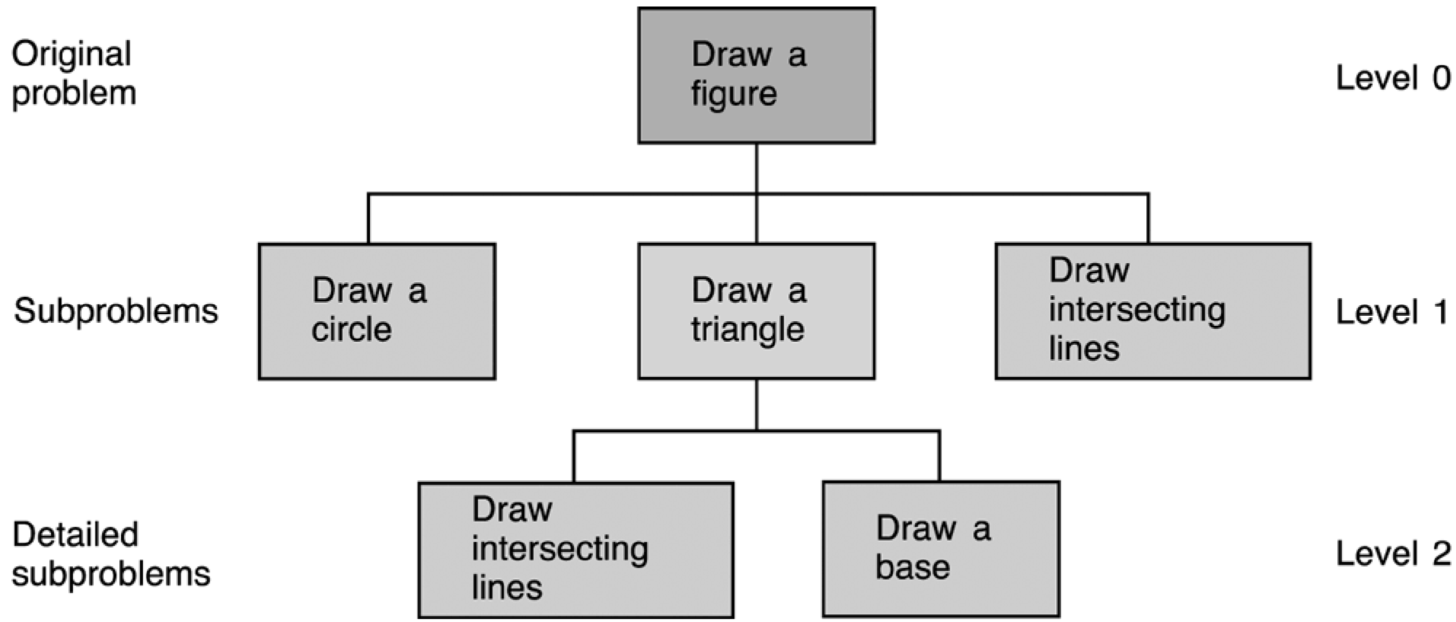
Example: Top-down approach

- Drawing a Stick Figure in the screen as an example of problem solving with Top-down design approach.
- We can draw this figure with the basic three components
  - Circle
  - Intersecting lines
  - Base line

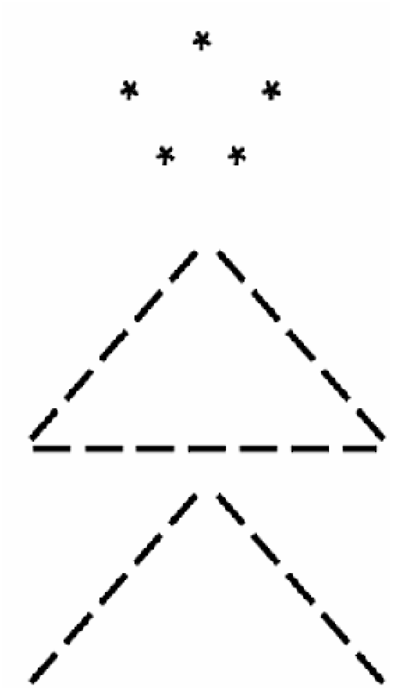


Structure Chart for Drawing Stick Figure

- Structure chart is an software engineering documentation tool.



Function `main` for Stick Figure



```c
/*
 * 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);
}
```

Void Functions without Arguments

- Functions that do not have arguments and return no values.
  - Output is normally placed in some place else (e.g. screen)
- Why would you want to do these?
  - They can help with top down design of your program.
  - Instead of writing all of your code in your `main` function, separate it into separate functions for each subproblem.

Void Functions Without Arguments

- Function Prototypes
- Function Definitions
- Local variables.
- Placement of Functions in a Program
- Program Style
- Advantages of Using Function Subprograms
  - **Procedural Abstraction**
  - **Reuse of Functions.**

Function Prototype (1)

```
/* This program draws a circle in the screen */

#include <stdio.h>

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

int main(void)
{
   draw_circle();
   return (0);
}

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

Function Prototype (2)

- Like other identifiers in C, a function must be declared before it can be referenced.
- To do this, you can add a **function prototype** before `main` to tell the compiler what functions you are planning to use.
- A function prototype tells the C compiler:

1. The data type the function will return
   - For example, the `sqrt` function returns a type of double.
2. The function name
3. Information about the arguments that the function expects.
   - The `sqrt` function expects a double argument.

- So the function prototype for `sqrt` would be:

```
double sqrt(double);
```

More on void Functions

- `void draw_circle(void);` is a void function
  - **Void function -** does not return a value
    - The function just does something without communicating anything back to its caller.
  - If the arguments are void as well, it means the function doesn’t take any arguments.
- Now, we can understand what our main function means:

```
int main(void)
```

- This means that the function `main` takes no arguments, and returns an `int`

Function Definition (1)

```
/* This program draws a circle in the screen */
#include <stdio.h>
/* Function prototypes */
void draw_circle(void); /* Draws a circle */

int main(void)
{
   draw_circle();
   return (0);
}
/* Draws a circle */
void draw_circle(void) {
   printf(“  * *\n”);
   printf(“ *   *\n”);
   printf(“  * *\n”);
}
```

Function Definition (2)

- The prototype tells the compiler what arguments the function takes and what it returns, but not what it does.
- We define our own functions just like we do the `main` function
  - **Function Header** – The same as the prototype, except it is not ended by the symbol ;
  - **Function Body –** A code block enclosed by {}, containing variable declarations and executable statements.
- In the function body, we define what actually the function does
  - In this case, we call `printf` 3 times to draw a circle.
  - Because it is a void function, we can omit the return statement.
- Control returns to `main` after the circle has been drawn.

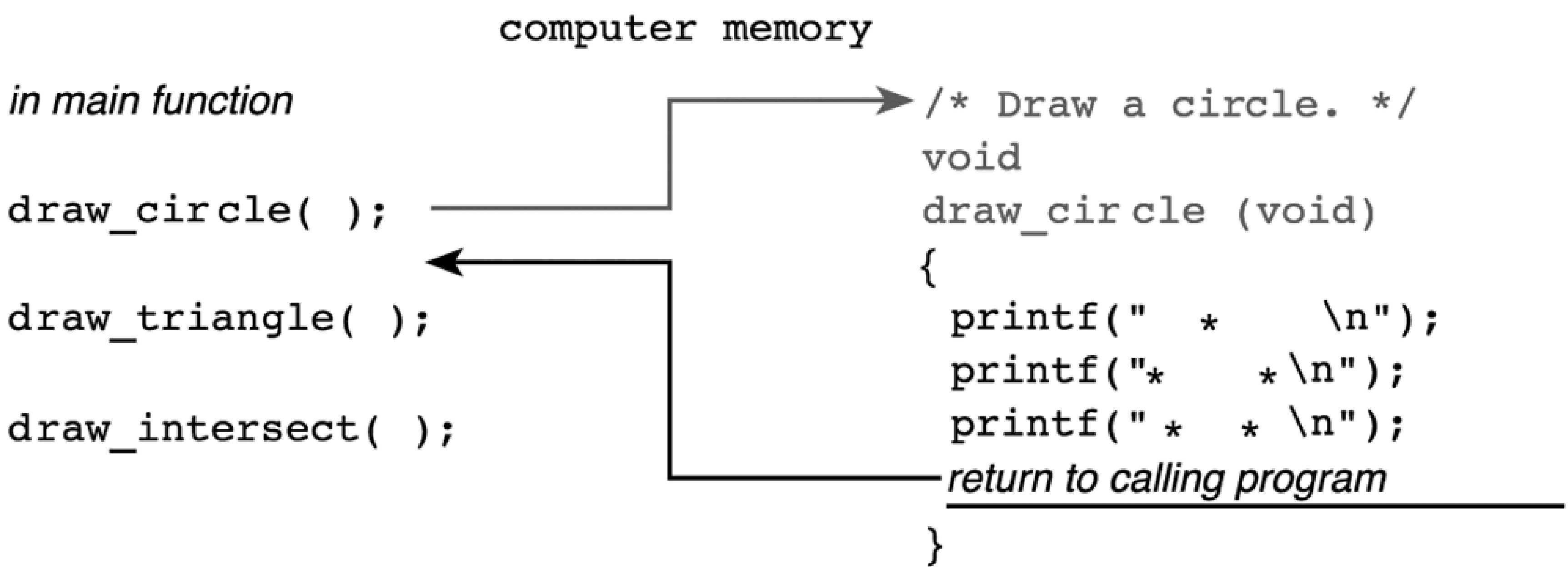
Placement of Functions in a program

- In general, we will declare all of our function prototypes at the beginning (after `#include` or `#define`)
- This is followed by the `main` function
- After that, we define all of our functions.
- However, this is just a convention.
- As long as a function's prototype appears before it is used, it doesn't matter where in the file it is defined.
- The order we define them in does not have any impact on how they are executed

Execution Order of Functions

- Execution order of functions is determined by the order of execution of the function call statements.
- Because the prototypes for the function subprograms appear before the `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.
- After compiling the `main` function, the compiler translates each function subprogram.
- At the end of a function, control always returns to the point where it was called.

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



Program Style

- Each function should begin with a comment that describes its purpose.
- If the function subprograms were more complex, we would include comments on each major algorithm step just as we do in function `main`.
- It is recommended that you put prototypes for all functions at the top, and then define them all after `main`.

Advantages of Using Function Subprograms

- There are two major reasons:

1. A large problem can be solved easily by breaking it up into several small problems and giving the responsibility of a set of functions to a specific programmer.
    - It is easer to write two 10 line functions than one 20 line one and two smaller functions will be easier to read than one long one.
2. They can simplify programming tasks because existing functions can be reused as the building blocks for new programs.
    - Really useful functions can be bundled into libraries.

Procedural Abstraction

- **Procedural Abstraction –** A programming technique in which a `main` function consists of a sequence of function calls and each function is implemented separately.
- All of the details of the implementation to a particular subproblem is placed in a separate function.
- The main functions becomes a more abstract outline of what the program does.
  - When you begin writing your program, just write out your algorithm in your main function.
  - Take each step of the algorithm and write a function that performs it for you.
- Focusing on one function at a time is much easier than trying to write the complete program at once.

Reuse of Function Subprograms

- Functions can be executed more than once in a program.
  - Reduces the overall length of the program and the chance of error.
- Once you have written and tested a function, you can use it in other programs or functions.

A good use of void functions – A separate function to display instructions for the user.

```c
/*
 * 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>
```