Introduction

Data structure:

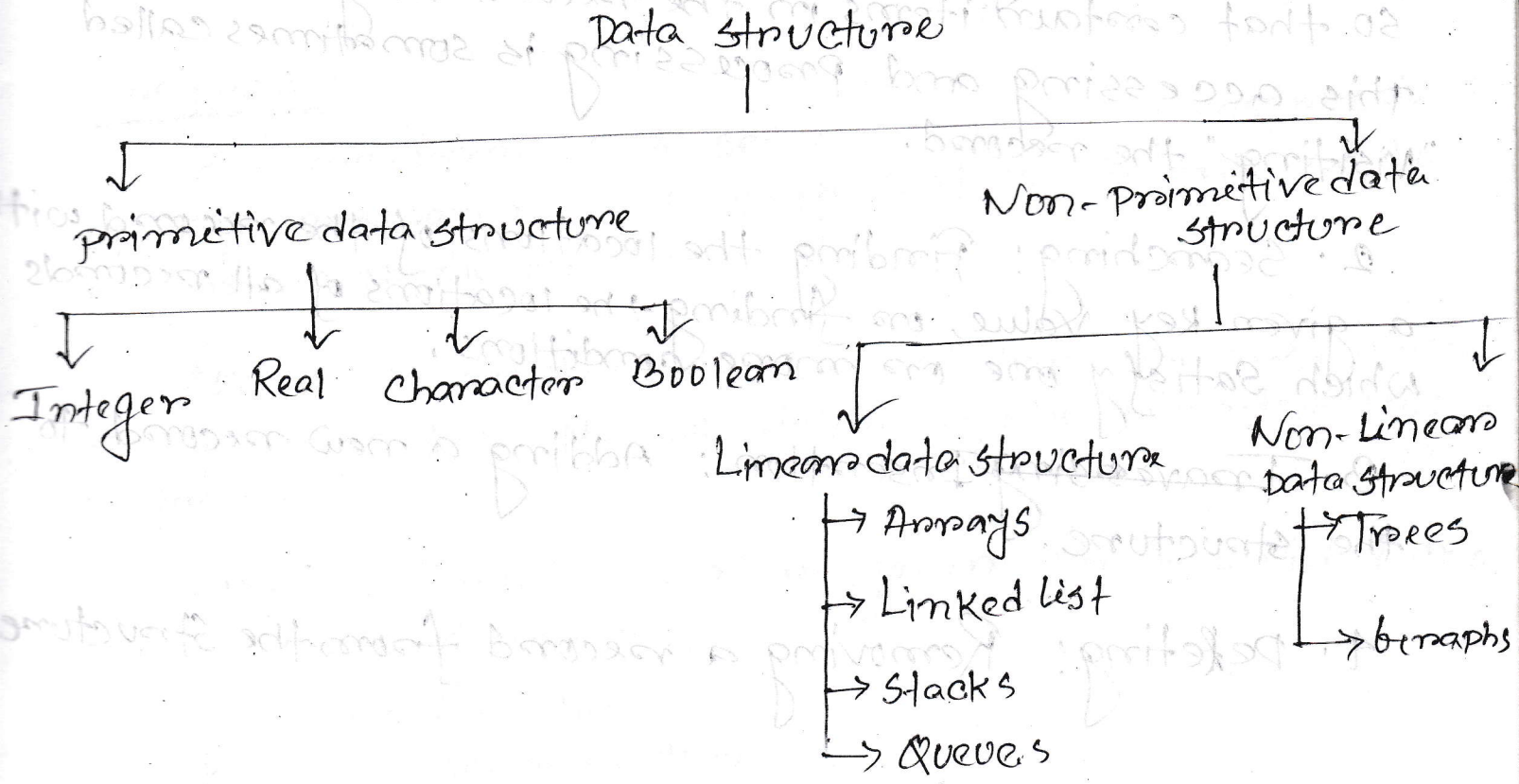
Data may be organized in many different ways; the logical and mathematical model of a particular organization of data is called a data structure.

The choice of a particular data model depends on two considerations.

First, it must be rich enough in structure to mirror the actual relationships of the data in the real world.

On the otherhand, the structure should be simple enough that one can effectively process the data when necessary

Classification of data structure:



Primitive data structure:

The basic data structures that directly operate upon the machine instructions.

Non-primitive data structure:

More complicated data structures and are derived from primitive data structures.

Data Structure Operations:

The following four operations play a major role in data structure -

1. Traversing: Accessing each record exactly once so that certain items in the record may be processed. this accessing and processing is sometimes called "visiting" the record.

2. Searching: Finding the locations of the record with a given key value, or finding the locations of all records which satisfy one or more conditions.

3. ~~Traversing~~ Insertion: Adding a new record to the structure.

4. Deleting: Removing a record from the structure.

The following two operations, which are used in special situations, will also be considered.

1. Sorting: Arranging the records in some logical order.

2. Merging: Combining the records in two different sorted files into a single sorted file.

Algorithm:

An algorithm is a well-defined list of steps for solving a particular problem. Each and every algorithm can be divided into three sections.

1. input section
2. operational or processing section
3. output section

Program:

A program is a set or sequence of instructions of any programming language that can be followed to perform a particular task.

For a particular problem, at first we may write an algorithm then the algorithm may be converted into a program.

Like an algorithm, a program can be divided into three sections such as,

a. input section
b. processing section
c. output section

Complexity:

The complexity of an algorithm is the function which gives the running time and/or space in terms of the input size.

1. Time Complexity: Relate to the execution time of the algorithm. Depends on the number of element comparis and number of element movement.

2. Space Complexity: Relate to space (memory) needs in the main memory for the data used to implement the algorithm for solving any problem.

3. Time Space Tradeoff: Sometimes the choice of data structure involves a time space tradeoff: by increasing the amount of space for storing the data, one may be able to reduce the time needed for processi the data or vice-versa.

Asymptotic Analysis:

Asymptotic analysis of an algorithm refers to defining the mathematical boundation/framing of its run time performance.

Using asymptotic analysis, we can very well conclud the best case, average case and worst case scenario of an algorithm.

Using the time required by an algorithm falls under three types.

1. Best case: Minimum time required for program execution.
2. Average case: Average time required for program execution.
3. Worst case: Maximum time required for program execution.

Asymptotic Notations:

Asymptotic notation is a way of expressing the cost of an algorithm. Goal of asymptotic notation is to simplify analysis by getting rid of unneeded information.

Following are the commonly used asymptotic notations to calculate the running time complexity of an algorithm.

- O notation
- Ω notation
- Θ notation

Big oh Notation (O): The notation $O(n)$ is the formal way to express the upper bound of an algorithm's running time.

It measures the worst case time complexity or the longest amount of time an algorithm can possibly take to complete.

For example, for a function $f(n)$

$O(f(n)) = \{g(n)$: there exists $c > 0$ and n_0 such that $f(n) \leq c \cdot g(n)$ for all $n > n_0\}$

Omega Notation (Ω): The notation $\Omega(n)$ is the formal way to express the lower bound of an algorithm running time.

It measures the best case time complexity or the best amount of time an algorithm can possibly take to complete.

For example, for a function $f(n)$

$\Omega(f(n)) = \{g(n)$: there exists $c > 0$ and n_0 such that $g(n) \geq c \cdot f(n)$ for all $n > n_0\}$

Theta Notation (Θ): The notation $\Theta(n)$ is the formal way to express both the lower bound and the upper bound of an algorithm's running time. It is represented as follows -

$\Theta(f(n)) = \{g(n)$ if and only if $g(n) = O(f(n))$ and $g(n) = \Omega(f(n))$ for all $n > n_0\}$

Floor function:

$\lfloor x \rfloor$ called the floor of x, denotes the greatest integer that does not exceed x.

Ex: $\lfloor 3.14 \rfloor = 3$, $\lfloor \sqrt{5} \rfloor = 2$, $\lfloor -8.5 \rfloor = -9$, $\lfloor 7 \rfloor = 7$

Ceiling Function:

$\lceil x \rceil$ called the ceiling of x, denotes the least integer that is not less than x

Ex: $\lceil 3.14 \rceil = 4$, $\lceil \sqrt{5} \rceil = 3$, $\lceil -8.5 \rceil = -8$, $\lceil 7 \rceil = 7$

Algorithmic Notation:

1. Identifying Number: Algorithm 4.3 refers to the third algorithm in chapter 4. Algorithm P5.3 refers to the algorithm in solved problem 5.3 in chaper 5. "P" indicates that the algorithm appears in a problem.

2. Comments: Each step may contain a comment in brackets that indicates the main purpose of the step.

3. Variable Names: Use capital letters as in MAX and DATA.

4. Assignment statement : Use dot - equal

5. Input and output

Read : Variables Names

Write : Messages and/or Variables Names

6. Procedures.

Control Structure:

1. Sequence logic (Sequential flow)

Algorithm

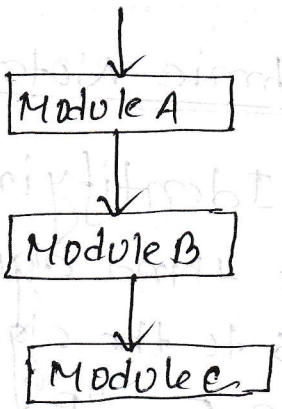
:

Module A

Module B

Module C

:



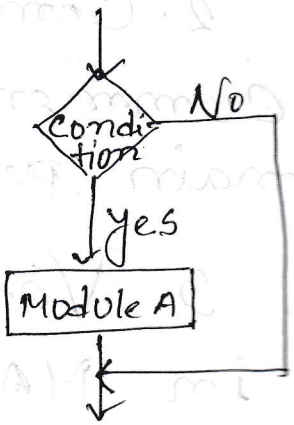
2. Selection Logic (conditional flow)

a. Single Alternative

IF Condition THEN

Module A

END of IF Statement



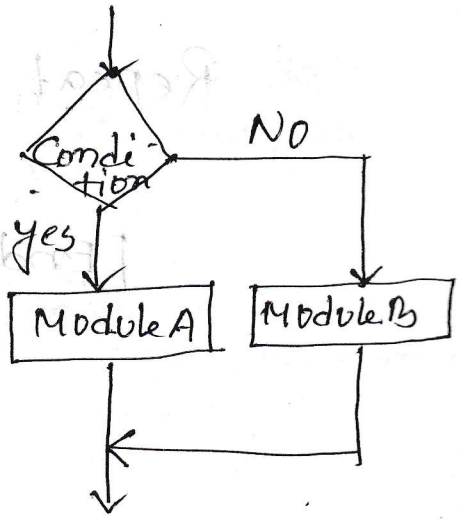
b. Double Alternative

If Condition, then:

[Module A]

Else: [Module B]

[End of If structure]



c. Multiple Alternative

If condition (1), then

[Module A1]

Else If condition (2), then

[Module A2]

⋮

Else If condition (M), then

[Module AM]

Else: [Module B]

[End of IF structure]

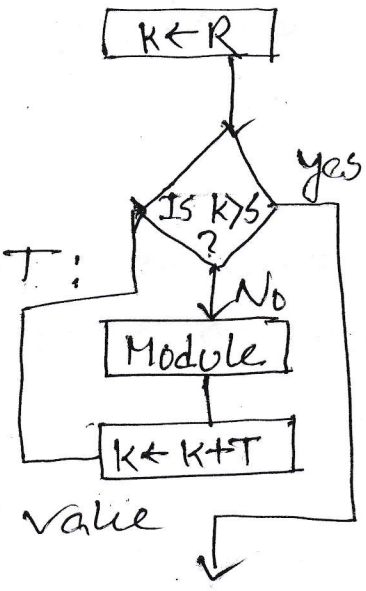
3. Iteration Logic (Repetitive Flow)

a) Repeat for Loop:

Repeat for K = R to S by T:

[Module]

[End of Loop]



R = initial value, S = end value or test value

T = increment

b) Repeat while loop:

Repeat while condition:

[Module]

[End of loop]