LEARNING OUTCOMES

On successful completion of this course,student should be able to

1. demonstrate advantages and disadvantages of specific algorithms and data structures,
2. select basic data structures and algorithms for autonomous realization of simple programs or program parts
3. determine and demonstrate bugs in program, recognise needed basic operations with data structures
4. formulate new solutions for programing problems or improve existing code using learned algorithms and data structures,
5. Students develop knowledge of basic data structures for storage and retrieval of ordered or unordered data. Data structures include: arrays, linked lists, binary trees, heaps, and hash tables.
6. Students develop knowledge of applications of data structures including the ability to implement algorithms for the creation, insertion, deletion, searching, and sorting of each data structure.

UNIT-1 FUNDAMENTAL NOTATIONS

1.1 Program development life cycle

Program Development Life Cycle (PDLC) is a systematic way of developing quality software. It provides an organized plan for breaking down the task of program development into manageable chunks, each of which must be successfully completed before moving on to the next phase.

- Defining the Problem –
  The first step is to define the problem. In major software projects, this is a job for system analyst, who provides the results of their work to programmers in the form of a program specification. The program specification defines the data used in program, the processing that should take place while finding a solution, the format of the output and the user interface.
- Designing the Program –
  Program design starts by focusing on the main goal that the program is trying to achieve and then breaking the program into manageable components, each of which contributes to this goal. This approach of program design is called top-bottom program design or modular programming. The first step involve identifying main routine, which is

the one of program's major activity. From that point, programmers try to divide the various components of the main routine into smaller parts called modules. For each module, programmer draws a conceptual plan using an appropriate program design tool to visualize how the module will do its assign job.

- Coding the Program –
  Coding the program means translating an algorithm into specific programming language. The technique of programming using only well defined control structures is known as Structured programming. Programmer must follow the language rules, violation of any rule causes error. These errors must be eliminated before going to the next step.
- Testing and Debugging the Program –
  After removal of syntax errors, the program will execute. However, the output of the program may not be correct. This is because of logical error in the program. A logical error is a mistake that the programmer made while designing the solution to a problem. So the programmer must find and correct logical errors by carefully examining the program output using Test data. Syntax error and Logical error are collectively known as Bugs. The process of identifying errors and eliminating them is known as Debugging.
- Documenting the Program –
  After testing, the software project is almost complete. The structure charts, pseudocodes, flowcharts and decision tables developed during the design phase become documentation for others who are associated with the software project. This phase ends by writing a manual that provides an overview of the program's functionality, tutorials for the beginner, in-depth explanations of major program features, reference documentation of all program commands and a thorough description of the error messages generated by the program.
- Deploying and Maintaining the Program –
  In the final phase, the program is deployed (installed) at the user's site. Here also, the program is kept under watch till the user gives a green signal to it. Even after the software is completed, it needs to be maintained and evaluated regularly. In software maintenance, the programming team fixes program errors and updates the software.

1.2 Problem solving Concepts: Top Down and Bottom up Design

problem solving is an efficient and organized method to define the problem which can be a issue or condition that presents ambiguity, complication, complexity and creating a enormous number of probable solutions in numerous condition.

Programming refers to the method of creating a sequence of instructions to enablethe computer to perform a task. It is done by developing logic and then writing instructions in a programming language. A program can be written using various programming practices available. A programming practice refers to the way of writing a program and is used along with coding style guidelines. Some of the commonly used programming practices include top-down programming, bottom-up programming and structured programming,

1.2.1Top Down Programming

Top-down programming focuses on the use of modules. It is therefore also known as modular programming. The program is broken up into small modules so that it is easy to trace a particular segment of code in the software program. The modules at the top level are those that perform general tasks and proceed to other modules to perform a particular task. Each module is based on the functionality of its functions and procedures. In this approach, programming begins from the top level of hierarchy and progresses towards the lower levels. The implementation of modules starts with the main module. After the implementation of the main module, the subordinate modules are implemented and the process follows in this way. In top-down programming, there is a risk of implementing data structures as the modules are dependent on each other and they nave to share one or more functions and procedures. In this way, the functions and procedures are globally visible. In addition to modules, the top-down programming uses sequences and the nested levels of commands.

1.2.2 Bottom-up Programming

Bottom-up programming refers to the style of programming where an application is constructed with the description of modules. The description begins at the bottom of the hierarchy of modules and progresses through higher levels until it reaches the top. Bottom-up programming is just the opposite of top-down programming. Here, the program modules are more general and reusable than top-down programming.

It is easier to construct functions in bottom-up manner. This is because bottom-up programming requires a way of passing complicated arguments between functions.

1.2.3 Structured Programming

Structured programming is concerned with the structures used in a computer program. Generally, structures of computer program comprise decisions, sequences, and loops. The decision structures are used for conditional execution of statements (for example, 'if statement). The sequence structures are used for the sequentially executed statements. The loop structures are used for performing some repetitive tasks in the program.

Structured programming forces a logical structure in the program to be written in an efficient and understandable manner. The purpose of structured programming is to make the software code easy to modify when required. Structured programming focuses on reducing the following statements from the program.

1. 'GOTO' statements.
2. 'Break' or 'Continue' outside the loops.
3. Multiple exit points to a function, procedure, or subroutine. For example, multiple 'Return' statements should not be used.
4. Multiple entry points to a function, procedure, or a subroutine.

Structured programming generally makes use of top-down design because program structure is divided into separate subsections. A defined function or set of similar functions is kept separately. Due to this separation of functions, they are easily loaded in the memory. In addition, these functions can be reused in one or more programs. Each module is tested individually. After testing, they are integrated with other modules to achieve an overall program structure. Note that a key characteristic of a structured statement is the presence of single entry and single exit point. This characteristic implies that during execution, a structured statement starts from one defined point and terminates at another defined point.

1.3 Data Type, Variable and Constants

1.3.1 Data and Data Item

Data are simply collection of facts and figures. 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 group items;those that are not are called elementary items. For example, a student's name may be divided into three sub items – [first name, middle name and last name] but the ID of a student would normally be treated as a single item. In the above example (ID, Age, Gender, First, Middle, Last, Street, Area) are elementary data items, whereas (Name, Address) are group data items.

1.3.2 Data Type

Data type is a classification identifying one of various types of data, such as floating-point, integer, or Boolean, that determines the possible values for that type; the operations that can be done on values of that type; and the way values of that type can be stored. It is of two types: Primitive and non-primitive data type. Primitive data type is the basic data type that is provided by the programming language with built-in support. This data type is native to the language and is supported by machine directly while non-primitive data type is derived from primitive data type. For example- array, structure etc.

1.3.3 Variable

It is a symbolic name given to some known or unknown quantity or information, for the purposeof allowing the name to be used independently of the information it represents. A variable name in computer source code is usually associated with a data storage location and thus also its contents and these may change during the course of program execution.

1.3.4 Record

Collection of related data items is known as record. The elements of records are usually called fields or members. Records are distinguished from arrays by the fact that their number of fields is typically fixed, each field has a name, and that each field may have a different type.

1.3.5 Constant

A constant is a value or an identifier whose value cannot be altered in a program. For example: 1, 2.5, "C programming is easy", etc.

As mentioned, an identifier also can be defined as a constant.

```
const double PI = 3.14
```

Here, `PI` is a constant. Basically what it means is that, `PI` and `3.14` is same for this program.

Below are the different types of constants you can use in C.

1. Integer constants

An integer constant is a numeric constant (associated with number) without any fractional or exponential part. There are three types of integer constants in C programming:

- decimal constant(base 10)
- octal constant(base 8)
- hexadecimal constant(base 16)

For example:

```
Decimal constants: 0, -9, 22 etc

Octal constants: 021, 077, 033 etc

Hexadecimal constants: 0x7f, 0x2a, 0x521 etc
```

In C programming, octal constant starts with a 0 and hexadecimal constant starts with a 0x.

2. Floating-point constants

A floating point constant is a numeric constant that has either a fractional form or an exponent form. For example:

```
-2.0

0.0000234

-0.22E-5
```

Note: E-5 = 10^{-5}

3. Character constants

A character constant is a constant which uses single quotation around characters. Forexample: 'a', 'l', 'm', 'F'

4. Escape Sequences

Sometimes, it is necessary to use characters which cannot be typed or has special meaning in C programming. For example: newline (enter), tab, question mark etc. In order to use these characters, escape sequence is used.

1.6 Pointers

A pointer is a variable whose value is the address of another variable, i.e., direct address of the memory location. Like any variable or constant, you must declare a pointer before using it to store any variable address. The general form of a pointer variable declaration is −

```
type *var-name;
```

Here, type is the pointer's base type; it must be a valid C data type and var-name is the name of the pointer variable. The asterisk * used to declare a pointer is the same asterisk used for multiplication. However, in this statement the asterisk is being used to designate a variable as a pointer. Take a look at some of the valid pointer declarations −

```
int    *ip;    /* pointer to an integer */
double *dp;    /* pointer to a double */
float  *fp;    /* pointer to a float */
char   *ch     /* pointer to a character */
```

The actual data type of the value of all pointers, whether integer, float, character, or otherwise, is the same, a long hexadecimal number that represents a memory address. The only difference between pointers of different data types is the data type of the variable or constant that the pointer points to.

There are a few important operations, like (a) define a pointer variable, (b) assign the address of a variable to a pointer and (c) finally access the value at the address available in the pointer variable. This is done by using unary operator * that returns the value of the variable located at the address specified by its operand. The following example makes use of these operations −

```
#include<stdio.h>

int main (){

intvar=20;/* actual variable declaration */

int*ip;/* pointer variable declaration */

ip=&var;/* store address of var in pointer variable*/

printf("Address of var variable: %x\n",&var);

/* address stored in pointer variable */

printf("Address stored in ip variable: %x\n",ip);

/* access the value using the pointer */

printf("Value of *ip variable: %d\n",*ip);

return0;

}
```

When the above code is compiled and executed, it produces the following result −

```
Address of var variable: bffd8b3c
Address stored in ip variable: bffd8b3c
Value of *ip variable: 20
```

1.7 DATA STRUCTURE

In computer science, a data structure is a particular way of storing and organizing data in acomputer's memory so that it can be used efficiently. Data may be organized in many differentways; the logical or mathematical model of a particular organization of data is called a datastructure. The choice of a particular data model depends on the two considerations first; it mustbe rich enough in structure to mirror the actual relationships of the data in the real world. On theother hand, the structure should be simple enough that one can effectively process the datawhenever necessary.

Need of data structure

- It gives different level of organization data.
- It tells how data can be stored and accessed in its elementary level.
- Provide operation on group of data, such as adding an item, looking up highest priorityitem.
- Provide a means to manage huge amount of data efficiently.

- Provide fast searching and sorting of data.

Selecting a data structure

Selection of suitable data structure involve following steps –

- Analyze the problem to determine the resource constraints a solution must meet.
- Determine basic operation that must be supported. Quantify resource constraint for eachoperation
- Select the data structure that best meets these requirements.

Each data structure has cost and benefits. Rarely is one data structure better than other inall situations. A data structure require :

- Space for each item it stores
- Time to perform each basic operation
- Programming effort.

Each problem has constraints on available time and space. Best data structure for the taskrequires careful analysis of problem characteristics.

1.7.1 Type of data structure

- Static data structure

A data structure whose organizational characteristics are invariant throughout its lifetime. Suchstructures are well supported by high-level languages and familiar examples are arrays andrecords. The prime features of static structures are

(a) None of the structural information need be stored explicitly within the elements – it is oftenheldin a distinct logical/physical header;

(b) The elements of an allocated structure are physically contiguous, held in a single segment ofmemory;

(c) All descriptive information, other than the physical location of the allocated structure, isdetermined by the structure definition;

(d) Relationships between elements do not change during the lifetime of the structure.

- Dynamic data structure

A data structure whose organizational characteristics may change during its lifetime. Theadaptability afforded by such structures, e.g. linked lists, is often at the expense of decreasedefficiency in accessing elements of the structure. Two main features distinguish dynamicstructures from static data structures. Firstly, it is no longer possible to infer all structural information from a header; each data element will have to contain information relating itlogically to other elements of the structure. Secondly, using a single block of contiguous storageis often not appropriate, and hence it is necessary to provide some storage management schemeat run-time.

- Linear Data Structure

A data structure is said to be linear if its elements form any sequence. There are basically twoways of representing such linear structure in memory.

a) One way is to have the linear relationships between the elements represented by means ofsequential memory location. These linear structures are called arrays.

b) The other way is to have the linear relationship between the elements represented by means ofpointers or links. These linear structures are called linked lists.

The common examples of linear data structure are arrays, queues, stacks and linked lists.

- Non-linear Data Structure

This structure is mainly used to represent data containing a hierarchical relationship betweenelements. E.g. graphs, family trees and table of contents.

1.8 BRIEF DESCRIPTION OF DATA STRUCTURES

Array

The simplest type of data structure is a linear (or one dimensional) array. A list of a finitenumber n of similar data referenced respectively by a set of n consecutive numbers, usually 1, 2,3..... n. if we choose the name A for the array, then the elements of A are denoted bysubscript notationA 1, A 2, A 3 ..... A nor by the parenthesis notation

A (1), A (2), A (3) ..... A (n)

or by the bracket notation

A [1], A [2], A [3] .....A [n]

Example:

A linear array A[8] consisting of numbers is pictured in following figure.

Linked List

A linked list or one way list is a linear collection of data elements, called nodes, where the linearorder is given by means of pointers. Each node is divided into two parts:

- The first part contains the information of the element/node
- The second part contains the address of the next node (link /next pointer field) in the list.

There is a special pointer Start/List contains the address of first node in the list. If this specialpointer contains null, means that List is empty.

Tree

Data frequently contain a hierarchical relationship between various elements. The data structurewhich reflects this relationship is called a rooted tree graph or, simply, a tree.

Graph

Data sometimes contains a relationship between pairs of elements which is not necessarilyhierarchical in nature, e.g. an airline flights only between the cities connected by lines. This datastructure is called Graph.

Queue

A queue, also called FIFO system, is a linear list in which deletions can take place only at oneend of the list, the Font of the list and insertion can take place only at the other end Rear.

Stack

It is an ordered group of homogeneous items of elements. Elements are added to and removedfrom the top of the stack (the most recently added items are at the top of the stack). The lastelement to be added is the first to be removed (LIFO: Last In, First Out).

1.9 DATA STRUCTURES OPERATIONS

The data appearing in our data structures are processed by means of certain operations. In fact,the particular data structure that one chooses for a given situation depends largely in thefrequency with which specific operations are performed.

The following four operations play a major role in this text:

- Traversing: accessing each record/node exactly once so that certain items in the recordmay be processed. (This accessing and processing is sometimes called “visiting” therecord.)
- Searching: Finding the location of the desired node with a given key value, or findingthe locationsof all such nodes which satisfy one or more conditions.
- Inserting: Adding a new node/record to the structure.
- Deleting: Removing a node/record from the structure.
- Sorting:Arranging elememts in either ascending or descending order.

Very Short Answer Questions

(i) What is data type?

(ii) Define information.

(iii) Define data structure.

(iv) Write the name of two built-in data structure.

(v) What do you understand by linear and non-linear data structure?

(vi) List the four major operations performed on linear data structure.

(vii) Define algorithm.

(viii) Define program testing.

(ix) Define structured programming.

(x) Define debugging.

(xi) Define documentation.

(xii) Define variable.

(xiii) What is a constant?

(xiv) Define array.

(xv) Define pointer.

(xvi) Define pointer variable.

(xvii) What is a structure?

(xviii) Define global variable.

(xix) Write a statement that displays the address of an integer variable.

(xx) Scope of a variable

Short answer type questions 4 Marks

1. Explain Program Development Life Cycle.
2. Explain various types of data type in detail.
3. What is data structure? Give difference between the data and information.
4. Give difference between linear and non-linear data structures.
5. Differentiate between primitive and non-primitive data structure.
6. Explain various type of constants in detail.
7. Give the advantages of modular programming.
8. What is pointer variable? Give the advantages of pointer variable.
9. Explain structured programming in detail.
10. What is function? Explain global and local variables used in functions.
11. Write a program to swap the contents of two variables using pointers.

Long Answer Type questions 10 Marks

1. Explain program development life cycle.
2. What are linear and non-linear data structures? Explain with example.
3. What is structured programming? Explain top- down and bottom-up design for solving any

problem. Illustrates with examples.

Write the short notes on
(i) Top down Design
(ii) Bottom up Design
(iii) Data and Information

UNIT-2 ARRAYS

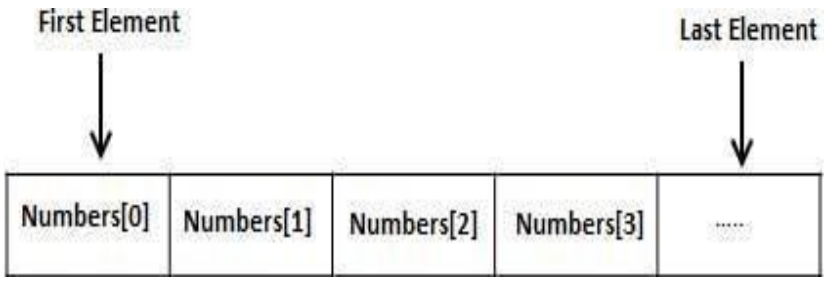
2.1Concept of Arrays

Array is a data structure which can hold a fix number of items and these items should be of the same type. Most of the data structures make use of arrays to implement their algorithms. Instead of declaring individual variables, such as number0, number1, ..., and number99, we declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.

All arrays consist of contiguous memory locations. The lowest address corresponds to the firstelement and the highest address to the last element.

The array may be categorized as :

- One dimensional array
- Two dimensional array
- Multidimensional array



2.1.1 One Dimensional Array

A one-dimensional array (or single dimension array) is a type of linear array. Accessing its elements involves a single subscript which can either represent a row or column index.

As an example consider the C declaration

`intanArrayName[10];`

Syntax :datatypeArrayname[sizeofArray];

`int` type. In

In the given example the array can contain 10 elements of any value available to the C, the array element indices are 0-9 in this case. For example, the expressions `anArrayName[0]` and `anArrayName[9]` are the first and last elements respectively.

One Dimensional Arrays can be initialized as follows:

Examples –

```
// A character array in C
char arr1[] = {'g', 'e', 'e', 'k', 's'};


// An Integer array in C
int arr2[] = {10, 20, 30, 40, 50};


// Item at i'th index in array is typically accessed
// as "arr[i]".  For example arr1[0] gives us 'g'
// and arr2[3] gives us 40.
```

As we know now 1-d array are linear array in which elements are stored in the successive memory locations. The element at which the first element is stored in memory is called its base address. Now consider the following example :

int arr[5];

Element	34	78	98	45	56
Memory Address	arr[0] = 100	arr[1] = ?	arr[2] = ?	arr[3] = ?	arr[4] = ?

Here we have defined an array of five elements of integer type whose first element is at base address 100. i.e, the element arr[0] is stored at base address 100. Now for calculating the starting address of the next element i.e. of a[1], we can use the following formula :

Base Address (B)+ No. of bytes occupied by element (C) * index of the element (i)

/* Here C is constant integer and vary according to the data type of the array, for e.g. for integer the value of C will be 2 bytes, since an integer occupies 2 bytes of memory. */

Now, we can calculate the starting address of second element of the array as :

arr[1] = 100 + 2 * 1 = 102/*Thus starting address of second element of array is 102 */

Similarly other addresses can be calculated in the same manner as :

arr[2] = 100 + 2 * 2 = 104

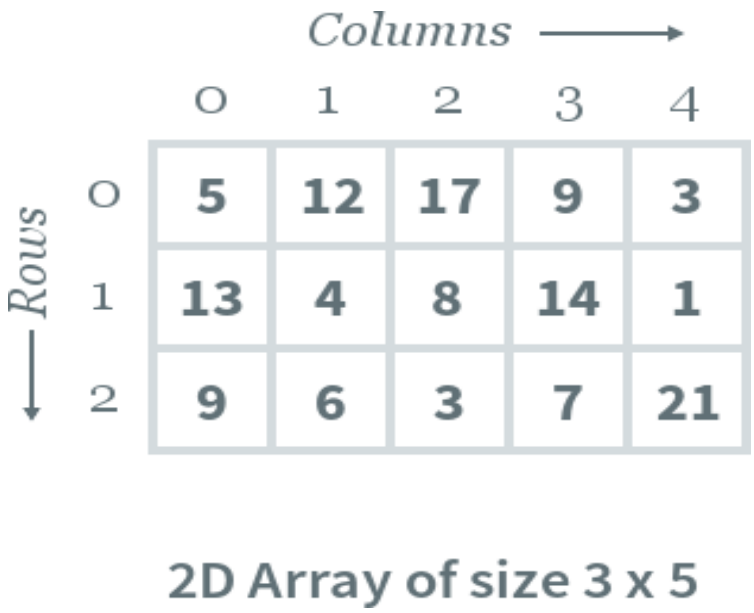
arr[3] = 100 + 2 * 3 = 106

arr[4] = 100 + 2 * 4 = 108

2.1.2 Two Dimensional Array

An array of one dimensional arrays is known as 2-D array. The two dimensional (2D) array in C programming is also known as matrix. A matrix can be represented as a table of rows and columns.

Consider following 2-D array, which is of the size 3×5. For an array of size N×M, the rows and columns are numbered from 0 to N−1 and columns are numbered from 0 to M−1, respectively. Any element of the array can be accessed by arr[i][j] where $0 \leq i < N$ and $0 \leq j < M$. For example, in the following array, the value stored at arr[1][3] is 14.



2D array declaration:

To declare a 2D array, you must specify the following: Row-size: Defines the number of rows Column-size: Defines the number of columnsType of array: Defines the type of elements to be stored in the array i.e. either a number, character, or other such datatype. A sample form of declaration is as follows:

```
typearr[row_size][column_size]
```

A sample C array is declared as follows:

```
intarr[3][5];
```

2D array initialization:

An array can either be initialized during or after declaration. The format of initializing an array during declaration is as follows:

```
typearr[row_size][column_size]={{elements},{elements}...}
```

An example is given below:

```
intarr[3][5]={{5,12,17,9,3},{13,4,8,14,1},{9,6,3,7,21}};
```

Initializing an array after declaration can be done by assigning values to each cell of 2D array, as follows.

```
typearr[row_size][column_size]
arr[i][j]=14
```

An example of initializing an array after declaration by assigning values to each cell of a 2D array is as follows:

```
intarr[3][5];
arr[0][0]=5;
arr[1][3]=14;
```

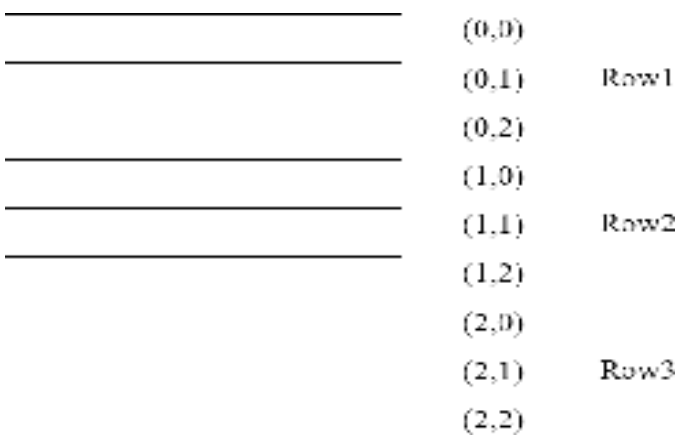
Processing 2D arrays:

The most basic form of processing is to loop over the array and print all its elements, which can be done as follows:

```
typearr[row_size][column_size]={{elements},{elements}...}
for i from0 to row_size
for j from0 to column_size
printarr[i][j]
```

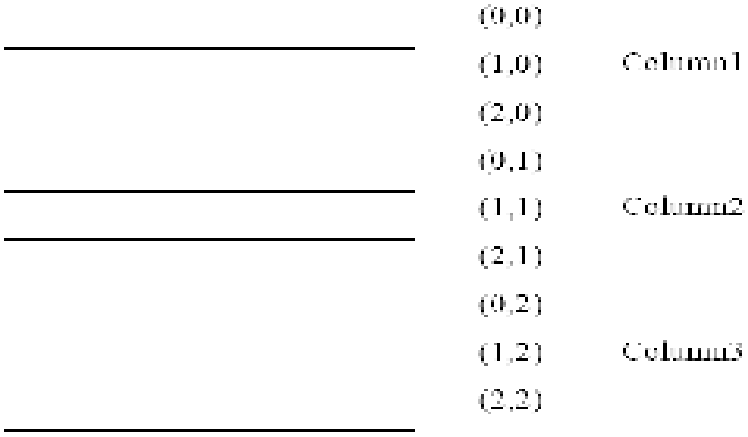
Representation of two dimensional arrays in memory

A two dimensional ‘m x n’ Array A is the collection of m X n elements. Programming languagestores the two dimensional array in one dimensional memory in either of two ways-
Row Major Order: First row of the array occupies the first set of memory locations reserved for the array; Second row occupies the next set, and so forth.



To determine element address A[i,j]:
Location (A[i,j]) =Base Address + (N x (I - 1)) + (j - 1)
For example:
Given an array [1…5,1…7] of integers. Calculate address of element T[4,6], where BA=900.
Sol) I = 4 , J = 6
M= 5 , N= 7
Location (T [4,6]) = BA + (7 x (4-1)) + (6-1)
= 900+ (7 x 3) +5
= 900+ 21+5
= 926

Column Major Order:Order elements of first column stored linearly and then comes elements of next column



To determine element address A[i,j]:
Location (A[i,j]) =Base Address + (M x (j - 1)) + (i - 1)
For example:
Given an array [1…6,1…8] of integers. Calculate address element T[5,7], where BA=300

Sol) I = 5 , J = 7
M= 6 , N= 8
Location (T [4,6]) = BA + (6 x (7-1)) + (5-1)
= 300+ (6 x 6) +4

= 300+ 36+4

= 340

2.2 Operations on array
a) Traversing: means to visit all the elements of the array in an operation is called traversing.
b) Insertion: means to put values into an array
c) Deletion / Remove: to delete a value from an array.
d) Sorting: Re-arrangement of values in an array in a specific order (Ascending or Descending) is called sorting.
e) Searching: The process of finding the location of a particular element in an array is called searching.
a) Traversing in Linear Array:
It means processing or visiting each element in the array exactly once; Let 'A' is an array stored in the computer's memory. If we want to display the contents of 'A', it has to be traversed i.e. by accessing and processing each element of 'A' exactly once.

Algorithm: (Traverse a Linear Array) Here **LA** is a Linear array with lower boundary **LB** and upper boundary **UB**. This algorithm traverses **LA** applying an operation Process to each element of **LA**.

1. [Initialize counter.] Set K=LB.
2. Repeat Steps 3 and 4 while K≤UB.
3. [Visit element.] Apply PROCESS to LA[K].
4. [Increase counter.] Set k=K+1.
[End of Step 2 loop.]
5. Exit.

The alternate algorithm for traversing (using for loop) is :

Algorithm: (Traverse a Linear Array) This algorithm traverse a linear array **LA** with lower bound **LB** and upper bound **UB**.

1. Repeat for K=LB to UB
Apply PROCESS to LA[K].
[End of loop].
2. Exit.

This program will traverse each element of the array to calculate the sum and then calculate& print the average of the following array of integers.

(4, 3, 7, -1, 7, 2, 0, 4, 2, 13)

b) Insertion:

Insert operation is to insert one or more data elements into an array. Based on the requirement, a new element can be added at the beginning, end, or any given index of array.

Here, we see a practical implementation of insertion operation, where we add data at the end of the array −

Algorithm

Let LA be a Linear Array (unordered) with N elements and K is a positive integer such that K<=N. Following is the algorithm where ITEM is inserted into the K^{th} position of LA −

```
1. Start
2. Set J = N
3. Set N = N+1
4. Repeat steps 5 and 6 while J >= K
5. Set LA[J+1] = LA[J]
6. Set J = J-1
7. Set LA[K] = ITEM
8. Stop
```

Example

Following is the implementation of the above algorithm −

```
#include<stdio.h>


main(){

int LA[]={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("LA[%d] = %d \n", i, LA[i]);
```

```
}

   n = n +1;

while( j >= k){
LA[j+1]= LA[j];
      j = j -1;
}

   LA[k]= item;

            printf("The array elements after insertion :\n");

for(i =0; i<n; i++){
printf("LA[%d] = %d \n", i, LA[i]);
}
}
```

When we compile and execute the above program, it produces the following result −

Output

```
The original array elements are :
LA[0] = 1
LA[1] = 3
LA[2] = 5
LA[3] = 7
LA[4] = 8
The array elements after insertion :
LA[0] = 1
LA[1] = 3
LA[2] = 5
LA[3] = 10
LA[4] = 7
```

```
LA[5] = 8
```

c) Deletion Operation

Deletion refers to removing an existing element from the array and re-organizing all elements of an array.

Algorithm

Consider LA is a linear array with N elements and K is a positive integer such that K<=N. Following is the algorithm to delete an element available at the K^{th} position of LA.

```
1. Start
2. Set J = K
3. Repeat steps 4 and 5 while J < N
4. Set LA[J] = LA[J + 1]
5. Set J = J+1
6. Set N = N-1
7. Stop
```

Example

Following is the implementation of the above algorithm −

```
#include<stdio.h>

main(){

int LA[]={1,3,5,7,8};

int k =3, n =5;

int i, j;


printf("The original array elements are :\n");


for(i =0; i<n; i++){

printf("LA[%d] = %d \n", i, LA[i]);

}


  j = k;
```

```
while( j < n){
LA[j-1]= LA[j];
    j = j +1;
}

  n = n -1;

printf("The array elements after deletion :\n");

for(i =0; i<n; i++){
printf("LA[%d] = %d \n", i, LA[i]);
}
}
```

When we compile and execute the above program, it produces the following result −

Output

```
The original array elements are :
LA[0] = 1
LA[1] = 3
LA[2] = 5
LA[3] = 7
LA[4] = 8
The array elements after deletion :
LA[0] = 1
LA[1] = 3
LA[2] = 7
LA[3] = 8
```

d) Search Operation

You can perform a search for an array element based on its value or its index.

Algorithm

Consider LA is a linear array with N elements and K is a positive integer such that K<=N. Following is the algorithm to find an element with a value of ITEM using sequential search.

```
1. Start
2. Set J = 0
3. Repeat steps 4 and 5 while J < N
4. IF LA[J] is equal ITEM THEN GOTO STEP 6
5. Set J = J +1
6. PRINT J, ITEM
7. Stop
```

Example

Following is the implementation of the above algorithm −

```
#include<stdio.h>

main(){

int LA[]={1,3,5,7,8};

int item =5, n =5;

int i =0, j =0;


printf("The original array elements are :\n");


for(i =0; i<n; i++){

printf("LA[%d] = %d \n", i, LA[i]);

}


while( j < n){

if( LA[j]== item ){

break;

}


   j = j +1;
```

```
}


printf("Found element %d at position %d\n", item, j+1);

}
```

When we compile and execute the above program, it produces the following result −

Output

```
The original array elements are :
LA[0] = 1
LA[1] = 3
LA[2] = 5
LA[3] = 7
LA[4] = 8
Found element 5 at position 3
```

e) Update Operation

Update operation refers to updating an existing element from the array at a given index.

Algorithm

Consider LA is a linear array with N elements and K is a positive integer such that K<=N.

Following is the algorithm to update an element available at the K^{th} position of LA.

```
1. Start
2. Set LA[K-1] = ITEM
3. Stop
```

Example

Following is the implementation of the above algorithm −

```
#include<stdio.h>

main(){

int LA[]={1,3,5,7,8};

int k =3, n =5, item =10;

int i, j;


printf("The original array elements are :\n");
```

```
for(i =0; i<n; i++){
printf("LA[%d] = %d \n", i, LA[i]);
}

LA[k-1]= item;

printf("The array elements after updation :\n");

for(i =0; i<n; i++){
printf("LA[%d] = %d \n", i, LA[i]);
}
}
```

When we compile and execute the above program, it produces the following result −

Output

```
The original array elements are :
LA[0] = 1
LA[1] = 3
LA[2] = 5
LA[3] = 7
LA[4] = 8
The array elements after updation :
LA[0] = 1
LA[1] = 3
LA[2] = 10
LA[3] = 7
LA[4] = 8
```

f) Sorting:

Sorting an array is the ordering the array elements in ascending (increasing from min to max) or descending(decreasing from max to min) order.
Bubble Sort:

The technique we use is called “Bubble Sort” because the bigger value gradually bubbles theirway up to the top of array like air bubble rising in water, while the small values sink to thebottom of array. This technique is to make several passes through the array. On each pass,successive pairs of elements are compared. If a pair is in increasing order (or the values areidentical), we leave the values as they are. If a pair is in decreasing order, their values areswapped in the array.

Pass = 1	Pass = 2	Pass = 3	Pass=4
2 1 5 7 4 3	**1 2** 5 4 3 7	**1 2** 4 3 5 7	**1 2** 3 4 5 7
1 **2 5** 7 4 3	1 **2 5** 4 3 7	1 **2 4** 3 5 7	1 **2 3** 4 5 7
1 2 **5 7** 4 3	1 2 **5 4** 3 7	1 2 **4 3** 5 7	1 2 3 4 5 7
1 2 5 **7 4** 3	1 2 4 **5 3** 7	1 2 3 4 5 7	
1 2 5 4 **7 3**	1 2 4 3 5 7		
1 2 5 4 3 7			

- Underlined pairs show the comparisons. For each pass there are size-1 comparisons.
- Total number of comparisons= $(size-1)^2$

Algorithm: (Bubble Sort) BUBBLE (DATA, N)
Here DATA is an Array with N elements. This algorithm sorts the elements in DATA.

```
1. for pass=1 to N-1.
2.    for (i=0; i<= N-Pass; i++)
3.       If DATA[i]>DATA[i+1], then:
                    Interchange DATA[i] and DATA[i+1].
              [End of If Structure.]
       [End of inner loop.]
   [End of Step 1 outer loop.]
4. Exit.
```

Chapter-2 Arrays
Very Short Answer Type Questions 2 Marks

(i) What do you mean by an array?

(ii) Declaration of single dimensional array.

(iii) What do you understand by index or subscript of an array?

(iv) What is base address of array?

(v) Give formula to calculate the size of one dimensional array.

(vi) Define two dimensional arrays.

(vii) What is row major order?

(viii) Write formula to calculate length of two dimensional arrays.

(ix) What do you mean by multidimensional array?

(x) How does a structure differ from an array?

Short Answer Type Questions 4 Marks

1. What is linear array? How is it represented in memory?

2. Consider the one dimensional float array LA. Base address is 2000,w = 4, LB = 1 and UB = 8. Calculate the address of LA[4] and LA[6].

3. Write the steps for traversing a linear array.

4. Write an algorithm for search operation of an array.

5. Write a program which defines the concept of traversing of an array.

Long Answer Type questions 10 Marks

1. Explain how two dimensional array is represented in memory.

2. Explain the various operations associated with two dimensional arrays.

3. Write a program in ‘C’ to add two matrices.

4. Write a program in ‘C’ to multiply two matrices.

5. Write a program in C to add two matrices using pointers.