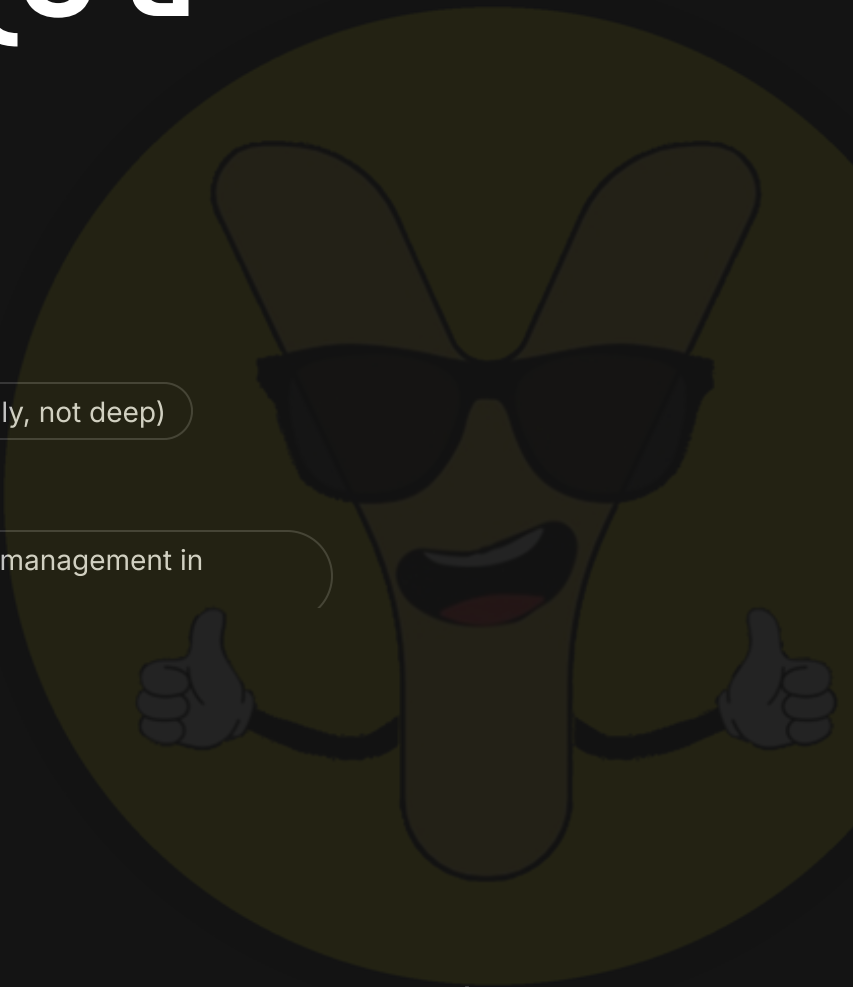
AKTU · 1ST YEAR · SEMESTER 1 · ACS101

Programming Languages (C & Python)

Unit 3 — Advanced Programming Concepts (C)

- 1D array — declaration, input, output, traversal
- Simple operations — find max, min, sum, average
- Strings in C — gets/puts, strlen, strcpy, strcmp
- Pointer — concept of address and pointer variable (concept only, not deep)
- Structures
- File Handling

Real connect: Arrays store sensor readings, image pixels, and stock prices. Pointers are the foundation of memory management in embedded systems.



DECODED BY

Chirag Jain

Founder, Ycotes · Gold Medalist in Artificial Intelligence, IIT Madras · taught 10,000+ students

Unit 3 Overview: Advanced Programming Concepts in C

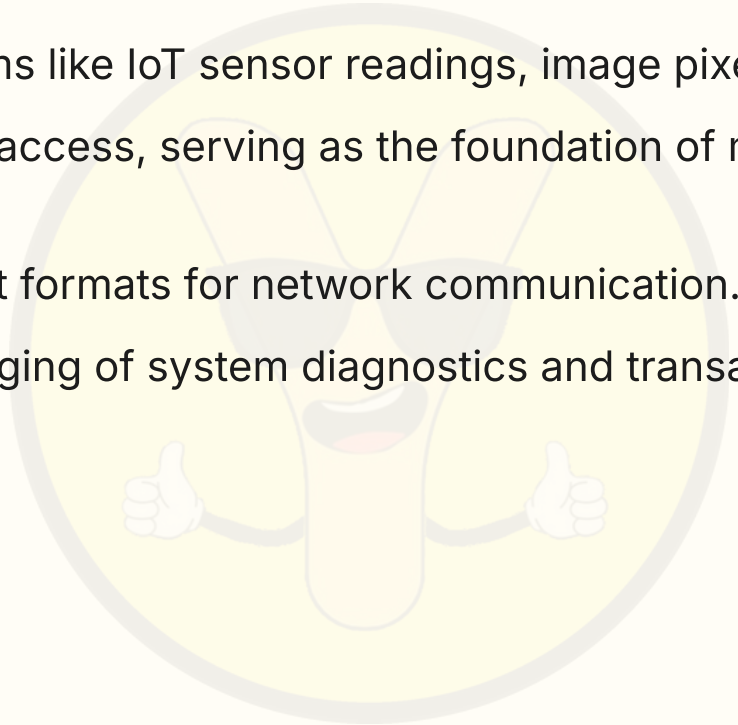
This unit covers structured data handling, memory referencing, complex user-defined types, and persistent storage.

- 1D Arrays: Contiguous memory storage for homogeneous data collections.
- Simple Operations: Algorithms for traversal, maximum, minimum, sum, and average.
- Strings: Character arrays and standard library functions for manipulation.
- Pointers: Direct memory address referencing concepts.
- Structures: User-defined data types grouping diverse variables.
- File Handling: Reading from and writing to external disk files.

Real-World Connection

Advanced C concepts map directly to industrial applications and hardware constraints.

- Arrays store continuous data streams like IoT sensor readings, image pixels, and historical stock prices.
- Pointers provide low-level memory access, serving as the foundation of memory management in embedded systems.
- Structures organize complex packet formats for network communication.
- File handling ensures persistent logging of system diagnostics and transaction records.



1D Array Declaration and Memory Layout

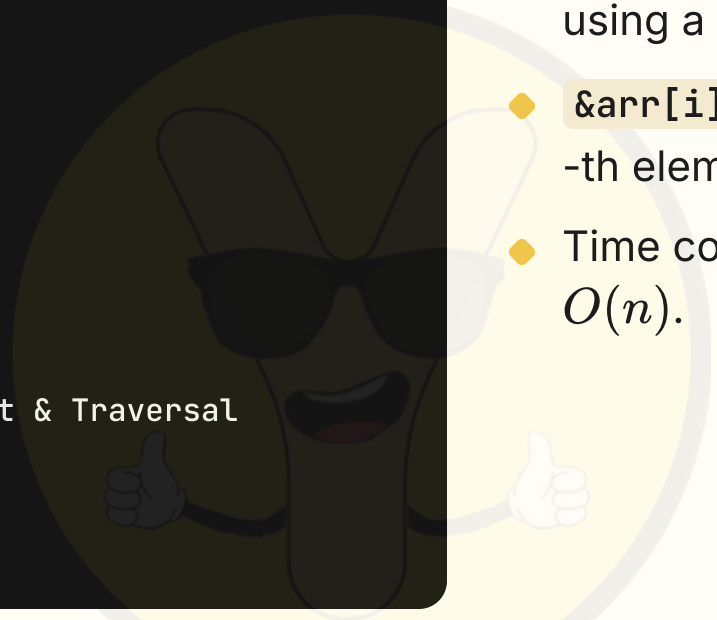
ONE-DIMENSIONAL ARRAY

A fixed-size collection of elements of the same data type stored in contiguous memory locations.

- Syntax: `data_type array_name[size];`
- Example: `int marks[5];` allocates 20 contiguous bytes (assuming 4 bytes per integer).
- Indexing: Elements are indexed from `0` to `size - 1`.
- Advantage: Enables efficient random access using index offsets.

1D Array Input, Output, and Traversal

```c
#include <stdio.h>
int main() {
    int arr[5], i;
    printf("Enter 5 numbers: ");
    for(i = 0; i < 5; i++) {
        scanf("%d", &arr[i]); // Input
    }
    printf("Array elements: ");
    for(i = 0; i < 5; i++) {
        printf("%d ", arr[i]); // Output & Traversal
    }
    return 0;
}
```



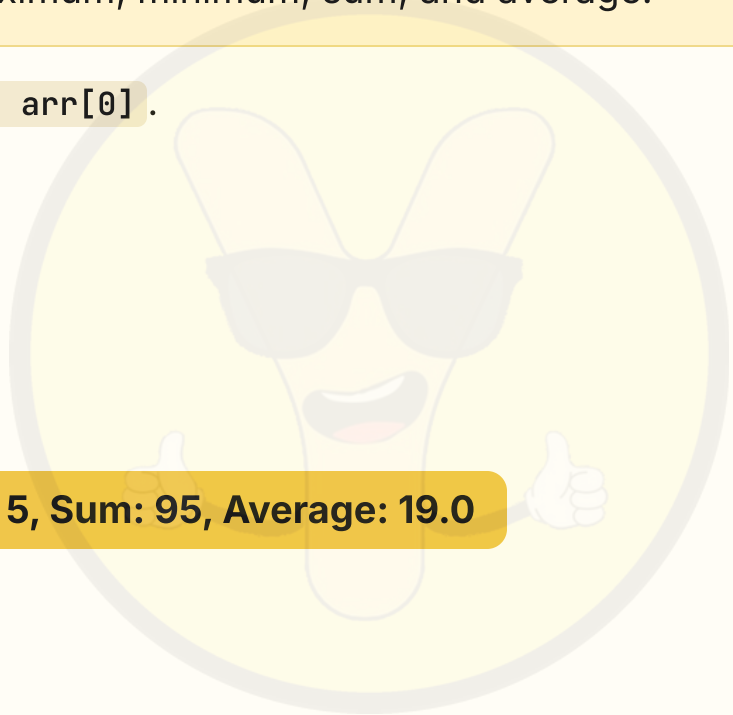
- Traversal visits every element sequentially using a loop.
- `&arr[i]` passes the memory address of the i-th element to `scanf`.
- Time complexity for complete traversal is $O(n)$.

Operations: Find Max, Min, Sum, and Average

Given an array of 5 integers, find the maximum, minimum, sum, and average.

1. Initialize `sum = 0`, `max = arr[0]`, `min = arr[0]`.
2. Loop from `i = 1` to `4`.
3. Add each element to `sum`.
4. If `arr[i] > max`, update `max = arr[i]`.
5. If `arr[i] < min`, update `min = arr[i]`.
6. Calculate `average = (float)sum / 5`.

Array [10, 25, 5, 40, 15] → Max: 40, Min: 5, Sum: 95, Average: 19.0



Program for Max, Min, Sum, and Average

```c
#include <stdio.h>
int main() {
    int a[5] = {10, 25, 5, 40, 15], i, sum = 0;
    int max = a[0], min = a[0];
    for(i = 0; i < 5; i++) {
        sum += a[i];
        if(a[i] > max) max = a[i];
        if(a[i] < min) min = a[i];
    }
    float avg = (float)sum / 5;
    printf("Sum:%d, Max:%d, Min:%d, Avg:%.2f", sum, max,
min, avg);
    return 0;
}
```



- Single-pass traversal computes all four aggregate metrics efficiently.
- Type casting `(float)sum` prevents integer division truncation.

Strings in C: Basics and I/O

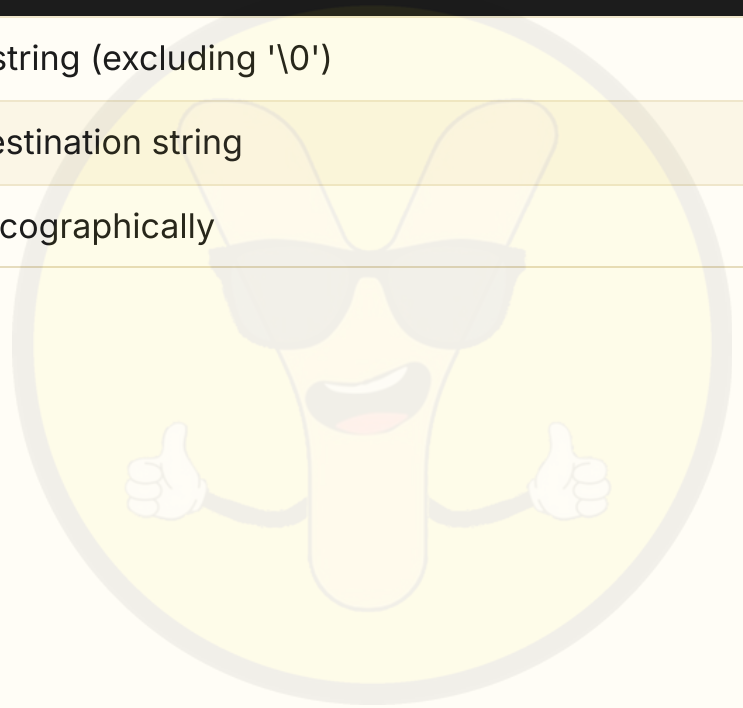
NULL-TERMINATED STRING

A one-dimensional array of characters terminated by a null character '\0' (ASCII value 0).

- Declaration: `char name[20] = "AKTU";`
- `gets(str)` reads a line of text until a newline (deprecated due to buffer overflow risks).
- `puts(str)` prints the string followed by a newline character.
- Always ensure destination arrays are large enough to hold the null terminator.

Standard String Library Functions (<string.h>)

Function	Description	Syntax Example
strlen()	Returns the length of the string (excluding '\0')	int len = strlen(str);
strcpy()	Copies source string to destination string	strcpy(dest, src);
strcmp()	Compares two strings lexicographically	int res = strcmp(s1, s2);



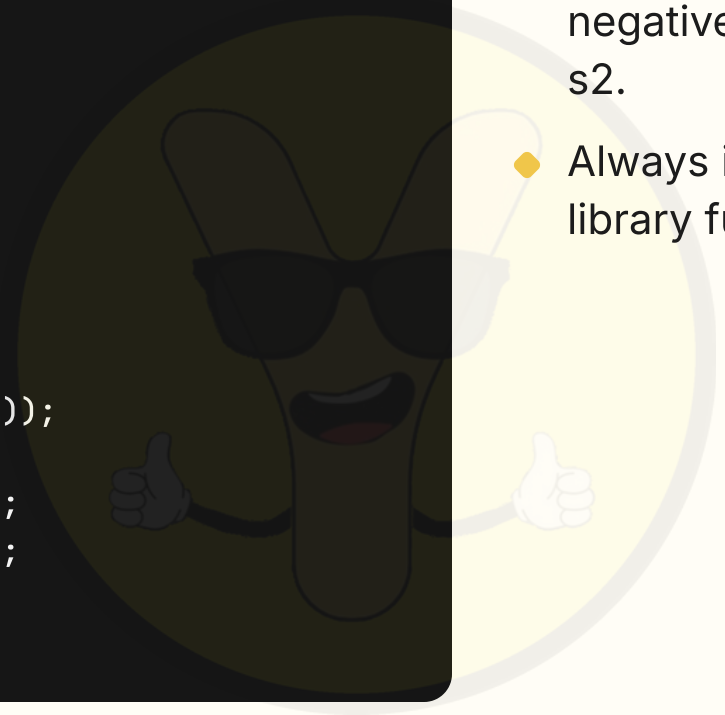
String Operations Example

```c
#include <stdio.h>
#include <string.h>
int main() {
    char str1[20] = "Hello";
    char str2[20];

    strcpy(str2, str1);
    printf("Copied: %s\n", str2);

    printf("Length: %lu\n", strlen(str1));

    int cmp = strcmp("Apple", "Banana");
    printf("Compare result: %d\n", cmp);
    return 0;
}
```



- `strcmp` returns 0 if strings are equal, a negative value if s1 < s2, and positive if s1 > s2.
- Always include `<string.h>` when using these library functions.

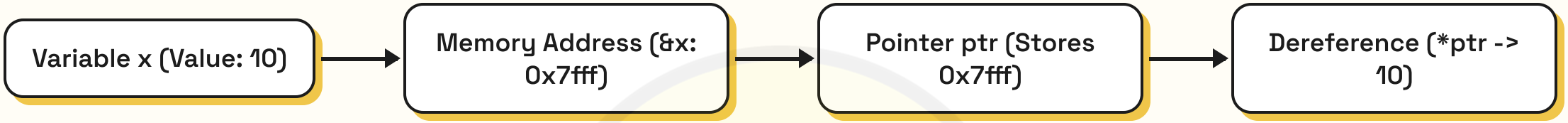
Pointers: Address and Variable Concept

POINTER VARIABLE

A variable that stores the memory address of another variable rather than its actual value.

- Address-of operator (`&`): Returns the memory address of a variable.
- Dereference operator (`*`): Accesses the value stored at the address held by a pointer.
- Declaration: `int *ptr;` declares a pointer to an integer.
- Crucial for dynamic memory allocation, passing arguments by reference, and direct hardware manipulation.

Concept of Pointer Mapping



Relationship between a normal variable, its memory address, and a pointer variable.

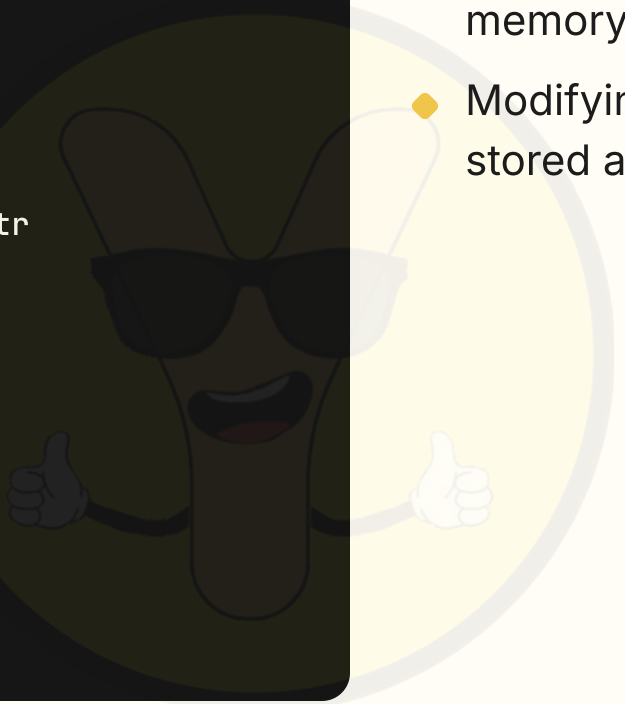
Basic Pointer Usage

```c
#include <stdio.h>
int main() {
    int var = 25;
    int *ptr;

    ptr = &var; // Store address of var in ptr

    printf("Value of var: %d\n", var);
    printf("Address of var: %p\n", &var);
    printf("Value via pointer: %d\n", *ptr);

    *ptr = 50; // Modify var through pointer
    printf("New value of var: %d\n", var);
    return 0;
}
```



- `%p` is the format specifier used to print memory addresses in hexadecimal.
- Modifying `*ptr` directly updates the value stored at the referenced memory location.

Structures in C

STRUCTURE

A user-defined data type that allows grouping items of different types under a single name.

- Keyword: `struct`
- Syntax: `struct Student { char name[50]; int roll; float marks; };`
- Members are accessed using the dot (`.`) operator.
- Contrasts with arrays: arrays store homogeneous elements; structures store heterogeneous elements.

Structure Declaration and Usage

```c
#include <stdio.h>
struct Student {
    char name[50];
    int roll;
    float marks;
};
int main() {
    struct Student s1;
    printf("Enter roll and marks: ");
    scanf("%d %f", &s1.roll, &s1.marks);

    printf("Roll: %d, Marks: %.2f\n", s1.roll, s1.marks);
    return 0;
}
```



- `struct Student s1;` creates a variable of type `struct Student`.
- Memory allocated is the sum of sizes of all members (plus padding alignment).

File Handling in C

FILE POINTER

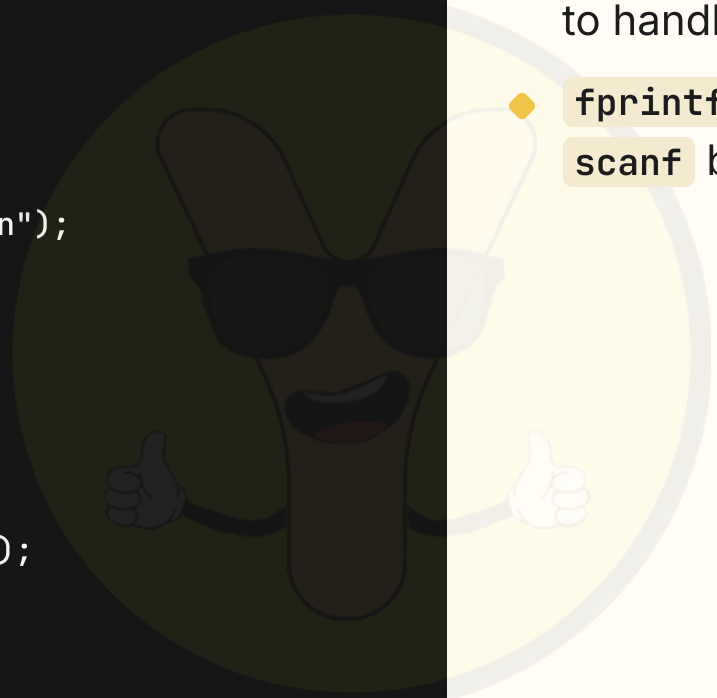
A mechanism that enables C programs to perform input and output operations on secondary storage files.

- Data type: `FILE *`
- Opening a file: `fopen("filename.txt", "mode");`
- Modes: `"r"` (read), `"w"` (write), `"a"` (append).
- Closing a file: `fclose(fptr);` prevents resource leaks and data corruption.

File Writing and Reading Example

```c
#include <stdio.h>
int main() {
    FILE *fptr;
    // Writing to file
    fptr = fopen("test.txt", "w");
    fprintf(fptr, "Hello AKTU Students\n");
    fclose(fptr);

    // Reading from file
    char buffer[100];
    fptr = fopen("test.txt", "r");
    fgets(buffer, 100, fptr);
    printf("Read from file: %s", buffer);
    fclose(fptr);
    return 0;
}
```



- Always check if `fptr == NULL` after `fopen()` to handle file opening errors.
- `fprintf` and `fscanf` work like `printf` and `scanf` but target streams.

Exam corner

- **Q1** Write a C program to find the maximum and minimum elements in a 1D array.
- **Q2** Explain the difference between `gets()` / `puts()` and standard `scanf()` / `printf()` for strings, mentioning safety issues.
- **Q3** What is a pointer? Explain the concepts of address-of (`&`) and dereference (`*`) operators with an example.
- **Q4** Differentiate between arrays and structures with suitable examples for each.
- **Q5** Write a C program to create a structure representing a Book (title, author, price) and display its details.
- **Q6** Explain the various file opening modes (`r` , `w` , `a`) in C file handling with syntax.
- **Q7** Write a complete C program to write a user-defined string into a file and read it back to display on the console.

Unit 3 done. You are exam-ready.

Revise the Exam corner once more, then open Unit 4 on theycotes.com. Every AKTU subject, unit by unit, free. Notes by Chirag Jain, Ycotes.

