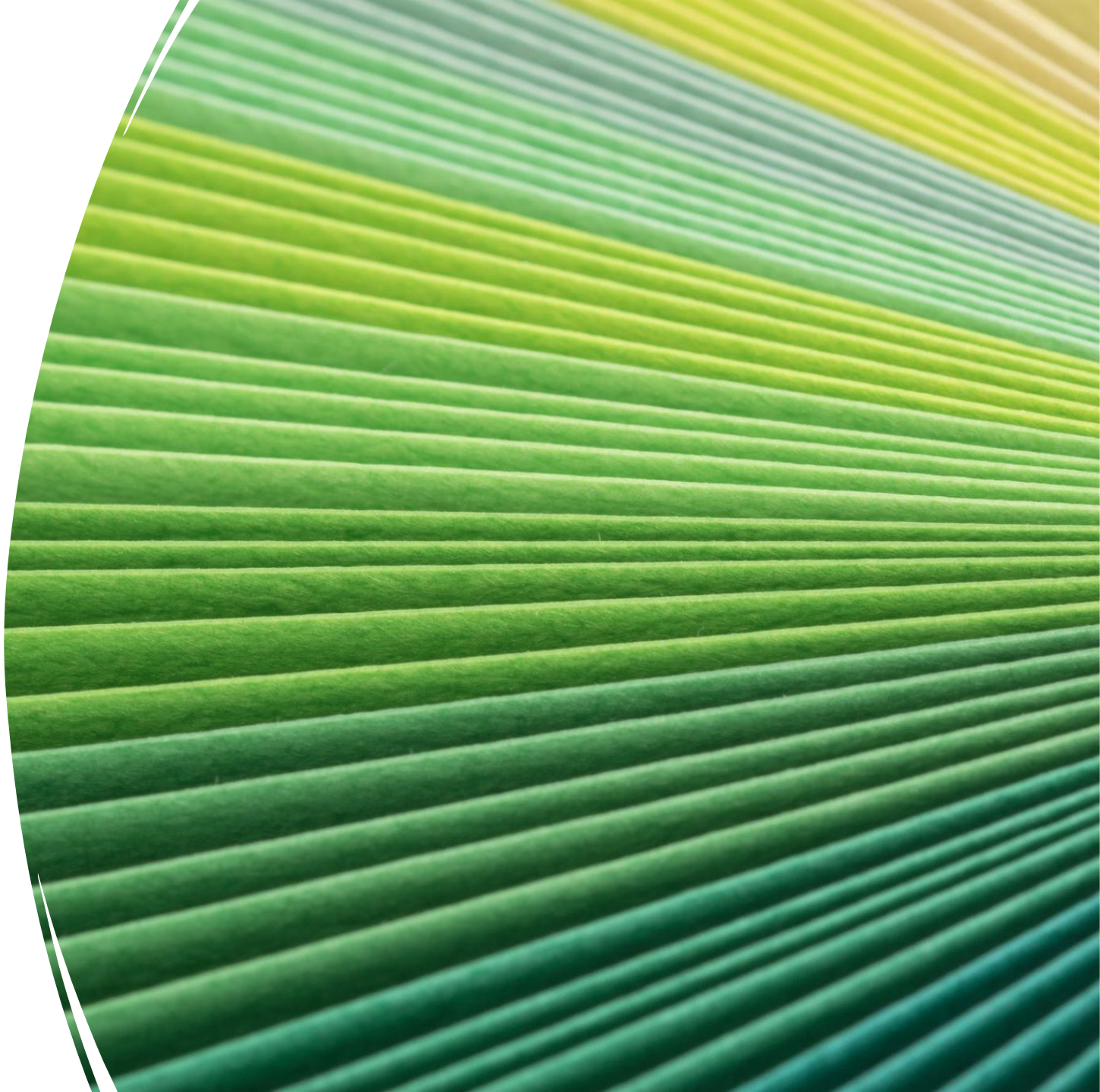
Introduction to Computing and Programming

Functions

Recap

- Multi-Dimensional Arrays
- Basics of Function
- Scope of Functions



Content

- Function Arguments
- Mid-sem Paper Discussion
- Function with Arrays

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

Function Name

Function Paramerters

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

Return Type

Body of the function

Return value

Scope of the variables

- Scope of the variables defined in a function?

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

The values of the variables: **num1**, **num2**, and **results** are purely local in this function.

Once the execution is over, these variables are not available for other parts of the program

Functions – A Simplified Example

- A simplified version of the same function that finds the maximum of two integers: m, n

```
int getMax(int num1, int num2) {
return ((num1 > num2) ? num1 : num2);
}
```

Functions – main()

- How do we call getMax function from main()?

```
int main(void) {
1   int m = 10, n = 27;
    printf("\nm = %d, n = %d\n", m, n);
2   int max = getMax(m,n);
3   printf("\nMax = %d\n", max);
4

5   return 0;

    }
```

Function that finds max(m,n)

```
/* The complete Program – finding max of m and n */
#include <stdio.h>

int getMax(int num1, int num2) {
        return ((num1 > num2) ? num1 :  num2);
}

int main(void) {
        int m = 10, n = 27;
        printf("\nm = %d, n = %d\n", m, n);

        int max = getMax(m,n);
        printf("\nMax = %d\n", max);
        return 0;
}
```

Function getMax() is defined before main(). So function declaration is implicit

Where is the function defined?

```
/* The complete Program – finding max of m and n */
#include <stdio.h>

/* Function Declaration is required */
int getMax(int, int);

int main(void) {
        int m = 10, n = 27;
        printf("\nm = %d, n = %d\n", m, n);

        /* Calling the function */
        int max = getMax(m,n);

        printf("\nMax = %d\n", max);
        return 0;
}
int getMax(int num1, int num2) {
        return ((num1 > num2) ? num1 :  num2);
}
```

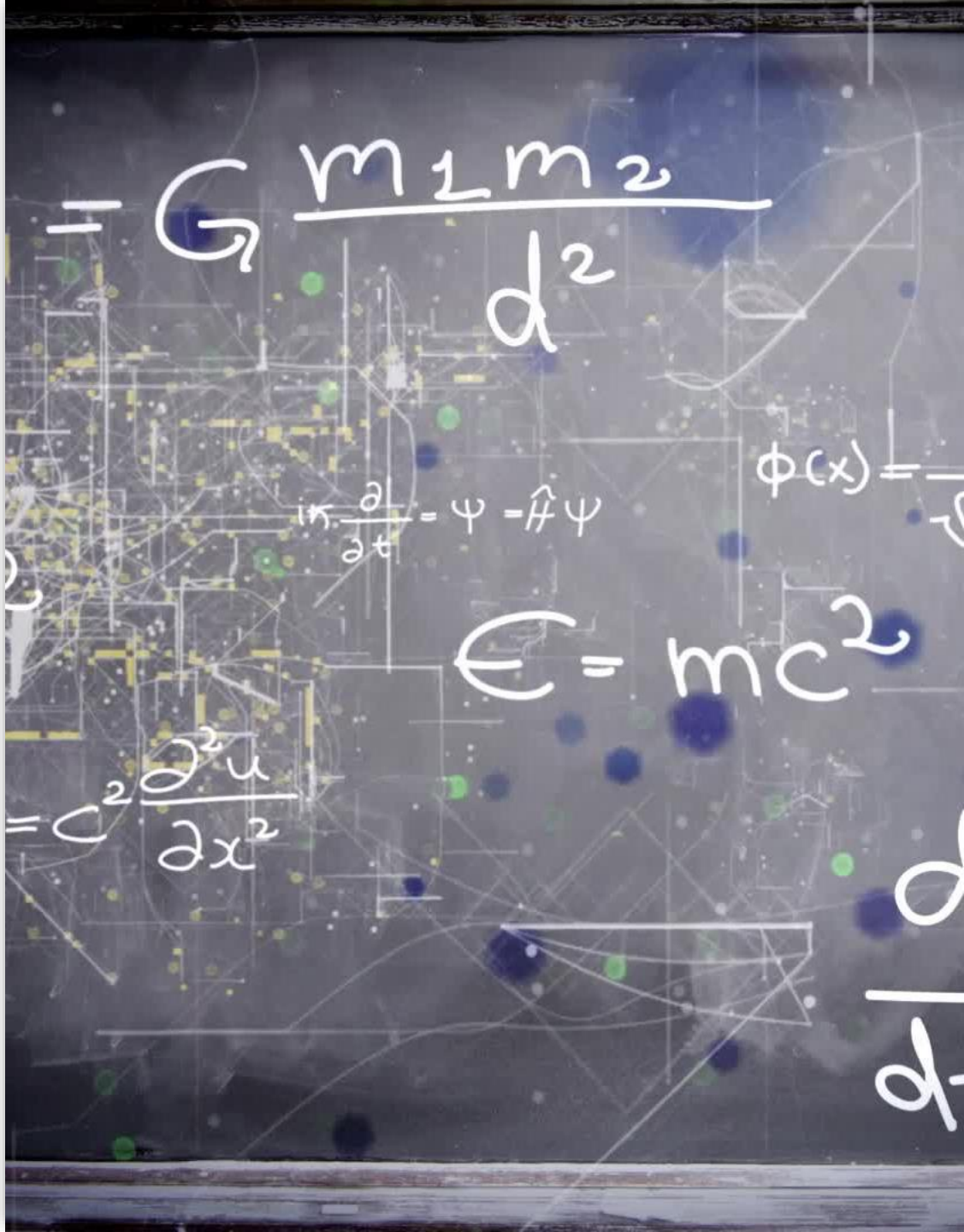
Function getMax() is defined afer main().

So the function declaration is needed

Function Arguments

- A function argument (or parameter) is a **value passed to a function** when it is called.
- The function can use these values to perform its task.
- **Two types:**

1. Formal Arguments (declared in the function definition): Formal parameters behave like local variables inside the function and are **created upon entry** into the function and **destroyed upon exit.**
2. Actual Arguments (provided during the function call)



Function Arguments Example

1. Formal Arguments:

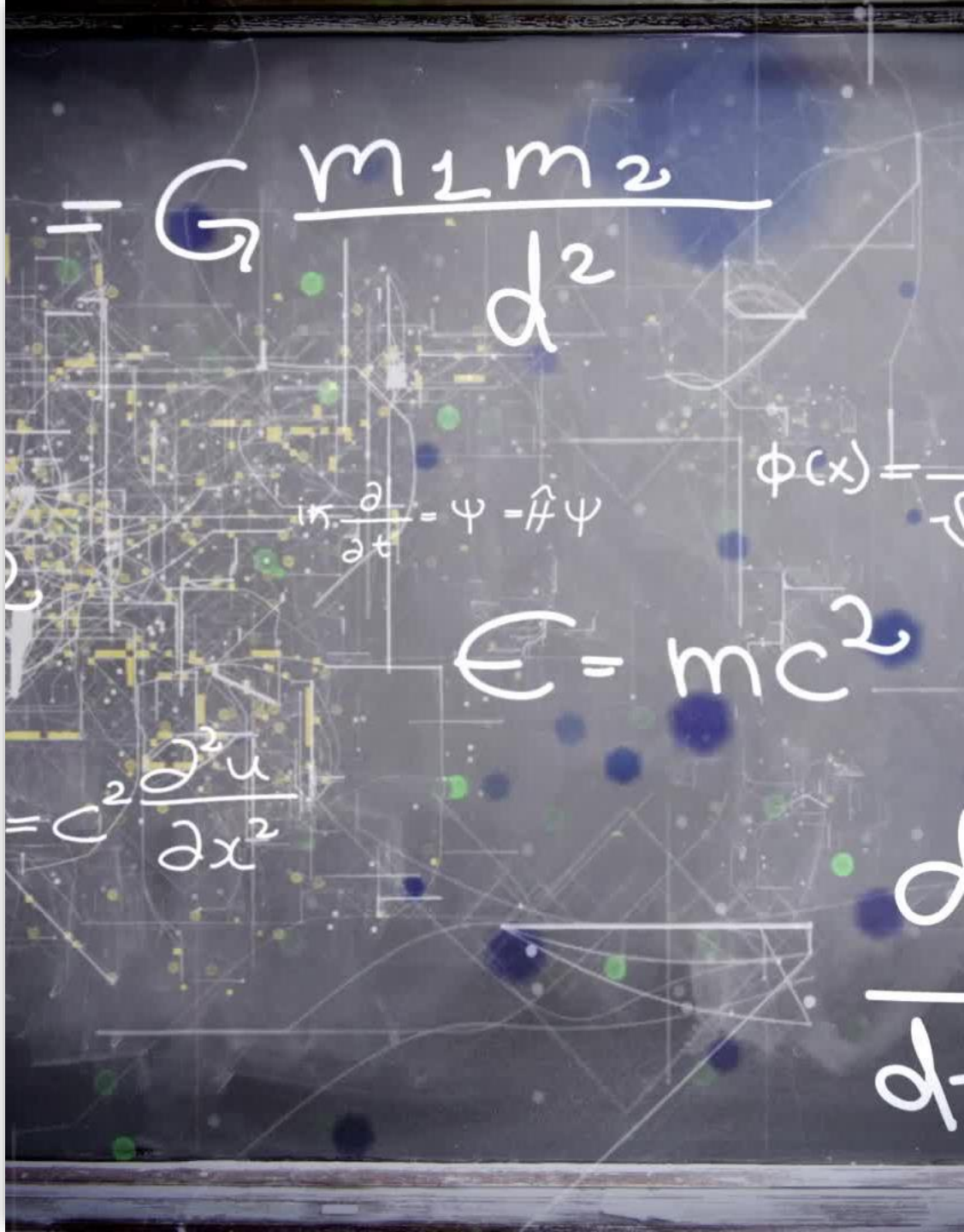
Example: int add(**int a, int b**) {

return a + b;

}

2. Actual Arguments: Arguments are passed to the function when it is called

Example: int result = add(**5, 10**); // 5 and 10 are actual arguments



Function Call with Arguments

- Two ways to call a function:
  - **Call by Value**
  - **Call by Reference**
- arguments can be passed to a function using any o the above way

Call by Value

A copy of the actual argument is passed to the function.

Modifying the parameter inside the function does not affect the original argument.

Example:

```
void changeValue(int x) {
x = 20;
}
int main() {
int num = 10;
changeValue(num);
printf("%d", num); // Output: 10
}
```

20

x

10

num

Another Example of Call by Value

```
#include <stdio.h>
// Function to swap two numbers using call by value
void swapByValue(int a, int b) {
    int temp = a;
    a = b;
    b = temp;
    printf("Inside swapByValue function: a = %d, b = %d\n", a, b);
}
int main() {
    int x = 10, y = 20;

    printf("Before swapByValue function: x = %d, y = %d\n", x, y);
    // Call by value
    swapByValue(x, y);

    printf("After swapByValue function: x = %d, y = %d\n", x, y);
    return 0;
}
```

10 20

a → 10, 5

a = a + b;
b = a - b;
a = a - b;

Output: Before swapByValue function: x = 10, y = 20
Inside swapByValue function: a = 20, b = 10
After swapByValue function: x = 10, y = 20

Call by Reference

A reference (address) to the actual argunment is passed to the function.

Modifying the parameter inside the function does affect the original argument.

Example:

```
void changeValue(int *x) {
 *x = 20;
}

int main() {
 int num = 10;
 changeValue(&num);
 printf("%d", num); // Output: 20
}
```

Another Example of Call by Reference

```
#include <stdio.h>
// Function to swap two numbers using call by reference
void swapByReference(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
    printf("Inside swapByReference function: a = %d, b = %d\n", *a, *b);
}
int main() {
    int x = 10, y = 20;
    printf("Before swapByReference function: x = %d, y = %d\n", x, y);
   // Call by reference
    swapByReference(&x, &y);
     printf("After swapByReference function: x = %d, y = %d\n", x, y);

    return 0;
}
```

10 20

1000 10 20 20 10 2000

x y

Output: Before swapByReference function: x = 10, y = 20
Inside swapByReference function: a = 20, b = 10
After swapByReference function: x = 20, y = 10

A Simple Function Example

- **Example:** void PrintGreetings() {

```
printf("\nWelcome to SNU");
printf("\nSNU is an Institute of Eminence\n");
}
```

- **The above function has**

 no arguments and

 no return value

 But prints the greetings message

- Functions can also have arguments (one or more) depending on the specific task in hand

A Simple Function Example

- **Example:** void PrintGreetings() {

```
printf("\nWelcome to SNU");
printf("\nSNU is an Institute of Eminence\n");
}
```

- **The above function has**

 no arguments and

 no return value

 But prints the greetings message

- Functions can also have arguments (one or more) depending on the specific task in hand

Function with one argument

- Compute and print the sum of the first N natural Numbers
- This function does not need to return any value.

```
#include <stdio.h>
/* Method 1 – Using FOR loop */
void getSumN( int n ) {
int i = 0, sum = 0;
for (i = 0; i < n + 1; i++) {
sum += i;
}
printf("\nMethod - 1: \nN = %d, SUM = %d\n", n, sum);
}
int main(int argc, char *argv[]) { /*accepts command-line
arguments: int argc: Argument count (the number of
arguments passed from the command line, char *argv[])
:Argument vector (an array of pointers to the command-line
arguments*/
int m = 10;
getSumN(m); /* Calling Method 1 */
return 0;
}
```

./a.out
2 3

An Updated getSumN(n)

- Compute and print the sum of the first N natural Numbers
- Another approach using two variables.

```
#include <stdio.h>
void getSumN( int n ) { /* Method 2 - Using two variables,
left(MIN) & right(MAX) */
int i = 1, sum = 0;
int left = i, right = n;
while (left < right) {
sum += left + right;
left++; right--;
}
/* Sum Correction for ODD numbers */
sum += (n % 2 == 1) ? ((int) n/2 + 1) : 0;
printf("\nMethod - 2:\nN = %d, SUM = %d\n", n, sum);
}
int main(int argc, char *argv[]) {
int m = 10;
getSumN(m); /* Calling Method - 2 */
return 0;
}
```

A small exercise

Write the function declaration & definition for BMI calculation

A small exercise- Solution

```
#include <stdio.h>

// Function declaration
float calculateBMI(float weight, float height);

// Main function
int main() {
    float weight, height, bmi;

    // Input weight and height
    printf("Enter weight in kilograms: ");
    scanf("%f", &weight);

    printf("Enter height in meters: ");
    scanf("%f", &height);

// Call the BMI function
    bmi = calculateBMI(weight, height);

// Output the calculated BMI
    printf("Your BMI is: %.2f\n", bmi);

    return 0;
}

// Function definition
float calculateBMI(float weight, float height) {
    return weight / (height * height);  // BMI formula
}
```

Function with Two arguments

- **Problem:** Compute the sum of 2 numbers
- This function computes and prints the sum of 2 numbers

```
#include <stdio.h>

void add( int m, int n ) {
    int sum = 0;
    sum = m + n;
    printf("\nSum = %d\n", sum);
}

int main(int argc, char *argv[]) {
    int m = 21, n = 14;
    add(m, n);
    return 0;
}
```

Function with no return Value

Function with no return value

✧ **Function that prints if a number is divisible by 7 or not?**

```
#include <stdio.h>

void div7( int n ) {
        if ( (n % 7) == 0 )
                printf("\n%d is divisible by 7\n", n);
        else
                printf("\n%d is NOT divisible by 7\n", n);

        return;
}

int main(int argc, char *argv[]) {
        int n = 28;
        div7(n);

        return 0;
}
```

return type is void

Optional (not needed!!)

Function with no return Value

Function with return value

- Problem: Compute the GCD of two numbers
- This function returns GCD of two given numbers to the main()

```
#include <stdio.h>
Int getGCD( int m, int n ) {    /* Computing GCD of m and n where m > n */
        int temp;
        while ( ( m % n) != 0 ) {
                temp= m % n;
                m = n;
                n = temp;
        }
        return n;
}
int main(int argc, char *argv[]) {
        int m = 21, n = 14, k;
        print("\n m  = %d, n = %d", );
        k = getGCD(m, n);
        print("\nGCD = %d", k);
        return 0;
}
```

Function getGCD() returns the GCD of m and n and stores GCD in the variable k

Mid-semester pattern discussion

- Consists of **~ total 10 to 12 questions** (Objective, theory, and Programming questions);
- Marks: **~ 20 marks**
- Duration **~ 1 to 1.5 Hours;**
- **These types of Questions can be asked:**
  - Type 1: **MCQ questions:** What will be output of the following programs
  - Type 2. Point out the **errors**, if any in the following C statements
  - Type 3: Evaluate the following **expression/ Number System**
  - Type 4: Theory Question (The concepts taught in the lecture)
  - Type 5: Write the C Program for the question
  - Type 6: Fill in the blanks with logic

Mid-Semester Syllabus

- All the topics covered till today (24th Sept)
- **Topics:** Introduction to Basic Fundamentals of Computers, Introduction to Programming, Identifiers and Constants, Data Types, Number System, Operators, Logical Expressions, Managing input & output, Conditional statements, Decision making & Branching, Decision making & loops, Arrays, Functions
- **Note: kindly refer lecture slides as well as textbooks mentioned in the lecture 1 slides for detailed theory & practice purpose.**
- I will upload the **question bank today** for your reference & the practice of coding.
- **Thursday (26th Sept) class would be of revision class**; **Send all the questions or topics that you want to revise;** Attendance will be given to all the students.
- I will upload the **question bank of Array & Function** today.
- We will be taking **graded lab 2 from 7th to 11th Oct.**

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 ... = $a^n + b^n + c^n + d^n + \ldots$
- 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

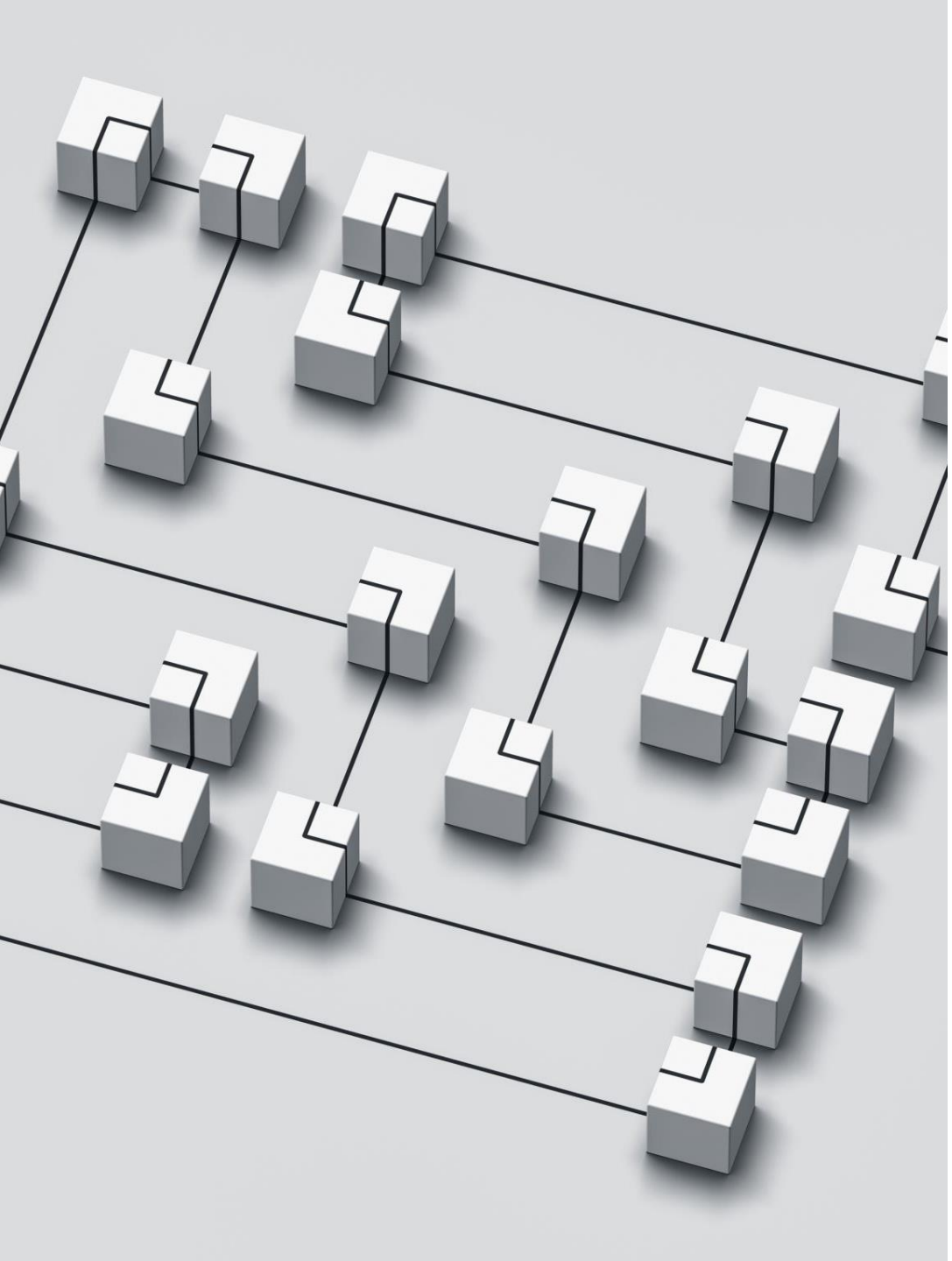
```
int isArmstrong(int num) {
    int act = num, rem, result = 0, n = 0;
    while (act != 0){
        act /= 10; n++;
    }
    act = num;
    while (act != 0) {
        rem = act % 10;
        result += pow(rem, n);
        act /= 10;
    }
    return ((result == num) ? 1 : 0);
}
```

You may or may not use Math Library

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 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

- Function with Arrays
- Macro & Inline functions
- Recursion