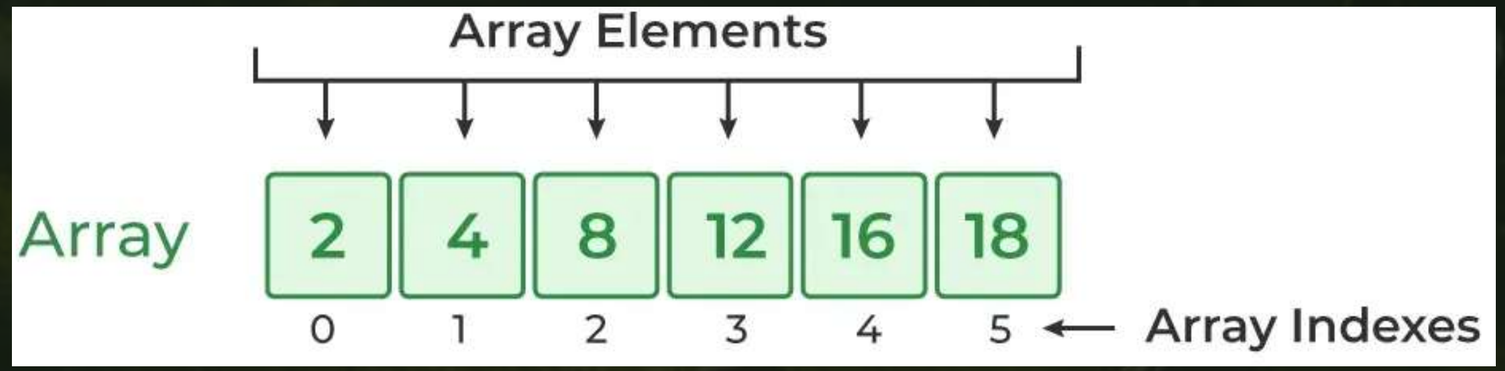
GRASSROOTS 2.0
DAY 2

TOPICS COVERED:

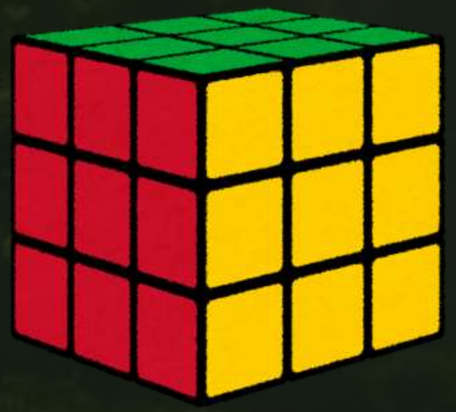
- Array , string
- Function, Recursion
- Calculator (mini project)

What is an Array?

- An Array is a Collection of elements of the same data type
- Stored in contiguous memory locations
- Accessed using an index (starts from 0)
- Useful for storing multiple values under one name

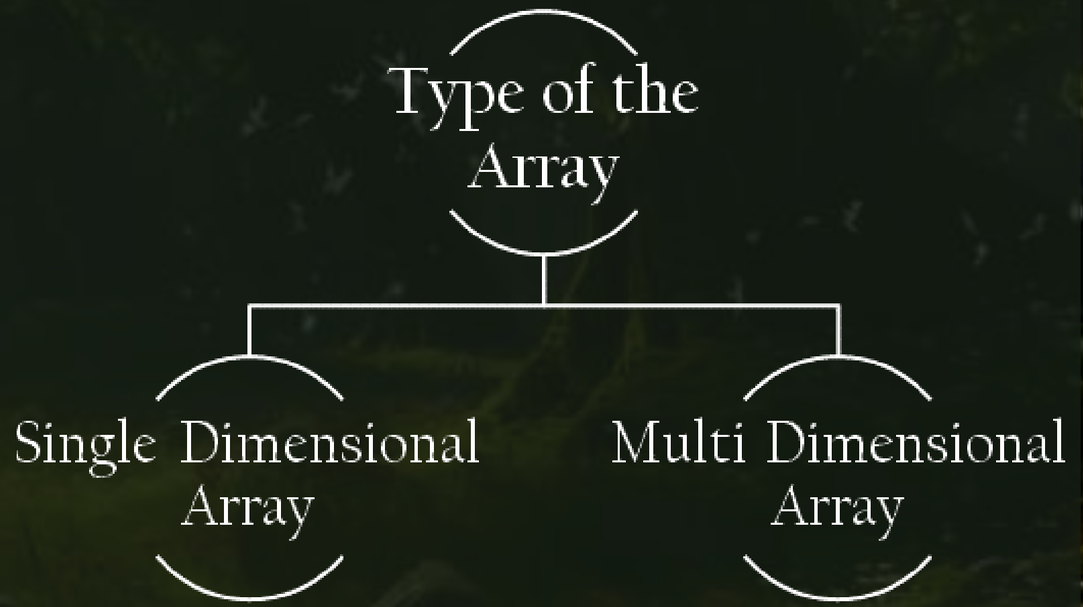


Real life Examples of Arrays



Why Arrays??

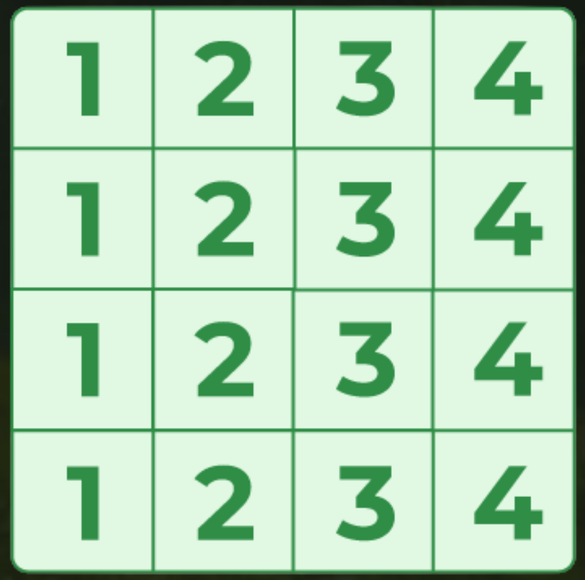
- Avoid multiple variable declarations
- Efficient in data storage & iteration
- Useful for searching & sorting
- Passing bulk data to functions



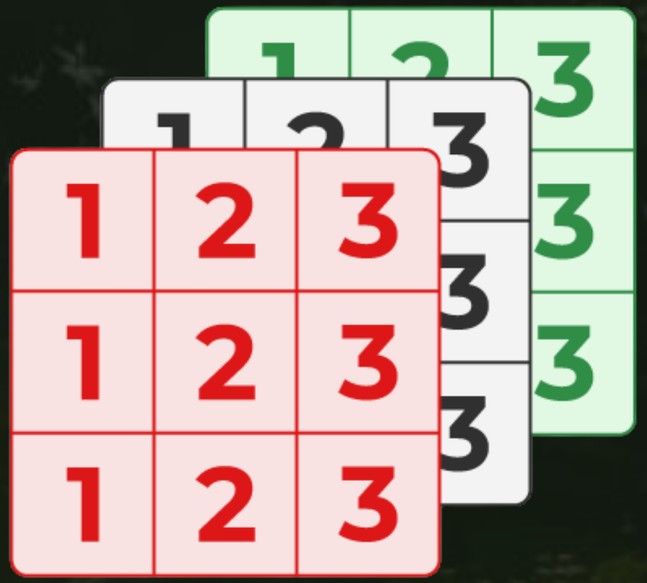
Types of Arrays

1D Array

2D Array



3D Array



Declaration of Array

```
// array of 6
int a[6];
```

Collection of 6 variables

Data Type

Array Name

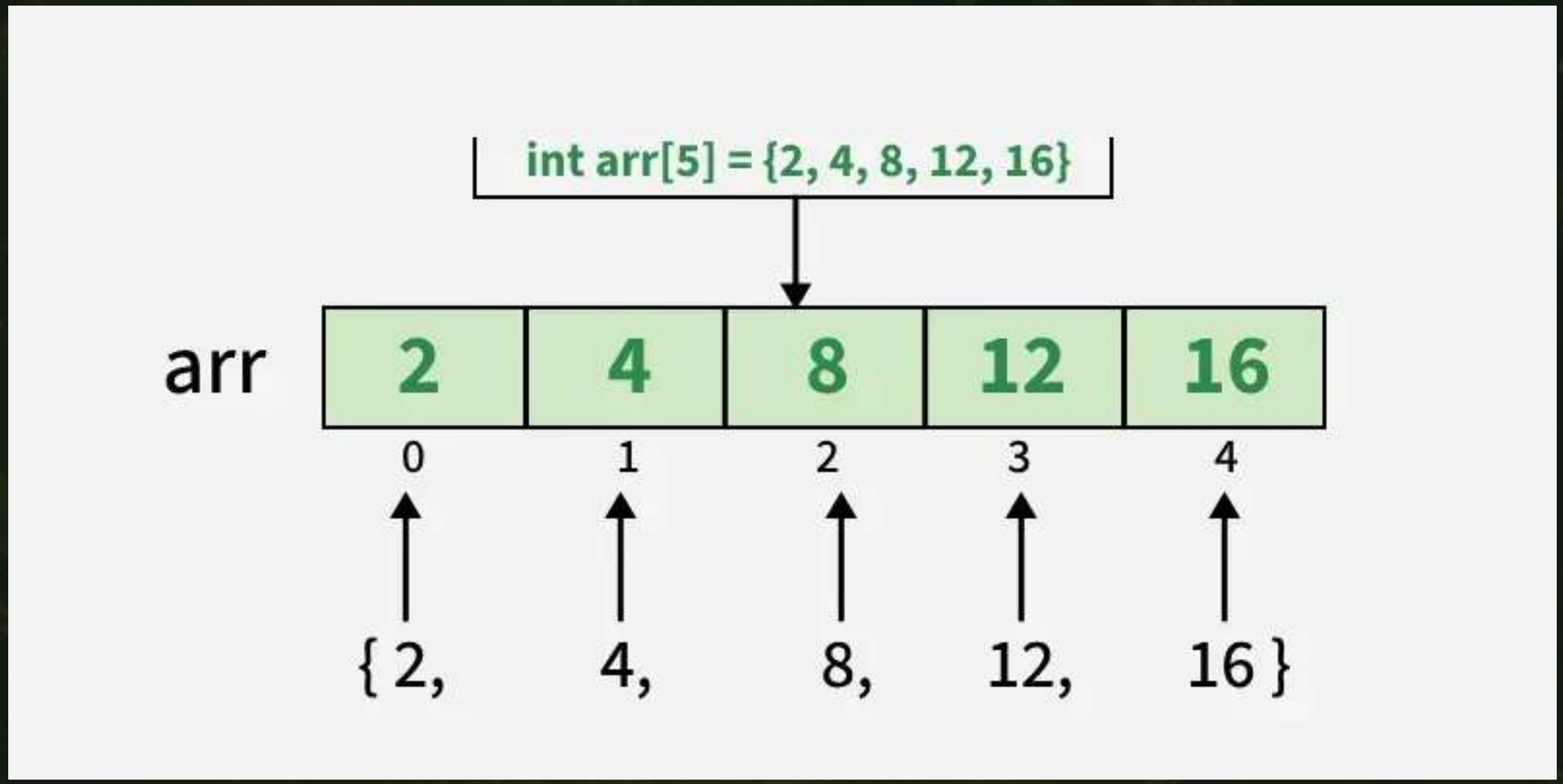
Array Declaration

Arr [5]; Size of Array = 5

Memory Allocated

Array Initialization

- Initialization in C is the process to assign some initial value to the variable.
- When the array is declared or allocated memory, the elements of the array contain some garbage value. So, we need to initialize the array to some meaningful values.

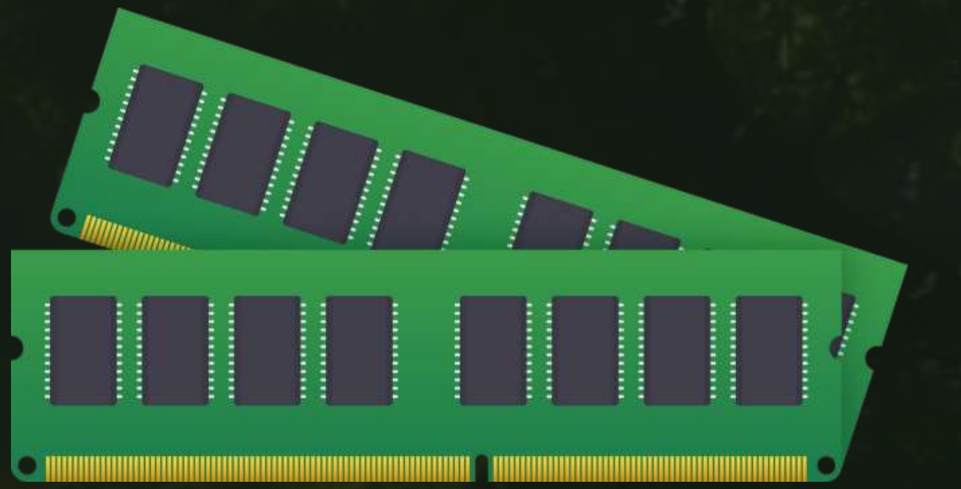

int arr[5] = {2, 4, 8, 12, 16}
arr
2
4
8
12
16
0
1
2
3
4
{ 2,
4,
8,
12,
16 }

Accessing Array Elements

- Access by index: arr[0], arr[1]...
- Example: arr[0] = 10;
- printf("%d", arr[2]);
- ⚠ Accessing out-of-bound indices causes undefined behavior

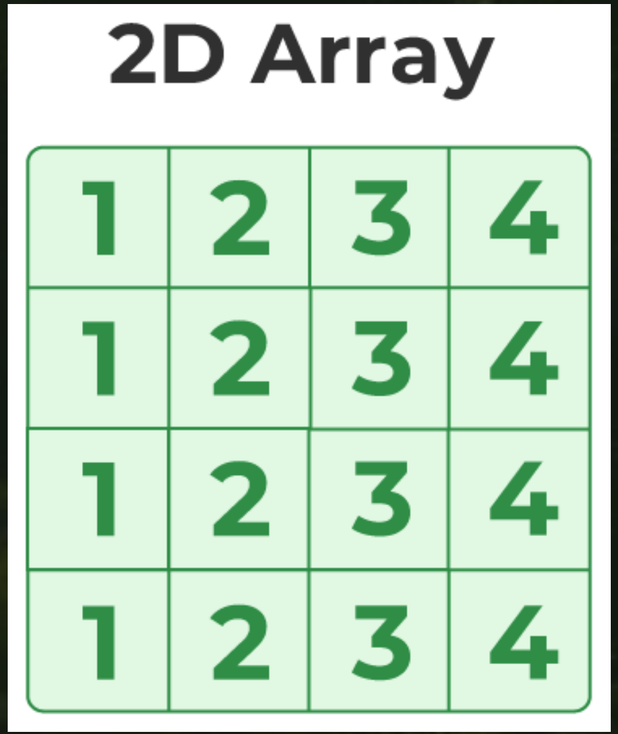
Memory Representation

- Example: int arr[5] = {10,20,30,40,50};
- For int: 4 bytes per element
- Addresses increase sequentially
- arr[0]=10 at 1000, arr[1]=20 at 1004, ... arr[4]=50 at 1016



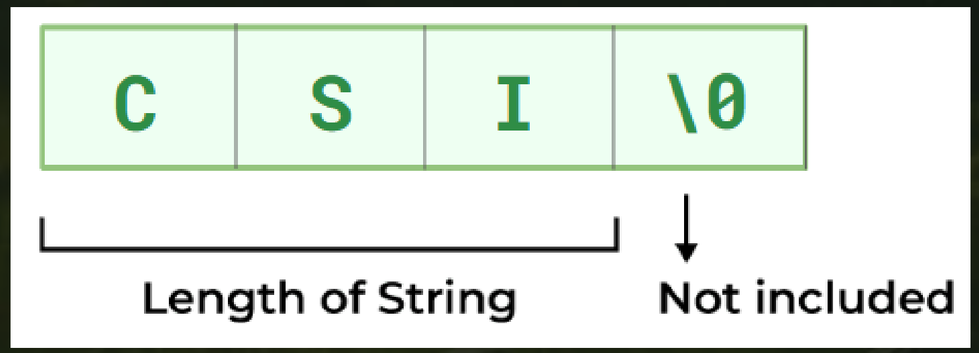
2D Arrays

- Declaration: int matrix[3][3];
- Initialization with rows & columns
- Access: matrix[1][2] = 3



Strings (Character Arrays)

- Strings are character arrays ending with '\0'
- char name[6] = "Hello";
- Equivalent: {'H','e','l','l','o','\0'}

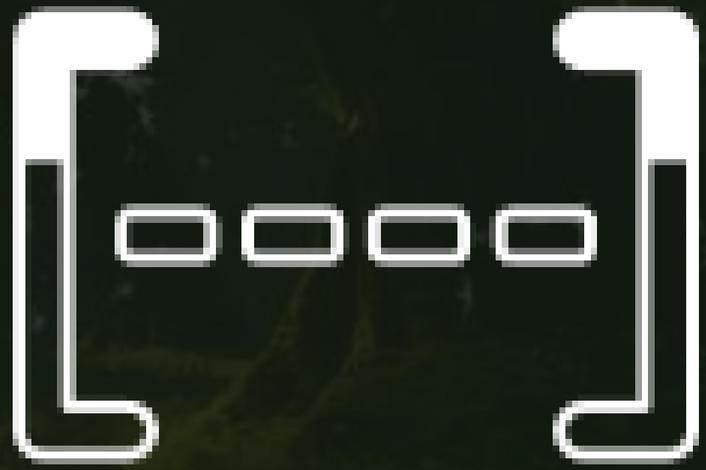


Common Mistakes

- Accessing out-of-bound elements
- Forgetting null character in strings
- Using uninitialized arrays
- Treating arrays as value types

Advantages of Arrays

- Easy to declare & use
- Random access by index
- Better memory management
- Foundation for advanced data structures

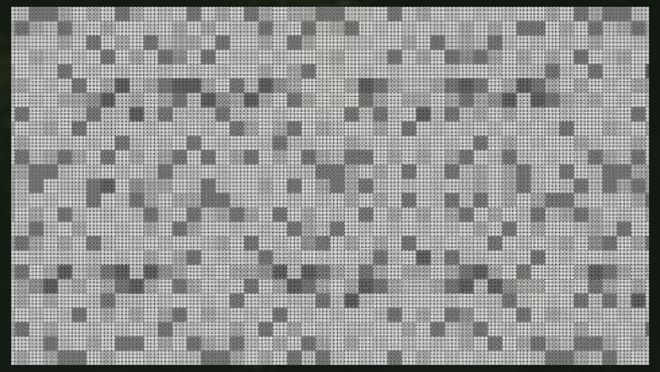


Limitations of Arrays

- Fixed size (cannot grow/shrink)
- All elements must be same type
- Memory wastage if oversized
- No built-in bounds checking

Real-Life Examples

- Storing student marks
- Matrix operations
- Image pixels
- Array of strings



Summary

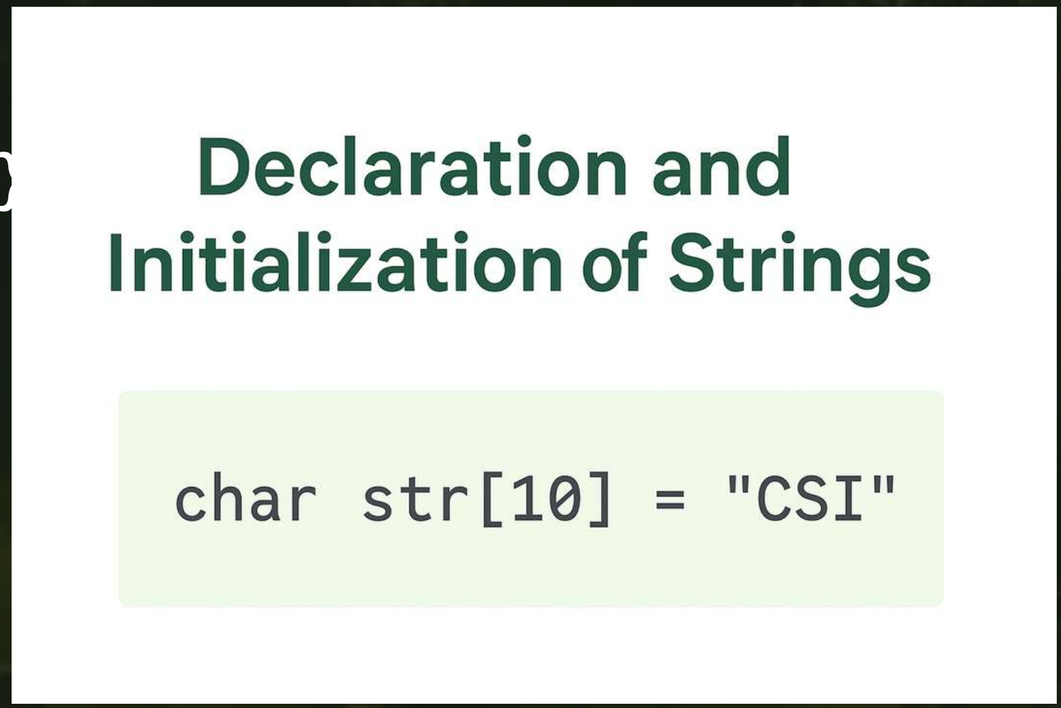
- Array = Collection of similar data stored contiguously
- Types: 1D, 2D, Multidimensional
- Array name = pointer to first element
- Strings = Character arrays
- Efficient but fixed size

Introduction to Strings in C

- A string in C is an array of characters stored in continuous memory.
- Each string ends with a null character ('\0') to mark its termination.
- C does not have a special string type like other languages.
- Strings are created using character arrays or pointers.
- Example:
    - char str[10] = "Hello";

Declaration and Initialization

- Ways to declare strings:
- char str1[] = "Hello";
- char str2[6] = {'H','e','l','l','o','\0'}
- char *str3 = "Hello";



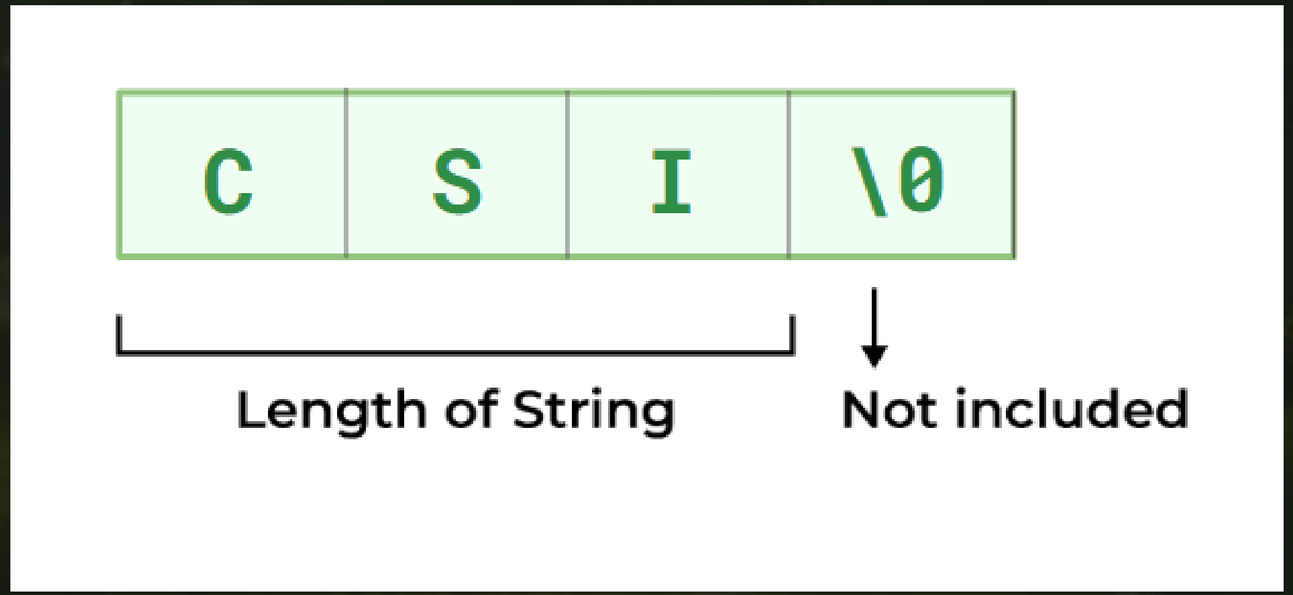
Input and Output of Strings

- **Input**
  - a. **scanf("%s", str);** → reads word (stops at space)
  - b. **gets(str);** → ⚠ unsafe, avoid
  - c. **fgets(str, size, stdin);** → safe, can read spaces

- **Output**
  - a. **printf("%s", str);** → print string
  - b. **puts(str);** → print + newline

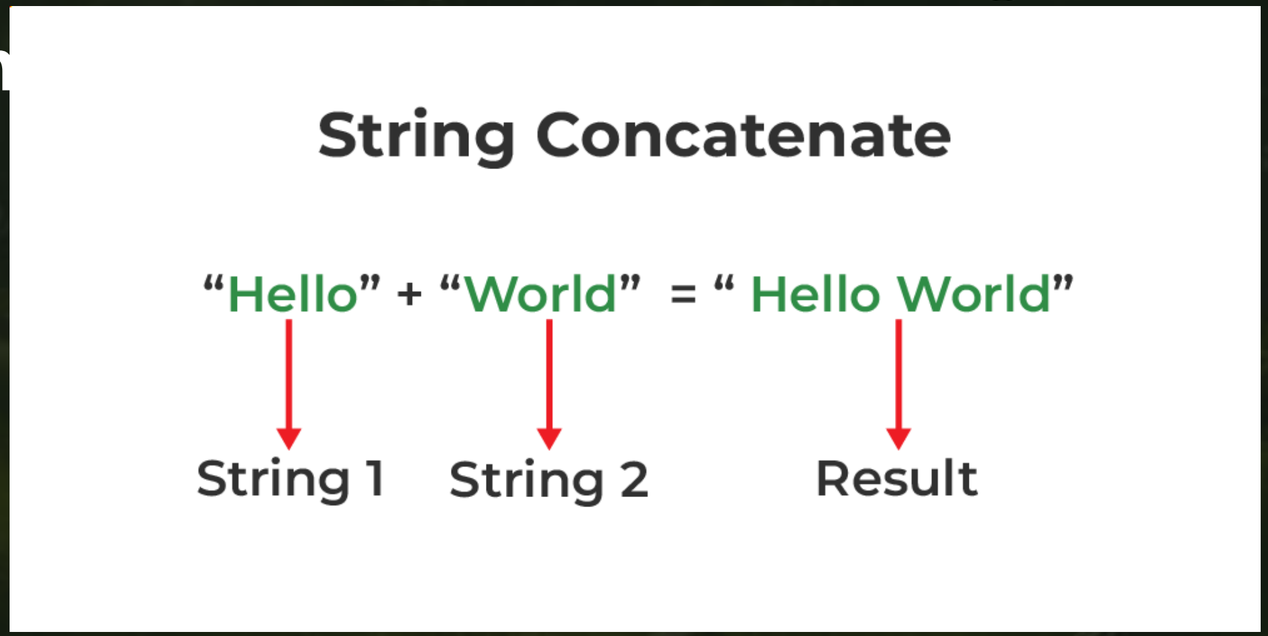
String Functions (string.h)

strlen(str) → Returns length of the string (excluding \0)



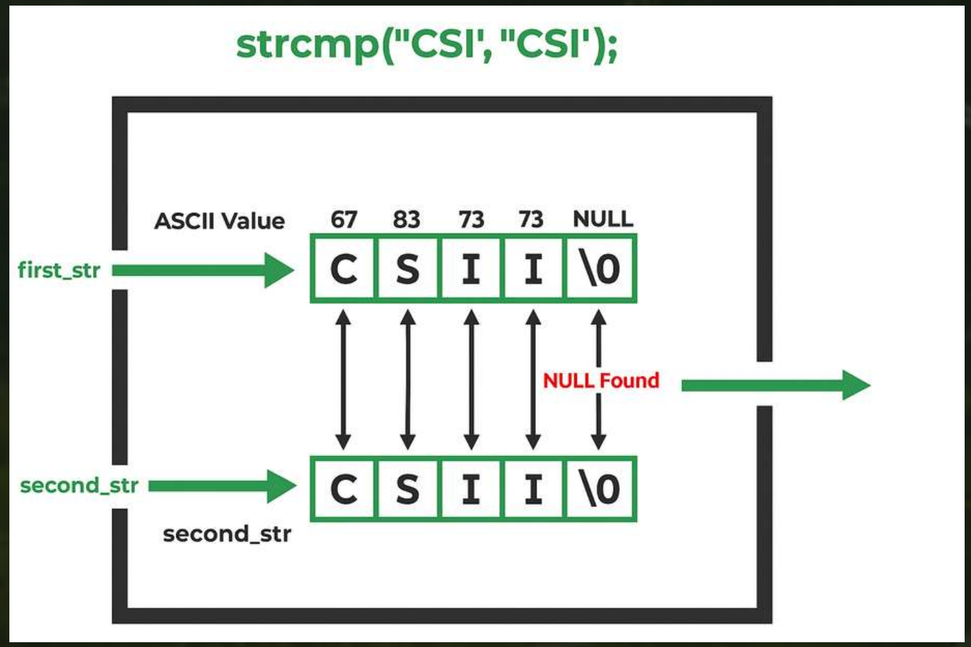
String Functions (string.h)

strcat(s1, s2) → Appends one string to anoth



String Functions (string.h)

- strcmp(s1, s2) → Compares two strings (returns 0 if equal).



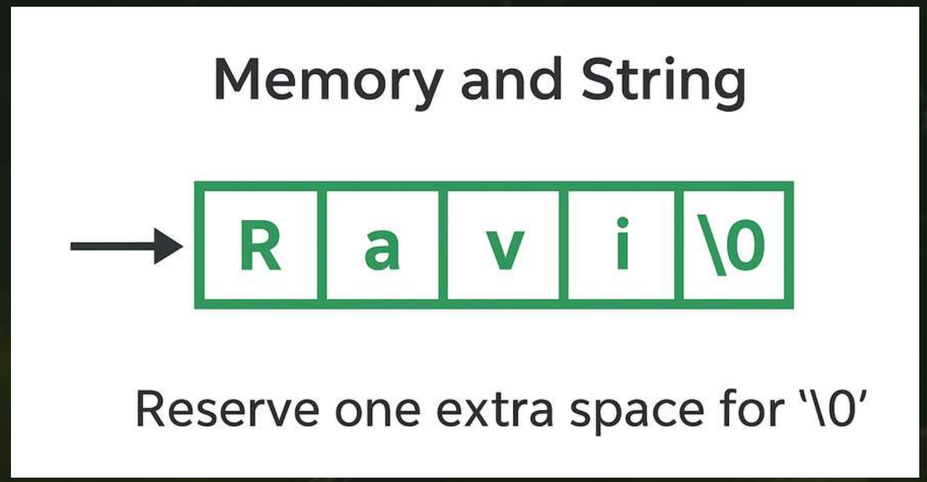
String Functions (string.h)

Other functions:

- **strcpy(dest, src)** → Copies one string into another.
- **strchr(str, ch)** → Finds first occurrence of a character.
- **strstr(str, sub)** → Finds first occurrence of a substring.

Memory and String

- Strings are stored as continuous blocks of memory.
- Always allocate one extra slot for the null terminator '\0'.
- **Example:**
  - char name[6] = "Ravi";



Common Pitfalls in C Strings

- **Forgetting the null terminator ('\0') →** leads to undefined behavior.
- **Buffer overflow →** occurs when writing beyond allocated memory.
- **Using gets() →** unsafe, prefer fgets() instead.
- **Comparing strings with == →** wrong, always use strcmp().

Example Program

```c
#include <stdio.h>
#include <string.h>

int main() {
    char s1[20] = "Hello";    //destination string
    char s2[] = "World";      //source string

    strcat(s1, s2);           //concatenate s2 to s1
    printf("%s", s1);         //output: HelloWorld
    return 0;
}
```

Summary

- Strings are simply arrays of characters.
- Every string must be null-terminated ('\0').
- Can be declared in multiple ways (array or pointer).
- Use string.h functions for common operations.
- Always watch out for memory limits and buffer overflows.

Functions

- A function is a piece of code that performs a specific task.
- functions are also called modules or procedures.

it can be used **multiple** times

increase code reusability

Function Syntax

Function Declaration

```
void printHello( );
```

> Tell the compiler

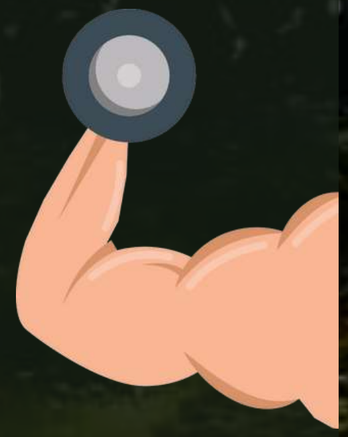
Function Syntax

Function Definition

```
void printHello() {
printf("Hello");
}
```

- void printHello() → defines a function named printHello that takes no input and returns nothing.
- { printf("Hello"); } → body of the function that tells it what task to perform when called.

Do the Work <



Function Syntax

Function Call

```
int main() {
printHello( );
return 0;
}
```

- printHello(); → calls the function to actually do its job.
- It runs the code inside printHello, carrying out the work defined there.

Use the Work <

Function Advantages

- **Reusability** – Write once, use many times.
- **Modularity** – Break program into smaller parts.
- **Readability** – Code is easier to read and maintain.
- **Simplicity** – Reduces program complexity.
- **Testing** – Functions can be tested independently.

Problem

Add two integers using functions

```c
// Function declaration
int add(int a, int b);       //parameter/formal parameter

int main() {
   int num1, num2, result;

   // Taking input from user
   printf("Enter first number: ");
   scanf("%d", &num1);

   printf("Enter second number: ");
   scanf("%d", &num2);

   // Function call
   result = add(num1, num2);      //argument/actual parameter

   // Displaying the result
   printf("The sum of %d and %d is: %d\n", num1, num2, result);

   return 0;
}
```

```c
// Function definition
int add(int a, int b)
{
   return a + b;
}
```

Problems

Q)Write a function if user is Indian it prints “Namaste” and if user is French it prints “Bonjour”

APPROACH

- Ask the user to enter their nationality (like "Indian" or "French").
- Check the input using a condition (if it matches "Indian", then do one task; if it matches "French", then do another).
- If the user is Indian, the function will print Namaste.
- If the user is French, the function will print Bonjour.
- Optionally, handle any other input by printing something like "Invalid choice".

Problems

Q)Write functions to calculate area of square, a circle & a rectangle.

APPROACH

- In the main function, call the greeting function after taking or assuming the user's nationality.
- The function then does its defined work: prints Namaste if Indian, or Bonjour if French.

Function Properties

- Execution always starts from main
- A function gets called directly or indirectly from main
- There can be multiple functions in a program

Function Types

Library function

Special functions inbuilt in C

Eg. scanf(), printf()

User-defined

declared & defined by programmer

Problems encountered when user- defined functions are not used

When a C program is coded utilizing only main function, the problems encountered are

- Difficulty in Debugging
- Difficulty in Testing
- Difficulty in Maintenance
- When same operation or calculation is to be repeated, space and time is wasted by repeating the program statements wherever they are needed.

Arguments vs Parameters

values that are passed in function call	values in function declaration & definition
used to send value	used to receive value
actual parameter	formal parameters

Note:
1)Function can only return one value at a time.
2)Changes to parameters in function don’t change the values in calling function
Because a copy of argument is passed to function.

Recursion

When a called function in turn calls another function a process of chaining occurs. Recursion is a special case of this process, where a function calls itself.

Example

```
void main ( )
{
printf ( " \n This is an example of recursion");
main ( ) ;        <-------- Recursive function call
}
```

Properties of Recursion

a. Anything that can be done with Iteration, can be done with recursion and vice-versa.

b. Recursion can sometimes give the most simple solution.

c. Base Case is the condition which stops recursion.

d. Iteration has infinite loop & Recursion has stack overflow

Advantages & Disadvantages

Advantages

- Easy solution for recursively defined problems.
- Complex programs can be easily written in less code.

Disadvantage

- Recursive code is difficult to understand and debug
- Terminating condition is must, otherwise it will go in infinite loop.
- Execution speed decreases because of function call and return activity many times.

Real Life Examples

- File directories – Opening nested folders.
- Tower of Hanoi – Solving disk moves.
- Math problems – Factorial, Fibonacci, power.
- Tree/Graph – Traversing nodes step by step.
- Undo/Redo – Tracing back previous states.

Problems

Print Hello world 5 times using recursion:

Approach:

- Create a recursive function that prints "Hello World".
- Each time it runs, decrease a counter (like from 5 to 0).
- Stop the recursion when the counter reaches 0.

Output:

Hello World
Hello World
Hello World
Hello World
Hello World

Problems

Print sum of n natural number using recursion

Approach:

- Define a recursive function that takes n as input.
- At each step, add n to the result of the function called with n-1.
- Stop when n becomes 0 (base case).

Output (for n = 5):

Sum = 15

Problems (factorial)

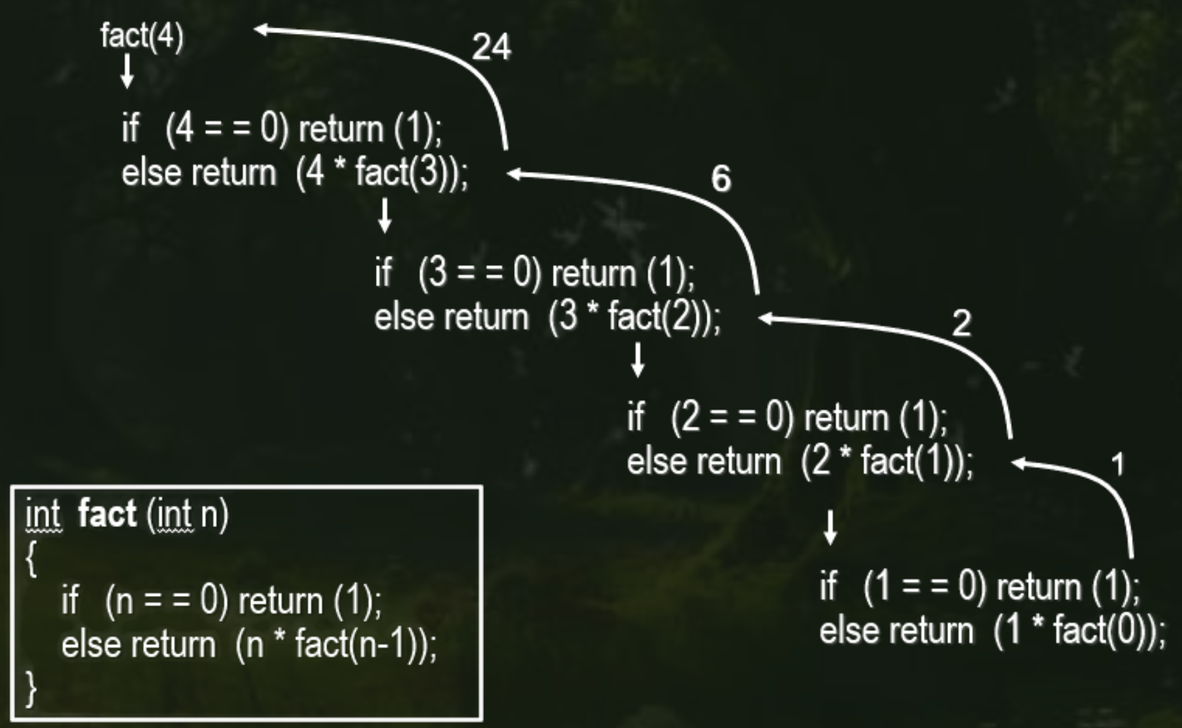
Print factorial of a number using recursion:

Approach:

- Define a recursive function that takes n as input.
- Multiply n with the result of the function called with n-1.
- Stop when n reaches 1 (base case).

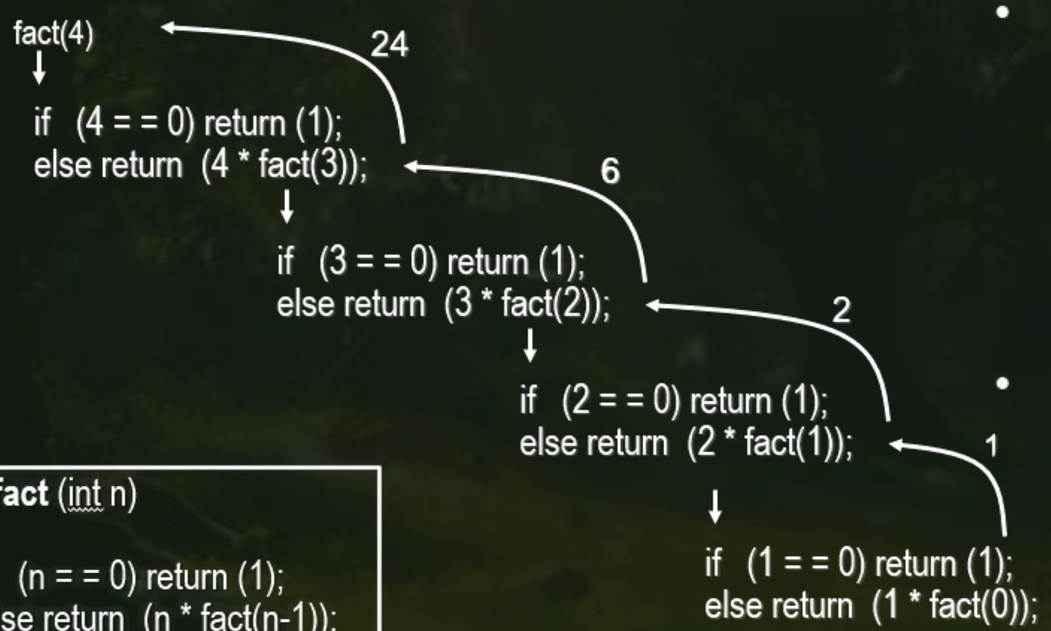
Output (for n = 5):

Factorial = 120




```
int fact (int n)
{
   if (n = = 0) return (1);
   else return (n * fact(n-1));
}
```

Problems (factorial)




```
int fact (int n)
{
   if (n = = 0) return (1);
   else return (n * fact(n-1));
}
```

- **First, the function calls will be processed**

  fact(4) = 4 * fact(3)
  fact(3) = 3 * fact(2)
  fact(2) = 2 * fact(1)
  fact(1) = 1 * fact(0)

- **The actual values return in the reverse order**

  fact(0) = 1
  fact(1) = 1 * 1 = 1
  fact(2) = 2 * 1 = 2
  fact(3) = 3 * 2 = 6
  fact(4) = 4 * 6 = 24

Problems

Write a recursive function to reverse a given number.

Approach:

- Create a recursive function that extracts the last digit of the number using modulus (%).
- Append this digit to the reversed number while reducing the original number using division (/).
- Continue recursion until the number becomes 0 (base case).

Input: 1234
Output: 4321

Problems (fibonacci)

Write a recursive function to generate the Fibonacci series up to n terms.

Approach:

- Define a recursive function for Fibonacci that returns 0 if n=0 and 1 if n=1.
- For other cases, return the sum of the function called with (n-1) and (n-2).
- This keeps breaking the problem into smaller Fibonacci numbers until base cases are reached.

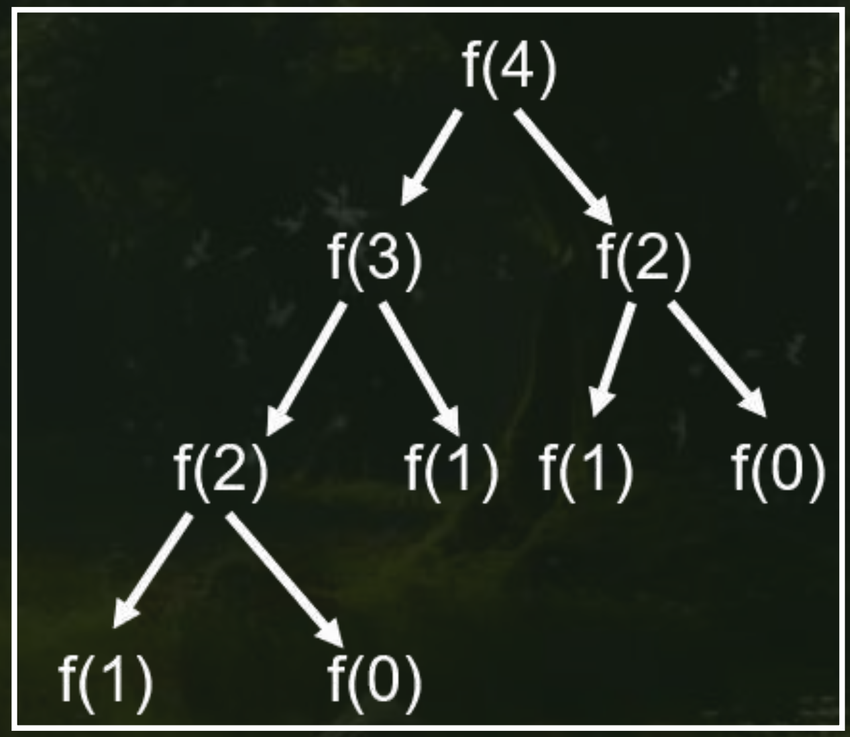
Input: 5 terms

Output: 0 1 1 2 3

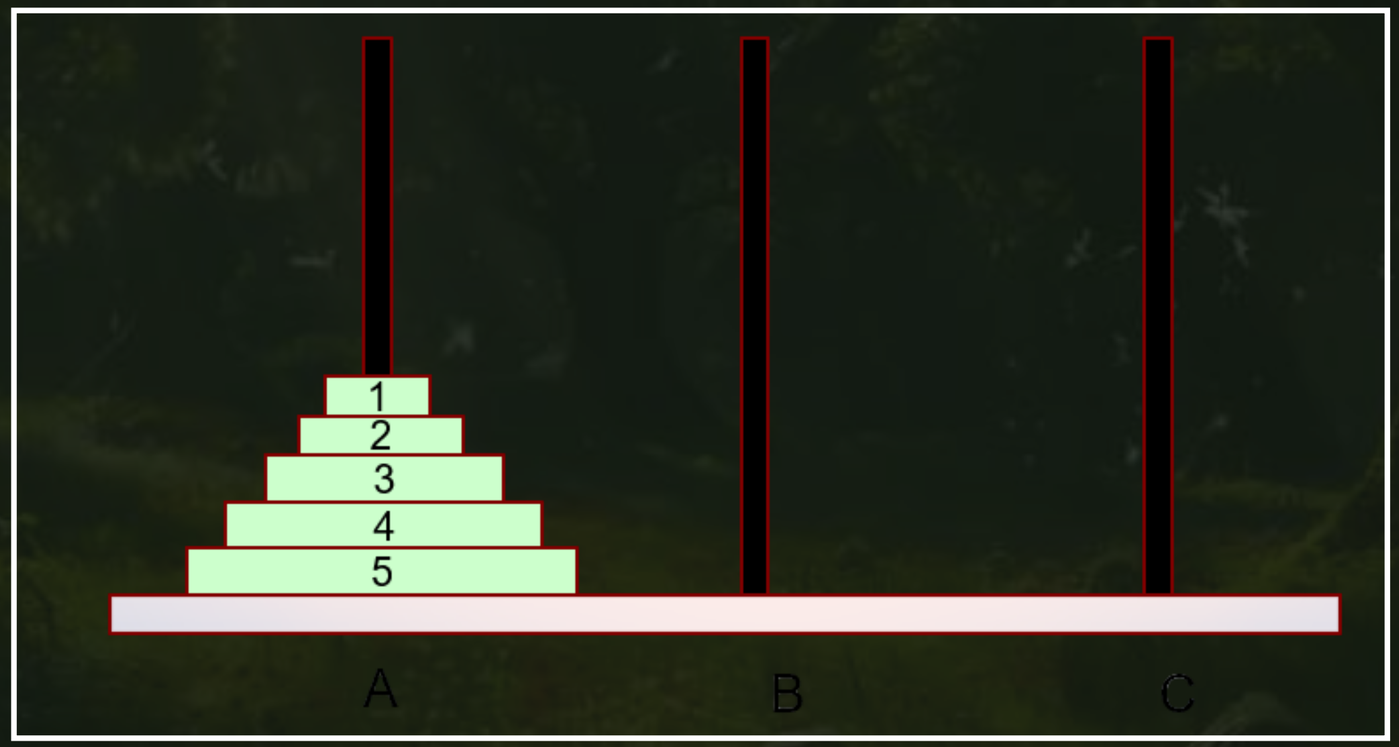
Problems (fibonacci)

```
f(0)   =   0
f(1)   =   1
f(n)   =  f(n-1)+f(n-2),if  n>1
```

```
int f(int n)
{
      if (n < 2)
            return (n);
      else
            return (f(n-1) + f(n-2));
}
```

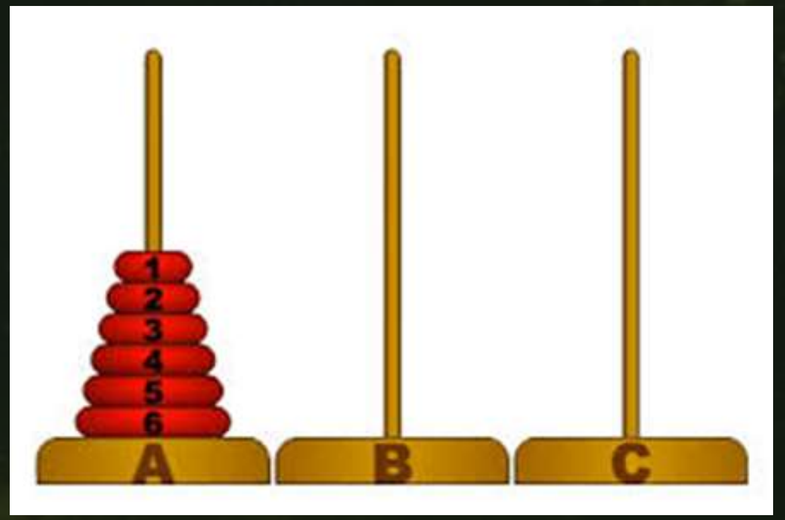


Tower Of Hanoi (Conceptual approach)

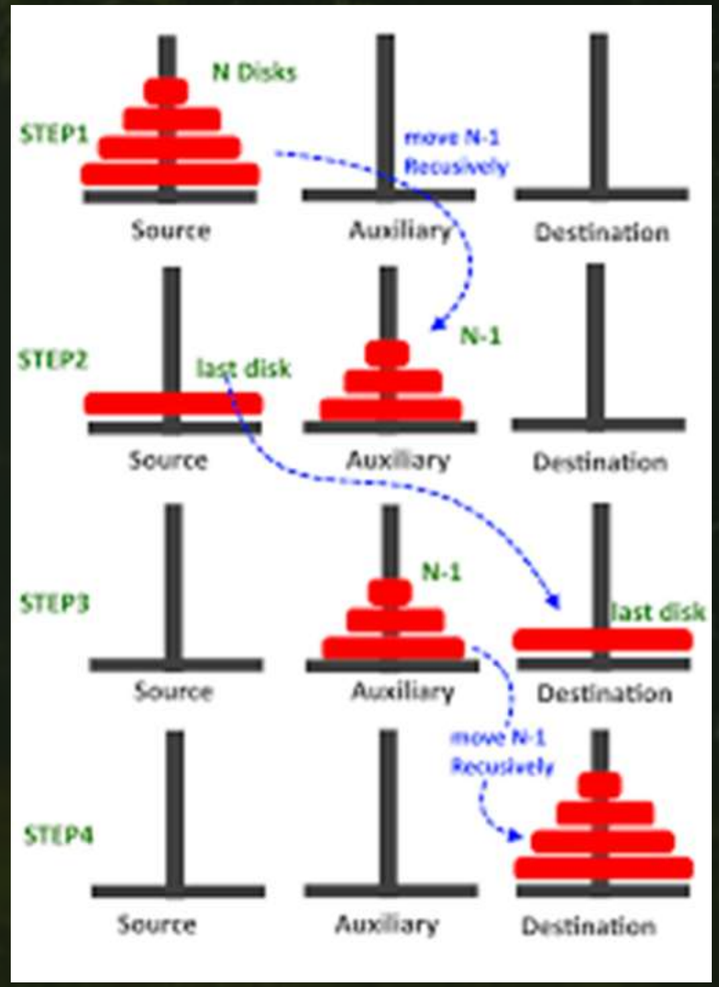


Tower Of Hanoi (Conceptual approach)

- **The problem statement**
  - Initially all the disks are stacked on the A pole.
  - Required to transfer all the disks to the C pole.
    - Only one disk can be moved at a time.
    - A larger disk cannot be placed on a smaller disk.
  - C pole is used for temporary storage of disks.



Tower Of Hanoi (Conceptual approach)



Calculator

Quiz Time