Data Structures

PAWAN KUMAR
ASSISTANT PROFESSOR
DEPARTMENT OF CSE
GJUS&T, HISAR

Contents

- Introduction to Data Structure
- Types of Data Structure
- Abstract data types
- Arrays
- linked lists
- Memory representations of Array and Linked List
- Operations on Array and Linked List
- Applications of arrays and linked lists
- Representing sets and polynomials using linked lists

Data Structure

- Data structure is way of storing and arranging data in such a way that the users can access, use and modify the data according to their requirements.

or

- Data may be organized in many different ways, the logical or mathematical model of a particular organization of data is called Data Structure.
- Data are values or set of values.
- A data item refers to a single unit of values.
- Data items that are divided into sub items are called group items.
- For example in database a person name can be divided as: First name, middle name, last name.

Types of Data Structure

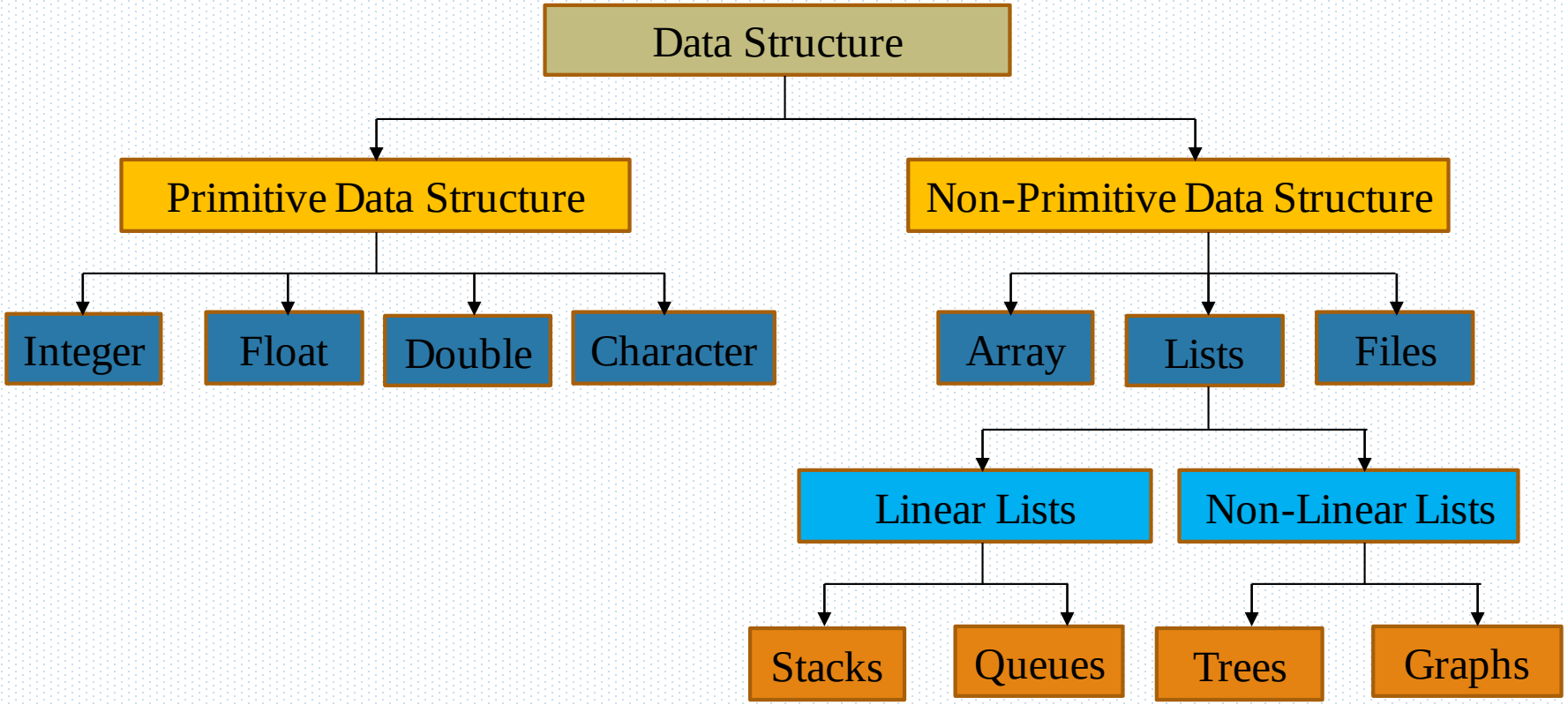


Figure 1: Hierarchy/ Types of Data Structure

Data Type v/s Data Structures

- In general both data type and data structure seems to be the same thing as both deals with the nature and organizing of data but among two one describes the type and nature of data while other represents the collections in which that data can be stored.
- Data type is the representation of nature and type of data that has been going to be used in programming or
- In other words data type describes all that data which share a common property.
- For example an integer data type describes every integer that the computers can handle.
- On other hand Data structure is the collection that holds data which can be manipulated and used in programming so that operations and algorithms can be more easily applied.
- For example tree type data structures often allow for efficient searching algorithms.
- We can say primitive data structures are also data types.

Primitive Data Structure

- Primitive Data Structures are the basic data structures that directly operate upon the machine instructions. They have different representations on different computers.
- Primitive data structures are those which are predefined way of storing data by the system. And the set of operations that can be performed on these data are also predefined.
- Data structures that can be manipulated directly by machine instructions are called primitive data structures.
- Primitive Data Structures are directly operated upon by machine-level instructions. These are In-Built too.
- They can be used to create our own data types (Non-Primitive Data Structures).

Primitive Data Structure

- Primitive data structures are char, int, float, double which are mainly used.
- Some other data types are byte, Boolean, short, long
- Characters are internally considered as int and floats also falls under double and the predefined operations are addition, subtraction, etc.
- But there are certain situations when primitive data structures are not sufficient for our job. There comes derived data structures and user defined data structures.
- Term "data type" and "primitive data type" are often used interchangeably.
- Primitive data types are predefined types of data, which are supported by the programming language.

Data Types Ranges

Data Type	Default Value	Default size	Type of number	Value-Range
boolean	false	1 bit	Flag	True- False
char	'\u0000'	2 byte	Character	Store a single character
byte	0	1 byte	whole	-128 to +127
short	0	2 byte	whole	-32,768 to 32,767
int	0	4 byte	whole	-2,147,483,648 to 2,147,483,647
long	0L	8 byte	whole	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	0.0f	4 byte	Factional	Sufficient for storing 6 to 7 decimal digits
double	0.0d	8 byte	Factional	Sufficient for storing 15 decimal digits

Non-Primitive Data Structure

- Non-primitive data structures are more complicated data structures and are derived from primitive data structures.
- They emphasize on grouping same or different data items with relationship between each data item.
- Derived data structures are also provided by the system but are made using primitive like an array, a derived data.
- It can be array of chars, array of integers, etc.
- The set of operations that can be performed on derived data structures are also predefined.

Non-Primitive Data Structure

- Finally there are user defined data types which the user defines using the primitive and derived data types using language constructs like structure or class and uses according to their needs. And the user has to define the set or operations that we can perform on them.
- User defines data types are array, linked lists, stack, queue, trees, graphs etc.

Abstract Data Type

- An abstract data type, sometimes abbreviated ADT, is a logical description of how we view the data and the operations that are allowed without regard to how they will be implemented. OR
- Abstract data type is special kind of data type, whose behavior is defined by a set of values and set of operations. The keyword “Abstract” is used as we can use these data types, we can perform different operations. But how those operations are working that is totally hidden from the user.
- It also means that we are concerned only with what the data is representing and not with how it will eventually be constructed. By providing this level of abstraction, we are creating an encapsulation around the data.
- The idea is that by encapsulating the details of the implementation, we are hiding them from the user’s view. This is called information hiding.

Abstract Data Type

- ADT is made of with primitive data types, but operation logics are hidden.
- Some examples of ADT are Stack, Queue, List etc.
- some operations of those mentioned ADT −
- Stack −
  - isFull(), This is used to check whether stack is full or not
  - isEmpty(), This is used to check whether stack is empty or not
  - push(x), This is used to push x into the stack
  - pop(), This is used to delete one element from top of the stack
  - peek(), This is used to get the top most element of the stack
  - size(), this function is used to get number of elements present into the stack

Abstract Data Type

- Queue –
  - isFull(), This is used to check whether queue is full or not
  - isEmpry(), This is used to check whether queue is empty or not
  - insert(x), This is used to add x into the queue at the rear end
  - delete(), This is used to delete one element from the front end of the queue
  - size(), this function is used to get number of elements present into the queue
- List –
  - size(), this function is used to get number of elements present into the list
  - insert(x), this function is used to insert one element into the list
  - remove(x), this function is used to remove given element from the list
  - get(i), this function is used to get element at position i
  - replace(x, y), this function is used to replace x with y value

Classification of Data Structure

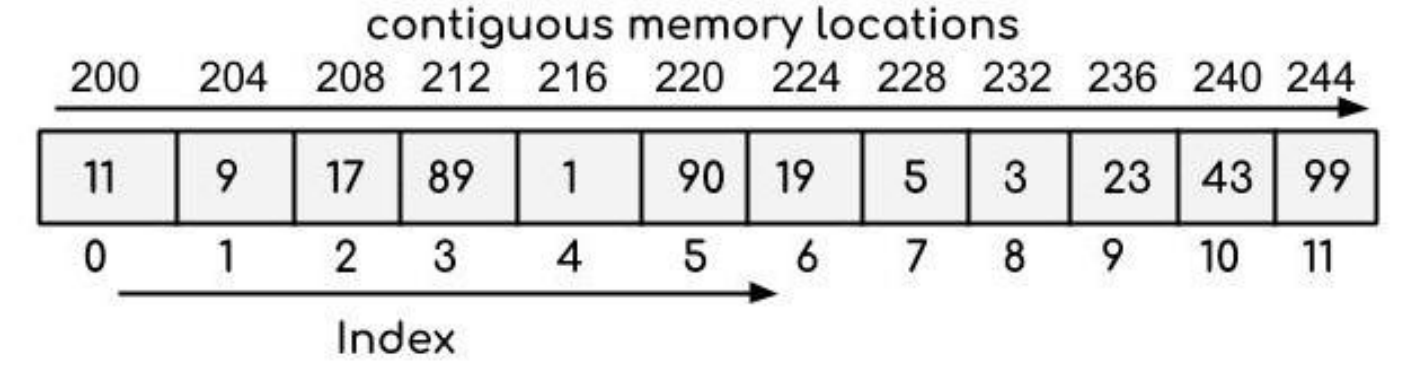
- Data structures can be classified as:
  - Linear
  - Non-Linear
- Linear: A data structure is said to be linear if its elements forms a sequence. For example- Array, list, stack, queue.
- To represent data structure in linear way, we use array and link list.
- Array use sequential memory locations and link list uses pointers or links.
- Non- Linear: A data structure is said to be non-linear if its elements does not form any sequence. For example- tree, graph.

Linear Array

- Array is a container which can hold a fix number *n* of items and these items should be of the same/ homogeneous type. OR
- A linear array is a list of a finite number *n* of homogeneous data elements such that:
  - Elements of the array are stored respectively in successively memory locations.
  - Elements of the array are referenced respectively by an *index set* consisting of *n* consecutive numbers.
- Most of the data structures make use of arrays to implement their algorithms.
- Here, *n* represents the length or size of array.
- To calculate the length of an array from the index set, we use formula:
  - Length = UB – LB + 1
  - Where, UB is largest index or upper bound
  - LB is lowest index or lower bound.

Representation of Array in Memory

- An array A can be represented in memory with continuous memory locations as shown in below figure.
- Starting memory location can be any in our computer system. Here, it is 200 and increased by 4 memory spaces, because each element need 4 bytes for storing.
- Here, we have an integer type array with size 12 or number of elements to be stored.
- Most preferably, we start indexing from 0. But we can start from 1 also.



Representation of Array in Memory

- Starting memory address in array A is called as **Base Address (BA)**.
- Using base address we can calculate or find the address of any element stored in array because each element is stored in sequence.
- LOC(A[K]) = BA(A) + w(K-LB)
  - Where, K = is index value for which address to find,
  - BA, is base address
  - w states word size or type of array
  - LB is lower bound or starting index
- For example, to find location of element at index 6.
  - LOC(A[6]) = 200 + 4(6-0) = 224

Operations on Array

- There are some basic operations which are performed on an array:
  - Traverse – Print/ process all the array elements one by one.
  - Insertion – Adds an element at the given index.
  - Deletion – Deletes an element at the given index.
  - Search – Searches an element using the given index or by the value.
  - Update – Updates an element at the given index.
  - Sorting – Arranging the elements in some type of order.
  - Merging – Combining two lists into a single list.

Traversing an Array

- Traversing: To Print/ process all the array elements one by one.
- Algorithm steps:
- 1. Set K=LB. //Initialize counter
- 2. Repeat step 3 and 4 while k <= UB
- 3. Apply process to A[K]. //visit element
- 4. Set K = K+1. //increase counter
- End of step 2 loop,
- 5. Exit.

- Here, process is an operation we want to print or read the element,

- For example- We have array A of size 5.
- Int A[5] = {3, 5, 7, 4, 6}
- Therefore, LB=0, UB=4 //Indexing start at 0
- So, step 1→ K=0
- Step 2→ while(k<=4)
- 3. Print A[K] //Here process is print element.
- 4. K increase with every time loop execution as 0 1 2 3 4 and stop as K>4.
- Output: 3 5 7 4 6

Traversing an Array

Program: Traverse an Array

```
#include <stdio.h>
main() {
   int  A[] = {1,3,5,7,8};
   int item = 10, k = 3, n = 5;
   int i = 0, j = n;
   printf("The original array elements are :\n");
   for(i = 0; i<n; i++) {
      printf("A[%d] = %d \n", i, A[i]);
 }  }
```

Insertion in Array

- Insert operation is to insert one or more data elements into an array.
- Based on the requirement, a new element can be added/ inserted at the beginning, end, or any given index of array.
- We insert an Item in array A with N elements at K^{th} position such that K<=N.

- INSERT(A, N, K, Item) //e.g. N= 10
- 1. Set J:= N // so, J=10
- 2. Repeat step 3, 4 while J>=K
- 3. Set A[J+1]:= A[J] //Move Jth element downward
- 4. Set J:= J-1 // Decrease counter, end of step 2 loop
- 5. Set A[K]:= Item. // Insert element at kth position
- 6. Set N:= N+1. // Increase value of N by 1.
- 7. Exit

Deletion in Array

- Delete operation is to delete one or more data elements from an array.
- Based on the requirement, a new element can be deleted from the beginning, end, or any given index of array.
- Deletion of any element from end of array presents no difficulty, we just need to delete the item and decrease size of array by 1.
- We delete an Item from array A with N elements at K^{th} position such that K<=N.

- Delete(A, N, K, Item)
- 1. Set Item:= A[K]
- 2. for (J=K to N-1):
- Set A[J]:= A[J+1]//Move J+1^{st} element upward
- //end of loop.
- 3. Set N:=N-1 //Reset/ Decrease value of N by 1
- 4. Exit

Linear Array

- Searching
- Sorting
- Merging
- These operation will be explained during covering searching and sorting topic.

Multi-dimensional Array

- Earlier discussed array also called as linear array or one-dimensional array.
- Simplest form of multidimensional array is the two-dimensional array.
- A two-dimensional array is, in essence, a list of one-dimensional arrays.
- To declare a two-dimensional (2-D) integer array of size [m][n], we would write something as follows − A[m][n] , where m is number of rows and n is number of columns.
- A 2-D array is collection of m*n data elements such that each element is specified by a pair of integers (such as J, K), called subscripts, with the property that

  1<=J<=m

  1<=K<=n
- Element of A with first subscript J and second subscript K will be denoted by

  $A_{J,K}$ or A[J,K]

Multi-dimensional Array

- 2-D array are called as matrices in mathematics and tables in business applications.
- Hence 2-D arrays are sometimes called matrix arrays.
- Length of any dimension in 2-D array is calculated by : UB-LB+1 // same as linear array
- Multi dimensional array can be 3-D, 4-D etc., but we will discuss only up to 2-D array
- 3-D array will be represented as A[m][n][o].
- Example- 2-D array

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Representation of 2-D Array

- A 2-D array of dimension m*n is represented as a rectangular array of elements like matrix.
- It will be represented in memory by a block of m*n sequential memory locations.
- In programming languages the array will be stored in two orders.
  - Row-major order
  - Column-major order
- To find location of first element in array we use,
- LOC(A[K])= BA(A) + w(K-1) //as per 1-D array
- In 2-D array, we have little change as they are array of 1-D arrays

Representation of 2-D Array

- **Row-major order:** In this order, each row will be stored one after another.
- As stated earlier we can start indexing either from 0 or 1. In taken example, it is 0.
- For example Array A[3][4] has 3 rows, so row1 then row 2and in last row3. OR
- { A[0][0], A[0][1], A[0][2], A[0][3], A[1][0], A[1][1], A[1][2], A[1][3], A[2][0], A[2][1], A[2][2], A[2][3]}
- In above line elements from each row are presented one by one.
- We can see, it is linear array of 12 elements (array of 3 linear arrays)

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Representation of 2-D Array

- To calculate address of some element in 2-D (row-major order) array we use following formula:
- LOC(A[J, K]) = BA(A) + w[n(J-0) +(K-0)] //when indexing start at 0
- LOC(A[J, K]) = BA(A) + w[n(J-1) +(K-1)] //when indexing start at 1
- **LOC(A[J, K]) = BA(A) + w[n(J-LB1) +(K-LB2)]** // Generalized
- Here LB1, LB2 are starting index for row and columns respectively. And UB1, UB2 are last index for rows and columns respectively.
- Where, J, K are dimensions (J^{th} row, K^{th} column) of element whose location to find,
- BA is base address of array, w is word size or type of array, n is number of columns and m is number of rows.
- Example- Find location of element A[2][3] with base address 100.
- Therefore, LOC(A[2][3]) = 100 + 4[4(2-0) + (3-0)] =100+4[4*2 +3] = 144

Representation of 2-D Array

- **Column-major order:** In this order, each column will be stored one after another.
- For example Array A[3][4] has 4 columns, so column1, then column2, column3 and in last column4. OR
- { A[0][0], A[1][0], A[2][0], A[0][1], A[1][1], A[2][1], A[0][2], A[1][2], A[2][2], A[0][3], A[1][3], A[2][3]}
- In above line elements from each column are presented one by one.
- We can see, it is linear array of 12 elements (array of 4 linear arrays)

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Representation of 2-D Array

- To calculate address of some element in 2-D (column-major order) array we use following formula:
- LOC(A[J, K]) = BA(A) + w[(J-0) +m(K-0)] // when indexing start at 0
- LOC(A[J, K]) = BA(A) + w[(J-1) +m(K-1)] // when indexing start at 1
- **LOC(A[J, K]) = BA(A) + w[(J-LB1) +m(K-LB2)]** // Generalized
- Where, J, K are dimensions (J^{th} row, K^{th} column) of element whose location to find.
- Here LB1, LB2 are starting index for row and columns respectively. And UB1, UB2 are last index for rows and columns respectively.
- BA is base address of array, w is word size or type of array, n is number of columns and m is number of rows.
- Example- Find location of element A[1][3] with base address 100.
- Therefore, LOC(A[2][3]) = 100 + 4[(1-0) + 3(3-0)] =100+4[1 +3*3] = 140

Matrices

- Vectors and Matrices are mathematical terms which refer to collection of numbers which are similar respectively to linear and 2-dimensional arrays. That is,
- An n-element vector V is a list of n numbers usually given in the form
- $V = (V_1, V_2, V_3, V_4 \ldots\ldots\ldots V_n)$
- An matrix of m*n order is an array of m.n numbers arranged in m rows and n columns as follows. E.g., a matrix or 2-D array of order 3*3
- $$\begin{matrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{matrix}$$
- A matrix with same number of rows and columns is called square matrix or n-square matrix.

Matrices

- A diagonal or main diagonal of an n-square matrix A consists of the elements A_{11}, A_{22}, A_{33}….A_{nn}.
- Sparse matrices are the matrices with high proportion of zero entries.
- Two general types of sparse matrices are as:
  - Triangular matrix
    - Lower Triangular matrix: A matrix where all entries above main diagonal are zero.
    - Upper Triangular matrix: A matrix where all entries below main diagonal are zero.
  - Tridiagonal matrix: A matrix where all non-zero entries may occur on the diagonal or on elements immediately above or below main diagonal.

$$\begin{bmatrix} 5 & 0 & 0 \\ 1 & 4 & 0 \\ 0 & 2 & 7 \end{bmatrix} \qquad \begin{matrix} 5 & 2 & 8 \\ 0 & 4 & 0 \\ 0 & 0 & 7 \end{matrix} \qquad \begin{matrix} 5 & 6 & 0 \\ 1 & 4 & 5 \\ 0 & 2 & 7 \end{matrix}$$

Lower Triangular Upper Triangular Tridiagonal

Pointer

- Pointers are the variables that are used to store the location of value present in the memory.
- A pointer P points to an element and stores its memory address.
- The process of obtaining the value stored at a location being referenced by a pointer is known as dereferencing.
- For example, we have an array DATA and P points to any element in DATA, i.e., If P contains the address of an element in DATA, then P is a pointer variable.

Pointer Array

- An array PTR is called a pointer array if each element of PTR is a pointer.
- Pointer array are useful when we need to point different size arrays/ groups with less memory consumption.
- For example, there are four groups in a class room and in each Groups (subject choice) total number of students sitting are 4, 9, 2, 6 respectively makes a total of 21 students.
- We are required to store all four groups in memory.

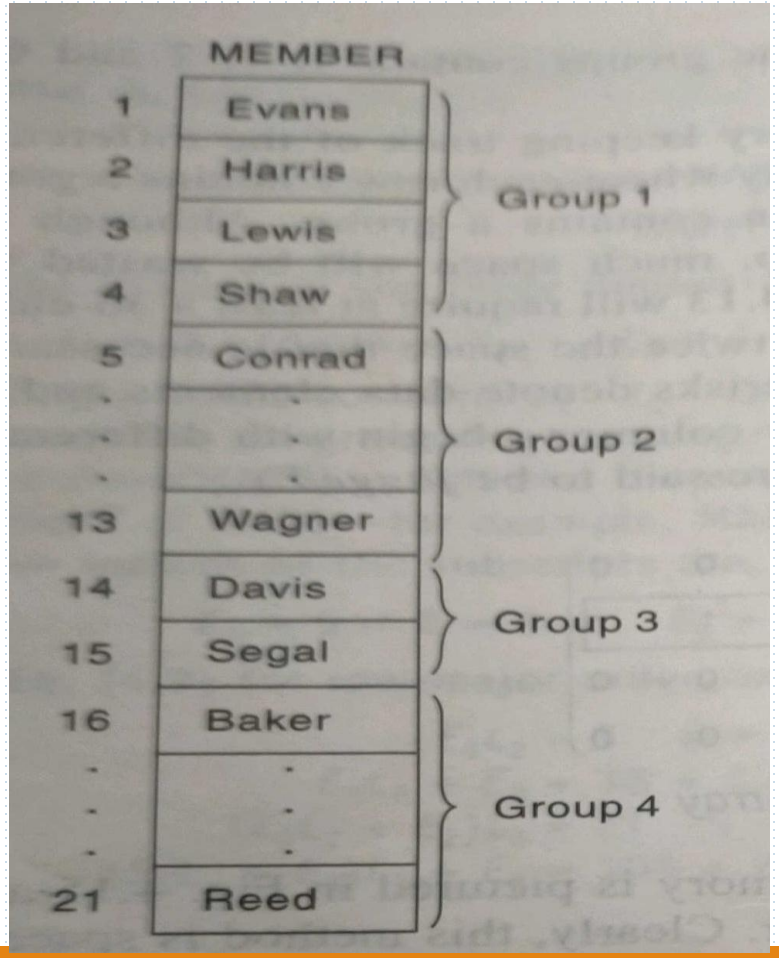
Group 1	Group 2	Group 3	Group 4
Evan	Conrad	Davis	Baker
Harris	Felt	Segal	Cooper
Lewis	Glass		Ford
Shaw	Hill		Gray
	King		Jones
	Penn		Reed
	Silver		
	Troy		
	Wagner		

Pointer Array

- First solution for this problem is to taken an 2-D array of 4*n where each row store a group or to use a 2-D array of n*4 where each column store a group. Here, n represents number of elements in largest group.
- In case, groups vary greatly will lead to more memory wastage. In our example, we require a array of 4*9 = 36 elements while we need to store only 21 elements and remaining 15 element spaces are wasted.
- $$A[4*9] = \begin{bmatrix} * & * & * & * & 0 & 0 & 0 & 0 & 0 \\ * & * & * & * & * & * & * & * & * \\ * & * & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ * & * & * & * & * & * & 0 & 0 & 0 \end{bmatrix}$$
- Entries with * are filled with elements and with 0 are blank. Here, we can see a lot of spaces are wasted in array.

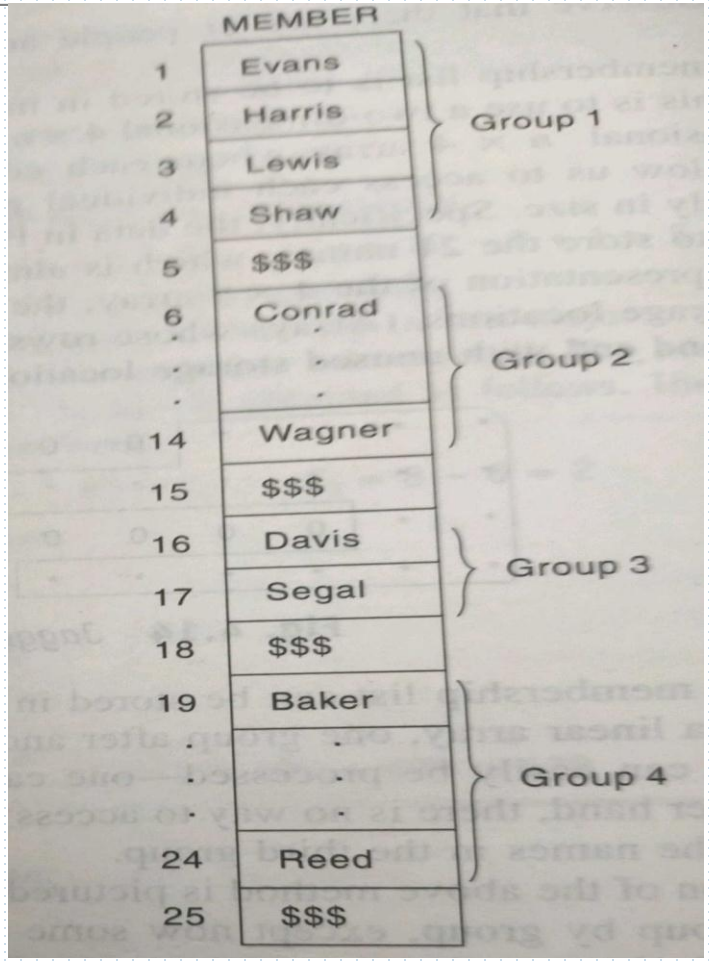
Pointer Array

- To minimize memory wastage, we take a linear array of size 21 and store all groups one after another respectively.
- Problem with this solution is that we are not able to recognize the start and end of groups.



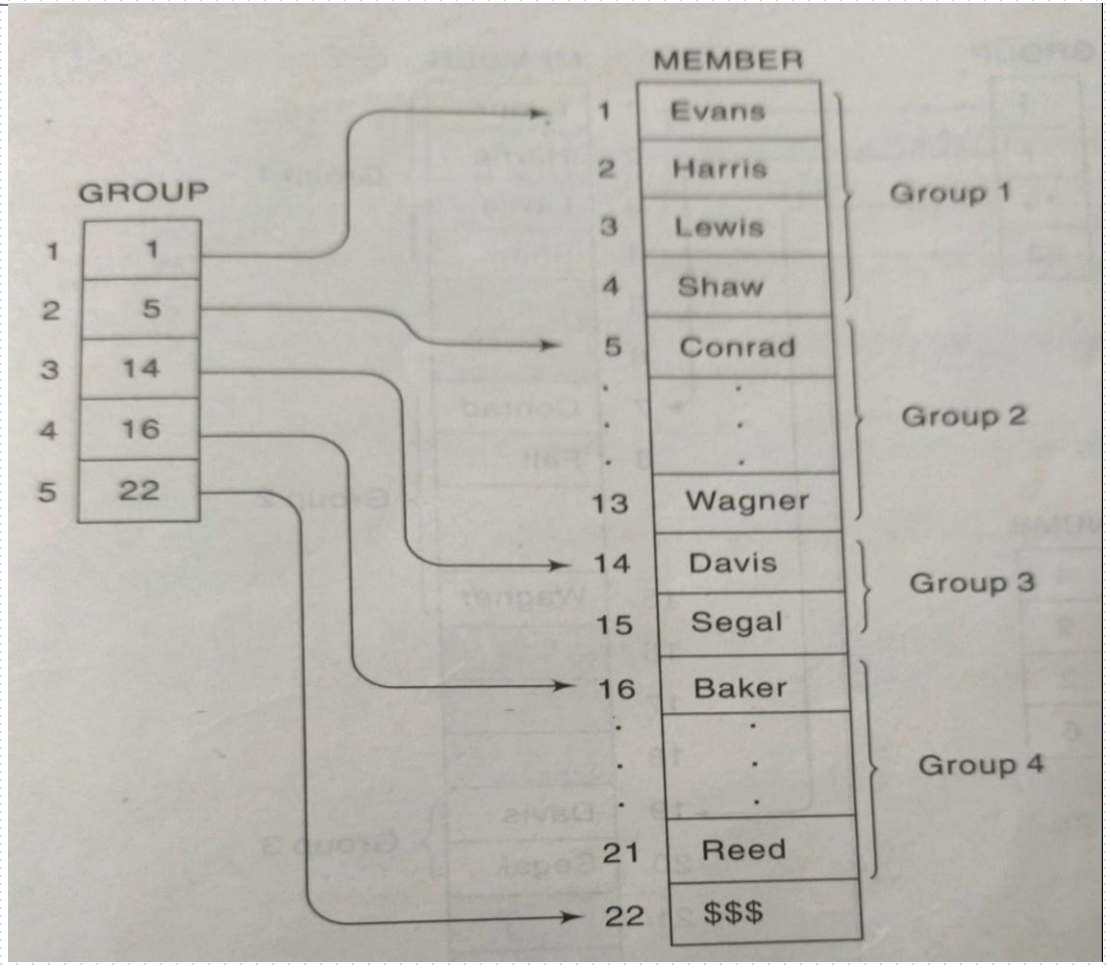
Pointer Array

- Therefore, we need to modify the solution and we have added some sentinel or marker such as 3 dollar signs as shown in figure to indicate end of any group.
- It uses some extra spaces but allow to identify each group individually.
- But main drawback with this solution is that if we need to find last group then also we need to traverse from beginning also.
- Therefore, it increase access (time) complexity.
- So, we use array pointer to overcome this issue.



Pointer Array

- We use pointer array which contains the location of each group.
- Here GROUP is a pointer array which store starting address of each group and MEMBER is the linear array which store all elements from each groups.
- So, to access a group L we can directly access the element starting at GROUP[L] and ending at GROUP[L+1]-1.
- Last entry in group will store location of last empty location in linear array.



Record

- A record is a collection of related data items, each of which is called a field or attribute and a file is a collection of similar records.
- Each data item itself may be a group item composed of sub-items.
- These sub-items are indecomposable are called elementary items or atoms or scalar.
- A record may be differ from linear array in following ways:
  - A record may be a collection of nonhomogeneous data, i.e., data items in a record may have different data types.
  - Data items in a record are indexed by attribute name, so there may not be a natural ordering of its elements

Structures

- Structure is another user defined data type available in C that allows to combine data items of different kinds while arrays allow to define type of variables that can hold several data items of the same kind.
- Structures are used to represent a record.
- To define a structure, you must use the struct statement. The struct statement defines a new data type, with more than one member.
- The format of the struct statement is as follows −

Syntax:

```
struct [structure tag]
{
   member definition;
   member definition;
   ...
   member definition;
}
[one or more structure variables];
```

- Example:
- struct Books
- {
- char title[50];
- char author[50];
- char subject[100];
- int book_id;
- } book;

Linked Lists

- A linked list or one-way list is a sequence of data structures, which are connected together via links.
- Linked lists is a linear collection of data elements, called nodes, where linear order is given by means of pointers.
- Linked List is a sequence of links which contains items.
- Each link contains a connection to another link.
- Linked list is the second most-used data structure after array.

Thank You!