Useful to have a variable that can store series of related values using an array.
For example: Suppose a program is required that we calculate the average age of six students. The ages of the student could be stored in six integer variables, "c".
However, a better solution will be to declare a six-element array
int age[6] = [18, 20, 19, 21, 22, 20]

CHARACTERISTICS OF AN ARRAY

1. They have a fixed size (in static arrays).
2. Elements share the same data type.
3. Indexed starting from zero in most languages.
4. Fast access due to contiguous memory.

OPERATIONS ON ARRAY

Array operations refer to the fundamental actions performed on array data structure to access, modify or manipulate stored elements.
Key operations include:-

1. TRAVERSAL - it is the process of visiting

ARRAY

An array is a collection of elements of the same data type stored in contiguous locations and accessed using an index. In Computer Science, an array is a data structure consisting of a group of element that are accessed by indexing.

Each data item of an array is known as an "element", and elements are referenced by a common name known as the "array name".

In Java, as most programming languages an array is a structure that holds multiple values of the same type. A Java array is also an object. Then an array can contain data of primitive data type.

As it is an object, an array must be declared and instantiated (represent a concept by an instance of a specific class) to create an object of a specific class)

Example:- int number[5] = [2, 5, 7, 11, 15]
0 1 2 3 4

Variables normally only stores a single value but, in some situations, it is

each array element exactly once in a systematic manner, from the first index to the last index.

2. INSERTION:- is the operation of adding new element at a specific position. Because array uses contiguous memory, insertion requires shifting elements to create space.

3. DELETION:- it removes an existing element from the array and shifts the remaining elements to maintain contiguity.

4. SEARCHING:- refers to locating the index of a given element within the array.

Types of Search

⇒ Linear Search:- Checking each element sequentially.

⇒ Binary Search (Sorted Array):- It uses divide and conquer to locate the key in $O(\log_2 n)$ time.

5. UPDATING:- is modifying the value stored at a specific array index. Updating does not require shifting elements.

RECORD

A record is a non-primitive composite data that allows programmers group different

related data under a single name. Unlike arrays with ~~heterogeneous~~ multiple values of the same data type, record stores values of different data types. Each value in a record is stored in a field and each field has its own name and data type.

Records are used to model real-world entities such as student, employees, books, vehicles, or an object to contain multiple attributes. They improve the organization, readability and structure of data in a program.

CHARACTERISTICS OF A ~~PROGRAM~~ RECORD

1. HETEROGENOUS DATA:- a record can contain fields of different types (eg integers, string, real).
2. FIXED STRUCTURE:- the numbers of fields and their data types are defined at the time of record creation.
3. NAME FIELD:- each field in a record has a name, which is used to access or modify.
4. GROUPED UNDER ONE IDENTIFIER: all fields belong to one unit in a record, making it easier to

handle the data as one structure.

6. STATIC MEMORY ALLOCATION (in most cases): memory of each field is set aside based on each data type when the record is created.

IMPORTANCE OF RECORDS

1. It allows grouping of related data into a single logical unit.
2. They reduce errors associated with parallel arrays
3. They make programs easier to maintain, understand and debug.
4. They are often used as the basis for more complex structures such as; arrays of records, linked list and file records.
5. They are useful in applications such as data base systems, where each record represents one row of a table.

COMPONENTS OF A RECORD

1. RECORD NAME:- It is the identifier representing the whole structure.
2. FIELDS/ATTRIBUTES:- They represent individual

items within the record.

8. FIELD DATA TYPE:- this defines the types of data each field stores.

Example:-

Record Name:- Student

Field/Attribute:- Name, Age, Matric NO, Dept, CGPA

Data types:- String, integer, String, integer, String, float.

OPERATIONS ON RECORDS

1. DECLARATION:- Creating the record structure.
2. INSTANTIATION:- Creating a variable of the record type.
3. ASSIGNMENT:- Storing values in a field.
4. ACCESS:- Retrieving field values.
5. UPDATE:- Changing the value of the field.
6. TRAVERSAL:- Iterating through an array of records
7. SEARCH:- Finding a record with a particular attribute.
8. COMPARISON:- Used in sorting operation.

ADVANTAGES OF RECORDS

1. Support for heterogenous data.
2. Improves data organization.
3. Better code readability.
4. Suitable for modelling real-world entities.
5. Useful for file handling and database design.

LIMITATION OF RECORDS

1. Size is fixed once defined.
2. Fields cannot be added or removed at runtime in most languages.
3. Not Suitable for storing large variable sized data without additional structure.

KEY CONCEPTS

DATA TYPE: in computer programming, a data type simply refers to a defined type of data, that is a set of possible values and basic operations of these values. Data types are important in programs because they classify data so that the translators (compiler or interpreter) can reserve appropriate memory storage to hold all possible values. E.g integers, real numbers, characters, strings, and boolean values all have very different representations in memory.

A data consists of:

1. A domain (set of values).
2. A set of operation that may be applied to the values.

CLASSIFICATION OF DATA TYPES

Some data items may be used singly while others may be combined together and arranged to form other items.

The former is classified as simple data

data types. Whereas the later is referred to as data structure.

Data structures are specialized formats for organizing, storing, and managing data in complex systems. The choice of data structure can significantly impact the performance and efficiency of algorithms.

TYPES OF DATA STRUCTURE

1. PRIMITIVE DATA STRUCTURE:- These are basic building blocks used to create more complex data structure. They represent the simplest form of data and cannot be broken into smaller data types. The simple data types are classified as follows:-

⇒ Integer:- Stores whole numbers e.g 5, 2.
⇒ Float:- Stores decimal numbers e.g 3.142.
⇒ Character (char):- Stores a single symbol letter.
⇒ Boolean (Bool):- Stores logical values. eg True or False, Yes or No.

2. NON-PRIMITIVE DATA STRUCTURE:- They are more complex structures that can store multiple values.

CLASSIFICATION OF NON-PRIMITIVE DATA STRUCTURE

It is broadly divided into two(2):-

(a) LINEAR DATA STRUCTURE:- They are data structures where individual data elements are stored and accessed linearly in the computer memory(List, Stack, Queues, and Array).

(b) NON-LINEAR DATA STRUCTURE:- Is a data structure in which the data elements are not stored linearly in the computer memory, but data items can be accessed using some techniques or rules e.g(Trees & Graphs)

ABSTRACT DATA TYPES (ADT):- an abstract data type is a mathematical model for data types, defining the data and the operation that can be performed on that data type without specifying the underlined implementation. It emphasizes what operations are allowed and how they behave, rather than how they are implemented in a computer program.

CORE CONCEPTS OF ADTs

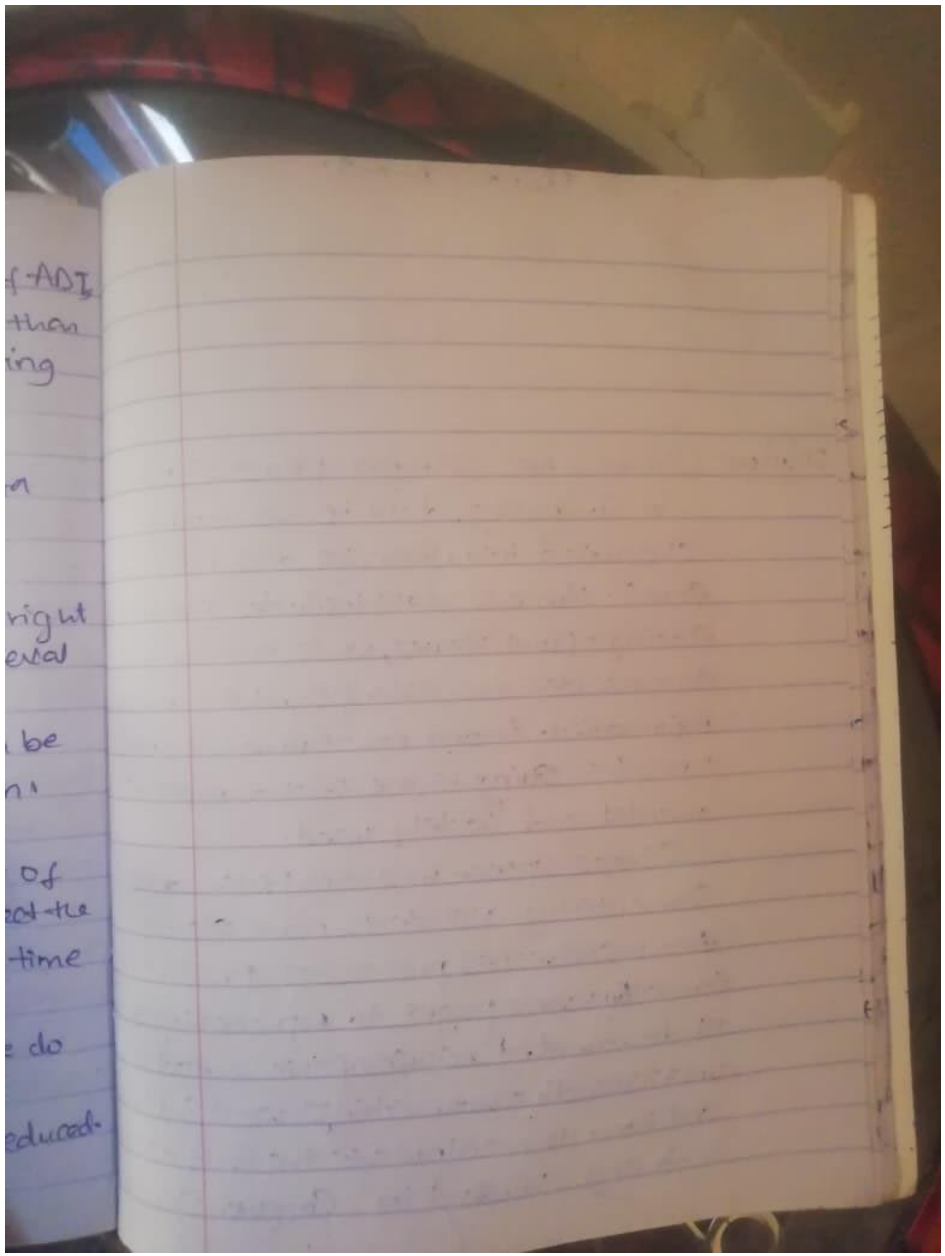
1. ABSTRACTION:- ADTs provide a high level perspective on data and operation, concealing the underlined implementation details.
2. OPERATIONS:- ADTs specify permisible operations such as insertion, deletion, traversal and searching without defining how these operations are executed.
3. ENCAPSULATION:- The internal representation of data is protected, users cannot access it directly, ensuring controlled interaction through defined operations.
4. IMPLEMENTATION FLEXIBILITY:- a single data item can be implemented using different data structures, allowing adaptability without changing how it is used.
5. PURPOSE:- ADTs facilitate modular design, improves code reusability, and enhances maintainability of software systems.

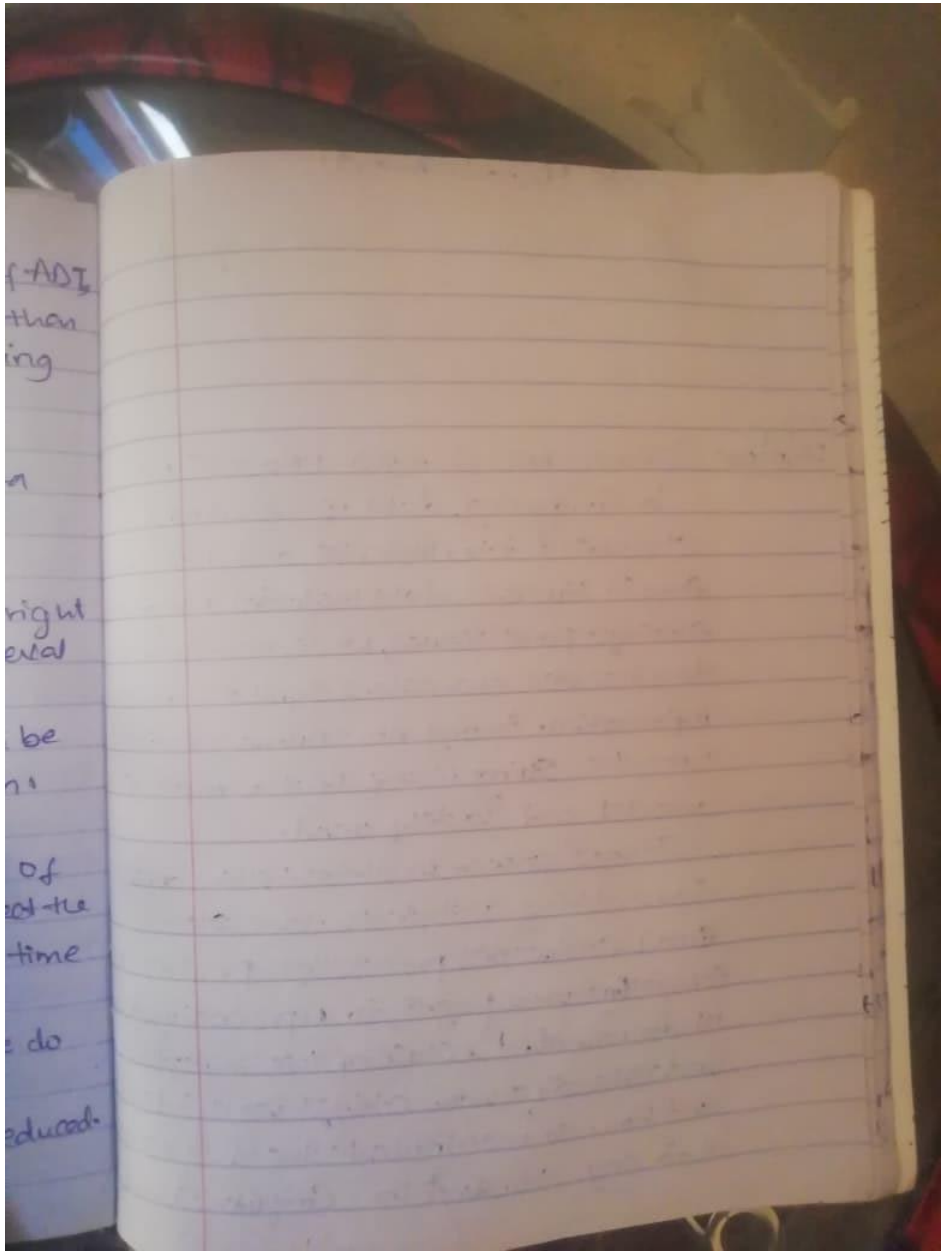
6. SEPARATION OF CONCERNS: Users of ADT, focus on what the ADT does rather than how it accomplishes it. Simplifying program development and usage.

ROLE OF DATA STRUCTURE IN PROBLEM SOLVING

Data Structures are Crucial for:-

1. Efficient data manipulation:- the right data structure enhances data retrieval and storage efficiency.
2. Flexibility:- Different Structures can be employed depending on the problem's requirement.
3. Algorithm efficiency:- the choice of data structure can significantly affect the performance of algorithms, impacting time and space complexities.
4. Software Program routines:- that make do with the data are made simpler.
5. Time and Storage Spaces are also reduced.





o any Student in IFT,

STRINGS DATA TYPES

The string is formally defined as an ordered sequence of characters. Characters may include:-

1. Alphabets (A-Z, a-z).
2. Digits (0-9).
3. Punctuation Symbols (! ? ,).
4. Special Symbols (#, $, %).
5. Whitespace characters (space, tab).

Strings are typically enclosed in quotation marks.

Depending on the programming language, either single (' ') or double (" ") quotes may be used.

For example: "University"

"2025/2026 Session"

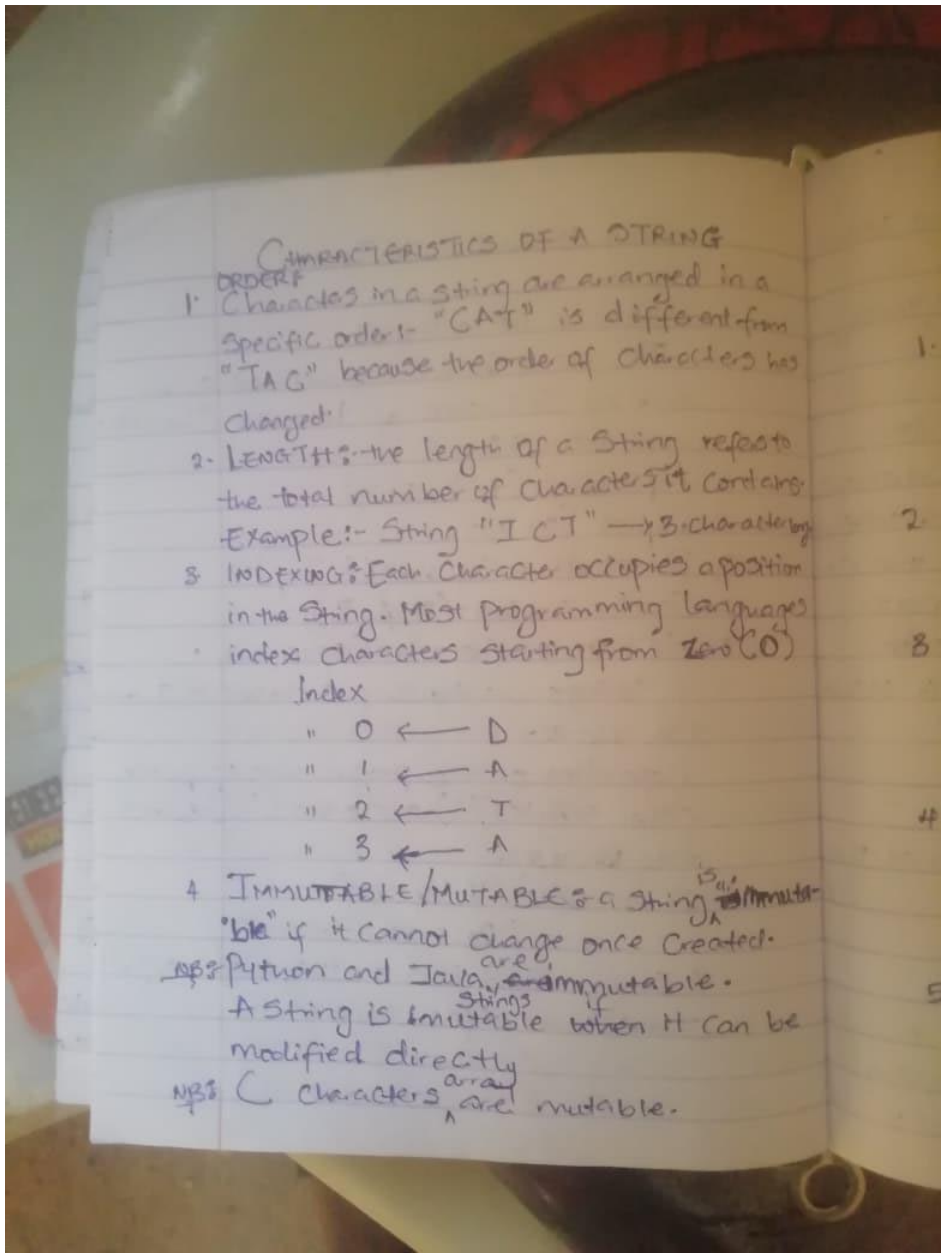
A String is treated as one Unit, even though it contains multiple characters.

Static & Dynamic Arrays.

24/11/2025 STRINGS AND STRING PROCESSING

In Computing, data is commonly classified into numeric and non-numeric forms. Numeric data includes integers and floating-point values, while non-numeric data includes characters, symbols, and textual information. Among non-numerical data types, the **String** is one of the most fundamental and widely used.

Strings appear in almost every application from entering a students name, storing email addresses, processing passwords, formatting web pages, to representing text in document. Because of this importance, understanding how strings work internally and how to manipulate them is essential for any student in Computer Science

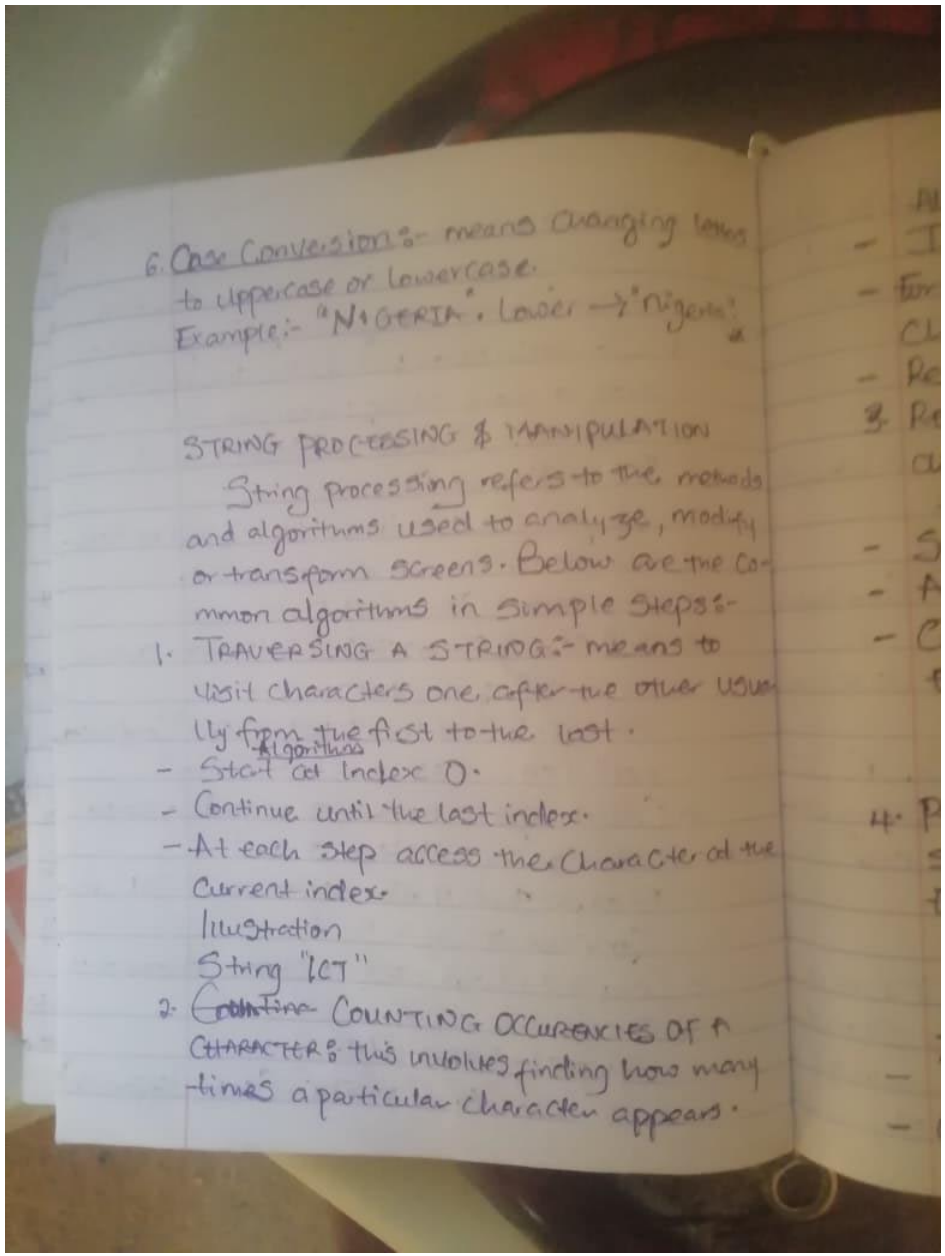


CHARACTERISTICS OF A STRING

1. ORDER: Characters in a string are arranged in a specific order:- "CAT" is different from "TAC" because the order of characters has changed.
2. LENGTH:- The length of a string refers to the total number of characters it contains. Example:- String "ICT" → 3 characters long
3. INDEXING: Each character occupies a position in the string. Most programming languages index characters starting from zero (0)

Index	
0	← D
1	← A
2	← T
3	← A

4. IMMUTABLE/MUTABLE: A string is "ble" if it cannot change once created.
NB: Python and Java strings are immutable.
A string is mutable when it can be modified directly
NB: C characters array are mutable.



6. Case Conversion:- means changing letters to uppercase or lowercase.
Example:- "NIGERIA". Lower → "nigeria"

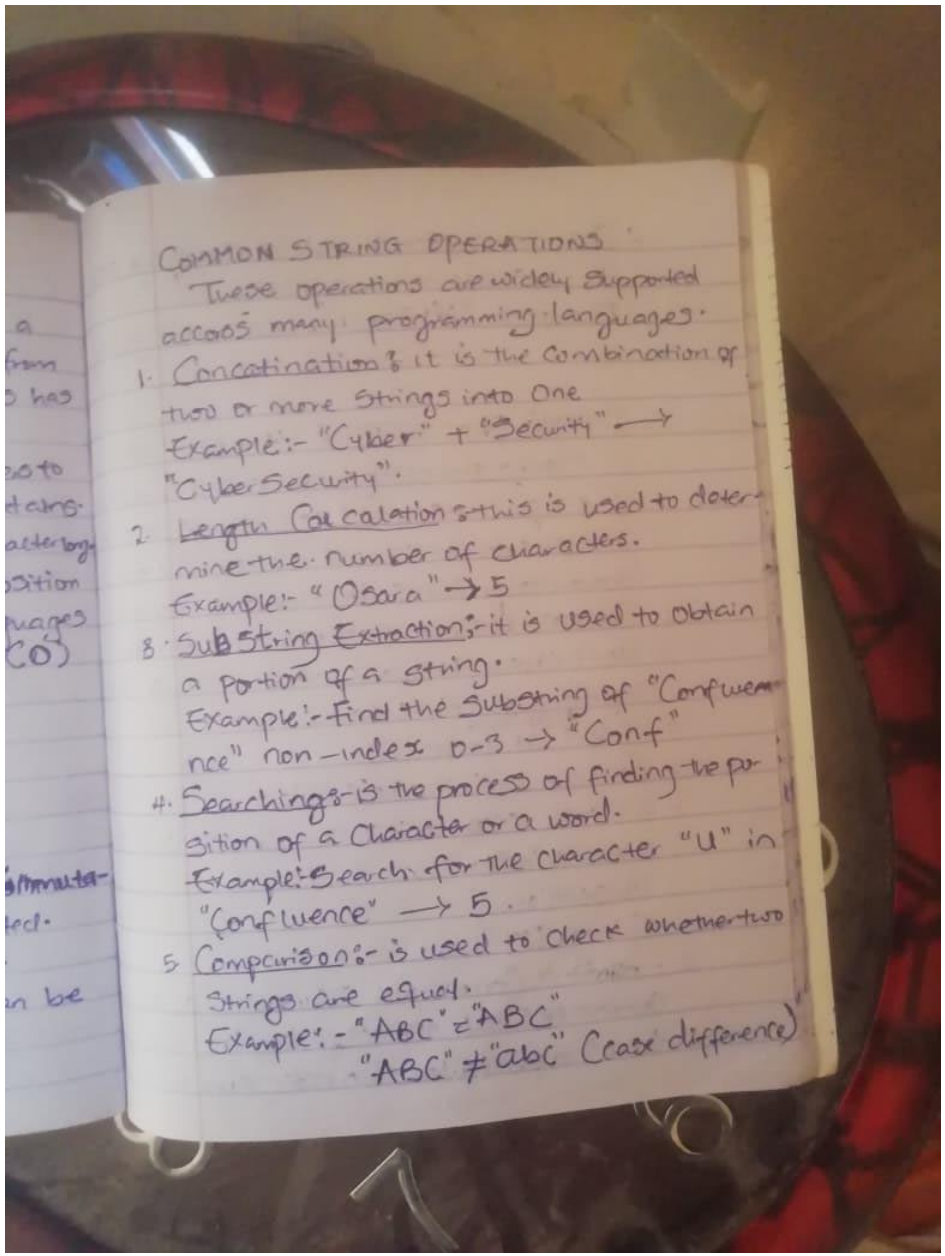
STRING PROCESSING & MANIPULATION

String processing refers to the methods and algorithms used to analyze, modify or transform screens. Below are the common algorithms in simple steps:-

1. TRAVERSING A STRING:- means to visit characters one after the other usually from the first to the last.
- Start at index 0.
- Continue until the last index.
- At each step access the character at the current index.

Illustration
String "ICT"

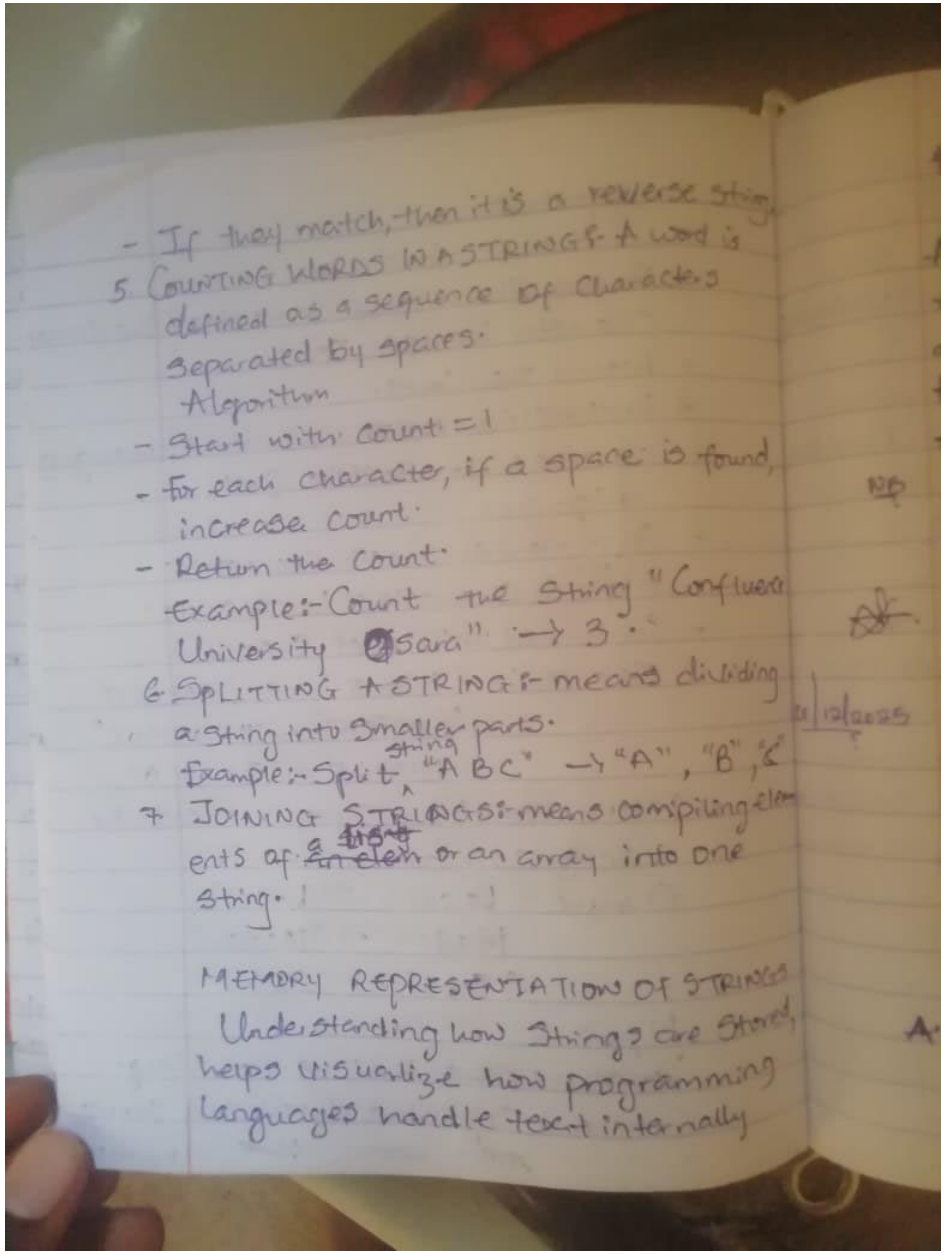
2. ~~Counting~~ COUNTING OCCURENCIES OF A CHARACTER: this involves finding how many times a particular character appears.



COMMON STRING OPERATIONS

These operations are widely supported across many programming languages.

1. Concatination :- it is the combination of two or more strings into one
   Example:- "Cyber" + "Security" → "Cyber Security".
2. Length Calculation :- this is used to determine the number of characters.
   Example:- "Osara" → 5
3. Sub String Extraction :- it is used to obtain a portion of a string.
   Example:- Find the Substring of "Confluence" non-index 0-3 → "Conf"
4. Searching :- is the process of finding the position of a character or a word.
   Example:- Search for the character "u" in "Confluence" → 5.
5. Comparison :- is used to check whether two strings are equal.
   Example:- "ABC" = "ABC"
   "ABC" ≠ "abc" (case difference)



- If they match, then it is a reverse string.

5. COUNTING WORDS IN A STRING:- A word is defined as a sequence of characters separated by spaces.

Algorithm

- Start with count = 1
- For each character, if a space is found, increase count.
- Return the count.

Example:- Count the string "Confluence University Ogbaru" → 3.

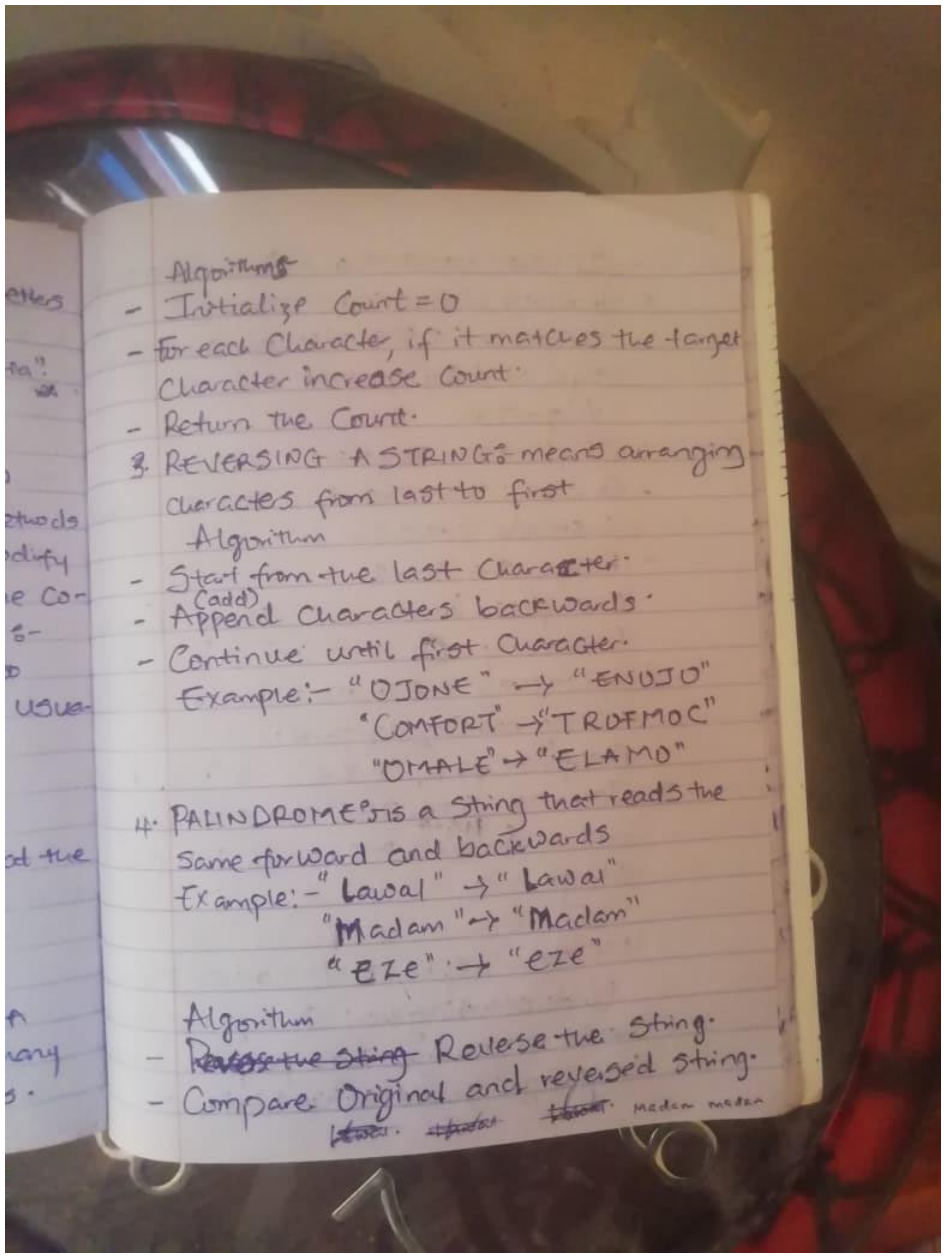
6. SPLITTING A STRING:- means dividing a string into smaller parts.

Example:- Split "A B C" → "A", "B", "C"

7. JOINING STRINGS:- means compiling elements of ~~an elem~~ or an array into one string.

MEMORY REPRESENTATION OF STRINGS

Understanding how Strings are Stored, helps visualize how programming languages handle text internally



Algorithm
- Initialize Count = 0
- For each Character, if it matches the target character increase Count.
- Return the Count.

3. REVERSING A STRING means arranging characters from last to first

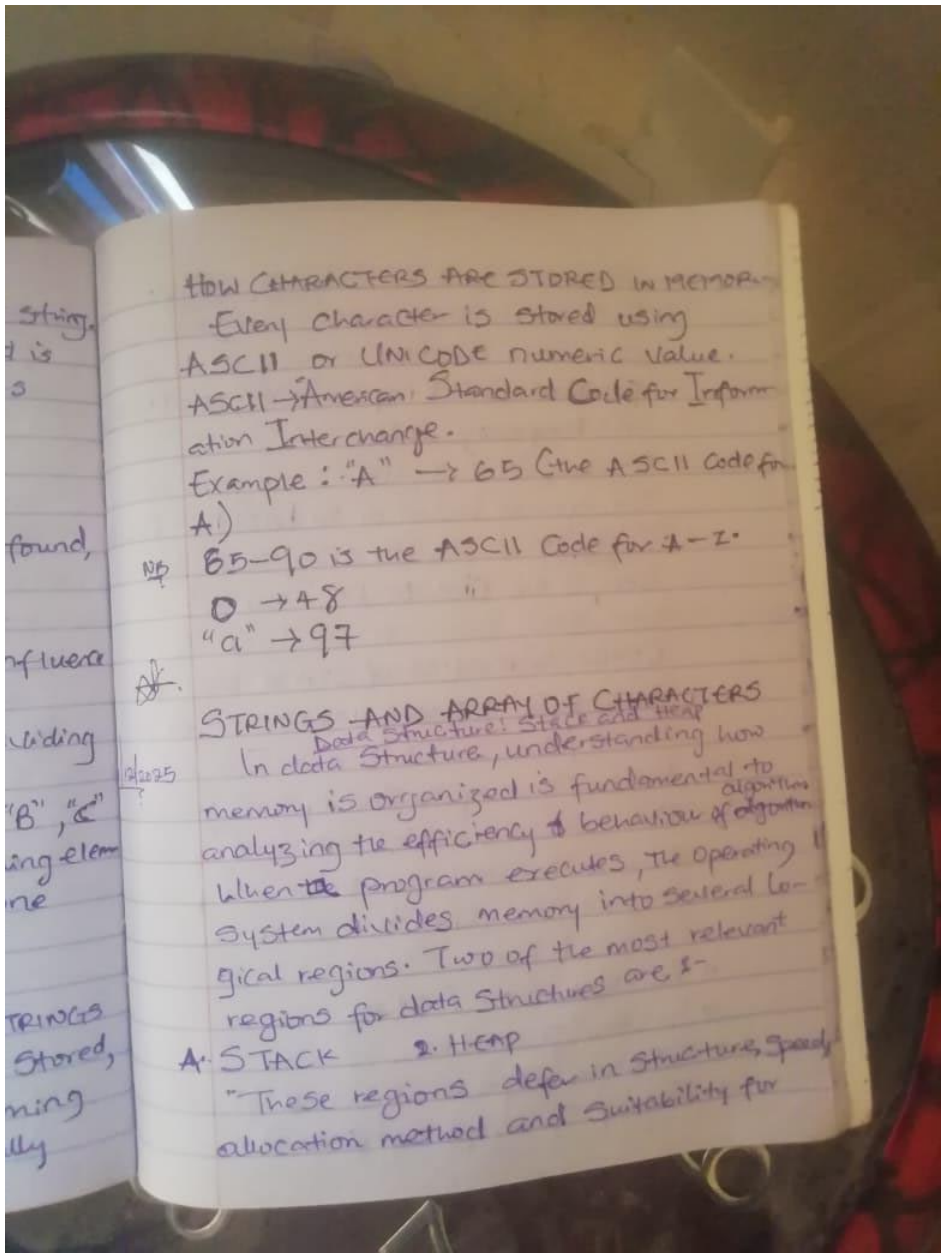
Algorithm
- Start from the last Character.
- Append (add) Characters backwards.
- Continue until first Character.

Example:- "OJONE" → "ENUJO"
"COMFORT" → "TROFMOC"
"OMALE" → "ELAMO"

4. PALINDROME is a String that reads the same forward and backwards

Example:- "Lawal" → "Lawal"
"Madam" → "Madam"
"eze" → "eze"

Algorithm
- ~~Reverse the String~~ Reverse the String.
- Compare Original and reversed string.



HOW CHARACTERS ARE STORED IN MEMORY

Every character is stored using ASCII or UNICODE numeric value.

ASCII → American Standard Code for Information Interchange.

Example : "A" → 65 (the ASCII code for A)

65–90 is the ASCII Code for A–Z.

0 → 48

"a" → 97

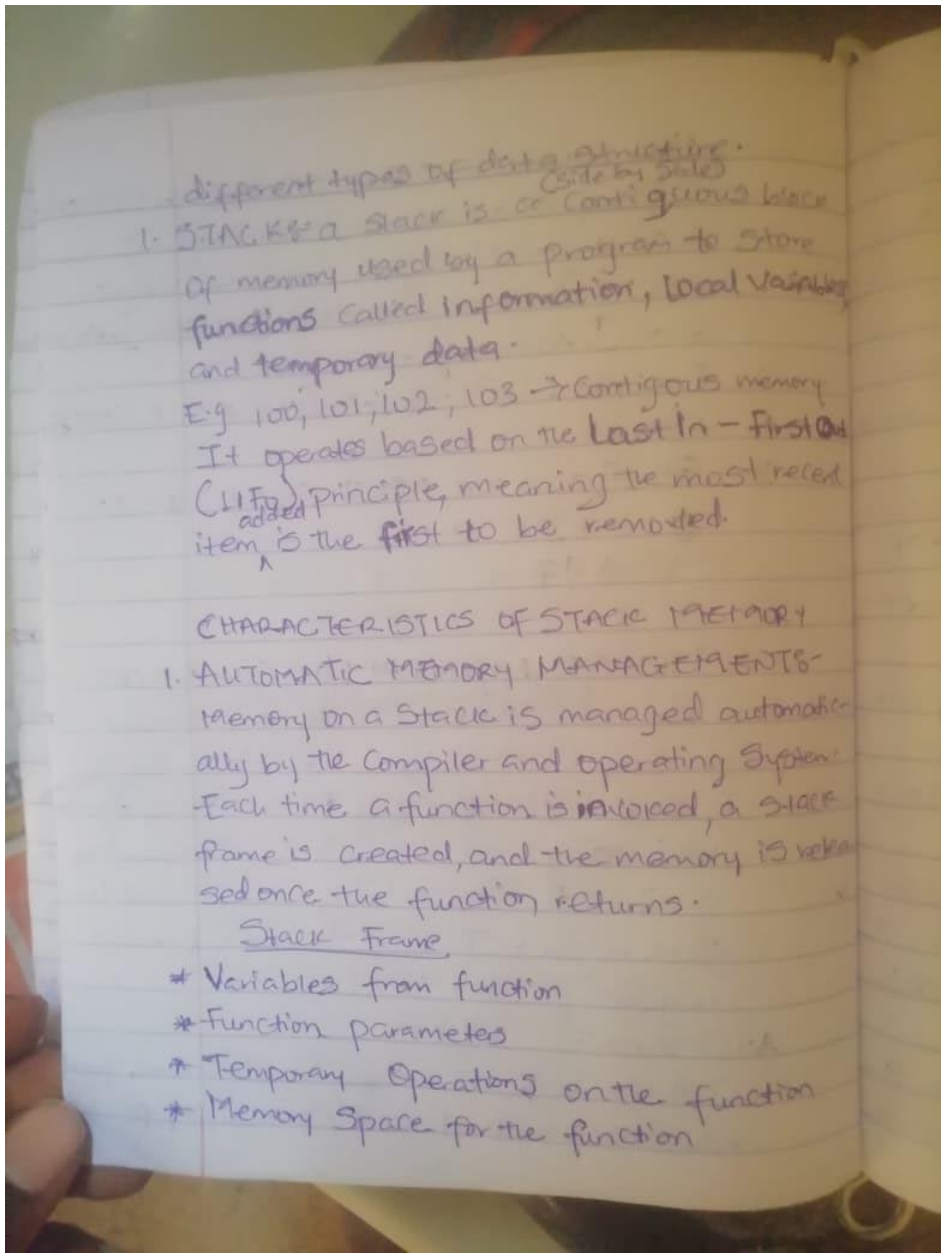
STRINGS AND ARRAY OF CHARACTERS

Data Structure: Stack and Heap

In data structure, understanding how memory is organized is fundamental to analyzing the efficiency & behaviour of algorithm. When the program executes, the operating system divides memory into several logical regions. Two of the most relevant regions for data structures are :-

A. STACK 2. HEAP

"These regions defer in structure, speed, allocation method and suitability for



different types of data structure.

1. STACK: a stack is a contiguous block of memory used by a program to store functions called information, local variables and temporary data.

E.g 100, 101, 102, 103 → Contigous memory

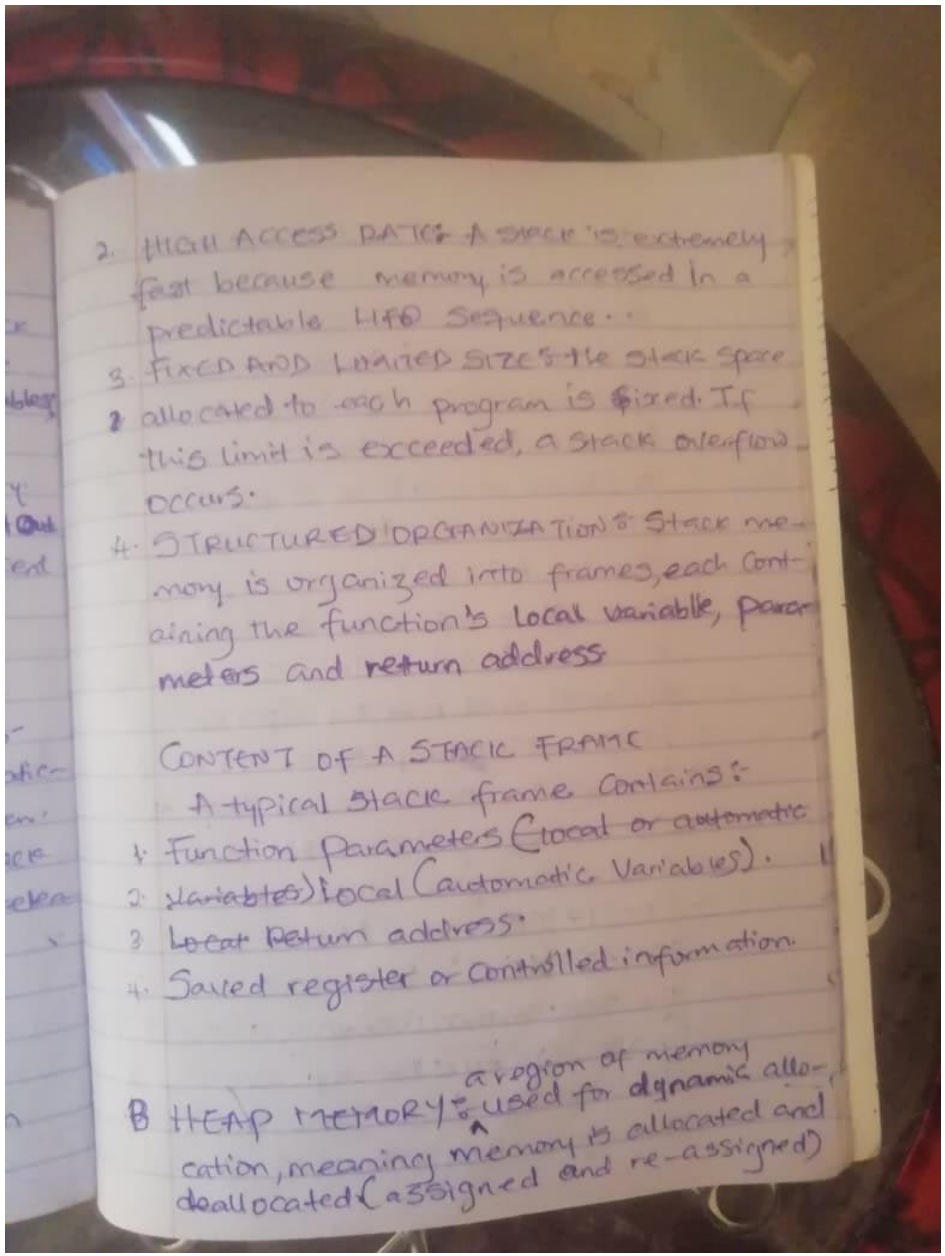
It operates based on the Last In - First Out (LIFO) principle, meaning the most recent item added is the first to be removed.

CHARACTERISTICS OF STACK MEMORY

1. AUTOMATIC MEMORY MANAGEMENT:
Memory on a stack is managed automatically by the compiler and operating system. Each time a function is invoked, a stack frame is created, and the memory is released once the function returns.

Stack Frame

* Variables from function
* Function parameters
* Temporary operations on the function
* Memory space for the function



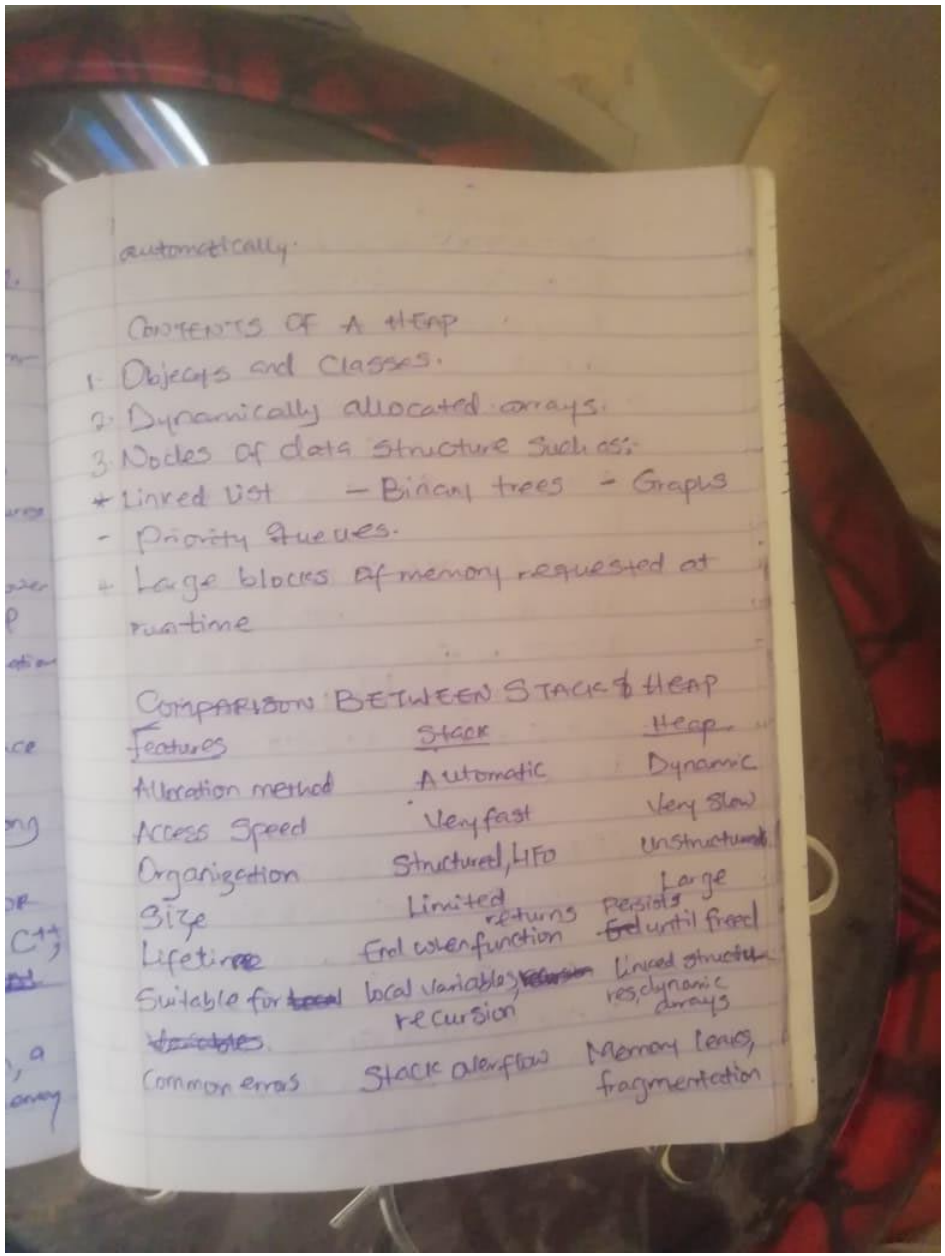
2. HIGH ACCESS RATE: A stack is extremely fast because memory is accessed in a predictable LIFO sequence.
3. FIXED AND LIMITED SIZE: The stack space allocated to each program is fixed. If this limit is exceeded, a stack overflow occurs.
4. STRUCTURED ORGANIZATION: Stack memory is organized into frames, each containing the function's local variable, parameters and return address

CONTENT OF A STACK FRAME

A typical stack frame contains:-

1. Function parameters (local or automatic variables).
2. ~~Variables)~~ Local (automatic Variables).
3. ~~Local~~ Return address.
4. Saved register or controlled information.

B HEAP MEMORY: a region of memory used for dynamic allocation, meaning memory is allocated and deallocated (assigned and re-assigned)



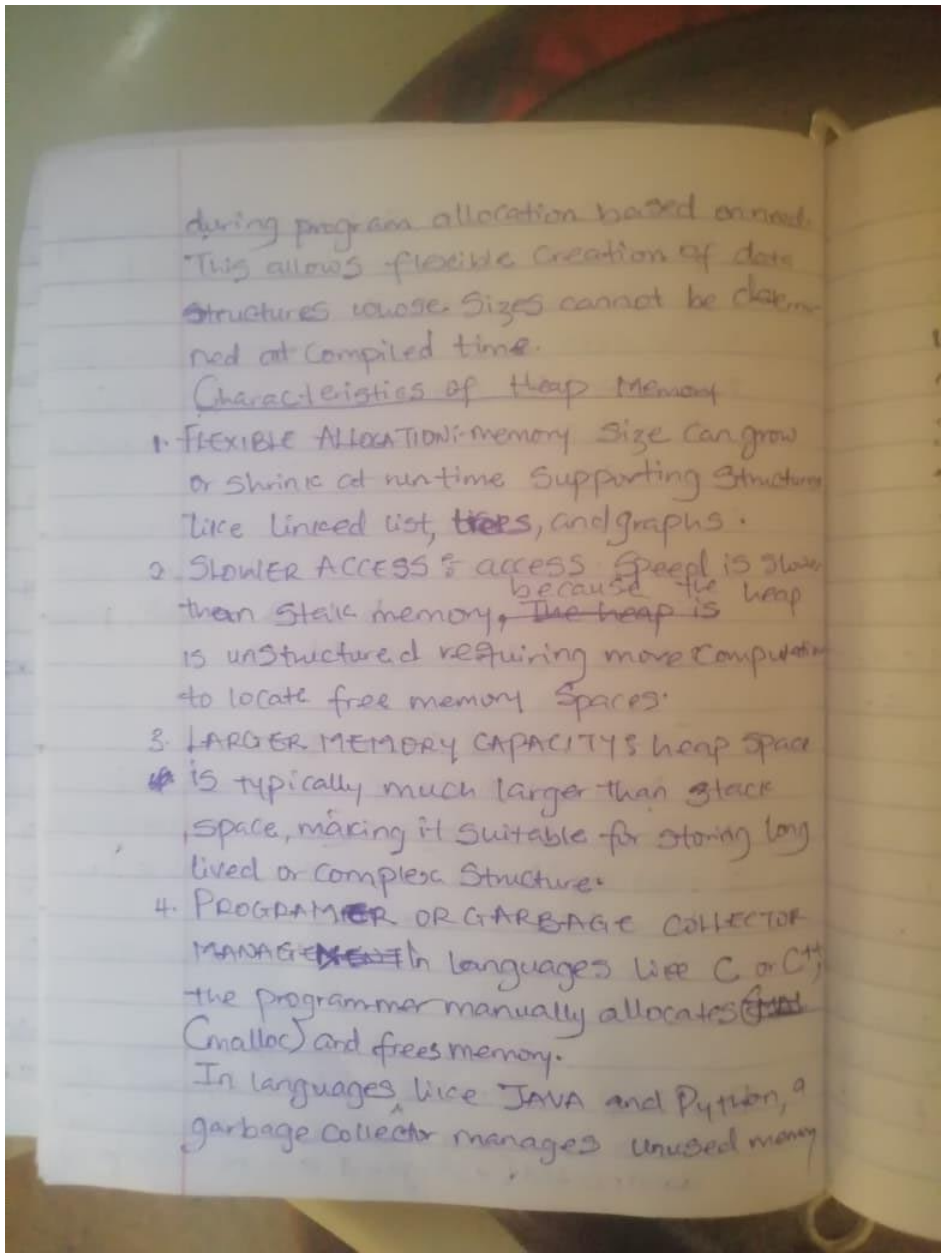
automatically.

CONTENTS OF A HEAP

1. Objects and classes.
2. Dynamically allocated arrays.
3. Nodes of data structure such as:
   - Linked list – Binary trees – Graphs
   - Priority queues.
4. Large blocks of memory requested at runtime

COMPARISON BETWEEN STACK & HEAP

Features	Stack	Heap
Allocation method	Automatic	Dynamic
Access Speed	Very fast	Very slow
Organization	Structured, LIFO	Unstructured
Size	Limited	Large
Lifetime	Ends when function returns	Persists until freed
Suitable for local variables	local variables, recursion	Linked structures, dynamic arrays
Common errors	Stack overflow	Memory leaks, fragmentation

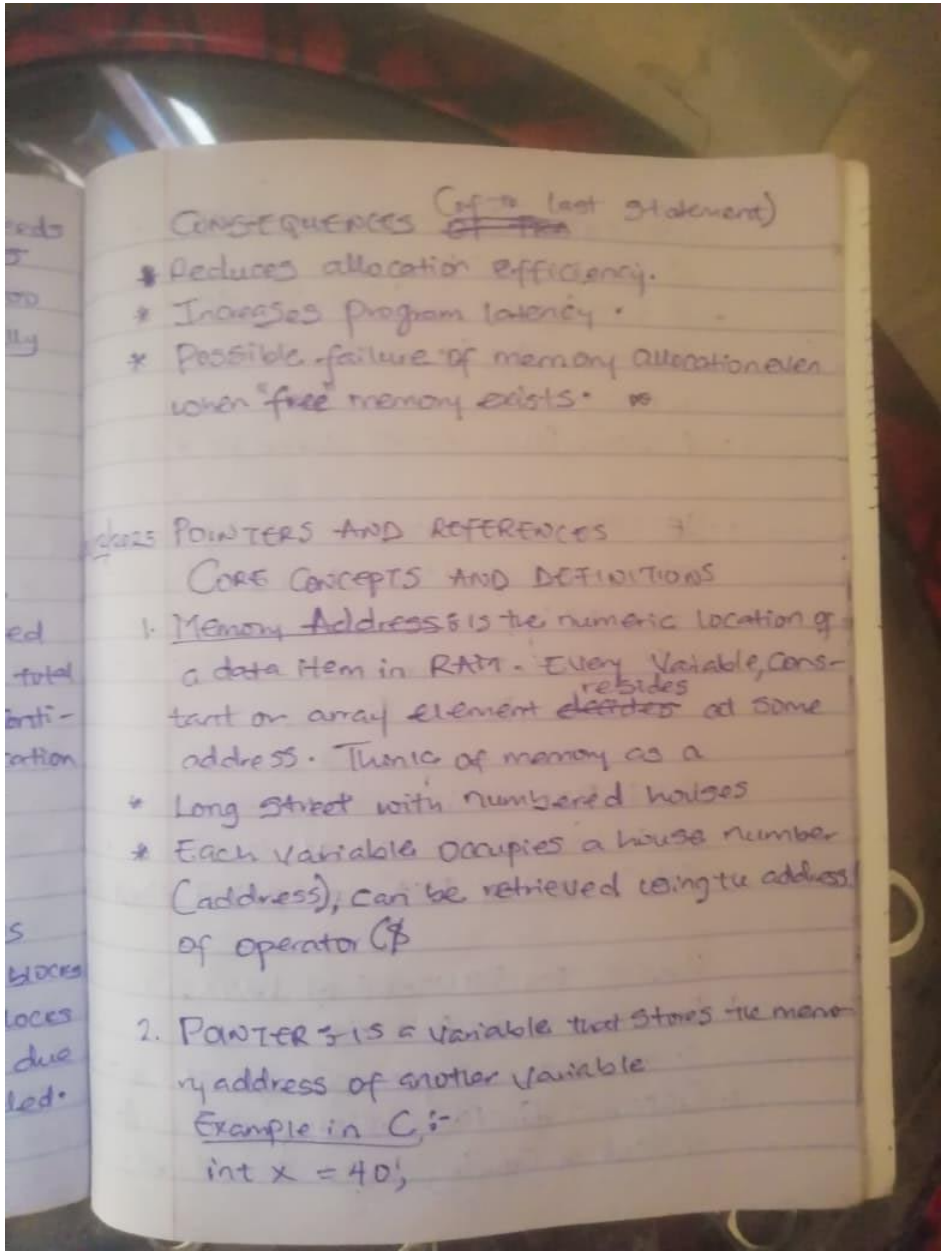


during program allocation based on need. This allows flexible creation of data structures whose sizes cannot be determined at compiled time.

Characteristics of Heap Memory

1. FLEXIBLE ALLOCATION: memory size can grow or shrink at runtime supporting structures like linked list, trees, and graphs.
2. SLOWER ACCESS: access speed is slower than stack memory, because the heap is unstructured requiring more computation to locate free memory spaces.
3. LARGER MEMORY CAPACITY: heap space is typically much larger than stack space, making it suitable for storing long lived or complex structure.
4. PROGRAMMER OR GARBAGE COLLECTOR MANAGED: In languages like C or C++, the programmer manually allocates (malloc) and frees memory.

In languages like JAVA and Python, a garbage collector manages unused memory



CONSEQUENCES (of the last statement)

* Reduces allocation efficiency.
* Increases program latency.
* Possible failure of memory allocation even when "free" memory exists.

POINTERS AND REFERENCES

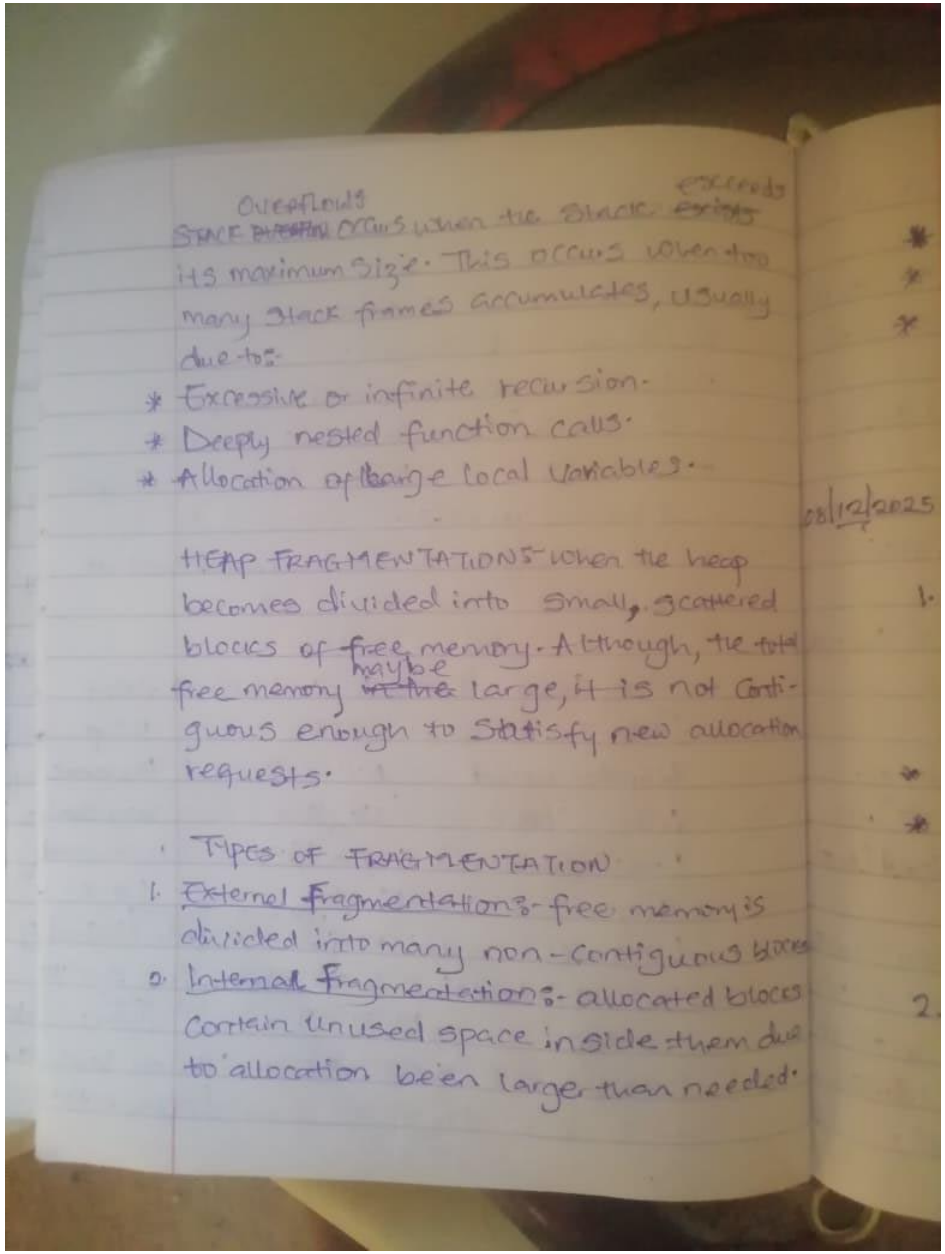
CORE CONCEPTS AND DEFINITIONS

1. Memory Address is the numeric location of a data item in RAM. Every variable, constant or array element resides at some address. Think of memory as a
   * Long street with numbered houses
   * Each variable occupies a house number (address), can be retrieved using the address of operator (&)

2. POINTER is a variable that stores the memory address of another variable

   Example in C:-

   int x = 40;



STACK OVERFLOWS occurs when the stack exceeds its maximum size. This occurs when too many stack frames accumulates, usually due to:

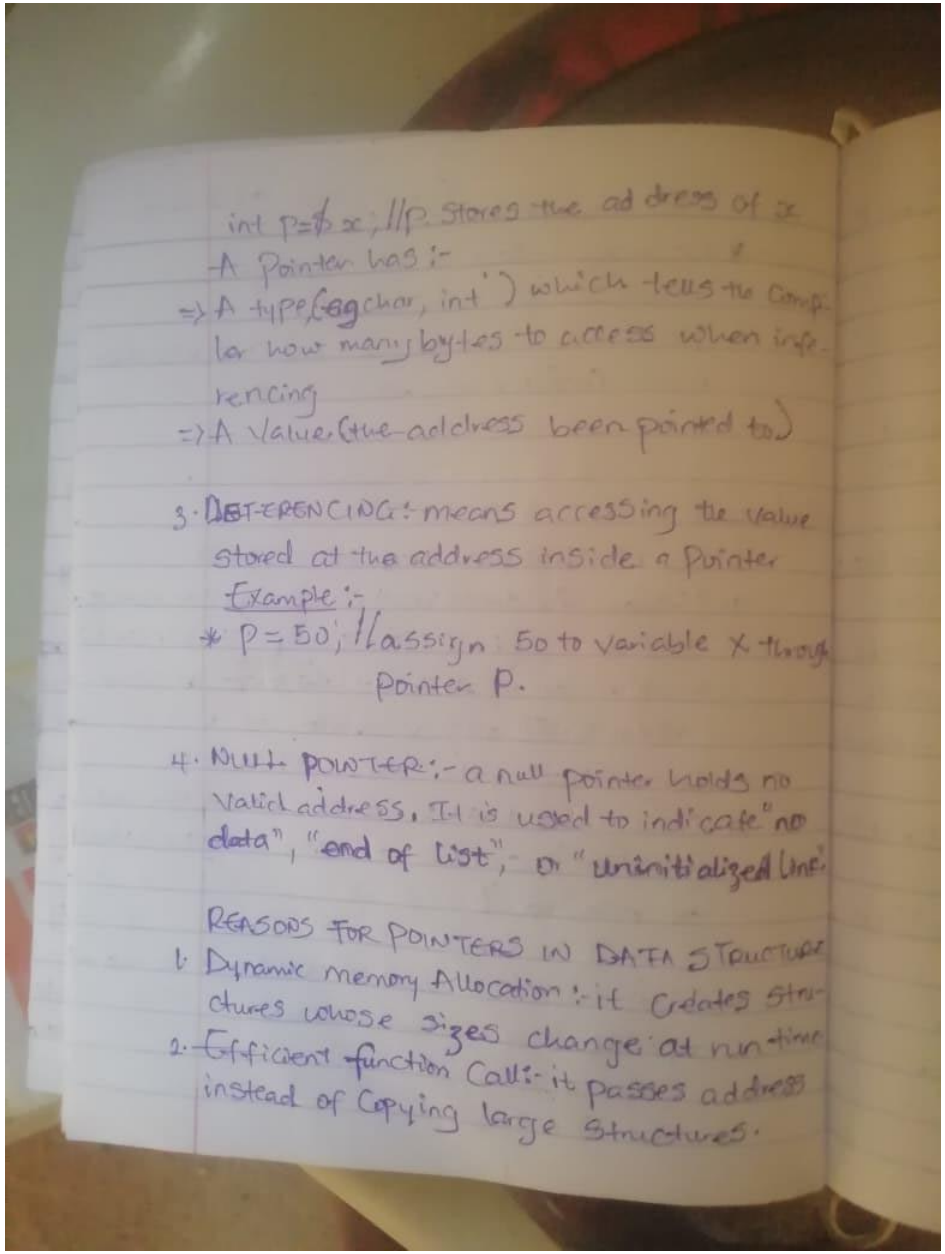
* Excessive or infinite recursion.
* Deeply nested function calls.
* Allocation of large local variables.

08/12/2025

HEAP FRAGMENTATION: When the heap becomes divided into small, scattered blocks of free memory. Although, the total free memory maybe large, it is not contiguous enough to satisfy new allocation requests.

TYPES OF FRAGMENTATION

1. External Fragmentation:- free memory is divided into many non-contiguous blocks
2. Internal Fragmentation:- allocated blocks contain unused space inside them due to allocation being larger than needed.



int p=&x; //p stores the address of x

-A Pointer has :-

=> A type (eg char, int *) which tells the compiler how many bytes to access when info. rencing

=> A value (the address been pointed to)

3. DEREFERENCING :- means accessing the value stored at the address inside a pointer

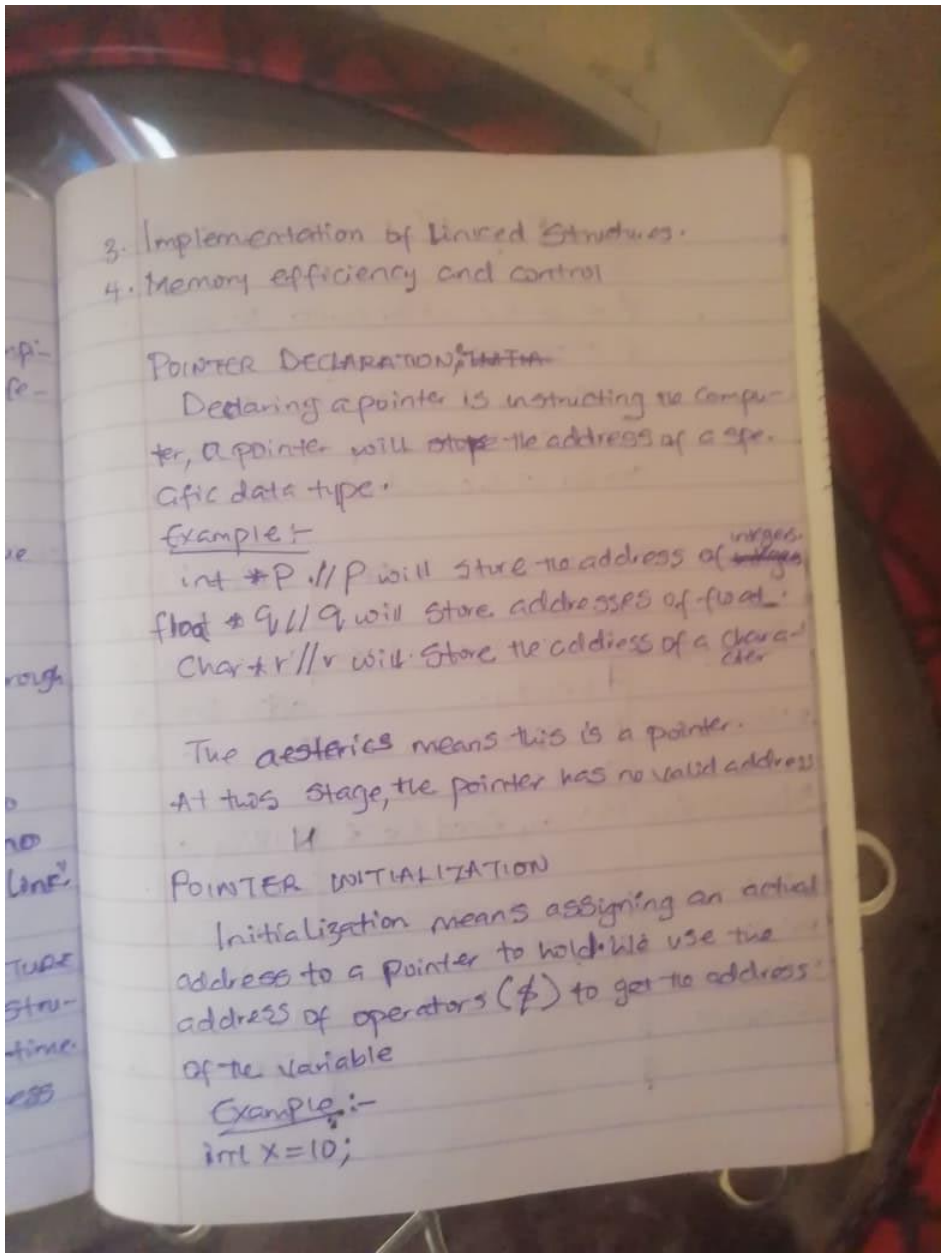
Example :-

* p = 50; // assign 50 to variable x through pointer p.

4. NULL POINTER :- a null pointer holds no valid address. It is used to indicate "no data", "end of list", or "uninitialized ptr".

REASONS FOR POINTERS IN DATA STRUCTURE

1. Dynamic memory Allocation :- it creates structures whose sizes change at runtime
2. Efficient function Call :- it passes address instead of copying large structures.



3. Implementation of Linked Structures.
4. Memory efficiency and control

POINTER DECLARATION

Declaring a pointer is instructing the computer, a pointer will store the address of a specific data type.

Example:-

int *P //P will store the address of integers.
float *q // q will store addresses of float.
Char *r // r will store the address of a character

The asterics means this is a pointer.
At this stage, the pointer has no valid address

POINTER INITIALIZATION

Initialization means assigning an actual address to a pointer to hold. We use the address of operators (&) to get the address of the variable

Example:-
int X=10;

int *P = &x //P now stores the address of x

x stores the value of 10

&x is the address of the operator

P stores &x

Using a Pointer means:

It means accessing or changing a value stored at the address the pointer holds.

COMMON POINTER ERRORS AND HOW TO AVOID THEM

Error	Description	Prevention
1. Dangling Pointer	Pointer refering to freed memory.	Set pointer to NULL after been freed.
2. Memory leak	Allocated memory not freed.	Ensure Malloc is free
3. Double free	Memory freed twice	Nullify pointer after been freed.
4. Uninitialized Pointer	Pointer contains garbage address	Always initialize pointers
5. Null Pointer deference	Using * Null	Check before deferencing.