Unit - I

Proceduhal Prog^m	Object Oriented Prog^m
1. Program into Small parts called functions	1. Program into Small parts called objects
2. Top Down Approach	2. Bottom up Approach
3. No Access specifiers (public, private)	3. Access specifiers there
4. less Secure	4. More Secure
5. No Code Reusability	5. Code Reusability there
Eg: C, Pascal	Eg: C++, Java

Features of C++

-) OOPs -) High Performance -) Rich Librahy

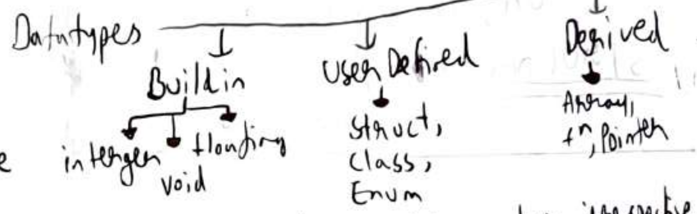
-) Low Level Manipulation

-) Multi Paradigm (Procedural + OOPs)

#include <iostream> -) Pre Processor Directives (including Libs)
flow of data making perform fast

cout → Predefined object of
cin → Similar ostream

Keyword → Reserved identifiers/words (names in prog^m)



Char, int, Short int, long int, float, double
(1) (2) (2) (4) (4) (8)

Void -) absence of value, Pointer & f^n do not return Any value

Struct -) Holds values irrespective of data type (grouping variables)

Array -) Contigous Coll^n of data with similar data types

Enum -) Way to attach name to nu.

F^n -) Block of Reusable code

Variable → to store value

Operators: Arithmetic, Relational, Logical, Assignment, Bitwise
(+, -, *, /, %) (==, !=, <, >, <=, >=) (&&, ||, !) (=, +=, ..) (&, |, ^, ~, <<, >>)

Unary (++, --)

Associativity → Determines the dir^n of evaluation

Operator Precedence → Determines in which order oper^s are evaluated

Type Conversion → One data type to Another
→ Implicit → Compiler Automatically
→ Explicit → User Manually

Control Flow Statements → Manage the order of exe^n of a prog^m

-) Decision Making → only Execute on Condition Eg: if, if-else, if-else-elseif Switch

-) Looping Statements → Looping code block Eg: for, while, do-while

-) Jump Statements → Breaks prog^m flow Eg: break, continue, goto, return

Functions → Library → buildin Eg: Sqrt() from <cmath>
→ User Defined Eg: a = add(b, c) //Actual Parameters
int add(int a, int b) //Formal Parameters

→ Call by Value (Just value) (int a, int b)
→ Call by Reference (int &a, int &b) (Actual Reference)

Scope → Variables Accessed in the region ({}) where they are created
→ Local
→ Global

Operator overloading see example

Class → Blueprint to create an object
Encapsulates Data & Methods in a single entity.

this → Special pointer that points to current object of a class

Object → instance of the Class

Structure

- → Class Name
- → Access specifier
  - → Public (Anyone)
  - → Private (only within class)
  - → Protected (within + Derived class)
- → Constructor
- → Destructor
- → Data Members
- → Member f^s

Constructor → Piece of code with same name as of class that runs automatically when object is created
↳ Default, Parametrized

Destructor → Piece of code that destroys object as soon as scope of object ends

OOPS → I Dont Care

Unit - 2

DS → Container that stores & organizes data in a specific way to use it efficiently

Data: Info to be stored
Structure: Way in which data is stored

store data one after another

Linear
→ Array
→ Linked list
→ Stack
→ Queue

Non - linear
Trees & Graphs

Why?
→ Better Time complexity
→ Better space complexity
→ Data Management
→ Solving complex problems
→ Data security
→ Data Retrieval

Primitive DS → int, float, double, char
Non - Primitive DS → Linear, Non-Linear
Linear → Direct Access & Sequential Access
(Array, Matrix) (LL, Stack, queue)
Non - Linear → Hierarchical & Unordered
(Trees, Heaps) (Graphs, sets, Hash Table)

Recursion → f^n calling itself Again & Again Base Cond^n → Cond^n to exit recurssion
↳ tail, head, tree, indirect (See Examples/ChatGPT)

Array → Same Data in continuous Memory, Always starts from 0, elements Accessed by indices, once declared, size is constant

Whenever you write C++ program, write Space & time complexity.

Row Major order → Row wise adding/accessing elements
Column Major Order → Column wise adding/accessing elements
→ 1D, 2D Arrays see examples
Dynamic Array → Array that can automatically resize to fit more elements
↳ vector

Sparse Matrix → Most elements of Matrix = 0] Array Representation
↳ Lessen Storage space + Faster Computation] Linkedlist "

Row	column	Value	Ptr to next

Array	Dynamic Array
1. Size fixed or static	1. Size Dynamic
2. In Stack	2. In Heap
3. No STL Support	3. STL Support
4. No Resizable	4. Resizable
Syntax, Eg.	Syntax, Eg.

Abstract Data type → theoretical Model of a Data Structure that tells what oper^n can be performed and what they does

Linked List → collec^n of elements in Lists that are connected/linked with each other
↓
No direct access + pointer space is needed

LL → SLL, DLL, CLL

Each item in List called Node →

Data	Ptr

See Egs

SLL ADT
↓
[insert, delete, traversal
↓
beginning, end, at any position

SLL → Only in 1 direction
DLL → In 2 directions

prev	Data	next

DLL ADT → Insert, Delete, Traversal → Beginning, end, at any position

Real Life Uses:
→ Memory Management by OS
→ Undo/Redo f^n
→ Browser forward & backward Navigation
→ Hash Tables
→ Stack & queue creation
→ Music/Video Playlist

Unit-3

Stack → Linear DS with FILO pattern
→ based on real life stack (of books, chairs...)

Operations:
1> Push 2> Pop 3> isEmpty 4> Peek
5> Display 6> isFull

Applications:
1> Balancing Symbols
2> Infix ⟷ Postfix
3) Redo - Undo in editors
4) Forward - Backward in web Browser
5) Algorithms like rat in maze, Sudoku Solver.

-) Has a reference to top Node

Queue → Linear DS With FIFO pattern
→ based on real life queue (for ticket)

Operations:

1) Enqueue 2> Dequeue 3> isEmpty 4) Front
5> Display 6> isFull

Applications:
1) CPU scheduling
2> Call Centers → Calls placed in queues
3> Traffic Flow
4> In Manufacturing to optimize production Lines
5) orders placed in Stock Exchange
6) Asynchronous Communication

-) Reference to Front end & Rear end

Unit -4

Tree → Non-linear DS that stores in a hierarchichical tree structure with root & sub trees of children with Parent

Root → topmost special Node
Parent → Predecessor of Node
Child → Successor of Node
Sibiling → Nodes with Same parent

Degree → No. of Children
Depth → Root to Node
Height → Leaf to Node

Traversal

- Depth First Search
  - Preorder
  - inorder
  - post order
- Breadth First Search (Level Search)

Properties

1) Recursive DS
2) N nodes N-1 edges

Binary Tree → every node at most 2 children

Applications:

1) Folder Structure
2) BST for Better searching Sorted Data
3) B- & B+ Trees used in databases
4) Comiler to build syntax trees
5) Decision Trees
6) Version Control Systems(git)

BST → Binary Tree where Left subtree is small & right subtree greater

Searching → $O(\log(N))$

Unit-5 Graphs → Non Linear DS with Multiple Links

Tree	Graphs
1. One path b/w 1 nodes	1. Multiple Paths b/w 2 nodes
2. Has Root Node	2. No Root Node
3. No Loops	3. Can have loops
4. Hierarchical Model	4. Network Model
Syntax, Eg:	Syntax, Eg
5. Tree is Undirected graph	

Graphs
- Directed → edge has dirⁿ associated to it (World wide web)
- Undirected → No dirⁿ associated (Social Network)
- Unweighted → All edges Same weight (World wide web)
- Weighted → Each Path has different weight (Eg: Google Maps)

Adjacency Matrix → Way to represent graph as Matrix (0s & 1s)

Adjacency List → List of Adjacent Nodes

BFS → traversal Algorithm

Only ones connected to node & nodes connected to them

DFS → traversing Algorithm

→ time (O(n))

S→A goto A shift

→ Not optimal

Hashing → Best searching

Linear → O(N)

Binary → O(logN)

Hashing → O(1)

Applications:
1) Databases
2) Symbol Tables
3) Memory Addressing
4) Data dictionaries

Hash Fⁿ → Fⁿ to get the key of Value
- Division Method (36%10 = 6)
- Mid Square ($16^2 = 256$ So 5)
- Digit Folding (12+3+16=31 So 12316 in 31)
- Multiplicative floor(table size * key * A) % table size (floor(10 * 23 * 0.618) % 10 So 2 → 23)

Good Hash fⁿ:
1) Simple
2) cost less
3) Less collisions
4) Hash key distributed Uniformly
5) Use all info by key

Collision → 2 values Same keys

Collision Resolution → Seperate Chaining (linked list)

0 [5]
1 [10] → [20] → [30]

→ simple, easy

→ Wastage of Space + Cache Performance not good

Collision Resolution → Open Addressing
- Linear Probing: rehash → (n+1)%ts
- Quadratic Probing ($hash+1^2$, $hash+2^2$, $hash+3^3$ ⋮)
- Double Hashing ($hash + 1*hash_2$, $hash + 2*hash2$ ⋮)

Priority Queue → Special queue where each element has priority to it

Eg: Patients in Hospital

2 Types → Min PQ, Max PQ

Applications:
1) CPU Scheduling
2) creating stacks
3) kth Largest Element

All queue's Applications

Operations:
1) isEmpty 2) insert 3) find Min
4) find Max 5) Remove

Heap → Complete Binary Tree where the root is min/max

Applications ↑

Children → 2n+1, 2n+2

Parent → (n-1)/2

Operations:
1) getMin 2) getMax
3) insert 4) delete
5) heapify