Unit-2 Notes → By Sweety Jushi

Structured Data Types

A data object that is constructed as an aggregate of other data object (or component) is called structured data object or data structure.

component may be

Elementary

data structure
↓ e.g
(Record, character string)

Specification.

① **Number of components**

Fixed Size → A datastructure may be fixed size if the number of components is invariant during its life time.
e.g Array, Records

Variable Size → A datastructure may be variable size if the number of component changes dynamically.
e.g Stacks, lists, sets, tables and files.

(ii) **Type of Each components**

Homogeneous Data Structure – If all its components are of same type e.g Array, Sets, files.

Hetegeneous Data Structure :→ If components are diff type
e.g Records, List

(iii) **Names to be used for selecting components** →

Different Data structure have different selection mechanism
e.g For Array → we use Subscript
For Stack → we can select or pop only Top element
For files → we can Access a particular Element

(iv) **Maximum number of components** :- For a variable size data structure such as a stack, a maximum size of structure in terms of no. of components may be specified.

⑤ Organization of components
↳ Two Types < linear Sequential organization
Linked " "

Operations on Data Structure

① component Selection operation
↳ Sequential Selection :- components are selected predetermined order. e.g vector
↳ Random Selection :- components are Selected randomly. e.g Linked List

(ii) whole data Structure operation
operation may take entire data Structure as an argument and produce a new data Structure as a results. e.g Language APL & SNOBOL4 provide such sets of whole data Structure operations.

(iii) Insertion/Deletion of components → Insertion and Deletion of components have major impact on Storage representation.

(iv) Creation/Destruction of Data Structure :- creation and distruction of data Structure have major impact on storage representation of data structure

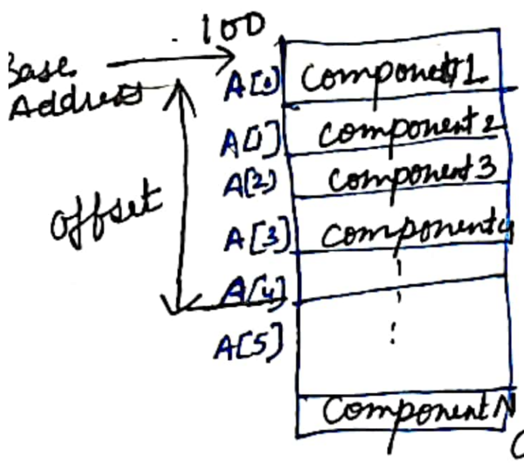
Implementation of dataStructure :-

Storage Representation
The storage representation for data structure include (i) Storage for components of Structure
(ii) An optional discriptor that Store some or all attribute of the Structure.

2 Basic Representation

ⓐ Sequential Representation ⓑ Linked Representation

Sequential Representation → In which Data Structure is Stored in single contiguous block of Storage that includes both descriptor and component



→ The Starting location of entire block is called base address..

offset :- The Relative position (location) of Selected component called offset.

char array

Actual location of Selected component = Base Address + offset

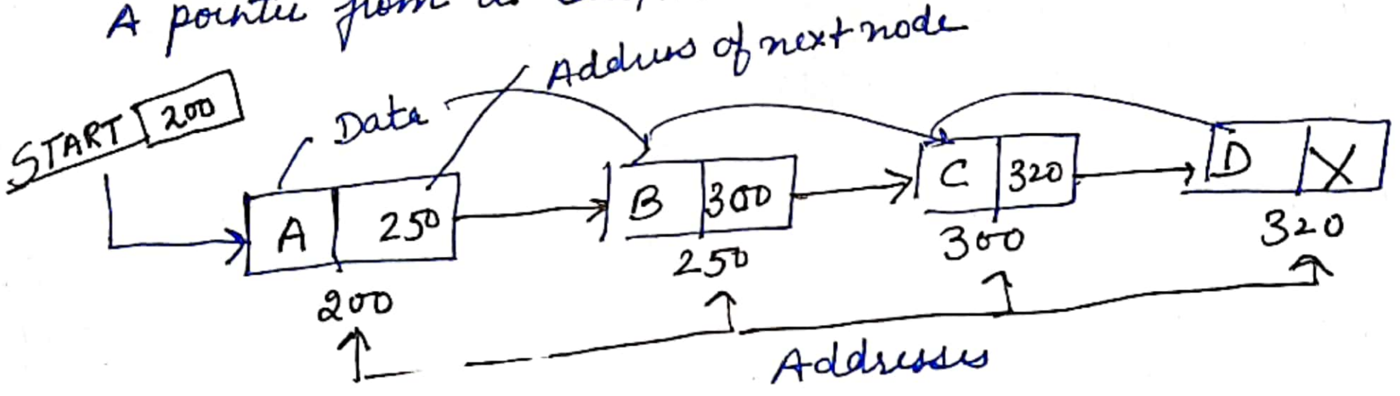
e.g

$A[5]$ = Base address + offset
100 + 5 = 105
↳ Because It is char array

(ii) Linked Representation → 1

In which Data Structure is stored in non-contiguous block of Storage that are linked through a pointer.

A pointer from a block A to Block B called link.



2 Central Problem in Storage Management

Garbage → when all access path to data object are destroyed but the data object continue to exist.

Disadvantage :- Data object can no longer be accessed by other parts of program i.e. so it is of no further use.

Vectors and Arrays

Vector :- It is a data structure composed of a fixed number of components of the same type organized as a simple linear sequence. It is a one-dimensional array.

a) Specification

1. Number of components → indicated by sequence of subscript ranges, one for each dimension.
2. Data type of each component → which is a single data type because the component are all of the same type.
3. Subscript to be used to select each component : first integer designating the first component, the second designating the second component & so on.

Declaration → A typical declaration for a vector is the Pascal declaration

V : array [-5,..5] of real; — Total 11 components

C declarations

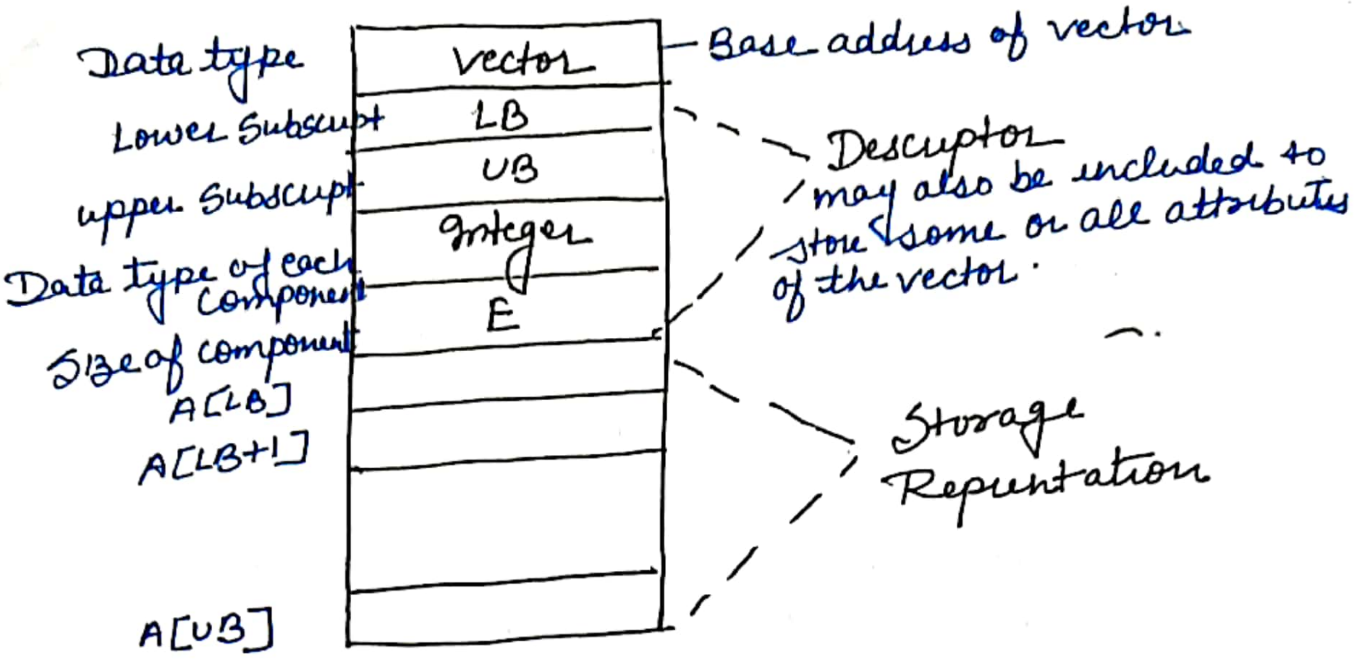
float a[10];

b) operations on vectors

(i) Create vector
(ii) Destroy vector
(iii) component Selection :- written as vector name followed by subscript of the component to be selected
V[2]
(iv) Insertion] Not allowed in case of vector ~~becoz~~ because
(v) Deletion] vectors are fixed size.
(vi) Modification of component.

Implementation :→ The homogeneity of components and fixed size of a vector make storage and accessing of individual components straightforward.

- Homogeneity → implies that size and structure of each component is same.
- Fixed size :- implies that no. & position of each component invariant throughout its life time.



<u>Multidimensional Array</u> :- ~~Are~~
Matrix :- composed of rows and columns.

@ <u>specification and Syntax</u>

B: array [1....10, -5 ... 5] of real;

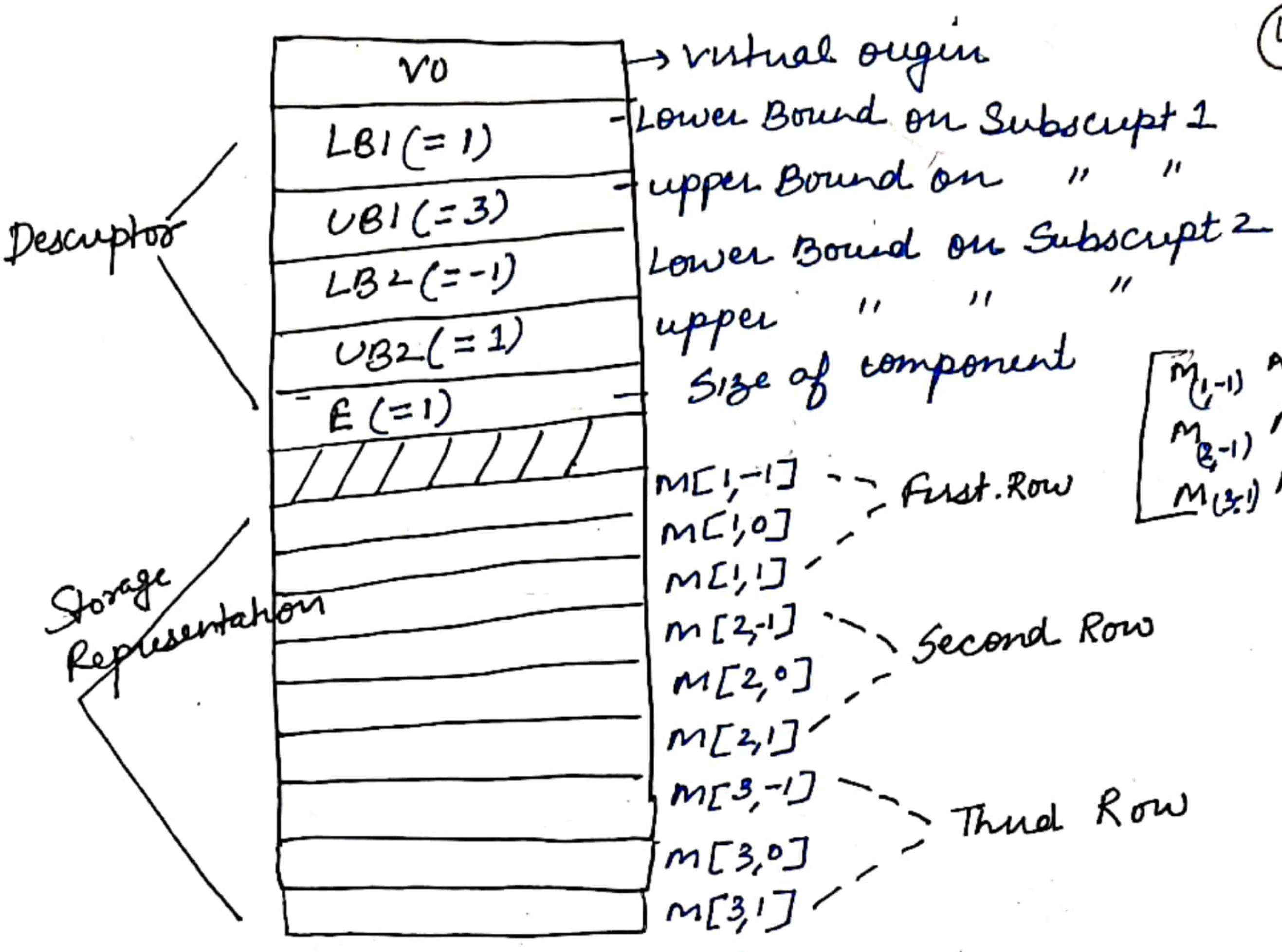
Selection of a component requires that one subscript be given for each dimension.

<u>Implementation</u>:-
Matrix is conveniently implemented by considering it as vector of vectors.

<u>Storage representation</u>

Row Major Storage representation
Data is stored in first row followed by second row and so on.

Column major Storage representation
Data is stored in first column followed by second column & so on.



$$\begin{bmatrix} M_{(1,-1)} & M_{(1,0)} & M_{(1,1)} \\ M_{(2,-1)} & M_{(2,0)} & M_{(2,1)} \\ M_{(3,-1)} & M_{(3,0)} & M_{(3,1)} \end{bmatrix}$$

[Row Major Form]

$$\begin{bmatrix} M(1,-1) & M(1,0) & M(1,1) \\ M(2,-1) & M(2,0) & M(2,1) \\ M(3,-1) & M(3,0) & M(3,1) \end{bmatrix}$$

Records / Structure

A datastructure composed of a fixed number of components of different types is usually termed a record.

Specification → component may be heterogeneous & they have symbolic name.

(A) • The component of records may be heterogeneous (i.e mixed data type) rather than homogeneous.

• The component of records are named with symbolic names (identifier) rather than indexed with ~~subscript~~

(B) Declaration: - Declaration for component allow size of each component.

```
Struct EmployeeType
{ int ID;
  int age;
  float Salary;
  char Dept;
} Employee;
```

component names are ID, age, Salary, Dept

Also called field.

(C) Major Attributes of Record

• The number of component → 4 component [ID, age, Salary, Dept]
• Type of each component → ~~ID, ag~~ int, int, float, char
• The Selector used to name each component

Employee . ID
Employee . Salary

Implementation: - The storage representation for a record is single sequential block of memory in which components are stored in sequence.

↳ component Selection: - is easily implemented because field names are known during translation

To access any member of a structure we use member access operator (•)

e.g Employee • Salary → Structure Member Name

Structure Variable Name

↳ Member Access operator

offset of any component may be computed during translation
The basic accessing formula used to compute the location of the Ith component is

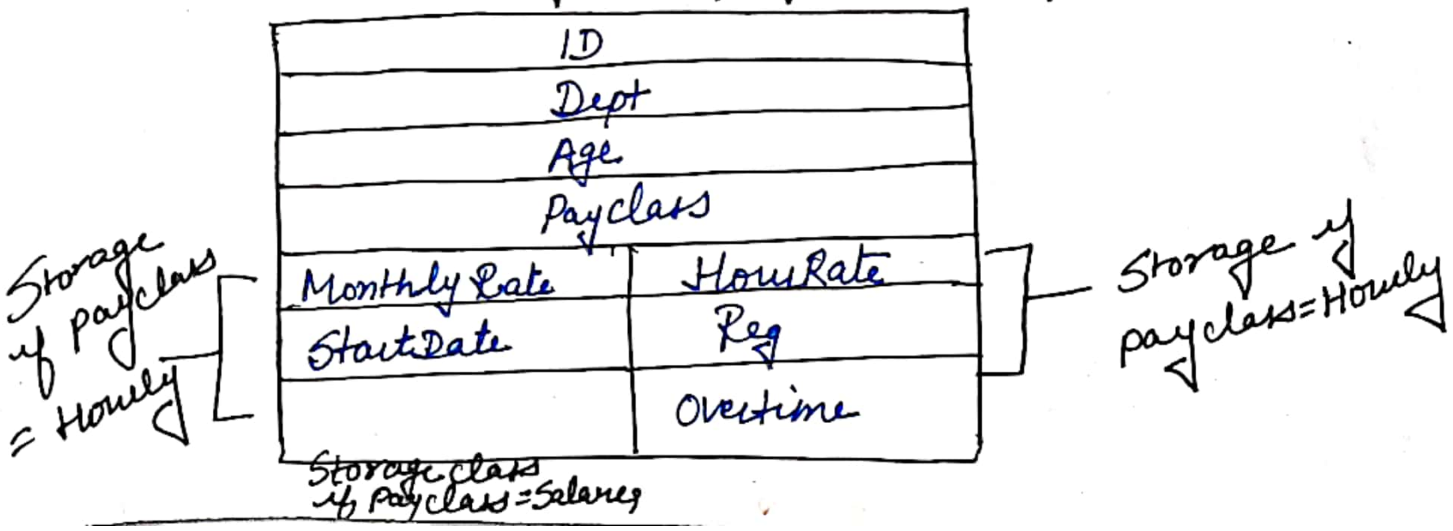
$$\underbrace{\text{lvalue}(R.I) = \alpha}_{\text{Base Address}} + \sum_{j=1}^{I-1} (\text{Size of } \overset{\text{is } j\text{th component}}{R_j})$$

Records And Array with Structured components

```
Struct Employee Type
{   int ID;
    int Age;
    float Salary
    char Dept;
    } Employee[500];
```

Variant Records :- Such a record ordinarily has one or more components that are common to all variants and several other components with names and data type that are unique to that variant.
→ Also known as union type

eg Employee payrole information Record have 2 variants
- one for Salaried employees paid by month
- one for employees paid by hour.



ID
Dept
Age
Payclass
MonthlyRate
StartDate
HourRate
Reg
Overtime
Storage of payclass = Hourly
Storage of payclass=Hourly
Storage class of Payclass=Salaried

Implementation

During translation, the amount of storage required for the components of each variant is determined and Storage is allocated in the record for largest possible variant.

Selection → selection of component is similar to ordinary record when no checking is to be done.

e.g Employee. MonthlyRate
Employee. Reg.

location of component → computed by adding offset to base address.

Sets :- A Set is a data object containing unordered collection of distinct values. i.e unique values without any particular order.

Basic operation

(i) **Membership**:- $(x \in S)$ implies x is member of Set S.

(ii) **Insertion/Deletion**

↳ Insert data value x in Set S if it is not a member of S.
Delete data value x from Set S if x is a member of S.

(iii) **Union** → Union(S1, S2) → contains all members of S1 and S2 with duplicate deleted.

(iv) **Intersection** → Intersection (S1, S2) → common value of S1 & S2

(v) **Difference** :- Difference(S1, S2) → contains value that are in S1 but not in S2.

Implementation of Sets

Bit String representation of Sets → This Representation is appropriate where size of universe of value small.

Suppose there are 5 elements in universe

$U = \{1, 2, 3, 4, 5\}$

and there are 3 elements in Set A.

$A = \{2, 4, 5\}$

Bit String representation of Set A = ~~{~~ 0 1 0 1 1

↳ Set 1 if ~~if~~ element in A
Set 0 " " not in A

$\{B\} = \{\}$

Bit string Representation of B = 0 0 0 0 0

$C = U$

Bit String Representation of C = 1 1 1 1 1

Let $A = \{2, 4, 5\}$ $B = \{1, 3, 5\}$

$A \cap B = \{5\}$

Bit Representation of A = 0 1 0 1 1

" " " B 1 0 1 0 1

" " " $A \cap B$ 0 0 0 0 1 $\Leftrightarrow \{5\}$

Hash-coded Representation :- Also known as scatter storage. This method used when universe of possible value is large.

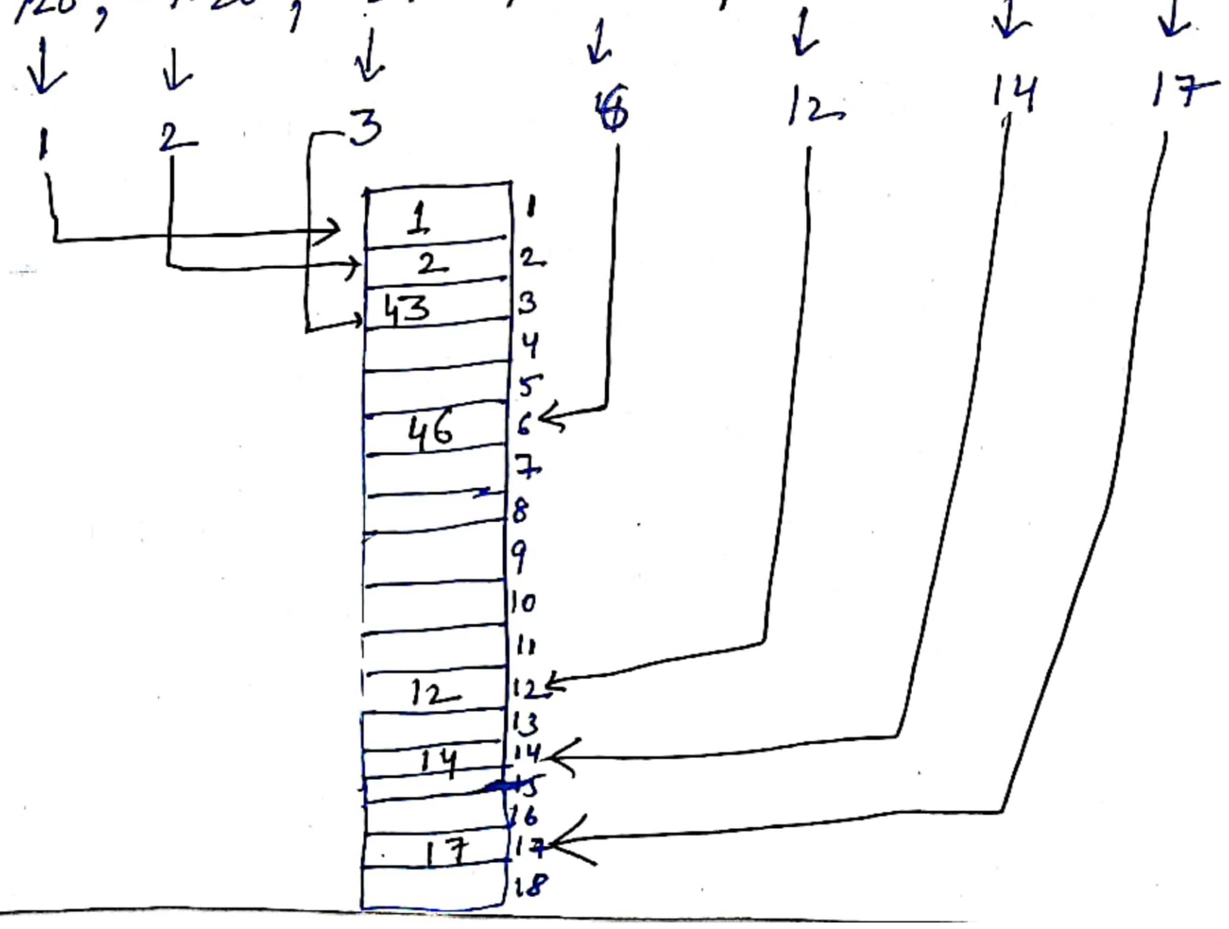
~~eg for~~.

Hashing is a technique of mapping values into hash tables by using hash function.

or Hashing is technique to convert a range of ~~indexes~~ key value into a range of indexes of array

$\{1, 2, 42, 4, 12, 14, 17\}$

$\{1\%20,\ 2\%20,\ 43\%20,\ 46\%20,\ 12\%20,\ 14\%20,\ 17\%20\}$



1
2
3
6
12
14
17
1
2
43
46
12
14
17
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18