MODULE 2

Syllabus:

CO2:	Make use of the concept of arrays to solve real world problems.		
M2.01	Summarize the definition, initialization and accessing of single and multi dimensional arrays.	1	Understanding.
M2.02	Develop programs using single and multidimensional arrays	1	Applying
M2.03	Illustrate the concept of divide and conquer method in solving problems.	1	Understanding.
M2.04	Develop C programs to implement searching (linear search and binary search) and sorting (selection sort and quicksort) algorithms.	2	Applying.
M2.05	Explain the representation of strings in C	1	Understanding.
M2.06	Develop C programs to perform different operations on strings	3	Applying
M2.07	Illustrate passing arrays as parameters to a function	2	Understanding.

Contents:

Arrays – definition, initialization and processing of arrays
Searching algorithms – Linear search, Binary Search.
Sorting algorithms – Selection sort, Quick sort,Passing arrays to functions .
Strings – Representation of strings in C – String input and output – String Processing – copy, concentrate, length,comparison, pattern searching etc – builtin String functions – Implementation of string functions.

ARRAYS

Definition:

- An array is a data structure
- Consisting of a collection of elements in a single variable
- Stored in contiguous location
- All the elements should be of the same data type(homogenous)
- Array elements can access by using indices
- They can be used to store collection of primitive datatypes such as int,float,double,char etc of any particular type.

Advantage of C Array

1) Code Optimization: Less code to the access the data.
2) Ease of traversing: By using the for loop, we can retrieve the elements of an array easily.
3) Ease of sorting: To sort the elements of the array, we need a few lines of code only.
4) Random Access: We can access any element randomly using the array.

Disadvantage of C Array

1) Fixed Size: Whatever size, we define at the time of declaration of the array, we can't exceed the limit. So, it doesn't grow the size dynamically .

Declaration:

Syntax:

```
dataType arrayName[arrayMaxSize];
```

For example,

```
Int mark[25];
```

Here, we declared an array, mark, of integer type. And its maximum size is 25. Meaning, it can hold 25 integer values.

Initialization:

1.Compile time initialization:user has to given values in the program itself.As same as variable initialization.

Syntax:

```
dataType arrayName[size] = { list_of_values };
```

.For example,

```
//integer array initialization

int rollnumbers[4]={ 2, 5, 6, 7};

//character array initialization

char name[4]={'R','A','M','\0'};
```

2.Run time initialization: User can enter different values during different runs of program.It is also used for initializing large arrays or array with user specified values,using scanf() function.

Accessing an element of array:

To access an individual element of an array, use the name of the array and name followed by the index of the element in square brackets. Array indices start at 0 and end at size-1:

```
array_name[index];
```

For example,

int Mark[5]={12 , 23 , 25 , 16 , 14 };

An integer array named as Mark and 5 in size.Elements are 12,23,25,16,14.

12	23	25	16	14
Mark[0]	Mark[1]	Mark[2]	Mark[3]	Mark[4]
0th index	1st index	2nd index	3rd index	4th index

```
printf("%d", Mark[0]);
```

This statement accesses the value of the **first element [0]** in Mark: that is 12.

Problem:Program to read and display an array

```
1. #include<stdio.h>
2. void main ()
3. {
4.     int arr[100],i,n;
5.     printf("Enter the size of the array?");
6.     scanf("%d",&n);
7.     printf("Enter the elements of the array?");
8.     for(i = 0; i<n; i++)
```

```
9.      {
10.         scanf("%d",&arr[i]);
11.     }
12.    printf("The array elements are:");
13.     for(i = 0; i<n; i++)
14.     {
15.         printf("%d\t",&arr[i]);
16.     }
```

Sample output:

Enter the size of the array? **5**

Enter the elements of the array?

11

12

13

14

15

The array elements are:

11 12 13 14 15

Sample program question:

1.Program to print the largest and second largest element of the array?

2.Program to display array in reverse order?

3.Program to find sum and average of an array?

4.Program to display even numbers in an array?

***Multi-dimensional arrays*:**

- A multi-dimensional array can be termed as an array of arrays
- that stores homogeneous data in tabular form.
- Data in multidimensional arrays are stored in row-major order.

The general form of declaring N-dimensional arrays is:

```
data_type array_name[size1][size2]....[sizeN];
```

For example:

Two dimensional array: int two_d[10][20];

Three dimensional array: int three_d[10][20][30];

Size of Multidimensional Arrays: The total number of elements that can be stored in a multidimensional array can be calculated by multiplying the size of all the dimensions.

For example:

- The array **int x[10][20]** can store total (10*20) = 200 elements.
- Similarly array **int x[5][10][20]** can store total (5*10*20) = 1000 elements.

Two-Dimensional Array:

- Two – dimensional array is the simplest form of a multidimensional array.
- array of one-dimensional array

Declaration:

```
data_type array_name[row_size][column_size];
```

For example,

Int matrix[2][2];

	Column 0	Column 1
Row 0	Matrix[0][0]	Matrix[0][1]
Row 1	Matrix[1][0]	Matrix[1][1]

Initialization:

First method: ***int matrix[2][2]={1,2,3,4};***

Second method*:* ***int matrix[2][2]={{1,2}, {3,4} };***

Third method(runtime):

```
for(row=0;row<2;row++)
{
        For(col=0;col<2;col++)
        {
                Scanf("%d",&matrix[row][col]);

}
```

Accessing Elements of Two-Dimensional Arrays:

Elements in Two-Dimensional arrays are accessed using the row indexes and column indexes.

Example:

matrix[0][1];

The above example represents the element present in the 0th row and first column.

Note: In arrays, if the size of an array is N. Its index will be from 0 to N-1.

int matrix[2][2]={1,2,3,4};
then,

matrix[0][0]=1
matrix[0][1]=2
matrix[1][0]=3
matrix[1][1]=4

Problem:Program to read and display a matrix?

```
int main(void)
{
        int matrix[5][5],row_size,col_size,row,col;
        printf("Enter row size\n");
        scanf("%d",&row);
        printf("Enter column size\n");
        scanf("%d",&col);
        printf("Enter elements of matrix\n");
        for (row = 0; row < row_size ; row++)
        {
    for (col = 0; col < col_size; col++)
    {
       printf("Enter element at matrix[%d][%d]: ",row, col);
       scanf("%d",&matrix[row][col]);
    }
  }


  // output each array element's value
       Printf("\n Matrix is:");
  for (row = 0; row < row_size ; row++)
  {
    for (col = 0; col < col_size; col++)
    {
       printf("%d\t",matrix[row][col]);
    }
        Printf("\n");
  }
```

Sample output:

```
Enter row size:2
Enter column size:2
Enter element at matrix[0][0]: 1
Enter element at matrix[0][1]: 2
Enter element at matrix[1][0]: 3
Enter element at matrix[1][1]: 4
Matrix is:
    1       2
    3       4
```

Sample program Questions:

1.Program to implement matrix operations:

i) Matrix sum/subtraction
ii) Matrix inverse
iii) Matrix transpose
iv) Matrix multiplication etc

Concept of Divide and Conquer method in solving problems

A **divide and conquer algorithm** is a strategy of solving a large problem by breaking the problem into smaller sub-problems solving the sub-problems, and combining them to get the desired output. To use the divide and conquer algorithm, **recursion** is used.
Here are the steps involved:

1. **Divide**: Divide the given problem into sub-problems using recursion.
2. **Conquer**: Solve the smaller sub-problems recursively. If the subproblem is small enough, then solve it directly.
3. **Combine:** Combine the solutions of the sub-problems recursively to solve the actual problem.

Examples for some standard algorithms that are of the Divide and Conquer algorithms variety. ie,Binary Search, Quicksort, Mergesort etc.

Linear search (Sequential search) algorithm

Linear Search is defined as a sequential search algorithm that starts at one end and goes through each element of an array until the desired element is found, otherwise the search continues till the end of the array.Linear search is applied on sorted or unsorted array.

Algorithm for linear search:

Linearsearch(array,key)

for each *item* in the array
if *item* equals *value* then return its index
exit for loop
if *item* not found in entire search returns -1

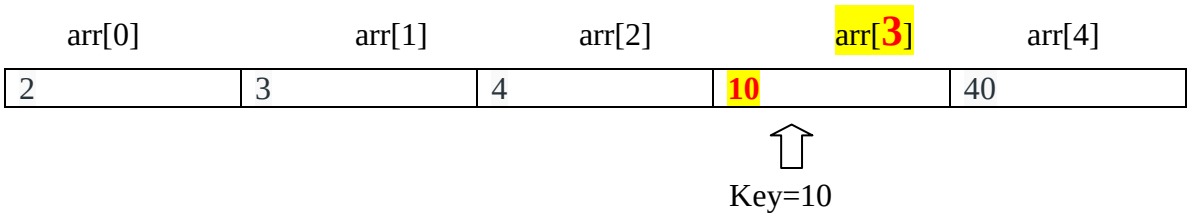
Program to implement linear search algorithm

```
// C code to linearly search key in arr[]. If key
// is present then return its index, otherwise
// return -1
```

```
#include <stdio.h>
int Linearsearch(int arr[], int N, int key)
{
        int i;
        for (i = 0; i < N; i++)
                if (arr[i] == key)
                        return i;
        return -1;
}
Void main()
{
        int arr[] = { 2, 3, 4, 10, 40 };
        int key = 10;
        N=5;
        // Function call
        int result = search(arr, N, key);
        if(result == -1)
                printf("Element is not present in array")
        else
                printf("Element is present at index %d", result);

}
```

Sample output:**Element is present at index 3**



Binary search algorithm

The basic steps to perform Binary Search are:

- Begin with the mid element of the whole array as a search key.
- If the value of the search key is equal to the item then return an index of the search key.
- Or if the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half.
- Otherwise, narrow it to the upper half.
- Repeatedly check from the second point until the value is found or the interval is empty.

Binary Search Algorithm can be implemented in the following two ways

1. Iterative Method
2. Recursive Method

1.Iterative method:

Algorithm

```
binarySearch(arr, key, low, high)
      repeat till low = high
           mid = (low + high)/2
              if (key == arr[mid])
                  return mid

              else if (key > arr[mid]) // key is on the right side
                 low = mid + 1
              else                 // key is on the left side
                 high = mid – 1
```

Program to implement binary search(iterative method)

```
#include<stdio.h>
int binarysearch(int array[],int key,int low,int high)
{
      while(low<=high)
      {
                int mid=(low+high)/2;
                if (array[mid]==key)
                        return mid;
                if(array[mid]<key)
                        low=mid+1;
                else
                        high=mid-1;
      }
      return -1;
}
void main()
{
      int array[5]={10,20,30,40,50};
      int n=5,key=30;  //n is size of array and key is searching element
      int result=binarysearch(array,key,0,n-1);
      if( result==-1) {printf("Element not found\n");}
      else {printf("Element found at %d\n",result);}

}
```

2.Recursive method:

Algorithm

```
binarySearch(arr, key, low, high)
        if low > high
           return False

        else
           mid = (low + high) / 2
              if key == arr[mid]
              return mid

           else if key > arr[mid]       // key is on the right side
              return binarySearch(arr, key, mid + 1, high)

           else                         // key is on the right side
              return binarySearch(arr, key, low, mid - 1)
```

Program

```
int binarySearch(int arr[], int low, int high, int key)
{
   if (high >= low) {
      int mid = low + (high - low) / 2;

      // If the element is present at the middle
      // itself
      if (arr[mid] == key)
         return mid;

      // If element is smaller than mid, then
      // it can only be present in left subarray
      if (arr[mid] > key)
         return binarySearch(arr, low, mid - 1, key);

      // Else the element can only be present
      // in right subarray
      return binarySearch(arr, mid + 1, high, key);
   }

   // We reach here when element is not
   // present in array
   return -1;
}
 #include<stdio.h>
int main(void)
{
   int arr[] = { 2, 3, 4, 10, 40 };
   int n = 5;
```

```
    int key = 10;
    int result = binarySearch(arr, 0, n - 1, key);
    if(result == -1)
       { printf("Element is not present in array");}
    else
       printf("Element is present at index %d", result);
    return 0;
}
```

For example:**search 23 in this array**

Step 1:

Low=0 **high=9**

0	1	2	3	**mid**= 4	5	6	7	8	9
8	9	10	11	16	23	25	66	70	100

Step 2:

					low=5	6	**mid=7**	8	**high=9**
2	5	8	12	16	23	38	56	72	91

Step 3:

					low=5 **mid=5**	**high=6**			
2	5	8	12	16	23	38	56	72	91

Array[mid]=key
*****Element found at mid****

Selection Sort algorithm

The **selection sort algorithm** sorts an array by repeatedly finding the minimum element (considering ascending order) from the unsorted part and putting it at the beginning.
The algorithm maintains two subarrays in a given array.

- The subarray which already sorted.
- The remaining subarray was unsorted.

In every iteration of the selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray.

Lets consider the following array as an example: **arr[] = {64, 25, 12, 22, 11}**
First pass:

- For the first position in the sorted array, the whole array is traversed from index 0 to 4 sequentially. The first position where **64** is stored presently, after traversing whole array it is clear that **11** is the lowest value.

64 25 12 22 11

- Thus, replace 64 with 11. After one iteration **11**, which happens to be the least value in the array, tends to appear in the first position of the sorted list.

11 25 12 22 64

Second Pass:

- For the second position, where 25 is present, again traverse the rest of the array in a sequential manner.

11 **25** 12 22 64

- After traversing, we found that **12** is the second lowest value in the array and it should appear at the second place in the array, thus swap these values.

11 **12** 25 22 64

Third Pass:

- Now, for third place, where **25** is present again traverse the rest of the array and find the third least value present in the array.

11 12 **25** 22 64

- While traversing, **22** came out to be the third least value and it should appear at the third place in the array, thus swap **22** with element present at third position.

11 12 **22** 25 64

Fourth pass:

- Similarly, for fourth position traverse the rest of the array and find the fourth least element in the array
- As **25** is the 4th lowest value hence, it will place at the fourth position.

11 12 22 **25** 64

Fifth Pass:

- At last the largest value present in the array automatically get placed at the last position in the array
- **The resulted array is the sorted array.**

11 12 22 25 64

Follow the below steps to solve the problem:

- Initialize minimum value(**min_idx**) to location 0.
- Traverse the array to find the minimum element in the array.
- While traversing if any element smaller than **min_idx** is found then swap both the values.
- Then, increment **min_idx** to point to the next element.
- Repeat until the array is sorted.

Algorithm

```
1.for i=1 up to n-1
2.      min_idx=i
3.      for j=i+1 up to n
```

```
4.              if array[j]  less than  array[min_idx] then
5.                      min_idx=j
6.              end if
7.      end for
8.      if min_index!=i then
9.              swap array[min_idx] and array[i]
10.     end if
11.end for
```

Program to implement selection sort algorithm in C

```
// C program for implementation of selection sort
#include <stdio.h>
void selectionSort(int arr[], int n)
{
        int i, j, min_idx;
        // One by one move boundary of unsorted subarray
        for (i = 0; i < n-1; i++)
        {
                // Find the minimum element in unsorted array
                min_idx = i;
                for (j = i+1; j < n; j++)
                if (arr[j] < arr[min_idx])
                        min_idx = j;

                // Swap the found minimum element with the first element
                if(min_idx != i)
                {
                        int temp = arr[min_idx];
                          arr[min_idx] = arr[i];
                         arr[i] = temp;
                }
        }
}


/* Function to print an array */
void printArray(int arr[], int size)
{
        int i;
        for (i=0; i < size; i++)
                printf("%d ", arr[i]);
        printf("\n");
}

int main()
{
        int arr[] = {64, 25, 12, 22, 11};
```

```
    int n = sizeof(arr)/sizeof(arr[0]);
    selectionSort(arr, n);
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}
```

Quick sort algorithm

QuickSort is a Divide and Conquer algorithm. It picks an element as a pivot and partitions the given array. There are many different versions of quickSort that pick pivot in different ways.

- Always pick the first element as a pivot.
- Always pick the last element as a pivot
- Pick a random element as a pivot.
- Pick median as the pivot.

The key process in **quickSort** is a partition(). The target of partitions is, given an array and an element x of an array as the pivot, put x at its correct position in a sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x.

Algorithm

Algorithm for recursive QuickSort function:

/* low –> Starting index, high –> Ending index */

1. quickSort(arr[], low, high)
2. if (low < high) then
3. j = partition(arr, low, high) // j is partitioning index, arr[j] is now at right place
4. quickSort(arr, low, j – 1) // Before j
5. quickSort(arr, j + 1, high) // After j

Algorithm for partition()

/* This function takes last element as pivot, places the pivot element at its correct position in sorted array, and places all smaller (smaller than pivot) to left of pivot and all greater elements to right of pivot */

1. partition (arr[], low, high)
2. i=low,j=high,pivot=low
3. while i<j then
4. while arr[i]<=a[pivot] do
5. Increment i by 1

6. While arr[j]>=arr[pivot] do
7. Decrement j by 1
8. If i<j then swap arr[i] and arr[j]
9. End while
10. Swap arr[j] and arr[pivot]
11. Return j
12. End partition

Program to implement quicksort in C

```
#include<stdio.h>
void quicksort(int [],int,int);
int partition(int [],int ,int );
void main()
{
        int i, count, number[25];
        printf("Enter limit ");
        scanf("%d",&count);
        printf("Enter %d elements: ", count);
        for(i=0;i<count;i++)
           scanf("%d",&number[i]);
        quicksort(number,0,count-1);
        printf("The Sorted Order is: ");
        for(i=0;i<count;i++)
           printf(" %d",number[i]);
}
void quicksort(int number[25],int low,int high)
{
        int j;
        if(low<high)
        {
           j=partition(number,low,high);
           quicksort(number,low,j-1);
           quicksort(number,j+1,high);
        }
}

int partition(int number[25],int low,int high)
{
           int i, j, pivot, temp;
           pivot=low;
           i=low;
           j=high;
           while(i<j)
           {
```

```
            while(number[i]<=number[pivot]&&i<high)
               i++;
            while(number[j]>number[pivot])
               j--;

            if(i<j)
            {
               temp=number[i];
               number[i]=number[j];
               number[j]=temp;
            }
         }
         temp=number[pivot];
         number[pivot]=number[j];
         number[j]=temp;
         return j;
}
```

String

String in C programming is a sequence of characters terminated with a null character '\0'.
Strings are defined as an array of characters
For example,

'h'	'e'	'l'	'l'	'O'	'\0'

Declaration:Declaring a string is as simple as declaring a one dimensional character array
Syntax: char stringName[size];
Eg,char str[20];
Here str is the name given to string and 20 is used to define length of string including null character.

Initialization:

You can initialize strings in a number of ways.

```
char c[] = "abcd";
char c[50] = "abcd";
char c[] = {'a', 'b', 'c', 'd', '\0'};
char c[5] = {'a', 'b', 'c', 'd', '\0'}
```

Access Strings:

Since strings are actually arrays in C, you can access a string by referring to its index number inside square brackets [].

This example prints the **first character (0)** in **str1**:

```
char str1[] = "Hello World!";
printf("%c", str1[0]);
```

String Operations in C

Inbuilt string handling functions

Function	Working
strlen()	computes string's length
strcpy()	copies a string to another
strcat()	concatenates(joins) two strings
strcmp()	compares two strings case sensitive checking
strcmpi()	compares two strings case insensitive checking
strlwr()	converts string to lowercase
strupr()	converts string to uppercase

1)Read string from user using scanf

```
#include <stdio.h>
void main()
{
        char str[20];
        printf("Enter the String: ");
        scanf("%s", str);
        printf("\nString: %s\n", str);
}
```

Smaple output:

Enter the string: hello world

String:hello

//scanf reads input until it encounters a whitespace

Use %[\n] instead %s

```
#include <stdio.h>
void main()
{
        char str[20];
        printf("Enter the 'hello world'");
        scanf("%[^\n]", str);
```

```
        printf("\nString: %s\n", str);

}
```

2)Make use of gets() and puts()

```
#include <stdio.h>
void main()
{
        char str[20];
        printf("Enter the String:");
        gets( str);
        puts(str);
}
```

Sample output:
Enter the string: Hello world
Hello world

- gets() is used to read input from keyboard until it encounters newline or End Of File(EOF)

2) strlen() vs sizeof()

- strlen returns the length of the string stored in array,however sizeof returns the total allocated size assigned to the array.

```
#include<stdio.h>
#include<string.h>
void main()
{
        char str[]="Hello World";
        int len=strlen(str);
        int size=sizeof(str);
        printf("strlen=%d\n sizeof=%d",len,size);
}
```

Sample output:
Strlen=11
Size=12

3) Program to copy a string to another without using string handling function

```
#include<stdio.h>
void main()
{
        char a[50], b[50];
        int i=0;
        printf(“Enter string1:”);
        gets(a);
        for(i=0;a[i]!=‘\0’;i++)
        {
```

```
            b[i]=a[i];
        }
        b[i]='\0';
}
```

Using Strcpy()

- It copies the string str2 into str1,including null character
- Syntax:strcpy(string1,string2);

```
#include <stdio.h>
#include <string.h>
void main()
{
        char s1[30];
        char s2[30] = "Hello World";
        strcpy(s1,s2);
        printf("String s1: %s", s1);
}
```

5) Program to concatenate two strings without using string handling function

```
#include <stdio.h>
void main()
{
        char str1[50], str2[25], i, j;
        printf("\nEnter first string: ");
        gets(str1);
        printf("\nEnter second string: ");
        gets(str2);
        for(i=0; str1[i]!='\0'; i++);
        for(j=0; str2[j]!='\0'; j++, i++)
        {
                str1[i]=str2[j];
        }
        str1[i]='\0';
        printf("\nOutput: %s",str1);
        }
}
```

Using Strcat()

- It concatenate two strings and returns the output string.
- Syntax:strcat(string1,string2);
- Concatenate string2 with string1

```
#include <stdio.h>
#include <string.h>
void main()
{
```

```
        char s1[10] = "Hello";
        char s2[10] = "World";
        strcat(s1,s2);
        printf("After concatenation: %s", s1);
}
```

Sample output:

After concatenation:HelloWorld

6) Program to check two strings are equal without using string handling functions

```
#include<stdio.h>
void main()
{
        char str1[100], str2[100];
        int i, flag=0;
        printf("Enter the First string :\n");
        gets(str1);
        printf("Enter the Second string :\n");
        gets(str2);
        for(i=0;str1[i]!='\0'&&str2[i]!='\0';i++)
        {
                if(str1[i]!=str2[i])
                {
                        flag=1;
                        break;
                }
        }
        if(flag==0&& str1[i]!='\0'&&str2[i]!='\0')
        printf("Strings are equal \n");
        else
        printf("Strings are not equal \n");

}
```

Using strcmp()

- It compares the two strings and returns an integer value.For same string function return 0 otherwise may return a negative or positive value based on comparison.
- Syntax:strcmp(string1,string2);
- if string1==string2 then get 0[zero] when use this function
- else positive or negative values

```
#include <stdio.h>
#include <string.h>
void main()
{
char s1[20] = "Hello";
char s2[20] = "World";
if (strcmp(s1, s2) ==0)
```

```
{
printf("string 1 and string 2 are equal");
}
else
{
printf("string 1 and 2 are different");
}
}
```

Sample output:

String 1 and 2 are different

7) Write a program to display the content of string in upper and lower case.

```
#include<stdio.h>
#include<string.h>
void main()
{
        char str[20];
        printf("Enter the string:");
        gets(str);
        printf("\nUpper Case String:%s",strupr(str));
        printf("\nLower Case String:%s",strlwr(str));
}
```