Introduction to Computing and Programming

Functions

Recap

- Basics of Function
  - What is the **need of a function?**
  - Example
- Function Arguments
  - Formal Parameters
  - Actual Parameters
- Function Call
  - Call by Value
  - Call by Reference

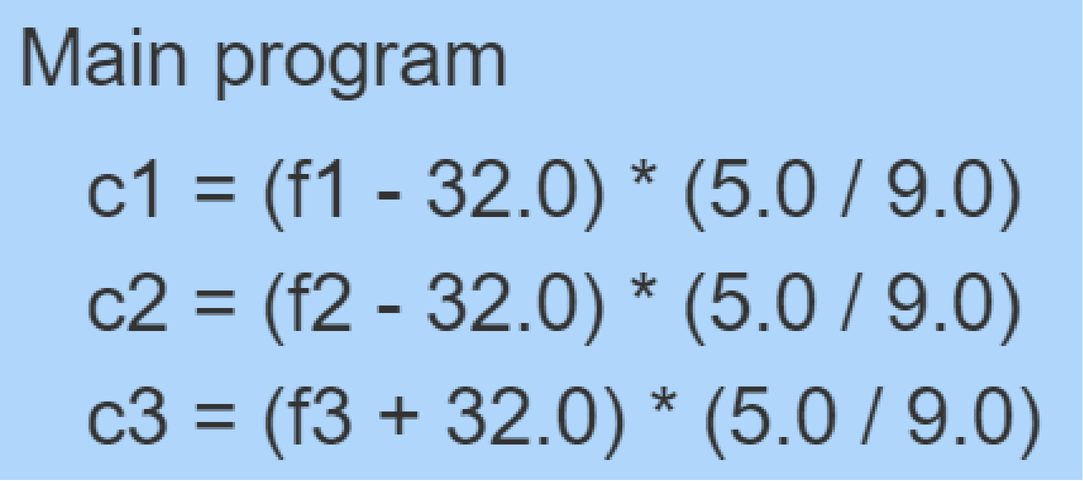
Content

- Function with Arrays
- Macro & Inline functions
- Mid-sem Answers discussion

Functions

Function basics & Motivations

- Program **redundancy can be reduced** by creating a grouping of predefined statements for repeatedly used operations, known as a function.



Main program

```
    c1 = (f1 - 32.0) * (5.0 / 9.0)
    c2 = (f2 - 32.0) * (5.0 / 9.0)
    c3 = (f3 + 32.0) * (5.0 / 9.0)
```

- This is Temperature conversion code;
- Cluttered repeated code; **Error in c3;**

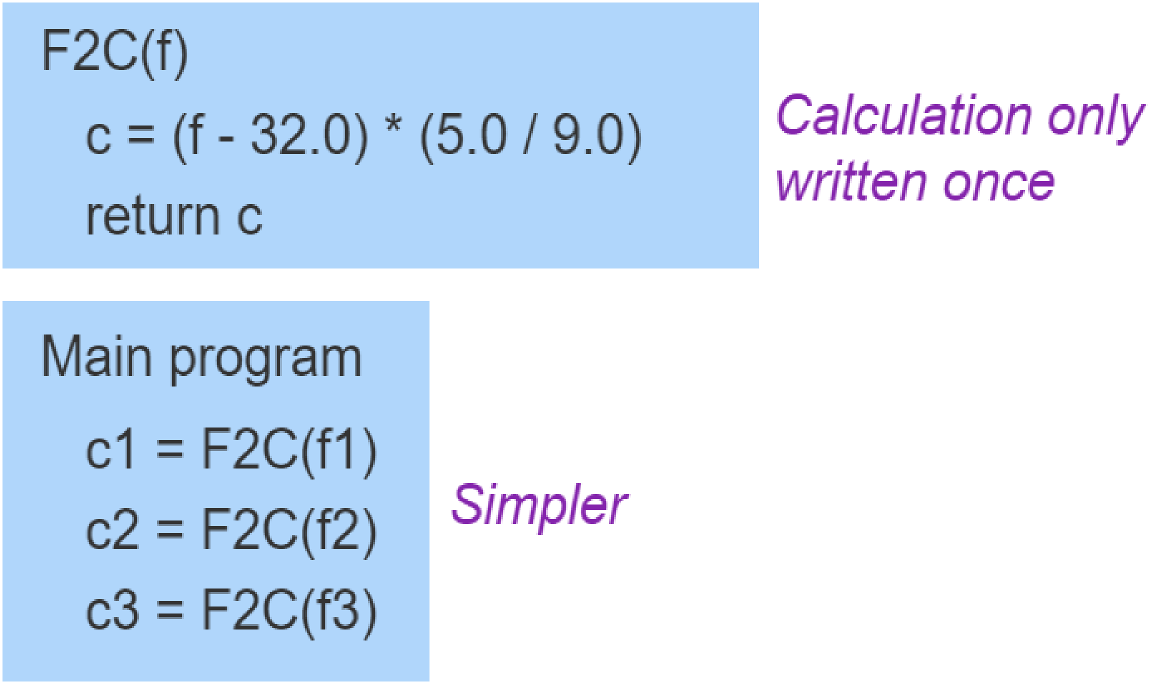
Function basics & Motivations Cont..

- // Function to convert Fahrenheit to Celsius

```
float F2C(float f) {
float c= (f – 32.0) * (5.0 / 9.0);
return c;}
```

- The impact is even greater when the operation has **multiple statements.**

- The main program is much simpler.



```
F2C(f)
    c = (f - 32.0) * (5.0 / 9.0)
    return c
```

Calculation only written once

```
Main program
    c1 = F2C(f1)
    c2 = F2C(f2)
    c3 = F2C(f3)
```

Simpler

Defining a Function

- The **general skeleton of a function** in C is as follows:

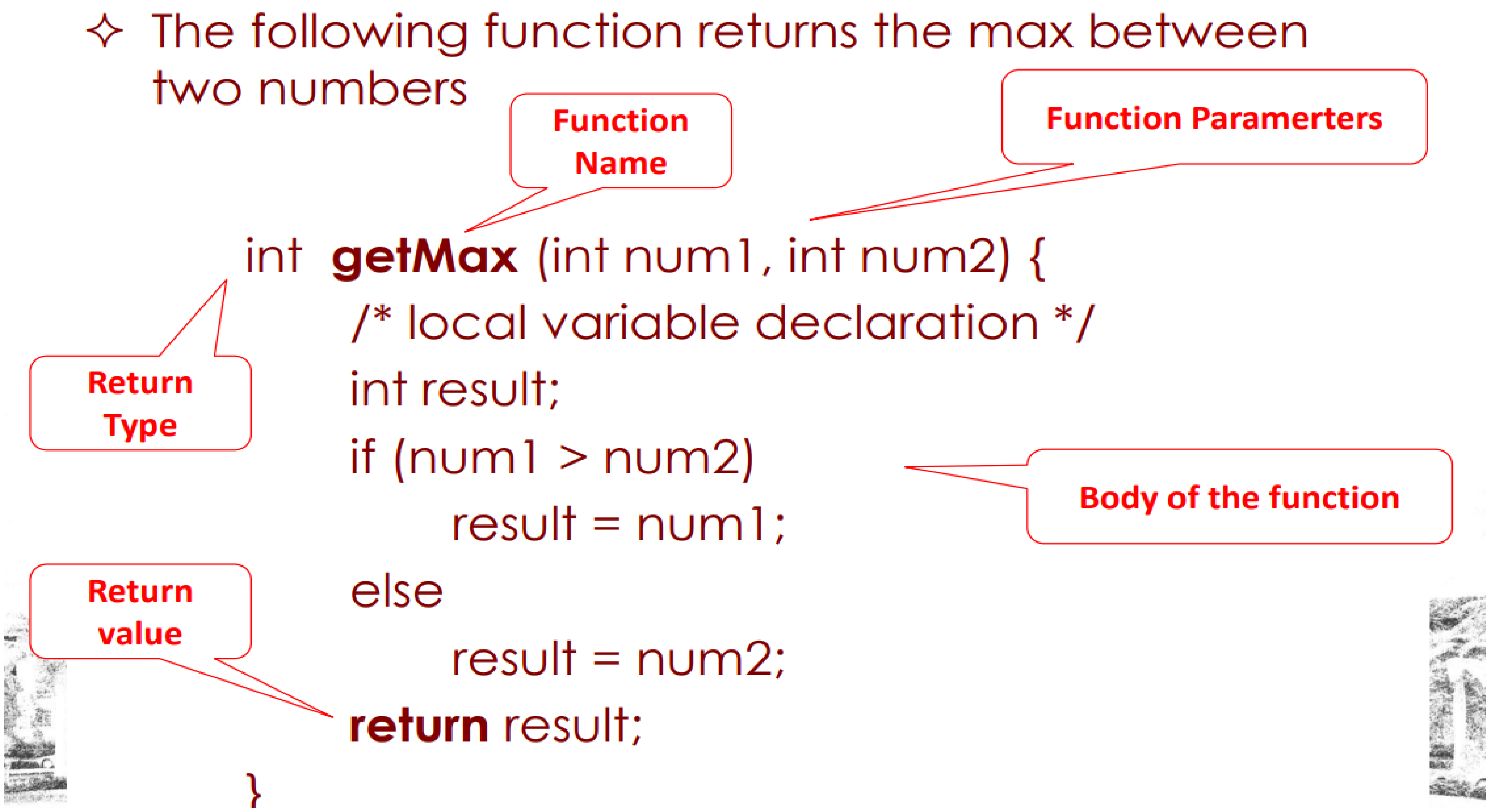
```
return_type function_name ( parameter list ) {
// body of the function
}
```

Example: int add(int a, int b){

```
                    return (a+b);}
```

- A **function definition** in C consists of:
  - a function header and
  - a function body
- **Function Declaration:**
  - Tells the compiler about a function's name, return type, and parameters
  - A function definition provides the actual body of the function

Function an Example:



✧ The following function returns the max between two numbers

```
int  getMax (int num1, int num2) {
    /* local variable declaration */
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
```

Function Name

Function Paramerters

Return Type

Body of the function

Return value

Functions with Arrays

Function with 1-D array

- ✧ Problem: Check a given number is present in the array or not?
- ✧ This function returns 1 if the number is present; 0, otherwise

```
#include <stdio.h>
void printValues( int a[], int n) {
        for ( int i =  0; i < n; i++) printf(" %d ", a[i]);
        printf("\n");
}
int Search( int a[], int n, int k) {
        for ( int i =  0; i < n; i++) {
                if ( k == a[i] ) return 1;
        }
        return 0;
}
int main(int argc, char *argv[]) {
        int n = 5;                              // Size of the array
        int a[5] = {12, 6, 2, 11, 5};           // An array of 5 elements
        int k = 5;                              // The element to be searched in the array
        printValues(a, n);
        int ret = Search(a, n, k);
        if (ret == 1) printf("\n%d is present in the array\n", k);
        else printf("\n%d is NOT present in the array\n", k);
        return 0;
}
```

15,5

Passing Values between Functions

Problem: Check and print all occurrences of a given number in an array

```
#include <stdio.h>

void printValue ( int val) {
    printf("  %d  ", val);
}
void Search( int a[], int n, int k) {
    for ( int i =  0; i < n; i++) {
        if ( k == a[i] )
            printValue(a[i]);
    }
}
int main(int argc, char *argv[]) {
    int n = 5, k = 5;
    int ret = Search(a, n, k);
    if (ret == 1) printf("\n%d is present in the array\n", k);
    else printf("\n%d is NOT present in the array\n", k);
    return 0;
}
```

Passing Values between Functions

Problem: Check and print all occurrences of a given number in an array

```
#include <stdio.h>

void printMessage(char ch[] ) {
   printf("%s\n", ch);
}
void printValue( int val ) {
      printf("  %d  ", val);
}
void Search( int a[], int n, int k) {
      printMessage("Printing all occurrences");
      for ( int i =  0; i < n; i++) {
            if ( k == a[i] )
                  printValue(a[i]);
      }
      printf("\n");
}
int main(int argc, char *argv[]) {
      int n = 10, k = 5;
      int a[10] = {2, 13, 71, 28, 5, 5, 4, 8, 5, 1};

      Search(a, n, k);
      return 0;
}
```

Functions with 2D arrays

✧ Function to print the values of a 2D array: **2 arguments**

```
const int N = 3;
void printValues(int arr[][N], ~~int m~~) {
        int i, j;
        for (i = 0; i < m; i++) {
                for (j = 0; j < N; j++) {
                        printf("%d ", arr[i][j]);
                }
                printf("\n");
        }
}

int main() {
        int arr[][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
        printValues(arr, 3);
        return 0;
}
```

1 2 3
4 5 6
7 8 9

arr[][]={

Functions with 2D arrays Cont..

✧ A Modified Version – Only One argument

```
const int M = 3;
const int N = 3;
void printValues(int arr[M][N]) {
    int i, j;
    for (i = 0; i < M; i++) {
        for (j = 0; j < N; j++) {
            printf("%d ", arr[i][j]);
        }
        printf("\n");
    }
}
int main() {
    int arr[][N] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    printValues(arr);
    return 0;
}
```

Functions with 2D arrays Cont..

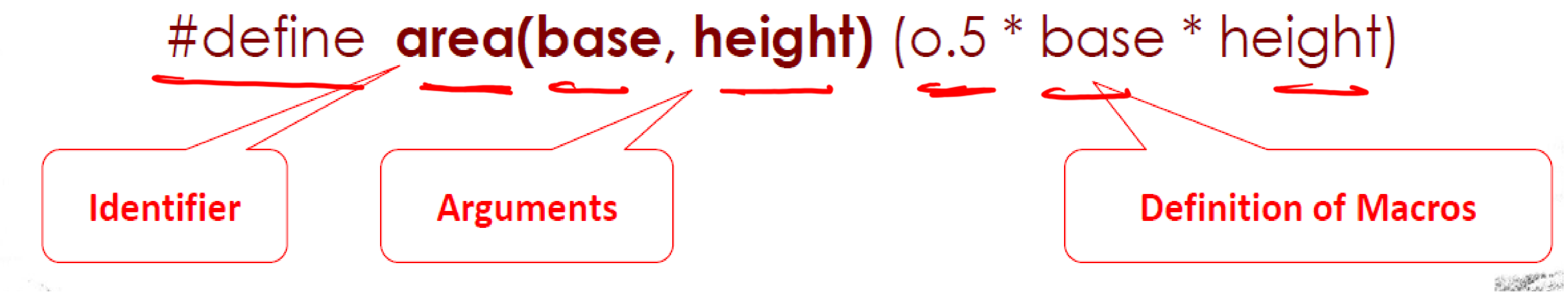
```
// The works only if your compiler is C99 compatible.
// n must be passed before the 2D array

    void printValues(int m, int n, int arr[][n]) {
        int i, j;
        for (i = 0; i < m; i++) {
            for (j = 0; j < n; j++) {
                printf("%d ", arr[i][j]);
            }
            printf("\n");
        }
    }
    int main() {
        int m = 3, n = 3;
        int arr[][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
        printValues(m, n, arr);
        return 0;
    }
```

Macros and Inline functions

- **Macros and Inline Functions:**
- **Use function if the sequence of steps is long**
- **If small, use macros or inline function**, to eliminate the need for retyping and time overhead.
- Need #define compiler directive
- Find **the area of a triangle (given: base and height)**

#define BMI(weight,height)

(weight/pow(heigt,2))



```
#define  area(base, height) (0.5 * base * height)
```

- Define it once and use it anywhere in the program

printf("\nArea = %f\n", area(4.0, 6.0));

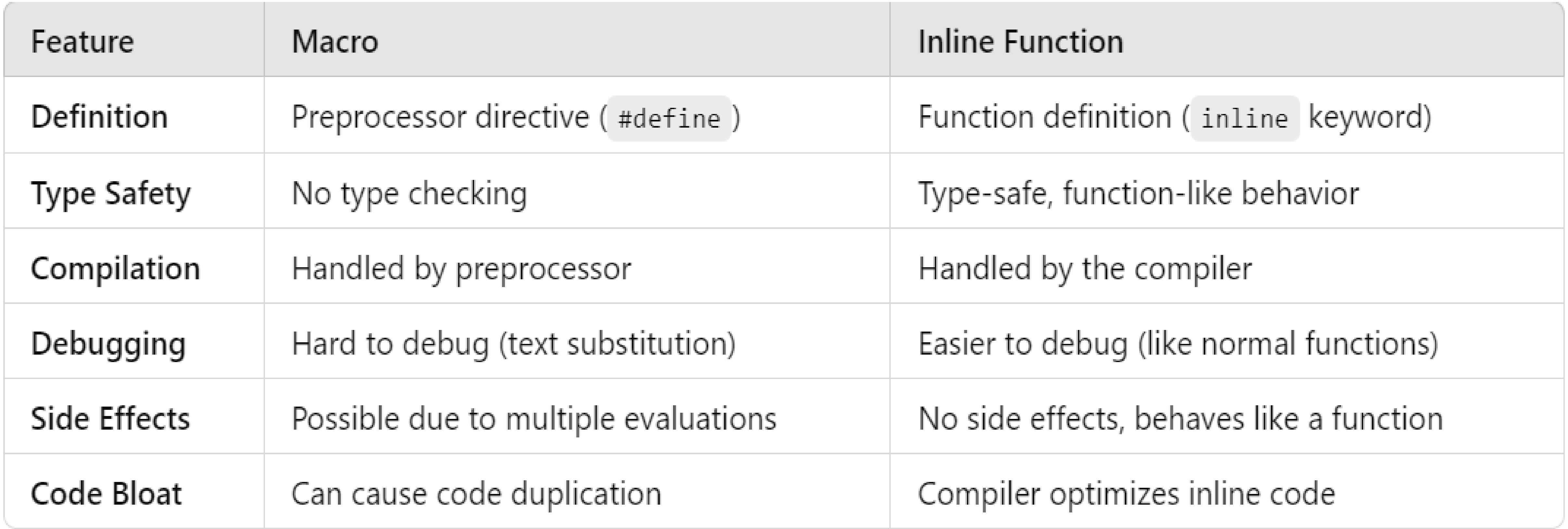
Macros and Inline functions Cont..

- **Macros:** Defined using **#define**, used for text substitution
- **Syntax**: #define SQUARE(x) ((x)*(x)) 5*5

  printf("%d", SQUARE(5)); // **Expands to ((5)*(5))**
- **Advantage**: Quick text substitution, fast.
- **Disadvantage**: No type checking
- **Inline Functions**: Defined using the **inline keyword**, used to optimize function calls by the compiler.
- **Syntax:** inline int square(int x) { return x * x; }

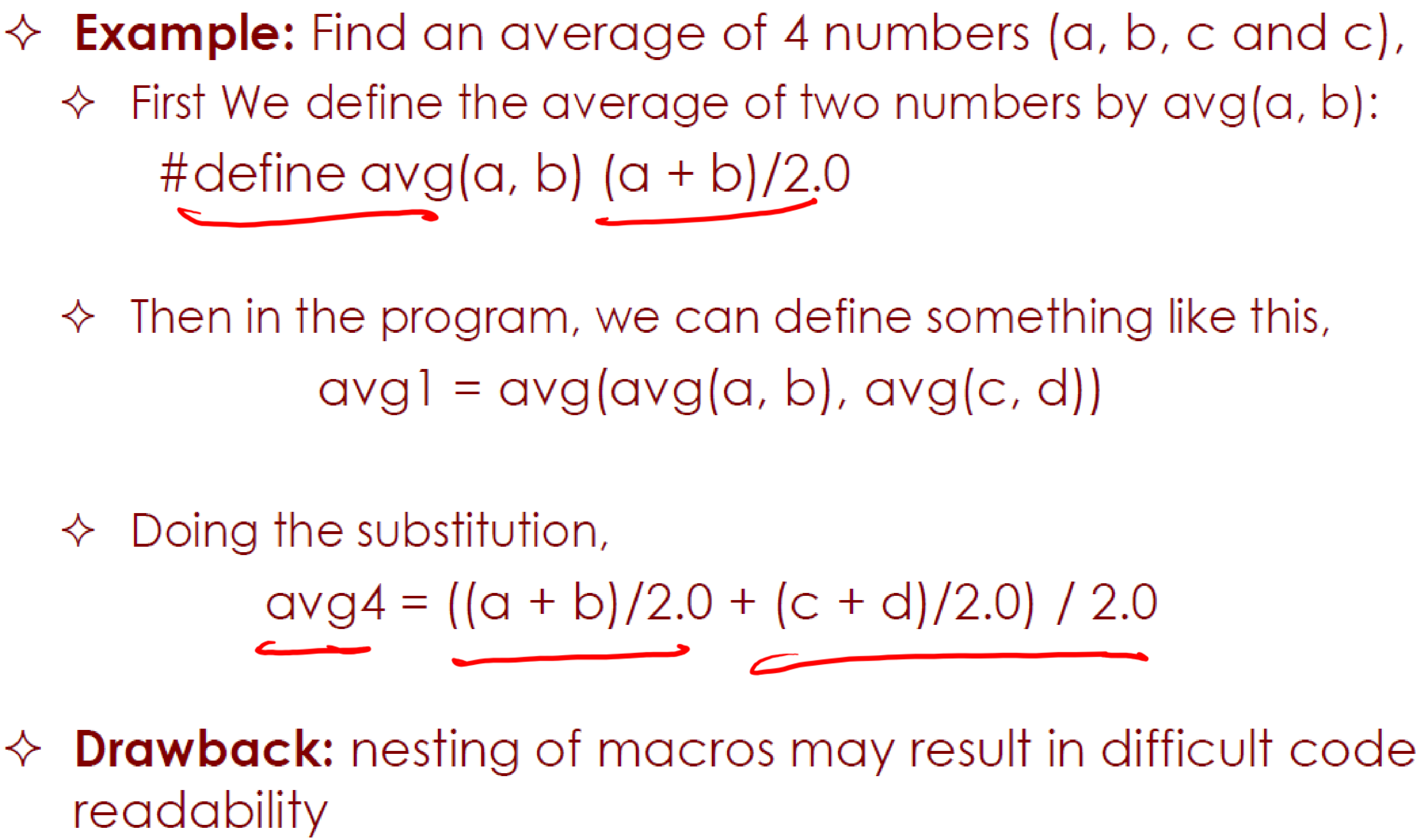
  printf("%d", square(5)); // **May be optimized by the compiler**
- **Advantages**: Type safety, debuggable, no side effects.
- **Disadvantages**: The compiler might ignore the inline suggestion
- **Both aim to reduce function call overhead but differ in implementation.**

Key difference between Macros & Inline function



Feature	Macro	Inline Function
Definition	Preprocessor directive (`#define`)	Function definition (`inline` keyword)
Type Safety	No type checking	Type-safe, function-like behavior
Compilation	Handled by preprocessor	Handled by the compiler
Debugging	Hard to debug (text substitution)	Easier to debug (like normal functions)
Side Effects	Possible due to multiple evaluations	No side effects, behaves like a function
Code Bloat	Can cause code duplication	Compiler optimizes inline code

Macros - Examples



- **Example:** Find an average of 4 numbers (a, b, c and c),
  - First We define the average of two numbers by avg(a, b):

    #define avg(a, b) (a + b)/2.0

  - Then in the program, we can define something like this,

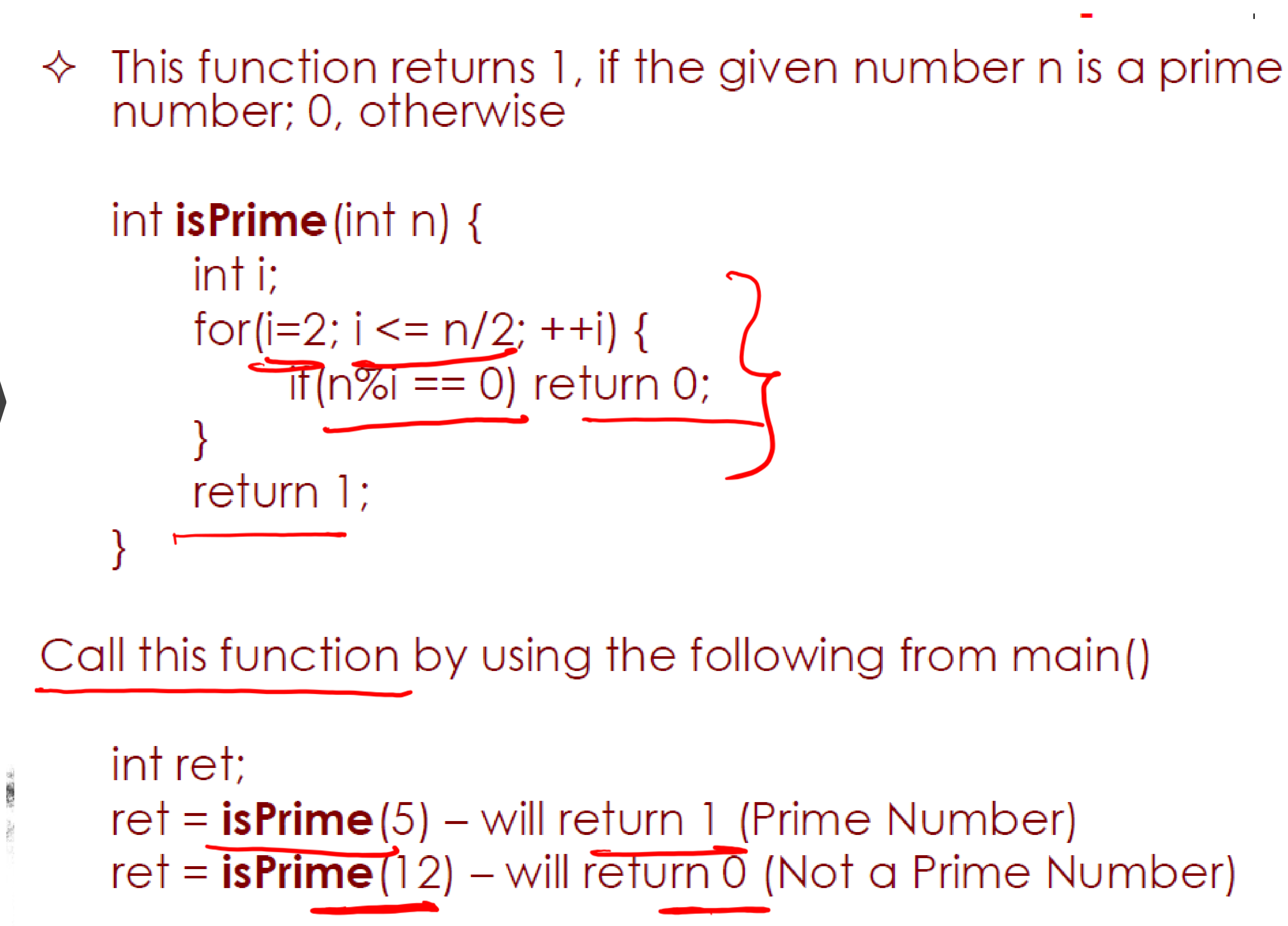
    avg1 = avg(avg(a, b), avg(c, d))

  - Doing the substitution,

    avg4 = ((a + b)/2.0 + (c + d)/2.0) / 2.0

- **Drawback:** nesting of macros may result in difficult code readability

Functions – A Few Examples



- This function returns 1, if the given number n is a prime number; 0, otherwise

```
int isPrime(int n) {
    int i;
    for(i=2; i <= n/2; ++i) {
        if(n%i == 0) return 0;
    }
    return 1;
}
```

Call this function by using the following from main()

```
int ret;
ret = isPrime(5) – will return 1 (Prime Number)
ret = isPrime(12) – will return 0 (Not a Prime Number)
```

Check An Armstrong Number

- A positive integer is called an Armstrong number of order n if $abcd \ldots = a^n + b^n + c^n + d^n + \ldots$
- For example:
  - $153 = 1^3 + 5^3 + 3^3$

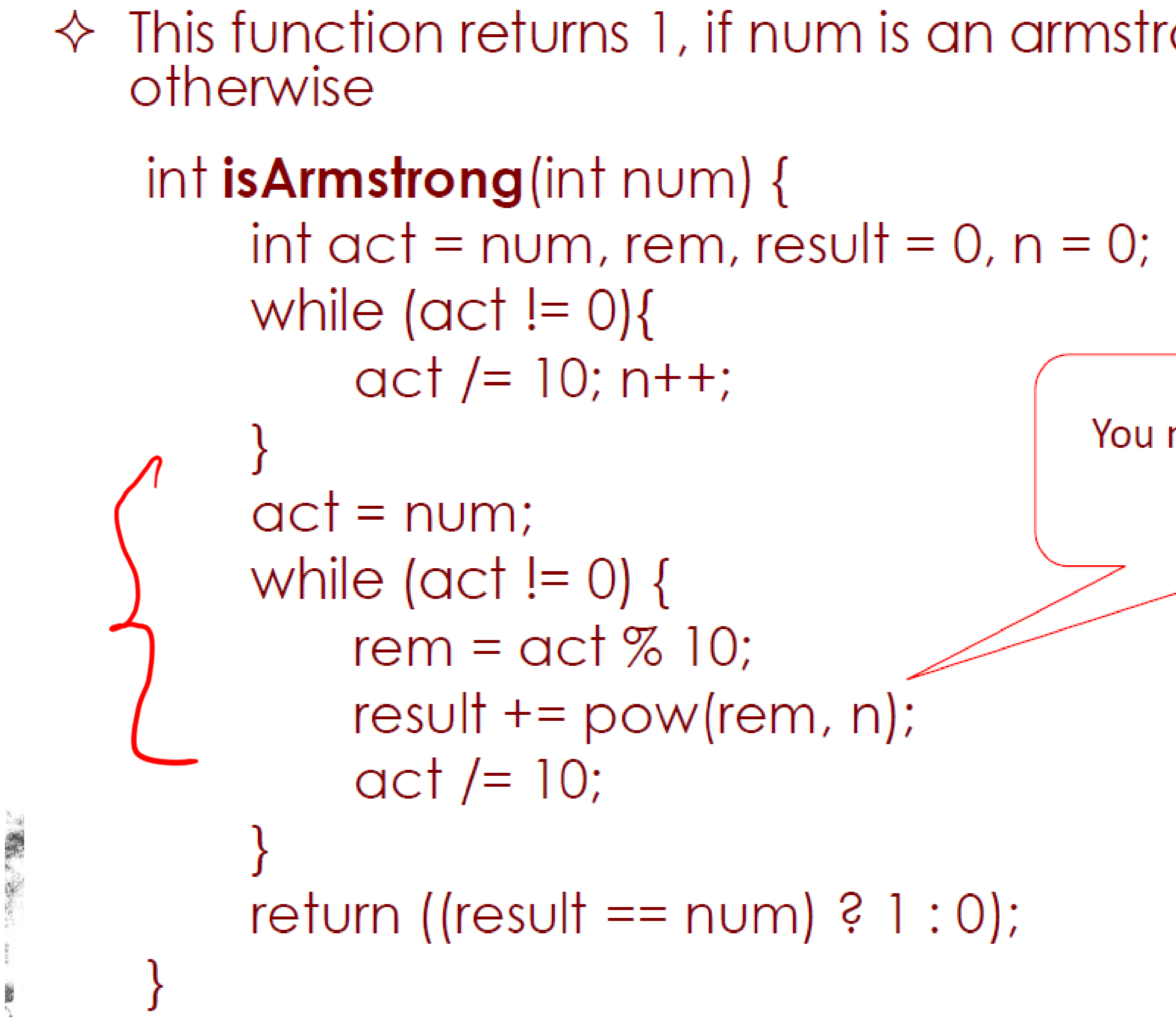
    $= 1*1*1 + 5*5*5 + 3*3*3$
  - $1634 = 1^4 + 6^4 + 3^4 + 4^4$

    $= 1*1*1*1 + 6*6*6*6 + 3*3*3*3 + 4*4*4*4$
  - $54748 = 5^5 + 4^5 + 7^5 + 4^5 + 8^5$

    $= 5*5*5*5*5 + 4*4*4*4*4 + 7*7*7*7*7$

    $+ 4*4*4*4*4 + 8*8*8*8*8$

Function - Armstrong Number



- This function returns 1, if num is an armstrong number; 0, otherwise

```
int isArmstrong(int num) {
      int act = num, rem, result = 0, n = 0;
      while (act != 0){
            act /= 10; n++;
      }
      act = num;
      while (act != 0) {
            rem = act % 10;
            result += pow(rem, n);
            act /= 10;
      }
      return ((result == num) ? 1 : 0);
}
```

You may or may not use Math Library

Fibonacci Series

- The Fibonacci sequence: A series in which the next term is obtained by summing up of pervious two terms. Let the first two terms be 0 followed by 1

□ The Fibonacci sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21, …

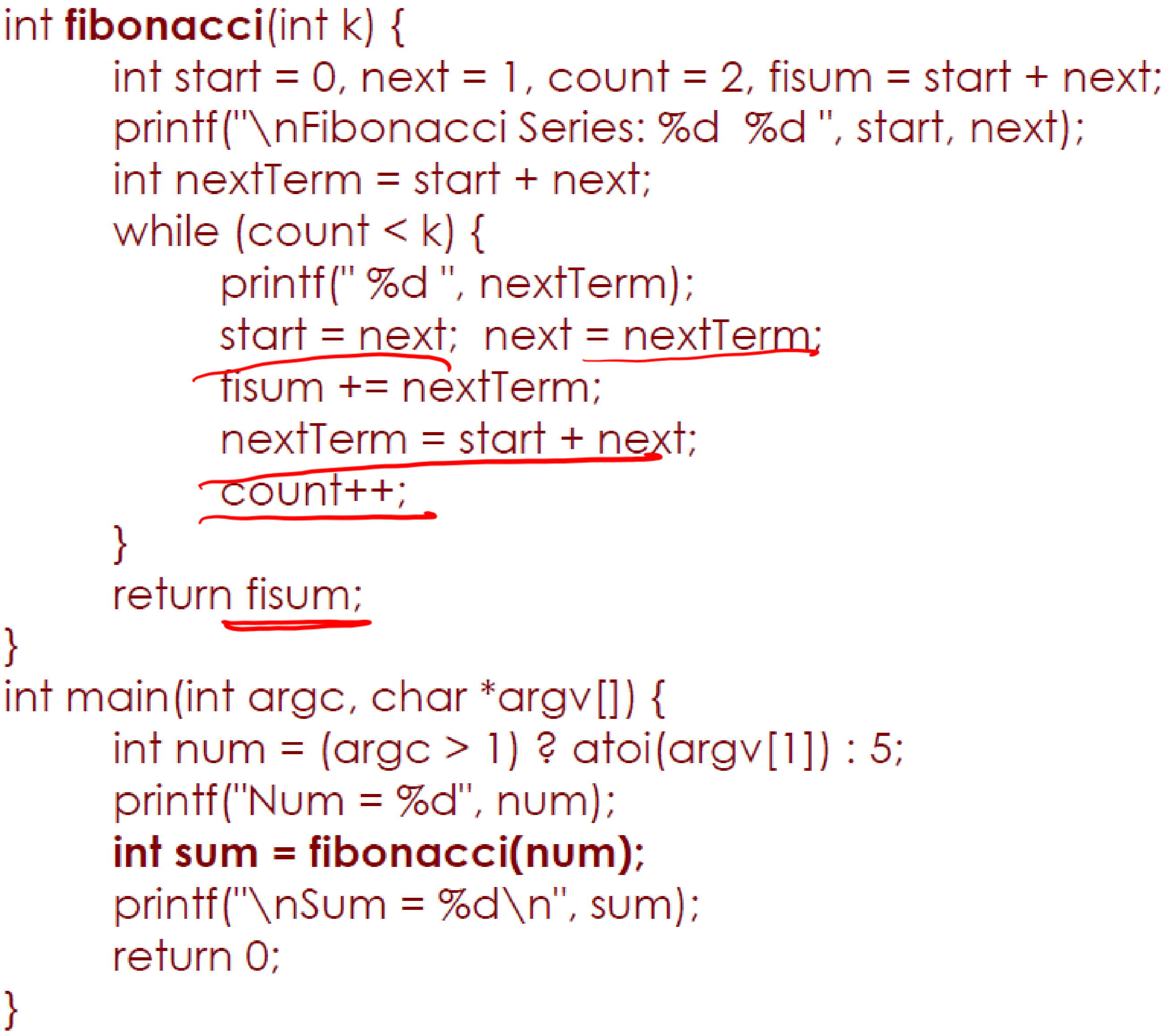
```
#include <stdio.h>


// Function to print Fibonacci series
void printFibonacci(int n) {
    int first = 0, second = 1, next;

    printf("Fibonacci Series up to %d terms:\n", n);

    for (int i = 0; i < n; i++) {
        if (i <= 1) {
            next = i; }// First two terms are 0 and 1
```

```
else {
            next = first + second; // Next term is the sum of the previous two
            first = second;        // Update first and second
            second = next;
        }
        printf("%d ", next); // Print the current term
    }
    printf("\n");
}
}
// Main function
int main() {
    int terms;
    // User input for number of terms
    printf("Enter the number of terms in the Fibonacci series: ");
    scanf("%d", &terms);
    // Print the Fibonacci series
    printFibonacci(terms);
    return 0;}
```

Fibonacci Series – upto k-terms

- write a function that:

compute and print the first k numbers in the Fibonacci Series and return the sum of the first k numbers of the Fibonacci series

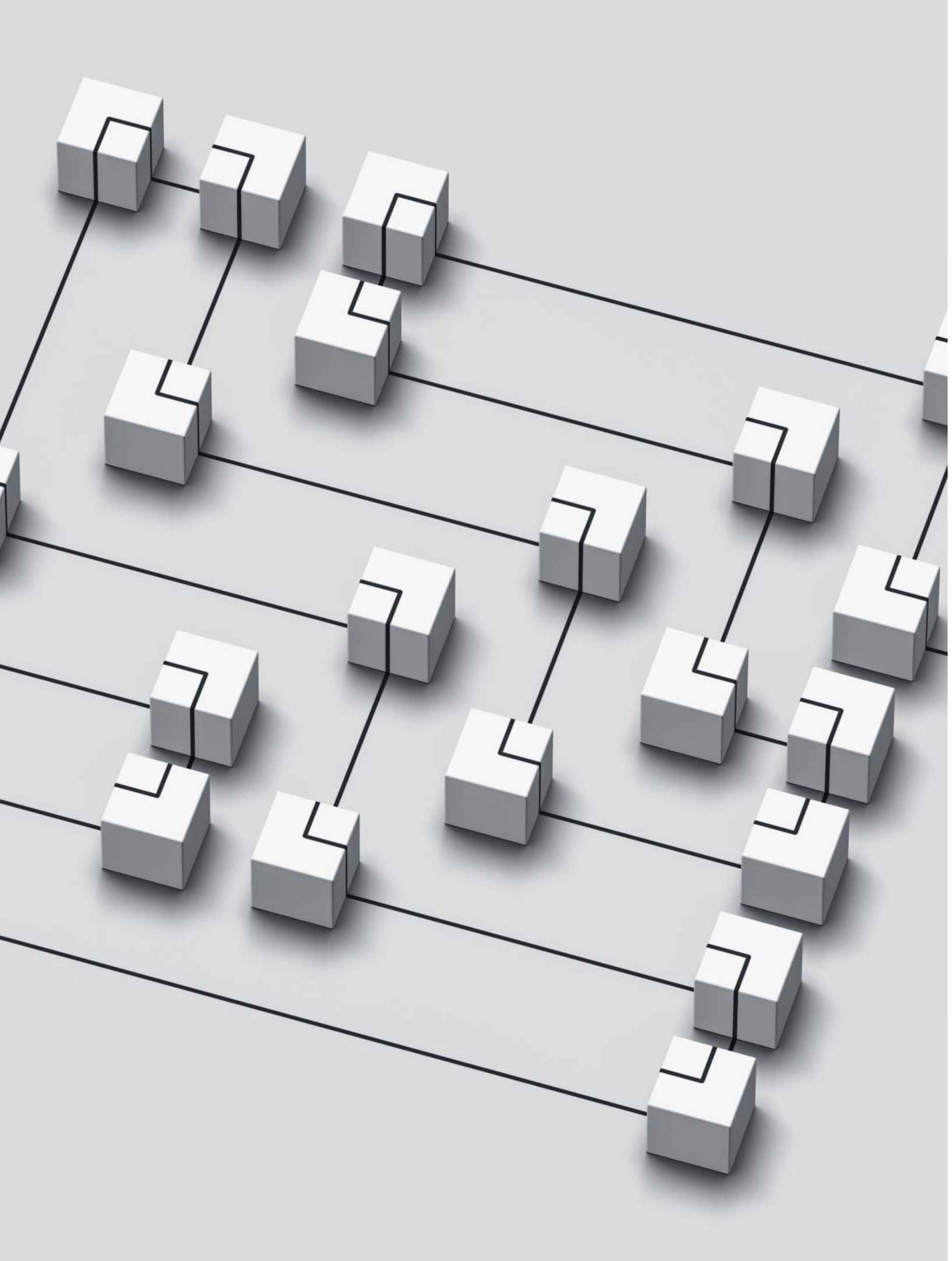


```
int fibonacci(int k) {
        int start = 0, next = 1, count = 2, fisum = start + next;
        printf("\nFibonacci Series: %d  %d ", start, next);
        int nextTerm = start + next;
        while (count < k) {
                printf(" %d ", nextTerm);
                start = next;  next = nextTerm;
                fisum += nextTerm;
                nextTerm = start + next;
                count++;
        }
        return fisum;
}
int main(int argc, char *argv[]) {
        int num = (argc > 1) ? atoi(argv[1]) : 5;
        printf("Num = %d", num);
        int sum = fibonacci(num);
        printf("\nSum = %d\n", sum);
        return 0;
}
```

Practice Questions

1. Write a function that takes a positive integer as input and displays all the positive factors of that number
2. Write a function to find and count the sum of only even digits in an integer
3. Write a function to count the number of Vowels, Consonants and symbols and print the same
4. Write a function to reverse the characters in an array of size k

5. Write a function to check whether a number can be expressed as the sum of two prime numbers
    - You may use a separate method to check primality



Upcoming Slides

- Recursion