Module 1

CHAPTER 1 Introduction to Data Structures

University Prescribed Syllabus

Introduction to Data Structures, Concept of ADT, Types of Data Structures-Linear and Nonlinear, Operations on Data Structures.

Syllabus Topic : Introduction to Data Structures

1.1 INTRODUCTION TO DATA STRUCTURES

- In the modern era Data and its information is considered as very important part related to any organization.
- Data is nothing but the collection of facts and figures or data is value or group of values which is in a particular format.
- While developing different types of applications, one has to store such data in a standard format. Only the storage of data is not sufficient, later on we have to perform various types of operations on such like insertion, deletion, modifications etc.
- Hence the data must be stored in a systematic format so that one can easily perform different operations on it.
- All the programming languages provide a set of built in data types to store the data such as int, float, char etc.
- In C programming we have seen that to store data various features are provided such as variable, array, structure, union etc.
- As the modern day programming problems are complex and large, all these features are not sufficient to store the huge data.

1.1.1 Data Structures

UQ. 1.1.1 *What is data structure ?*

MU- May 15, 2 Marks

UQ. 1.1.2 *Define Data Structure.*

MU - Dec. 19, 1 Marks

Definition : Data Structure is a way of collecting as well as organizing data in such a way that various operations can be performed on it in an effective way.

A data structure is a logical model of a particular organization of data.

- Data Structure is regarding the representation of data elements in terms of specific relationship, for the purpose of better organization and storage. For example, consider we have data regarding cricket player's name "Rohit" and age 30. Here the data "Rohit" is of **String** type and 30 is of **integer** type data.
- This data can be organized as a record like **Player** record. Now records of all the players can be collected and stored in a file or database as a data structure. For example, "Dhoni" 36, "Yuvraj" 36, "Hardik" 24.
- In simple language, we can say that Data Structures are structures which are programmed to store sequential data, so that it will be easy to perform various operations on it.
- The knowledge of data is represented by it which is to be organized in memory.
- The design and implementation of Data Structures is in such a manner that the complexity gets reduced and efficiency gets increased.

1.2 NEED OF DATA STRUCTURES

GQ. 1.2.1 *Why do we need data structure ?*

(2 Marks)

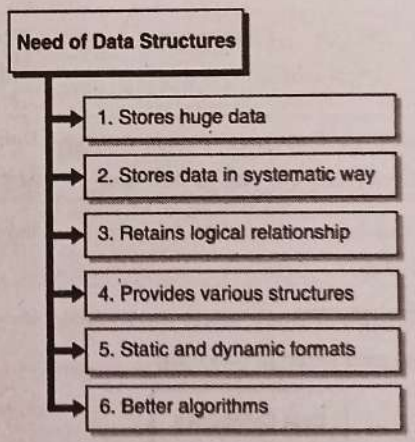


Fig. 1.2.1 : Need of Data Structures

1. Stores huge data

As we have seen that modern day computing problems are complex and large. The basic data elements provided by programming languages like variable, array, structures are not sufficient to store such huge data. Data structures provide a way to store such huge data.

2. Stores data in systematic way

After storing data, variety of operations has to be performed on data of various types like numeric, string, date etc. Data structures help to store the data in systematic way to ease the operations. It makes easy to store and manipulate data.

3. Retains logical relationship

Data structures help to retain the logical relationship between the data elements.

4. Provides various structures

Data structures provide various structures such as stack, queue, linked list etc. to store the data as per the need of application.

5. Static and dynamic formats

Data structures provide static as well as dynamic formats to store the data.

6. Better algorithms

Data structures provide better algorithms to apply on organized data to improve efficiency of program.

1.3 ELEMENTARY DATA STRUCTURE ORGANIZATION

GQ. 1.3.1 *Explain Elementary Data Structure Organization.* **(2 Marks)**

Definition : Data represents a single value or a group of multiple values assigned to entities.

- Data can be considered as raw, unprocessed recorded information.

Data Categories

Data is categorized in two forms :

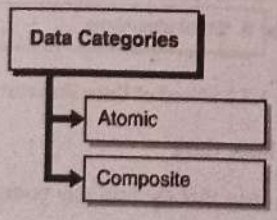


Fig. 1.3.1 : Data Categories

1. Atomic Data : Atomic data is a single, elementary piece of data. This type of data is difficult to further decompose. For example number 5 or name 'Kunal'.

2. Composite Data : Composite data is group of more than one atomic data. For example Full name consists of name and surname. Composite data can be further decomposed into multiple atomic data.

- **Data item :** Data item refers a single or group of values. It is either value of a variable or constant. E.g. rollno of a student.
- **Entity :** An entity is nothing but anything which has some properties to which values are assigned.
- **Information :** Information is the processed or meaningful data. The information is used to take some action.
- **Variable :** The meaningful name given to the memory location where the value is stored is known as the variable. The name given to the variable is an **identifier**. A variable is a program entity which is used to hold the value. One variable can store only one value at a time. The value stored in variable can be changeable during program execution.

1.4 DATA TYPE

GQ. 1.4.1 *Give the classification of data types.* **(3 Marks)**

- A program may use different types of data for example, digit, character, real numbers, etc.
- **Data types** are used to specify the compiler which types of data the program manipulates.
- C language has some predefined set of data types to handle various kinds of data that we use in our program. These data types have different storage capacities.

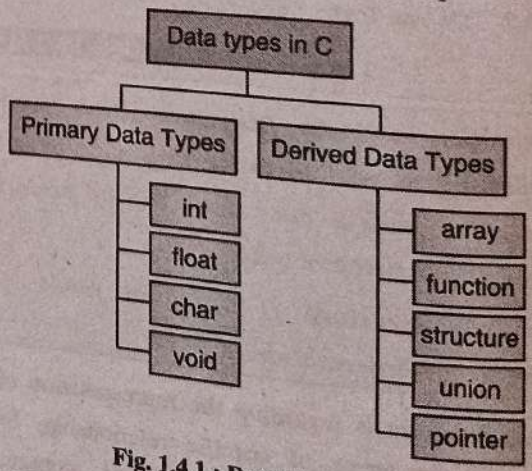


Fig. 1.4.1 : Data Types in C

I. Primary DataTtypes / Primitive Data Types

- These are basic data types.
- **Example :** int, char, float, void are the primary data types.

II. Derived DataTtypes

- Derived data types are derived from primary data types.
- **Example :** Array functions, structure, union and pointers are the derived data types.

1.4.1 Primary Data Types

GQ. 1.4.2 *State various primary data types along with their memory sizes.* **(3 Marks)**

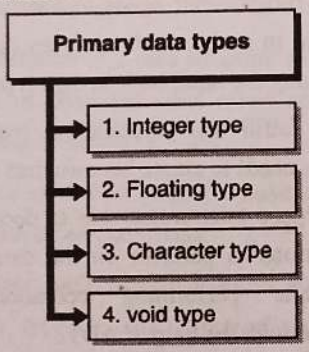


Fig. 1.4.2 : Primary Data Types

1. Integer type

Integers are used to store whole numbers like 1, 0, 15, 3049, etc.

Table 1.4.1 : Size and range of Integer type on 16-bit machine

Type	Memory Size (Bytes)	Range
int or signed int	2	– 32, 768 to 32767
unsigned int	2	0 to 65535
short int or signed short int	1	– 128 to 127
unsigned short int	1	0 to 255
long int or signed long int	4	– 2, 147, 483, 648 to 2, 147, 483, 647
unsigned long int	4	0 to 4, 294, 967, 295

2. Floating type

Floating types are used to store real numbers.

Table 1.4.2 : Size and range of Floating type on 16-bit machine

Type	Memory Size (Bytes)	Range
Float	4	3.4E-38 to 3.4E+38
Double	8	1.7E-308 to 1.7E+308
long double	10	3.4E-4932 to 1.1E+4932

3. Character type

Character types are used to store characters value.

Table 1.4.3 : Size and range of character type on 16-bit machine

Type	Memory Size (Bytes)	Range
char or signed char	1	-128 to 127
unsigned char	1	0 to 255

4. void type

void type means no value. This is usually used to specify the type of functions.

1.4.2 Derived Data Types

GQ. 1.4.3 *Explain the term : Derived data type.* **(3 Marks)**

Following elements comes under derived category :

a. **Array :** Array is a group of elements having similar data types.

b. **Function :** Function is a block of statements related to such a task which we want to execute repeatedly in our program.

c. **Structure :** Structure is a group of elements with different data types.

d. **Union :** Union is a group of elements with different data types. Unlike structure, we can store value in single union member at a time.

e. **Pointer** : Pointer is a variable which can store address of another variable.

– **Record** : Group of related data items is known as record.

– **Program** : Program is a set of instructions which can be interpreted and executed by the computer.

GQ. 1.4.4 *Write a 'C' program to accept 'int' and 'char' type of data and display the same.*

Program

```
# include <stdio.h>
# include <conio.h>
    void main ()
        {
            int no;
            char ch;

            printf("Enter a character : ");
            scanf("%c",&ch);

            printf("Enter an integer : ");
            scanf("%d",&no);

            printf("\n Integer is %d and character is
            %c",no,ch);

            getch();
        }
```

Output

```
C:\d1.exe
Enter a character : K
Enter an integer : 5

Integer is 5 and character is K
```

Syllabus Topic : Concept of ADT

1.5 CONCEPT OF ADT

UQ. 1.5.1 *Define ADT with an example.*

MU - Dec. 13, May 16, 3 Marks.

❑ *Definition : In computer science, an Abstract Data Type (ADT) is a mathematical model for data types, where a data type is defined by its behavior (semantics) from the point of view of a user of the data, specifically in terms of possible values, possible operations on data of this type, and the behavior of these operations.*

– Nowadays, programmer needs to handle some special kind of data. In such case it becomes difficult to store such data using the basic built in data type.

– Hence to fulfill the application requirement, the programmer needs to create his own data types.

– For this purpose programmer has to decide the type of data, operations to perform on the data and rules to follow while performing operations. This is implemented with the help of ADT.

– ADT refers to defining our own data types.

– ADT is useful tool which helps to specify logical properties of data type without the need of going into details.

– Simply we can say that Abstract Data type is collection of values and operations to be performed on those values. A mathematical construct is formed by this collection. The ADT refers to this mathematical construct.

– The definition of ADT can be written in another way also :

☞ **Another Definition of Abstract Data Type (ADT)**

ADT is set of D, F and A

where :

D (Domains) – Data objects

F (Functions) – Set of operations which can be carried out on data objects

A (Axioms) – Properties and rules of the operations.

1.5.1 Implementation of Data Structure

GQ. 1.5.2 *Write note on implementation of Data Structure.* **(2 Marks)**

- Data structures are generally based on the ability of a computer to fetch and store data at any place in its memory.
- Thus, the array and record data structures are based on computing the addresses of data items with arithmetic operations; while the linked data structures are based on storing addresses of data items within the structure itself.
- Many data structures use both principles, sometimes combined in non-trivial ways.
- The implementation of a data structure usually requires writing a set of procedures that create and manipulate instances of that structure.
- The efficiency of a data structure cannot be analyzed separately from those operations.
- This observation motivates the theoretical concept of an abstract data type, a data structure that is defined indirectly by the operations that may be performed on it, and the mathematical properties of those operations (including their space and time cost).

Syllabus Topic : Types of Data Structures

1.6 TYPES OF DATA STRUCTURES

UQ. 1.6.1 *Explain different types of data structures with example.*

MU - May 17, May 18, 5 Marks

In computer science, Data Structure is classified into two categories :

1. Primitive Data Structure
2. Non Primitive Data Structure

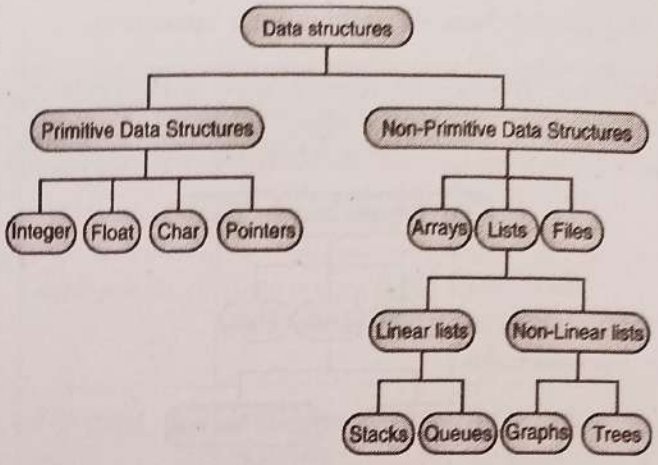


Fig. 1.6.1 : Classification of Data Structures

1.6.1 Primitive Data Structure

GQ. 1.6.2 *Define the term : Primitive data structure.* **(1 Mark)**

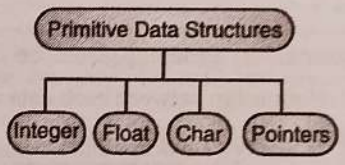


Fig. 1.6.2 : Primitive Data Structures

❑ *Definition : Primitive Data Structures are the basic data structures that directly operate upon the machine instructions. They have different representations on different computers.*

- These are the basic data types provided by various programming languages.
- **Examples :** Integers, Floating point numbers, Character constants, String constants and Pointers come under this category.

1.6.2 Non - Primitive Data Structures

GQ. 1.6.3 Define the term : Non primitive data structures. (1 Mark)

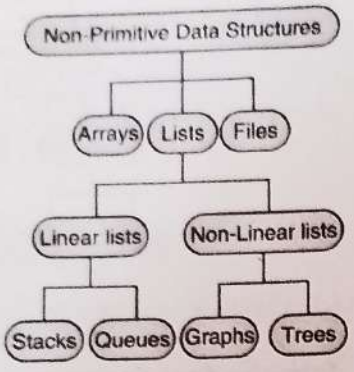


Fig. 1.6.3 : Non-Primitive Data Structures

- **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.
- **Examples :** Arrays, Lists and Files come under this category.

1.7 CLASSIFICATION OF LIST

The Lists are further divided into linear and non-linear data structures :

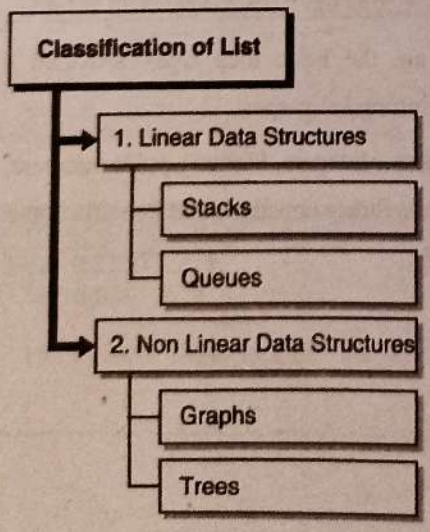


Fig. 1.7.1 : Classification of List

1.7.1 Linear Data Structures

UQ. 1.7.1 Explain the concept of linear list with example.
MU - Dec. 13, May 15, Dec. 16, Dec. 17, 3 Marks

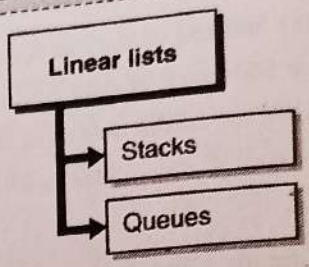


Fig. 1.7.2 : Linear Data Structures

❑ *Definition : The data structure where data items are organized sequentially or linearly one after another is called linear data structure.*

- Data elements in a liner data structure are traversed one after the other and only one element can be directly reached while traversing. All the data items in linear data structure can be traversed in single run.
- These kinds of data structures are very easy to implement because memory of computer also has been organized in linear fashion.

☞ **Examples of linear data structures are**

Stack and Queue

1. **Stack :** Stack is a data structure in which addition and deletion of element is allowed at the same end called as TOP of the stack. A Stack is a LIFO (Last In First Out) data structure where element that added last will be retrieved first.
2. **Queue :** A **Queue** is a data structure in which addition of element is allowed at one end called as REAR and deletion is allowed at another end called as FRONT. A Queue is a FIFO (First In First Out) data structure where element that added first will be retrieved first.

Syllabus Topic : Non Linear Data Structures

1.7.2 Non Linear Data Structures

UQ. 1.7.2 *Enlist non-linear data structures along with example.*

MU - Dec. 13, May 15, Dec. 16, Dec. 17, 3 Marks

UQ. 1.7.3 *Explain Linear and Non-Linear data structures.* MU - May 19, 5 Marks

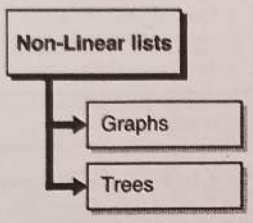


Fig. 1.7.3 : Non Linear Data Structures

Definition : *The data structure in which the data items are not organized sequentially or in linear fashion is called as* **non linear data structure.**

- In other words, a data element of the non linear data structure could be connected to more than one element to reflect a special relationship among them.
- All the data elements in non linear data structure cannot be traversed in single run.

Examples of non linear data structures Trees and Graphs

1. **Tree :** A Tree is collection of nodes where these nodes are arranged hierarchically and form a parent child relationship.
2. **Graph :** A Graph is a collection of a finite number of vertices and edges which connect these vertices. Edges represent relationships among vertices that stores data elements.

1.7.3 Difference between Linear and Non-Linear Data Structure

UQ. 1.7.4 *Differentiate linear and non-linear data structures with example.*

MU - Dec. 19, 4 Marks

Parameter	Linear Data Structure	Non-Linear Data Structure
Relation	Every item is related to its previous and next item.	Every item is attached with many other items.
Arrangement	Data is arranged in linear sequence.	Data is not arranged in linear sequence.
Traversing	Data items can be traversed in a single run.	Data cannot be traversed in a single run.
Implementation	Implementation is Easy.	Implementation is difficult.
Examples	Array, Stack, Queue, Linked List.	Tree, Graph.

Syllabus Topic : Operations on Data Structures

1.8 OPERATIONS ON DATA STRUCTURES

UQ. 1.8.1 *What are various operations possible on data structures ?.* MU - Dec.18, 5 Marks

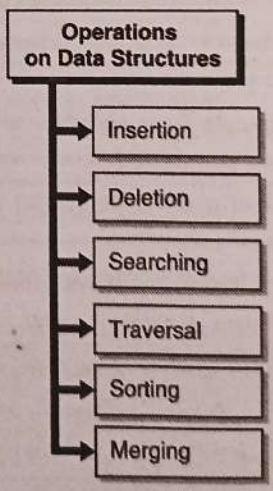


Fig. 1.8.1 : Operations on Data Structures

Following are the basic operations which can be performed on data structures :

Table 1.8.1 : Operations on Data Structures

Insertion	Insertion refers to addition of a new data element in a data structure.
Deletion	Deletion refers to removal of a data element from a data structure.
Searching	Searching refers to finding the specified data element in a data structure.
Traversal	Traversal of a data structure means visiting every element of data structures at least once.
Sorting	Arranging data elements of a data structure in a standard predetermined sequence is called sorting.
Merging	Combining elements of more than one data structures in third data structure is called as merging.

1.9 ARRAY

- Let's consider a situation where a program needs number of similar types of data elements to be stored. A variable is used to store one data element at a time. So to store number of data elements; number of variables are required. This solution has many problems :
  - As the list of variables increases the length of program also increases.
  - To manipulate those variables several assignment statements also needed.
  - Programmer needs to remember names of all the variables.
- An alternate solution to above situation is to store similar type of data elements is create an array.

> ❑ *Definition : Array is a collection of elements of similar data types referred by the same variable name. Contiguous memory block is allocated to all these array elements.*

- Array elements are of same data type i.e. once array is declared as integer, then all values which are present in an array will be of type integer only.

1.10 REPRESENTATION OF ARRAY

GQ. 1.10.1 *Write note on representation of array.* **(4 Marks)**

- Arrays are categorized according to the dimensions used to define it. Here dimension indicates the number of rows and columns used to set size of array.
- Array has categorized into following types.

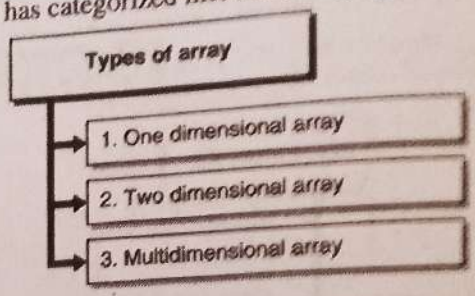


Fig. 1.10.1 : Types of array

1.11 ONE DIMENSIONAL ARRAY

GQ. 1.11.1 *How to representation 1D array.* **(2 Marks)**

> ❑ *Definition : An array with single subscript is called as one dimensional array.*

Declaration of one dimensional array Syntax

- Syntax for declaring one dimensional array is given below.

```
data_type array_name [ size ];
```

Example

- Declare an array to store percentage of 10 students

```
float percentage[10];
```

- Memory arrangement after declaring one dimensional array shown in Fig. 1.11.1.

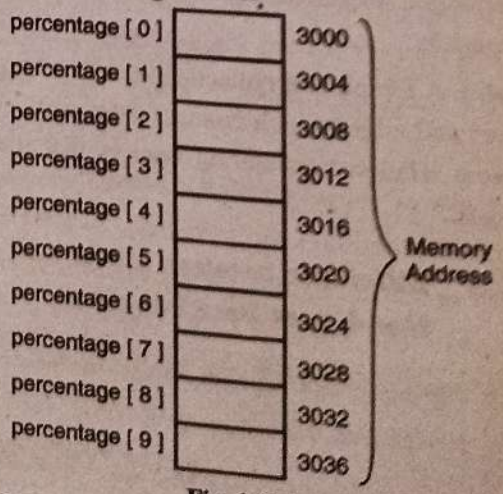


Fig. 1.11.1

Address calculation of 1D array

Address of arr[i] = base address + i * element_size

Address of percentages[3] = 3000 + 3 * sizeof(float)

= 3000 + 3 * 4 = 3012

GQ. 1.11.2 *Write a program to declare array to store percentage of 10 students. Accept percentage from user and print on the screen.* **(5 Marks)**

Program

```
#include <conio.h>
#include <stdio.h>
int main()
{
float percentage[10];
int i;
for ( i = 0 ; i <10 ; i++ )
{
printf ( "\nEnter Percentage of student %d :\t ", i+1 ) ;
scanf ( "%f", &percentage[i] ) ;
}

for ( i = 0 ; i <10 ; i++ )
{
printf ("\n Percentage of student %d :\t %4.2f", i+1,
                                        percentage[i] ) ;
}
return 1;
}
```

Array declaration

Accepting array elements from user.

Output

```
Enter Percentage of student 1 :  83.67
Enter Percentage of student 2 :  89.45
Enter Percentage of student 3 :  76.98
Enter Percentage of student 4 :  34.65
Enter Percentage of student 5 :  65.34
Enter Percentage of student 6 :  67.34
Enter Percentage of student 7 :  98.87
Enter Percentage of student 8 :  56.89
Enter Percentage of student 9 :  97.67
Enter Percentage of student 10 :         67.87

 Percentage of student 1 :       83.67
 Percentage of student 2 :       89.45
 Percentage of student 3 :       76.98
 Percentage of student 4 :       34.65
 Percentage of student 5 :       65.34
 Percentage of student 6 :       67.34
 Percentage of student 7 :       98.87
 Percentage of student 8 :       56.89
 Percentage of student 9 :       97.67
 Percentage of student 10 :      67.87
Process exited after 57.4 seconds with return value 1
Press any key to continue . . .
```

1.12 OPERATIONS ON ARRAY

GQ. 1.12.1 *List the operations on array.* **(2 Marks)**

- Apart from inputting and outputting elements into and from array, many operations can be performed on array shown in Fig. 1.12.1.

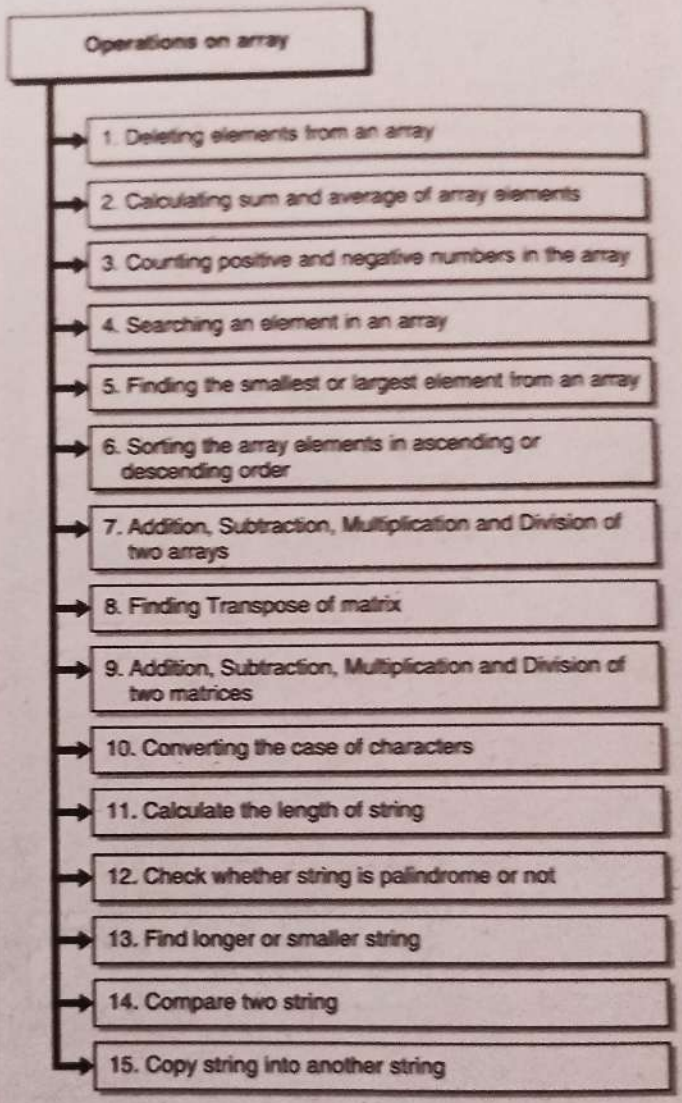


Fig. 1.12.1 : Operations on array

1. Deleting elements from array

- The deletion of elements of array doesn't affect array size.
- Following program shows the example of deleting the element at specified location in array. And also checks whether the given position is less than the array size or not.

GQ. 1.12.2 *Write a program to accept the position from user and delete the element at that position from array.* **(5 Marks)**

Program

```
#include <stdio.h>
int main()
{
int array[20], pos, i, n;
printf("Enter number of elements want to insert in the array\n");

scanf("%d", &n);
printf("Enter %d elements\n", n);
for ( i = 0 ; i < n ; i++ )
{
    scanf("%d", &array[i]);
}
printf("Enter the position of element to be deleted\n");

scanf("%d", &pos);

if ( pos >= n+1 )
{
    printf("Deletion not possible.\n");
}
else
{
    for ( i = pos - 1 ; i < n - 1 ; i++ )
    {
        array[i] = array[i+1];
    }
    printf("After deleting element at %d location the array is\n",pos);
    for( i = 0 ; i < n - 1 ; i++ )
    {
        printf("%d\n", array[i]);
    }
}
return 0;
}
```

Accepts elements for array from user

Accepts the position from user to for deletion

Prints if the position is greater than the array size

Shifts the elements to left

Output

```
D:\teju\c\deletearr.exe
Enter number of elements want to insert  in the array
6
Enter 6 elements
1
2
3
4
5
6
Enter the position of element to be deleted
5
After deleting element at 5 location the array is
1
2
3
4
6

--------------------------------
Process exited after 14.51 seconds with return value 0
Press any key to continue . . .
```

3. Counting positive and negative numbers in the array

GQ. 1.12.3 *Write a program to find out number of positive, negative and zero elements from an array.* **(5 Marks)**

Program

```
# include<stdio.h>
int main()
{
    int numbers[5];

    int i=0,cnt_pos=0,cnt_neg=0,cnt_zero=0;

for(i=0;i<5;i++)
{
printf("\n Enter a number:");
scanf("%d", &numbers[i]);
```

cnt_pos, cnt_neg, cnt_zero are used to count number of negative values, zero values, and positive values in the array.

Condition to check for negative number

```
if (numbers[i]<0)
{
    cnt_neg++;
}
if (numbers[i]==0)
{
    cnt_zero++;
}
if (numbers[i]>0)
{
    cnt_pos++;
}
}

printf("\n Number of  positive numbers= %d",cnt_pos);
printf("\n Number of zeros=%d", cnt_zero);
printf("\n Number of negative numbers= %d", cnt_neg);
return 1;
}
```

Condition to check for zero

Condition to check for positive number

Output

```
Enter number:5
Enter number:-1
Enter number:0
Enter number:6
Enter number:-6
Number of negative numbers= 2
Number of zeros=1
Number of  positive numbers= 2
```

4. Searching an element in an array

GQ. 1.12.4 Write a program to search a particular number in an array. If the number is present print "Number found" else print "Number not found". **(5 Marks)**

Program

```
#include<stdio.h>
#include<conio.h>
int main()
{
int array[6]={20,80,60,40,10,45};
int i, num, flag=1;

printf("array elements are:");
for(i=0;i<6;i++)
{
    printf("\t %d",array[i]);
}
printf("\n enter
an element to search:\t");
scanf("%d", &num);

for(i=0;i<6;i++)
{
    if(array[i]==num)
    {
        flag=0;
    }
}
if(flag==0)
{
printf("\n Number found ");
}
else
{
printf("\n Number not found ");
}
}
```

The variable **flag** is used here to determine the search is successful or not initially it is set to 1

Accepts a number to be searched in the array.

6 times the inner if condition gets checked.

If the number is found in the array then **flag** is set to **0.**

Prints if value of **flag** is **0.**

Prints if value of **flag** is **1**

Output

```
array elements are:     20    80    60    40    10    45
enter an element to search:   40

Number found
```

1.13 TWO DIMENSIONAL ARRAY

UQ. 1.13.1 Write short note on following : Storage representation of 2 Dimensional array.
MU - Dec. 11, 4 Marks

Definition : An array with two subscripts is called as **two dimensional array**. 2D arrays are mostly used to perform matrix operations.

Declaration of two dimensional array

Syntax

- Syntax for declaring two dimensional array is given below.

```
data_type array_name [ row size ][ column size ];
```

Example

- Declare an array to store a 2 × 2 matrix

```
int matrix[2][2];
```

- The 2 × 2 matrix stores 2*2=4 values. In our example data type of array is integer means the four values are of int type.
- In user's view the matrix looks like:

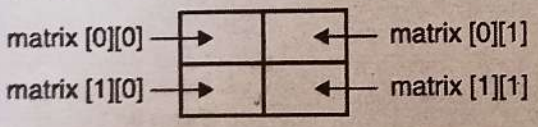


Fig. 1.13.1

- But in memory it will form different structure.
- Memory arrangement after declaring two dimensional array shown in Fig. 1.13.2.

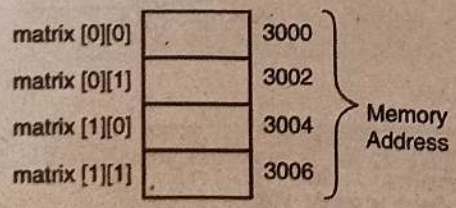


Fig. 1.13.2

1.14 MULTI DIMENSIONAL ARRAY

UQ. 1.14.1 Explain Multidimensional Array. How it is stored in memory?
MU - Dec. 13, 3 Marks

Definition : An array with more than two subscripts is called as **multidimensional array.**

Declaration of multidimensional array

- For simplicity we will study 3D array which has 3 subscripts.

Syntax

- Syntax for declaring multidimensional array is given below:

```
data_type array_name [size1][size2][size3];
```

Example

- Declare an multidimensional array

```
int array[2][2][2];
```

- This array stores 2*2*2 =8 values. In this example data type of array is integer means the values should be of int type.
- In user's view the multidimensional array looks like:

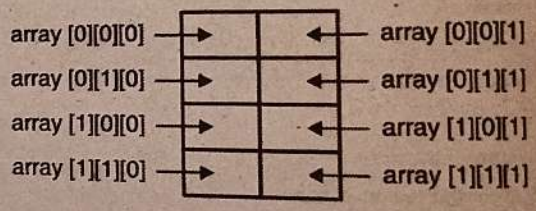


Fig. 1.14.1

- But in memory it will form different structure.
- Memory arrangement after declaring multidimensional array shown in Fig. 1.14.2.

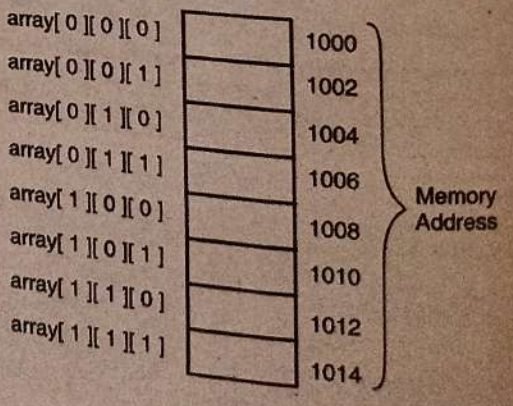
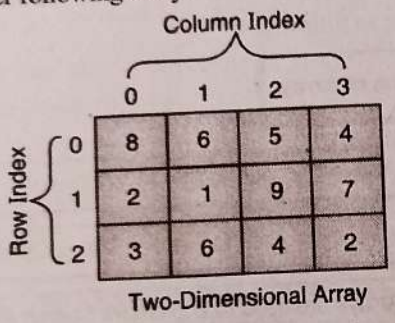


Fig. 1.14.2

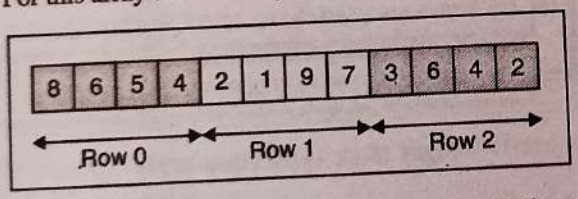
1.14.1 Row- and Column-major Order

- In computing, row-major order and column-major order describe methods for arranging multidimensional arrays in linear storage such as random access memory.
- In row-major order, consecutive elements of the rows of the array are contiguous in memory; in column-major order, consecutive elements of the columns are contiguous.
- In matrix notation, the first/left index indicates the row, and the second/right index indicates the column, e.g., a1, 2 is in the first row and in the second column.
- For row-major order, all elements of the first row come before all elements of the second row, etc.
- Consider following array.

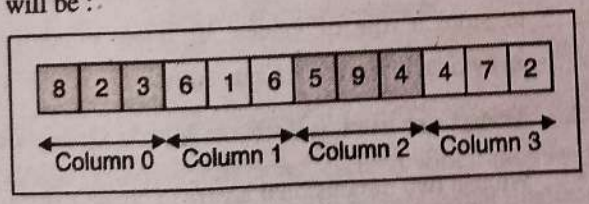


Two-Dimensional Array

- For this array the row-major representation will be :



- For column-major order, all elements of the first column come before all elements of the second column, etc.
- For the given array the column-major representation will be :



Address Calculation of 2D array

A. Row major representation

- Address of [i][j] = base address + i *c * element_size + j * element_size

  = base address + (i*c*+j) * element_size

Example

- Adresss of arr[1][2] (Consider base address as 1000)

  Arr[1][2 = 1000 + (1*3+2) * sizeof(int)

  = 1000 + 10

  = 1010

B. Column major representation

Address of [i][j] = base address + j *r * element_size + i * element_size

= base address + (j*r*+i) * element_size

Example

- Address of arr[1][2] (Consider base address as 1000)

  Arr[1][2] = 1000 + (2*4+1) * sizeof(int)

  = 1000 + 18

  = 1018

Advantages of linear representation

1. This representation is very easy to understand.
2. Programming is very easy.
3. It is very easy to move from one element to other.

Disadvantages of linear representation

1. Lot of memory area wasted.
2. Insertion and deletion of elements needs lot of data movement.
3. Execution time high.

UQ. 1.14.2 Given a two dimensional array Z1 (2 : 9, 9 : 18) stored in column major order with base address 100 and size of each element is 4 bytes. Find address of the element Z1 (4, 12).

MU - May 12, 4 Marks

Array : Z1(2:9, 9:18)

Here the array form is **A[Lr- - - - - Ur, Lc- - - - - Uc].**

In this case number of rows and columns are calculated using the following methods:

Number of rows (**M**) will be calculated as = **(Ur – Lr) + 1**

= (9-2) +1

= 8

Number of columns(**N**) will be calculated as = **(Uc – Lc) + 1**

= (18-9) + 1

= 10

Address of [i][j] = base address + j *r * element_size + i * element_size

= base address + (j*r*+i) * element_size

Address of Z1(4,12)

= 100 + (12*8+4) * 4

= 100 + 400 = 500

GQ. 1.14.3 *Write a program to accept values for 2 x 2 matrix and print them.* **(5 Marks)**

Program

```
# include<stdio.h>
#include<conio.h>
int main()
{
int matrix[2][2],i,j;
printf ( "\nEnter data for 2D matrix\n" );
for ( i = 0 ; i <2 ; i++ )
{
    for ( j = 0 ; j <2 ; j++ )
    {
     printf ( "\n matrix [%d] [%d] :\t ", i, j ) ;
     scanf ( "%d", &matrix[i][j] ) ;
    }
}
printf("\nThe matrix is:\n");
for ( i = 0 ; i <2 ; i++ )
{
    for ( j = 0 ; j <2 ; j++ )
    {
     printf ( " %d ", matrix[i][j] ) ;
    }
    printf("\n");
}
return 1;
}
```

Output

```
Enter data for 2D matrix
 matrix [0] [0] :        12
 matrix [0] [1] :        1
 matrix [1] [0] :        32
 matrix [1] [1] :        10
The matrix is:
 12  1
 32  10
--------------------------------
Process exited after 17.17 seconds with return value 1
Press any key to continue . . .
```

1.15 APPLICATIONS OF ARRAY

GQ. 1.15.1 *Write applications of array.* **(4 Marks)**

– Arrays are used in wide range of applications. Few of them are as follows :

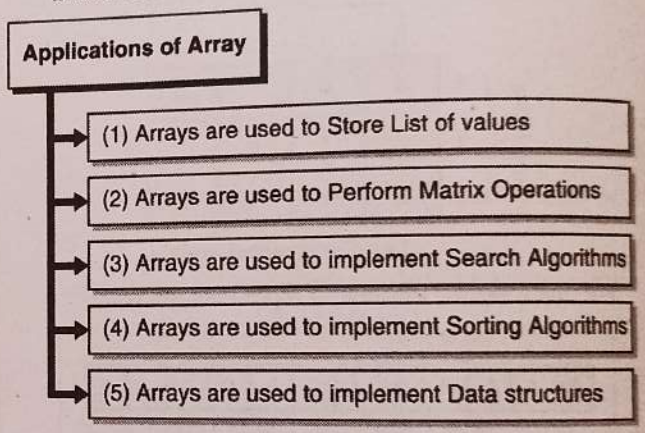


Fig. 1.15.1 : Applications array

1. Arrays are used to Store List of values

– Single dimensional arrays are used to store list of values of same data-type.

– In other words, single dimensional arrays are used to store a row of values. In single dimensional array data is stored in linear form.

2. Arrays are used to Perform Matrix Operations

– We use two dimensional arrays to create matrix.

– We can perform various operations on matrices using two dimensional arrays.

3. Arrays are used to implement Search Algorithms

- We use single dimensional arrays to implement search algorithms like :
  (i) Linear Search
  (ii) Binary Search

4. Arrays are used to implement Sorting Algorithms

- We use single dimensional arrays to implement sorting algorithms like :
  (i) Insertion Sort (ii) Bubble Sort
  (iii) Selection Sort (iv) Quick Sort
  (v) Merge Sort, etc.,

5. Arrays are used to implement Data structures

- We use single dimensional arrays to implement data structures like :
  (i) Stack Using Arrays (ii) Queue Using Arrays

Arrays are also used to implement CPU Scheduling Algorithms.

1.16 SPARSE MATRIX AND ITS REPRESENTATION

GQ. 1.16.1 *Explain sparse matrix with its representation.* **(4 Marks)**

- In computer programming, a matrix can be defined with a 2-dimensional array.
- Any array with 'm' columns and 'n' rows represents a mXn matrix.
- There may be a situation in which a matrix contains more number of ZERO values than NON-ZERO values. Such matrix is known as **sparse matrix**.

> *Definition : Sparse matrix is a matrix which contains very few non-zero elements.*

- When a sparse matrix is represented with 2-dimensional array, we waste lot of space to represent that matrix.
- For example, consider a matrix of size 100 X 100 containing only 10 non-zero elements.
- In this matrix, only 10 spaces are filled with non-zero values and remaining spaces of matrix are filled with zero.
- That means, totally we allocate 100 X 100 X 2 = 20000 bytes of space to store this integer matrix.
- And to access these 10 non-zero elements we have to make scanning for 10000 times.

Array Representation of Sparse Matrix

- 2D array is used to represent a sparse matrix in which there are three rows named as
- **Row :** Index of row, where non-zero element is located
- **Column :** Index of column, where non-zero element is located.
- **Value :** Value of the non zero element located at index – (row,column)

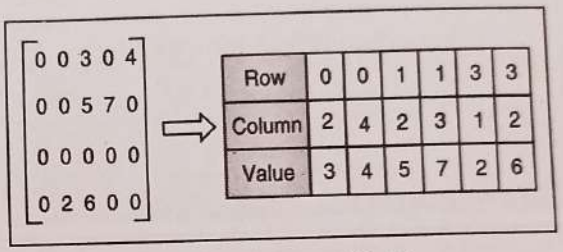


Fig. 1.16.1 : Sparse Matrix

Program for Sparse Matrix Representation using Array.

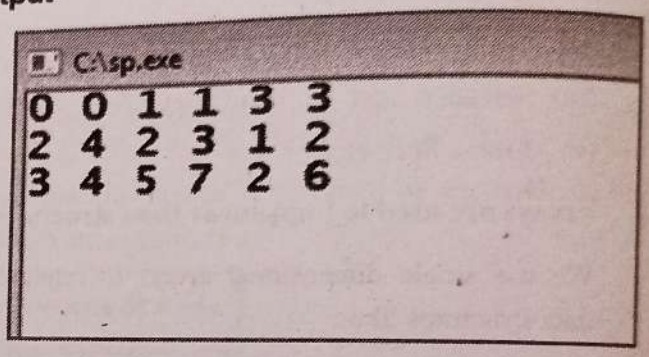
```
#include<stdio.h>

int main()
{
  int i,j,k,size = 0;

  int sparseMatrix[4][5] =
  {
    {0,0,3,0,4},
    {0,0,5,7,0},
    {0,0,0,0,0},
    {0,2,6,0,0}
  };

  for (i = 0; i < 4; i++)
    for (j = 0; j < 5; j++)
      if (sparseMatrix[i][j] != 0)
        size++;
```

Assume 4x5 sparse

```
/* number of columns in compactMatrix (size) must be
equal to number of non - zero elements in  sparseMatrix */
 int compactMatrix[3][size];

  k = 0;
for (i = 0; i < 4; i++)
  for (j = 0; j < 5; j++)
    if (sparseMatrix[i][j] != 0)
    {
      compactMatrix[0][k] = i;
      compactMatrix[1][k] = j;
      compactMatrix[2][k] = sparseMatrix[i][j];
      k++;
    }

for (i=0; i<3; i++)
  {
    for (j=0; j<size; j++)
      printf("%d ", compactMatrix[i][j]);

    printf("\n");
  }
getch();
}
```

Making of new matrix

Output



..Chapter Ends

❑❑❑