Programming for Engineers

Lecture 4: Function & Recursion

Course ID: EE057IU

Lecture Outline

- Functions (Chapter 5)
- Recursion (Chapter 5.14)

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

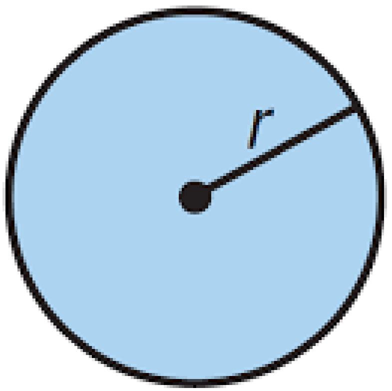
Example: Area and circumference of a circle

```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);
}
```

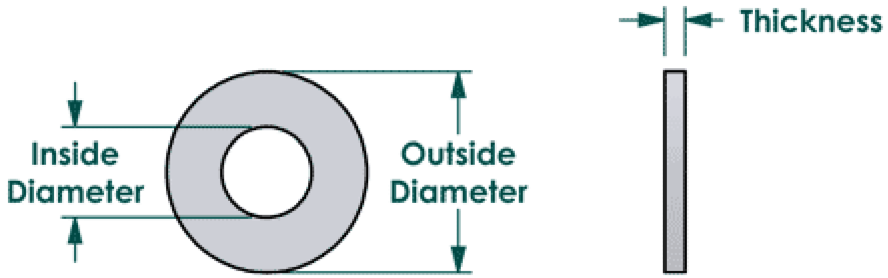
radius r

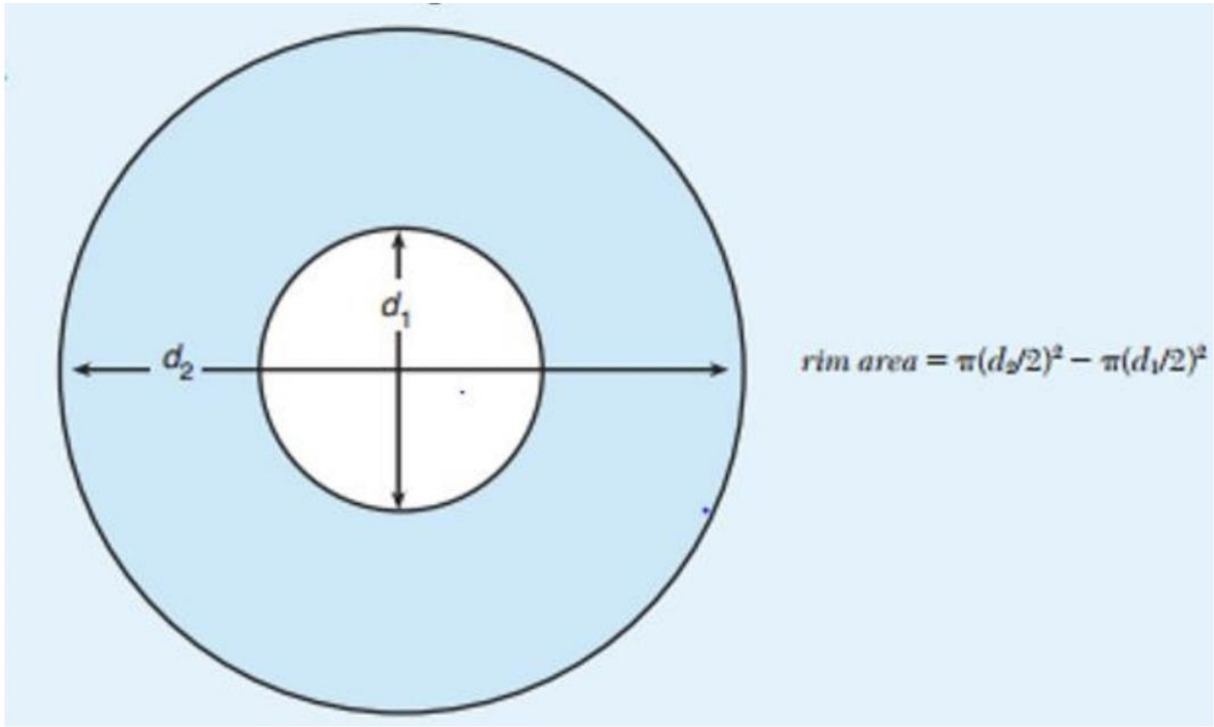
$$C = 2\pi r$$

$$A = \pi r^2$$



Example: Computing Rim Area of a Flat Washer





Example: Computing Rim Area of a Flat Washer

```c
#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;
```

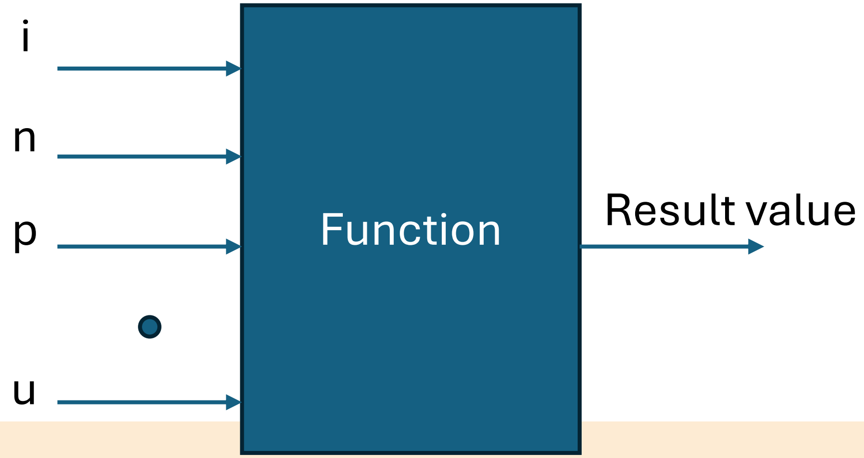
```c
    rim_area = PI * edge_radius * edge_radius - PI * hole_radius * hole_radius;

    /* 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 grams.\n", weight);

    return 0;
}
```

Functions

- Functions allow us to
  - Modularize a program
  - Reuse the code (Avoid Reinventing the Wheel)

- Two types:
  - Programmer/user write, called **programmer-defined** functions
  - Prepackaged functions in C standard library

- Structure
  - Input variables
  - Output value, which is returned
  - Function body (series of statements)



Functions

➢ Characteristics

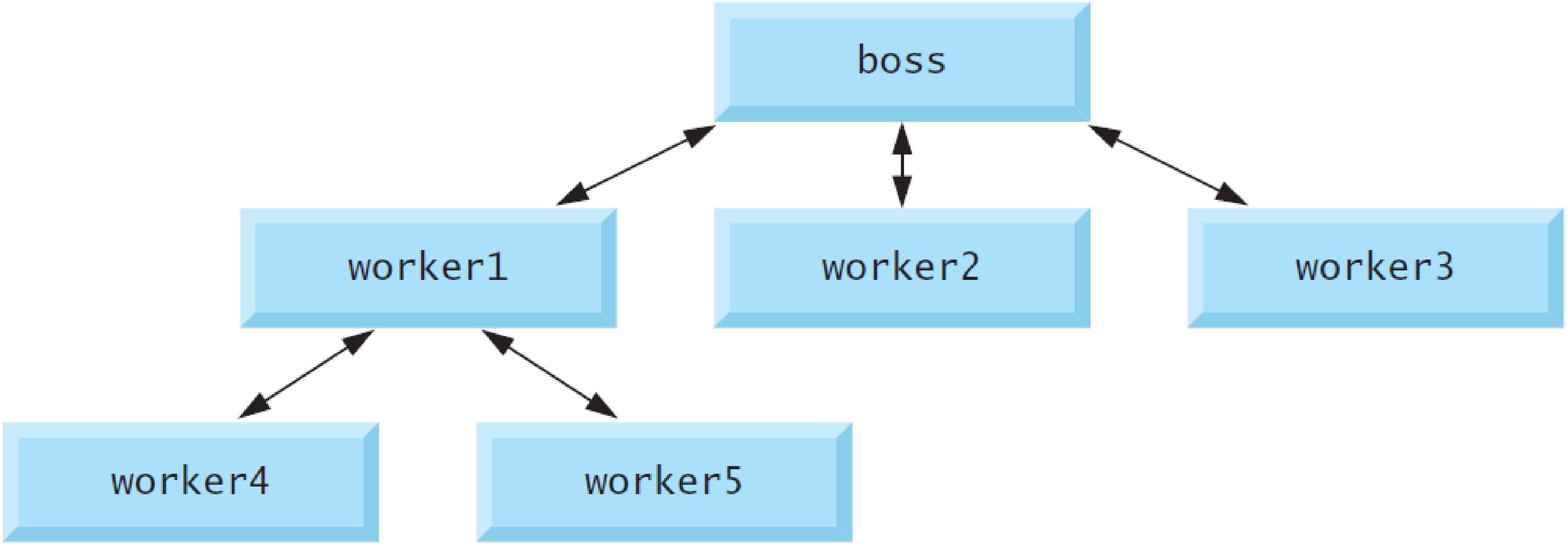
- Can be called as many points in a program
- Be written only once
- The statements are hidden from other functions

➢ Calling and Returnning from Functions

- Functions are invoked by a function call, which specifies the function name and provides information (as arguments)

Functions – 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



Functions – Math Library Functions (Chap 5.3)

- C's math library functions (header math.h)
  - Allow you to perform mathematical calculations
  - More complicated calculations can be found in <complex.h>

Table 1: Mathematical Functions in C

Function	Description	Example
sqrt(x)	square root of x	sqrt(900.0) is 30.0
cbrt(x)	cube root of x (C99 and C11 only)	cbrt(27.0) is 3.0
exp(x)	exponential function e^x	exp(1.0) is 2.718282
log(x)	natural logarithm of x (base e)	log(2.718282) is 1.0
log10(x)	logarithm of x (base 10)	log10(10.0) is 1.0
fabs(x)	absolute value of x as a floating-point number	fabs(-13.5) is 13.5
ceil(x)	rounds x to the smallest integer not less than x	ceil(9.2) is 10.0
floor(x)	rounds x to the largest integer not greater than x	floor(9.2) is 9.0
pow(x, y)	x raised to power y (x^y)	pow(2, 7) is 128.0

Table 2: Trigonometric Functions in C

Function	Description	Example
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 (in radians)	sin(0.0) is 0.0
cos(x)	trigonometric cosine of x (in radians)	cos(0.0) is 1.0
tan(x)	trigonometric tangent of x (in radians)	tan(0.0) is 0.0

Function Definitions

- Each program include a main function called standard library functions
- All variable inside function called local variable, can accessed only within function
- Each function has parameters to enable communication between calling function and called function

Format of a function definition:

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

Example of User-defined Function

square Function

```c
// fig05_01.c
// Creating and using a function.
#include <stdio.h>

int square(int number); // 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 number) { // number is a copy of the function's argument
   return number * number; // returns square of number as an int
}
```

```
1   4   9   16   25   36   49   64   81   100
```

Function Definition – Analyzing

```c
// fig05_01.c
// Creating and using a function.
#include <stdio.h>

int square(int number); // 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 number) { // number is a copy of the function's argument
   return number * number; // returns square of number as an int
}
```

Function square is invoked or called

Function square
- receives parameter x
- Passes the variable x to calculate the statement number*number

Function Definition – Analyzing

```c
// fig05_01.c
// Creating and using a function.
#include <stdio.h>

int square(int number); // 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 number) { // number is a copy of the function's argument
   return number * number; // returns square of number as an int
}
```

Function square

- x in the iterations is defined as typedef int
- Function also expect the integer variable

Function Definition – Analyzing

```c
// fig05_01.c
// Creating and using a function.
#include <stdio.h>

int square(int number); // 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 number) { // number is a copy of the function's argument
   return number * number; // returns square of number as an int
}
```

Function square

(1) Perform the calculation inside the statement

(2) Pass the value back to calling function via return

(3) Keyword int at the beginning of function indicate function need to return an integer to calling function

Function Definition – Analyzing

```c
// fig05_01.c
// Creating and using a function.
#include <stdio.h>

int square(int number); // 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 number) { // number is a copy of the function's argument
   return number * number; // returns square of number as an int
}
```

Function prototype

- Informs the compiler that square expects to receive an integer variable from the caller
- Informs compiler that square return an integer result to the caller

Function Definition … continue

- The compiler refers that
  - Correct return type
  - Correct number of arguments
  - Correct argument types
  - Arguments are in correct order

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

- 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 … continue

- 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.

Function Definition – Return Control

- Returns control to calling function after function execution
  - when a function is called, the program's execution **jumps** to that function. When the function is finished, execution must **jump back** to the exact place where it was called.
  - For, **void function**, it simply executes all statements in its body. Once the closing brace (}) of the function is reached, control automatically returns to the calling function
  - The statement return; It can be placed anywhere in the function body to stop execution early. It does not send any data back.
  - Returns the value of the expression to the caller by the statement - return *expression*;

Function Definition – *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.

Example of User-defined Function

maximum Function

```c
// fig05_02.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 = 0; // first integer entered by the user
   int number2 = 0; // second integer entered by the user
   int number3 = 0; // 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 maximum definition
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
```

```
Enter three integers: 47 32 14
Maximum is: 47
```

```
Enter three integers: 35 8 79
Maximum is: 79
```

In-class Practice

Write a program that inputs a series of integers and passes them one at a time to function isEven, which uses the remainder operator to determine whether an integer is even. The function should take an integer argument and return 1 if the integer is even and 0 otherwise.

Random-Number Generation (Chap 5.10)

- Why we need random-number generation?
  - Simulation, modeling, games, sampling and statistics, testing and debugging
  - Game: a program that can simulate coin tossing “head” or “tail”
  - Game: a program that can simulate dice-rolling game that provides randomly 6 integers 1 to 6.

- *rand* function
  - Defined in <stdlib.h> header
  - Syntax: i = *rand*();
  - Get a random number in a range [0, N]: i = *rand*() %(N+1);

Scaling and Shifting

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

RAND_MAX: At least 32767

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

- Shift
  - `r_shifted = r_scaled+M;`

Random Number Generation Code

```c
// fig05_04.c
// Shifted, scaled random integers produced by 1 + rand() % 6.
#include <stdio.h>
#include <stdlib.h>

int main(void) {

    for (int i = 1; i <= 10; ++i) {
        printf("%d  ", 1 + (rand() % 6)); // display random die value
    }

    puts("");
}
```

```
6   6   5   5   6   5   1   1   5   3
```

rand() is not truly random

- When the program starts, the **seed** establishes the starting point for a very long, predetermined sequence of pseudo-random numbers.

- Every time **`rand()` is called** in the program, it computes and returns the **next number in that sequence**.

- That's why when you call `rand()` 3 times, it returns 3 different values (the 1st, 2nd, and 3rd numbers in the sequence), but when you stop and recompile the code, run it again, it keeps giving out the same 3 numbers (because the starting seed defaults to 1 and the sequence is the same)

```c
int main() {
for(int i=1;i<=3;i++)
    printf("%d ", 1+ rand()%10);
    return 0;
```

Output

```
4 7 8
```

Run again

```c
int main() {
for(int i=1;i<=3;i++)
    printf("%d ", 1+ rand()%10);
    return 0;
```

Output

```
4 7 8
```

Run again

```c
int main() {
for(int i=1;i<=3;i++)
    printf("%d ", 1+ rand()%10);
    return 0;
```

Output

```
4 7 8
```

$srand$ – Randomizing with a seed

- Function *srand()* takes an int argument and **seeds** function *rand* to produce a different sequence of random numbers for each program execution.

```
// fig05_06.c
// Randomizing the die-rolling program.
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    printf("%s", "Enter seed: "); // Prompt for entering seed

    int seed = 0; // Number used to seed the random-number generator
    scanf("%d", &seed); // Read the seed from the user

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

    // Loop to generate and display 10 random die values
    for (int i = 1; i <= 10; ++i) {
        printf("%d ", 1 + (rand() % 6)); // Display random die value between 1 and 6
    }

    puts(""); // Print a newline character
    return 0; // Return success
}
```

```
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
```

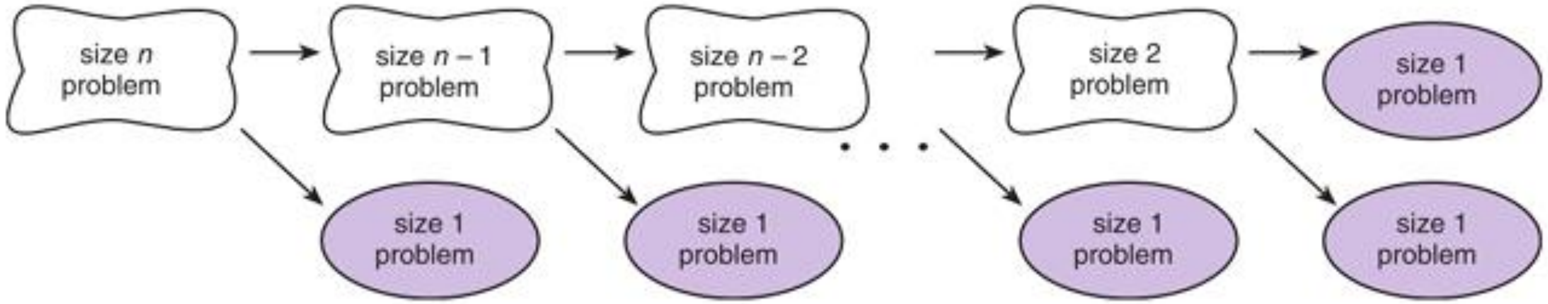
```
Enter seed: 67
6  1  4  6  2  1  6  1  6  4
```

Try this as a better solution for random number generator

```
#include <time.h>
srand(time(NULL));
```

Recursion (Chap 5.14)

- A recursive function is a function that calls itself either directly or indirectly through another function.
- Nature of recursion
  - One or more simple cases of the problem have a straightforward, nonrecursive solution.
  - The other cases can be redefined in terms of problems that are closer to the simple cases.



Recursively Calculating Factorials

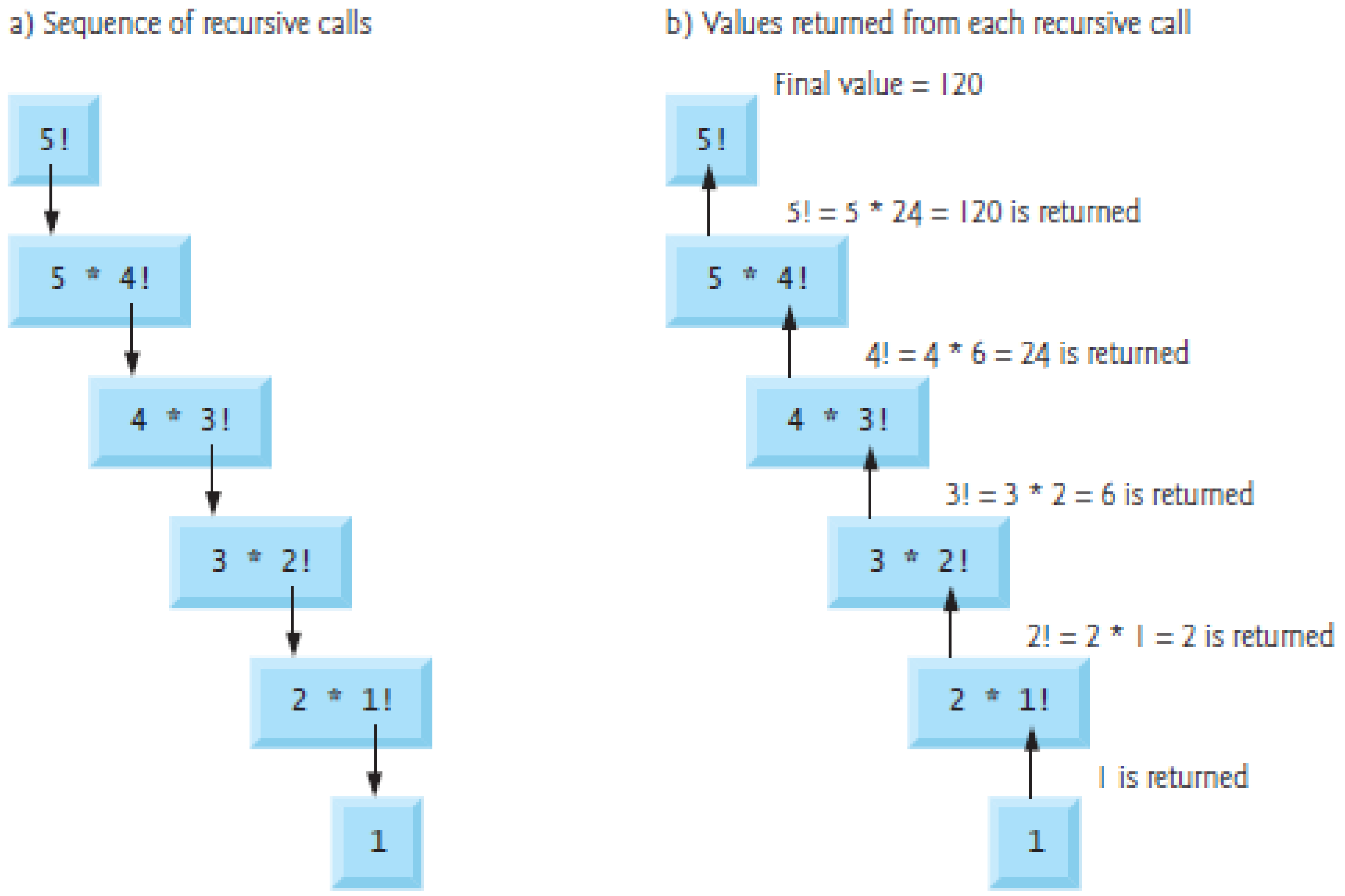
- The factorial of a non-negative integer n:
  - n! (pronounced “n factorial”)
  - $n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 1$
  - 1! equal to 1, 0! defined to be 1

- A recursive definition of the factorial function can be observed by the relationship
  - $n! = n \cdot (n-1)!$
  - For example: $5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$

    $5! = 5 \cdot (4 \cdot 3 \cdot 2 \cdot 1)$

    $5! = 5 \cdot (4!)$

Recursive evaluation of 5!



Recursive factorial function

```c
// fig05_09.c
// Recursive factorial function.
#include <stdio.h>

unsigned long long int factorial(int number);

int main(void) {
   // calculate factorial(i) and display result
   for (int i = 0; i <= 21; ++i) {
      printf("%d! = %llu\n", i, factorial(i));
   }
}

// recursive definition of function factorial
unsigned long long int factorial(int number) {
   if (number <= 1) { // base case
      return 1;
   }
   else { // recursive step
      return (number * factorial(number - 1));
   }
}
```

Recursive factorial function

```
0!  = 1
1!  = 1
2!  = 2
3!  = 6
4!  = 24
5!  = 120
6!  = 720
7!  = 5040
8!  = 40320
9!  = 362880
10!  = 3628800
11!  = 39916800
12!  = 479001600
13!  = 6227020800
14!  = 87178291200
15!  = 1307674368000
16!  = 20922789888000
17!  = 355687428096000
18!  = 6402373705728000
19!  = 121645100408832000
20!  = 2432902008176640000
21!  = 14197454024290336768
```

Example Fibonacci Series by Recursion

- The Fibonacci series
  - 0, 1, 1, 2, 3, 5, 8, 13, 21, …
  - Begins with 0 and 1
  - Each subsequent Fibonacci number = sum of previous two Fibonacci numbers

- The Fibonacci series may be defined recursively as follows:

  fibonacci(0) = 0

  fibonacci(1) = 1

  fibonacci(n) = fibonacci(n – 1) + fibonacci(n – 2)

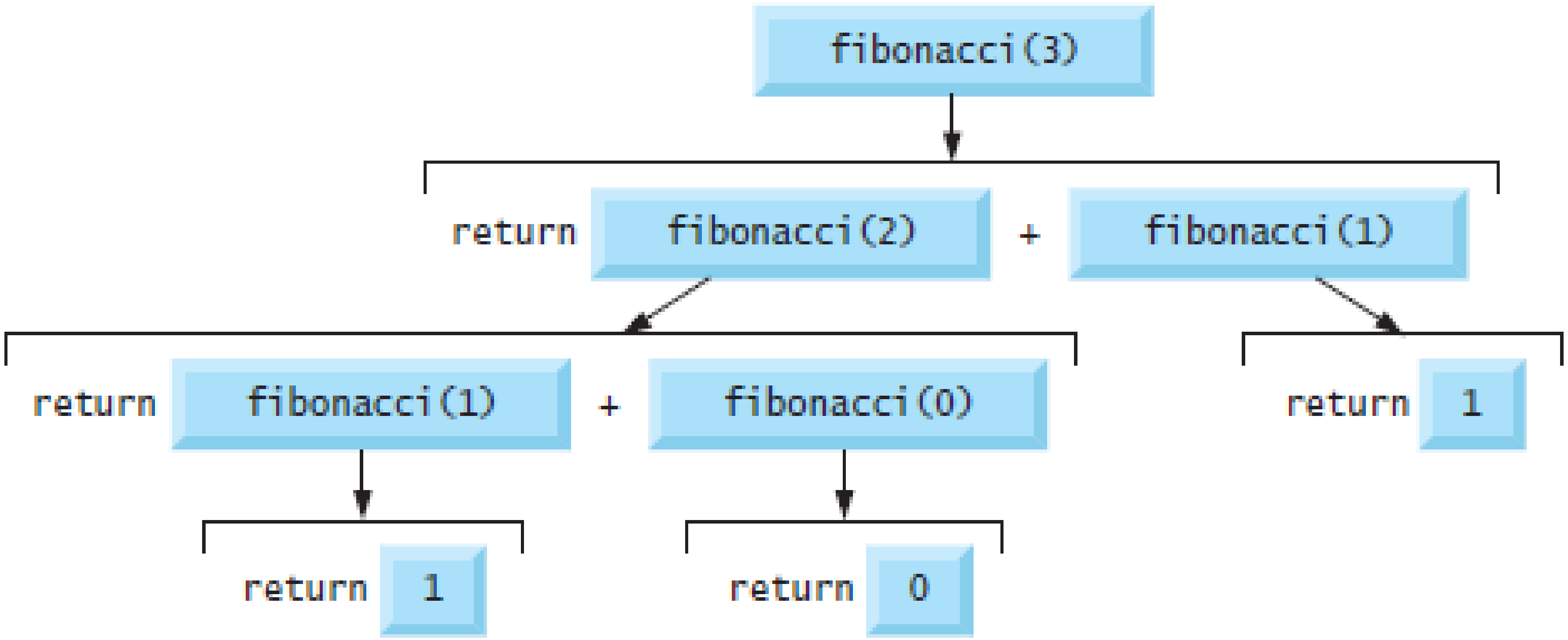
Recursive Fibonacci Function

```c
// fig05_10.c
// Recursive fibonacci function.
#include <stdio.h>

unsigned long long int fibonacci(int n); // function prototype

int main(void) {
   // calculate and display fibonacci(number) for 0-10
   for (int number = 0; number <= 10; number++) {
      printf("Fibonacci(%d) = %llu\n", number, fibonacci(number));
   }

   printf("Fibonacci(20) = %llu\n", fibonacci(20));
   printf("Fibonacci(30) = %llu\n", fibonacci(30));
   printf("Fibonacci(40) = %llu\n", fibonacci(40));
}

// Recursive definition of function fibonacci
unsigned long long int fibonacci(int n) {
   if (0 == n || 1 == n) { // base case
      return n;
   }
   else { // recursive step
      return fibonacci(n - 1) + fibonacci(n - 2);
   }
}
```

Recursive Fibonacci Function

```
Fibonacci(0) = 0
Fibonacci(1) = 1
Fibonacci(2) = 1
Fibonacci(3) = 2
Fibonacci(4) = 3
Fibonacci(5) = 5
Fibonacci(6) = 8
Fibonacci(7) = 13
Fibonacci(8) = 21
Fibonacci(9) = 34
Fibonacci(10) = 55
Fibonacci(20) = 6765
Fibonacci(30) = 832040
Fibonacci(40) = 102334155
```

Recursive Fibonacci Function - Diagram



Recursion vs Iteration

- Both iteration and recursion are based on a control statement: Iteration uses a repetition statement; recursion uses a *selection statement*.

- Both iteration and recursion involve repetition: Iteration explicitly uses a repetition statement; recursion achieves repetition through *repeated function calls*.

- Iteration and recursion each involve a termination test: Iteration terminates when the loop-continuation condition fails; recursion when a *base case is recognized*.

Recursion is expensive

- It repeatedly invokes the mechanism, and consequently the overhead, of function calls.
- This can be expensive in both processor time and memory space.
- Each recursive call causes another copy of the function to be created; this can consume considerable memory.
- The amount of memory in a computer is finite, so only a certain amount of memory can be used to store stack frames on the function call stack.
- If more function calls occur than can have their stack frames stored on the function call stack, a fatal error known as a *stack overflow* occurs

Inclass practice – Recursive function

Write a recursive function in **C** called `power` that calculates the result of raising a base integer b to a non-negative exponent integer e (b^e).

You must use **recursion** to solve this. Do not use the standard C library function `pow()`.

Function prototype:

```c
int power(int base, int exponent);
```

Let the user choose options:
Option 1: b and e are inputted by user.
Option 2: b and e are randomly generated
Ask the use to choose the option again if the input is invalid

Inclass practice – Recursive function

Write a recursive function in **C** called `power` that calculates the result of raising a base integer b to a non-negative exponent integer e (b^e).

$$\boxed{b^e = b \times b^{e-1}}$$

You must use **recursion** to solve this. Do not use the standard C library function `pow()`.

Function prototype:

```c
int power(int base, int exponent);
```

Inclass practice – Recursive function

```c
#include <stdio.h>

// Function Prototype
int power(int base, int exponent);

int main() {
    printf("2^4 = %d\n", power(2, 4));   // Expected: 16
    printf("5^0 = %d\n", power(5, 0));   // Expected: 1
    printf("3^1 = %d\n", power(3, 1));   // Expected: 3
    printf("10^3 = %d\n", power(10, 3)); // Expected: 1000

    return 0;
}

// Implement the recursive power function here
int power(int base, int exponent) {
    // 1. Base Case:
    if (exponent == 0) {
        return 1;
    }

    // 2. Recursive Step:
    else {
        // b^e = b * b^(e-1)
        return base * power(base, exponent - 1);
    }
}
```

In-class Practice

1st Solution

```c
// Online C compiler to run C program online
#include <stdio.h>

int isEven(int var);

int main() {
    int var1 = 0;
    int var2 = 0;
    int var3 = 0;
    int var4 = 0;
    printf("Please enter a series of integers: \n");
    scanf("%d%*c%d%*c%d%*c%d", &var1, &var2, &var3, &var4);

    printf("%d", var1);
    printf("%d", var2);
    printf("%d", var3);
    printf("%d", var4);

    printf("1st integer is: %d\n", isEven(var1));
    printf("2nd integer is: %d\n", isEven(var2));
    printf("3rd integer is: %d\n", isEven(var3));
    printf("4th integer is: %d\n", isEven(var4));

    return 0;
}

int isEven(int var){
    if (var%2 == 0)
        return 1;
    else
        return 0;
}
```

2nd Solution

```c
// Online C compiler to run C program online
#include <stdio.h>

int isEven(int var);

int main() {
    int var1 = 0;
    int num_series = 0;
    printf("Please enter the total number of integer series: \n");
    scanf("%d", &num_series);

    for (int i=0; i<num_series; i++)
    {
        printf("Please start to enter the integer: \n");
        scanf("%d", &var1);
        printf("The type of integer is: %d\n", isEven(var1));
    }

    return 0;
}

int isEven(int var){
    if (var%2 == 0)
        return 1;
    else
        return 0;
}
```