Course: Problem Solving Techniques using C

Course Code: 24BTPHY104

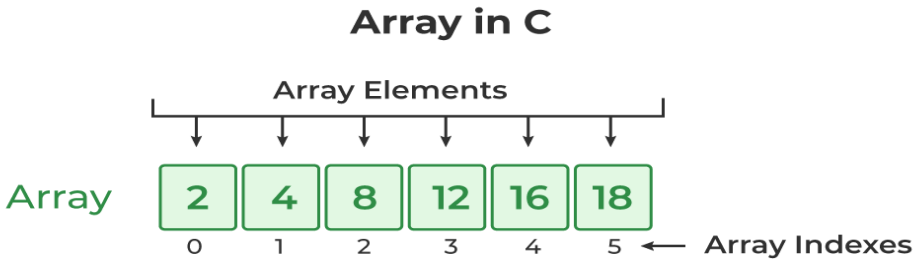
Module 3

3.ARRAYS – STRINGS – STORAGE CLASSES

3.1 Arrays

An Array is a collection of similar data elements. Array is one of the most used data structures in C programming. It is a simple and fast way of storing multiple values under a single name.

An array in C is a fixed-size collection of similar data items stored in contiguous memory locations. It can be used to store the collection of primitive data types such as int, char, float, etc., and also derived and user-defined data types such as pointers, structures, etc.



Pictorial representation of an array of 5 integers

10	20	30	40	50
A[0]	A[1]	A[2]	A[3]	A[4]

- An array is a collection of similar data items.
- All the elements of the array share a common name.
- Each element in the array can be accessed by the subscript (or index) and array name.
- The arrays are classified as:

1.Single dimensional array

2.Multidimensional array.

3.2 Single Dimensional Array.

a. A single dimensional array is a linear list of related data items of same data type.
b. In memory, all the data items are stored in contiguous memory locations.

Declaration of one-dimensional array (Single dimensional array)

Syntax:

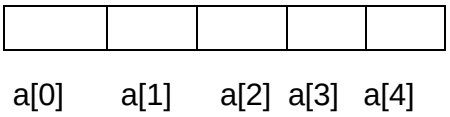
```
datatype  array_name[size];
```

- ✓ **datatype** can be int,float,char,double.
- ✓ **array_name** is the name of the array and it should be a valid identifier.
- ✓ **Size** is the total number of elements in array.

For example**:**

```
int  a[5];
```

The above statement allocates 5*2=10 Bytes of memory for the array **a.**




```
float b[5];
```

The above statement allocates 5*4=20 Bytes of memory for the array **b.**

a. Each element in the array is identified using integer number called as i**ndex.**
b. If n is the size of array, the array index starts from **0** and ends at **n-1.**

Storing Values in Arrays

i. Declaration of arrays only allocates memory space for array. But array elements are not initializedand hence values have to be stored.
ii. Therefore, to store the values in array, there are 3 methods

1.Initialization

2.Assigning Values

3.Input values from keyboard through **scanf()**

Initialization of one-dimensional array

- Assigning the required values to an array element before processing is called initialization.

```
data type array_name[expression]={v1,v2,v3…,vn};
```

Where

- ✓ datatype can be char,int,float,double
- ✓ array name is the valid identifier
- ✓ size is the number of elements in array
- ✓ v1, v2, v3……vn are values to be assigned.

- Arrays can be initialized at declaration time.Example:

int a[5]={2,4,34,3,4};

2	4	34	3	4
a[0]	a[1]	a[2]	a[3]	

The various ways of initializing arrays are as follows:

Initializing all elements of array (Complete array initialization)

1. Partial array initialization.
2. Initialization without size3.
3. String initialization

Initializing all elements of array:

- Arrays can be initialized at the time of declaration when their initial values are known in advance.
- In this type of array initialization, initialize all the elements of specified memory size.

- Example:

int a[5]={10,20,30,40,50};

10	20	30	40	50

Initialization without size

- In the declaration the array size will be set to the total number of initial values specified.
- The compiler will set the size based on the number of initial values.

Example:

int a[ ]={10,20,30,40,50};

In the above example the size of an array is set to 5

String Initialization

- Sequence of characters enclosed within double quotes is called as string.
- The string always ends with NULL character **(\0)**

```
char s[5]="SNPS";
```

We can observe that string length is 4, but size is 5 because to store NULL character we need one more location.

So pictorial representation of an array **s** is as follows

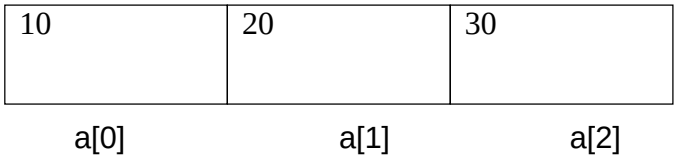
S	N	P	S	\0
S[0]	S[1]	S[2]	S[3]	S[4]

Assigning values to arrays

Using assignment operators, we can assign values to individual elements of arrays.

For example:

```
int a[3];
    a[0]=10;
    a[1]=20;
    a[2]=30;
```



Reading and writing single dimensional arrays.

To read array elements from keyboard we can use **scanf()** function as follows:

To read **0th** element: scanf("%d",&a[0]);To read **1st** element: scanf("%d",&a[1]); To read **2nd** element: scanf("%d",&a[2]);

……

…….

To read **nth** element : scanf("%d",&a[n-1]);

In general

To read **ith** element:

scanf("%d",&a[i]); where i=0; i<n; i++

To print array elements we can use **printf()** function as follows:

To print **0th** element: printf("%d",a[0]); To print **1st** element: printf("%d",a[1]); To print **2nd** element :printf("%d",a[2]);

……..

……..

To **nth** element : printf("%d",&a[n-1]);

In general

To read **ith** element:

printf("%d",a[i]); where i=0; i<n; i++

- **Write a C program to read N elements from keyboard and to print N elements on screen**.

```
/* program to read N elements from keyboard and to print N elements on
screen */
#include<stdio.h>
void main()
{
int i,n,a[10];
printf("enter number of array elements\n");
scanf("%d",&n);
printf("enter array elements\n");
for(i=0; i<n;i++)
```

```
{
scanf("%d",&a[i]);
}
printf("array elements are\n"):
for(i=0; i<n;i++)
{
printf("%d",a[i]);
}
}
```

2.Write a C program to find sum of n array elements.

```
/* program to find the sum of n array elements. */
#include<stdio.h>
void main()
{
int i,n,a[10],sum=0;
printf("enter number of array elements\n");
scanf("%d",&n);
printf("enter array elements\n");
for(i=0; i<n; i++)
{
scanf("%d",&a[i]);
}
for(i=0; i<n;i++)
{
sum=sum+ a[i];
}
printf("sum is %d\n",sum):
}
```

3.Write a c program to find largest of n elements stored in an array a.

```
#include<stdio.h>
void main()
{
int i,n,a[10],big;
printf("enter number of array elements\n");scanf("%d",&n);
printf("enter array elements\n");for(i=0; i<n;i++)
{
scanf("%d",&a[i]);
}
big=a[0];
for(i=0; i<n;i++)
{
if(a[i]>big)
big=a[i];
}
```

```
printf("the biggest element in an array is %d\n",big);
}
```

4.Write a C program to generate Fibonacci numbers using arrays.

```
#include<stdio.h>
    void main()
    {
        int i,n,a[10];a[0]=0;
        a[1]=1;
        printf("enter n\n");
        scanf("%d",&n); if(n==1)
        {
            printf("%d\t",a[0]);
        }
        if(n==2)
        {
            printf("%d\t %d\t",a[0],a[1]);
        }
        if(n>2)
        {
            printf("%d \t %d\t",a[0],a[1]);
            for(i=2;i<n;i++)
            {
                a[i]=a[i-1]+a[i-2];
                printf("%d\t",a[i]);
            }
        }
        }
```

3.3Two Dimensional arrays (Multidimensional array)

- In two dimensional arrays, elements will be arranged in rows and columns.
- To identify two dimensional arrays, we will use two indices (say i and j) where I index indicates rownumber and j index indicates column number.

Declaration of two-dimensional array

data_type array_name[exp1][exp2]; Or data_type array_name[row_size][column_size];

- **data_type** can be int, float,char,double.
- **array_name** is the name of the array.
- **exp1 and exp2** indicates number of rows and columns

For example:

int a[2][3];

✓ The above statements allocate memory for 3*4=12 elements i.e 12*2=24 bytes.

Initialization of two-dimensional array

Assigning or providing the required values to a variable before processing is called initialization.

Data_type array_name[exp1][exp2]={

{a1,a2,….an},
{b1,b2, .bn},
…….
……..
{z1,z2,…zn}

};

Data type can be int, float etc.

- exp1 and exp2 are enclosed within square brackets.
- both exp1 and exp2 can be integer constants or constant integer expressions (number of rows andnumber of columns).

a1 to an are the values assigned to 1st row.

- b1 to bn are the values assigned to 2nd row and

so on.Example:

```
int a[3][3]={
        {10,20,30},
        {40,50,60},
        {70,80,90}
};
```

10	**20**	**30**
40	**50**	**60**
70	**80**	**90**

Partial Array Initialization

- If the number of values to be initialized is less than the size of array, then the elements areinitialized from left to right one after the other.
- The remaining locations initialized to zero automatically.
- Example:

```
int a[3][3]={
{10,20},
{40,50},
{70,80}
};
```

Write a c program to copy one 2d array in to another 2d array

```
#include<stdio.h>
#include<conio.h>
void main()
{
int m,n,i,j,a[3][3],b[3][3];
clrscr();
printf("enter number of rows and columns\n");
scanf("%d %d",&m,&n);
printf("enter array  a elements\n");
for(i=0; i<m; i++)
{
for(j=0; j<n; j++)
{
scanf("%d", &a[i][j]);
}
}

for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
```

```
                    {
                    b[i][j]=a[i][j];
                    }
                    }
                    printf("matrix b  is \n");
                    for(i=0;i<m;i++)
                    {
                    for(j=0;j<n;j++)
                    {
                    printf("%d\t",b[i][j]);
                    }
                    printf("\n");
                    }
    }
```

Write a c program to find biggest element in a matrix or 2D array.

```
#include<stdio.h>void main()
{
        int m,n,i,j,a[3][3];clrscr();
        printf("enter number of rows and columns\n");scanf("%d %d",&m,&n);
        printf("enter array elements\n");for(i=0;i<m;i++)
        {
                for(j=0;j<n;j++)
                {
                        scanf("%d",&a[i][j]);
                }
        }
        big=a[0][0]; for(i=0;i<m;i++)
        {
                for(j=0;j<n;j++)
                {
                        if(big>a[i][j])
                        big=a[i][j];
                }
        }
        printf("big is %",big);
}
```

Write a C program to implement Matrix Multiplication

```
#include<stdio.h>void main()
{
        int m,n,i,j,sum,p,q,k,a[3][3],b[3][3],c[3][3];
        printf("enter number of rows and columns of matrix a \n");scanf("%d %d",&m,&n);
        printf("enter number of rows and columns of matrix b \n");scanf("%d %d",&p,&q);
        if(n!=p)
        {
                printf("multiplication not possible\n"):exit(0);
        }

        printf("enter matrix a elements\n");for(i=0;i<m;i++)
        {
                for(j=0;j<n;j++)
                {
                        scanf("%d",&a[i][j]);
                }
        }

        printf("enter array b elements\n");for(i=0;i<p;i++)
        {
                for(j=0;j<q;j++)
                {
                        scanf("%d",&b[i][j]);
                }
        }

        for(i=0;i<m;i++)
        {
                for(j=0;j<q;j++)
                {
                        {
                                c[i][j]=0; for(k=0;k<n;k++)
                                {
                                        c[i][j]= c[i][j]+a[i][k]*b[k][j];
                                }

                }
        }
        printf("resultant matrix a is \n");

    for(i=0;i<m;i++)
```

```
{
for(j=0;j<n;j++)
{
printf("%d\t",a[i][j]);
}
printf("\n");
}
printf("resultant matrix a is \n");
for(i=0;i<p;i++)
{
for(j=0;j<q;j++)
{
printf("%d\t",b[i][j]);
}
printf("\n");
}
printf("resultant matrix a is \n");
for(i=0;i<m;i++)
{
for(j=0;j<q;j++)
{
printf("%d\t",c[i][j]);
}
printf("\n");
}
}
```

Write a program to find sum of each row and sum of each column

```
#include<stdio.h>
void main()
{
int m,n,i,j,rsum,csum,a[3][3];
printf("enter number of rows and columns\n");
scanf("%d %d",&m,&n);
printf("enter array elements\n");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
                    scanf("%d",&a[i][j]);
```

```
                for(j=0;j<n;j++)
                {
                rsum=rsum+a[i][j];
                }
                printf("sum is %d",rsum);
                }
                for(i=0;i<m;i++)
                {
                csum=0;
                for(j=0;j<n;j++)
                {
                csum=csum+a[i][j];
                }
                printf("sum is %d",csum);
                }
7   }
     #include<stdio.h>void main()
{
      int m,n,i,j,sum=0,a[3][3];
      printf("enter number of rows and columns\n");scanf("%d %d",&m,&n);
      printf("enter array elements\n");for(i=0;i<m;i++)
      {
            for(j=0;j<n;j++)
            {
                  scanf("%d",&a[i][j]);
            }
      }
      for(i=0;i<m;i++)
      {

            sum=sum+a[i][j];

      }
      printf("sum is %d",rsum);
}
```

Real-world Applications of 1-D Arrays

- Lists and Sequences: Storing a list of names, numbers, or any type of data that needs to be organized sequentially.
- Counting and Accumulation: Keeping track of counts, scores, or incremental values.
- Data Retrieval: Accessing elements of a database or dataset.
- Mathematical Operations: Performing mathematical calculations using arrays.
- Text Processing: Storing and processing text or characters.

Real-world Applications of 2-D and 3-D Arrays

- Image Processing: Storing and manipulating pixel values in images.
- Game Development: Representing game boards, maps, and 3D environments.
- Scientific Computing: Storing and processing data from experiments and simulations.
- Matrices in Mathematics: Solving linear equations, transformations, and more.
- Databases: Organizing data in tabular form.

3.4 Strings

3.4.1 C Strings

In C language a string is group of characters (or) array of characters, which is terminated by delimiter \0 (null).

Thus, C uses variable-length delimited strings in programs.

3.4.2 Declaring Strings

C does not support string as a data type. It allows us to represent strings as character arrays. In C, a string variable is any valid C variable name and is always declared as an array of characters.

Syntax: - char string_name[size];

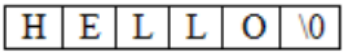
The size determines the number of characters in the string name.

Ex: - char city[10];
char name[30];

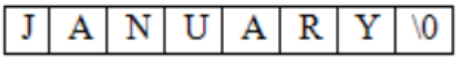
3.4.3 Initializing strings

There are several methods to initialize values for string variables.

Ex:- char str1[6]="HELLO";



Ex:- char month[]="JANUARY";



Ex:- char city[8]="NEWYORK";

char city[8]={„N","E","W","Y","O","R","K","\0"};

The string city size is 8 but it contains 7 characters and one-character space is for NULL terminator.

3.4.4 Operations on Strings

In this section, we will learn about different operations that can be performed on character arrays. But before we start with these operations, we must understand the way arithmetic operators can be applied to characters.

In C, characters can be manipulated in the same way as we do with numbers. When we use a character constant or a character variable in an expression, it is automatically converted into an integer value, where the value depends on the local character set of the system. For example, if we write

```
int i;
char ch 'A';
i = ch;
printf("%d", i);
// Prints 65, ASCII value of ch is 'A'
```

C also enables programmers to perform arithmetic operations on character variables and character constants. So, if we write

```
int i;
char ch = 'A';
i = ch + 10;
printf("%d", i);
```

// Prints 75, ASCII value of ch that is 'A' + 10

Character variables and character constants can be used with relational operators as shown in the example below,

```
char ch = 'C';

if (ch >= 'A' && ch <= 'Z')

printf("The character is in upper case");
```

1. Finding the Length of a String

```c
// C pro
#include <stdio.h>
#include <string.h>

int main()
{
    char Str[1000];
    int i;

    printf("Enter the String: ");
    scanf("%s", Str);

    for (i = 0; Str[i] != '\0'; ++i);

    printf("Length of Str is %d", i);

    return 0;
}
```

C:\Users\chira\OneDrive\Documents\C_DS prog\c length of string.c

Output:

```
Enter the String: Hello
Length of Str is 5
Process returned 0 (0x0)   execution time : 3.391 s
Press any key to continue.
```

2.Converting Characters of a String into Upper Case

```c
#include <stdio.h>
#include <conio.h>
#include <string.h>
int main()
{
    char s[100];
    int i;
    printf("Enter a string : ");
    gets(s);
    for (i = 0; s[i]!='\0'; i++) {
       if(s[i] >= 'a' && s[i] <= 'z') {
          s[i] = s[i] -32;
       }
    }
    printf("String in Upper Case = %s", s);
    return 0;
}
```

Output:

```
Enter a string : sapthagiri nps university
String in Upper Case = SAPTHAGIRI NPS UNIVERSITY
Process returned 0 (0x0)   execution time : 13.072 s
Press any key to continue.
```

3.Write a program to convert characters of a string into lower case.

```c
#include <stdio.h>
#include <ctype.h>


int main ()
{
    int i = 0;
    char c;
    char str[] = "SAPTHAGIRI NPS UNIVERSITY";

    while(str[i])
    {
        putchar(tolower(str[i]));
        i++;
    }

    return(0);
}

```

Output:

```
sapthagiri nps university
Process returned 0 (0x0)   execution time : 0.483 s
Press any key to continue.
```

The string Input/ Output functions with example:

Reading and Writing strings: -

C language provides several string handling functions for input and output.

String Input/Output Functions: - C provides two basic methods to read and write strings.

Using formatted input/output functions and using a special set of functions.

Reading strings from terminal: -

formatted input function: - scanf can be used with %s format specification to read a string.

Ex: - char name[10];
scanf("%s",name);

Here don't use '&' because name of string is a pointer to array.
The problem with scanf is that it terminates its input on the first white space it finds.
Ex: - NEW YORK
Reads only NEW (from above example).

Unformatted input functions: -

getchar(): - It is used to read a single character from keyboard. Using this function repeatedly we may read entire line of text

Ex: - char ch="z";
ch=getchar();

(2) gets(): - It is more convenient method of reading a string of text including blank spaces.

Ex: - char line[100];
gets(line);

Writing strings on to the screen: -
(1) Using formatted output functions: - printf with %s format specifier we can print strings in different formats on to screen.

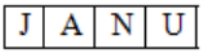
Ex: - char name[10];

printf("%s",name);

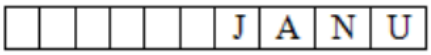
Ex: - char name[10];

printf("%0.4",name);

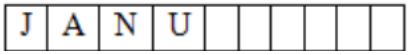
/* If name is JANUARY prints only 4 characters ie., JANU */



Printf("%10.4s",name);



printf("%-10.4s",name);



3.4.5 Arrays of strings

✓ Declare and Initialize Array of strings in C:

We have array of integers, array of floating-point numbers, etc... similarly we have array of strings also.

Collection of strings is represented using array of strings.

Declaration: -

Char arr[row][col];

where,

arr - name of the array

row - represents number of strings

col - represents size of each string

Initialization: -

char arr[row][col] = {list of strings};

Example: - char city[5][10] = { “DELHI”, “CHENNAI”, “BANGALORE”, “HYDERABAD”, “MUMBAI” };

D	E	L	H	I	\0				
C	H	E	N	N	A	I	\0		
B	A	N	G	A	L	O	R	E	\0
H	Y	D	E	R	A	B	A	D	\0
M	U	M	B	A	I	\0			

In the above storage representation memory is wasted due to the fixed length for all strings.

3.5 Storage Classes

- There are following storage classes which can be used in a C Program:
  - *i.* Automatic variables (Local variable)
  - *ii.* External variables
  - *iii.* Static variables
  - *iv.* Register variables

3.5.1 Local variables (automatic variables)

- These are the variables which are defined within a function.
- These variables are also called as automatic variables.
- The scope of these variables is limited only to the function in which theyare declared and cannot be accessed outside the function.

Example 1: (Automatic Variables)

```
int main()
{
int n1; // n1 is a local variable to main()
}
void func()
{
int n2;  // n2 is a local variable to func()
}
```

- In the above example, n1 is local to main() and n2 is local to func().
- This means you cannot access the n1 variable inside func() as it only exists inside main().
- Similarly, you cannot access the n2 variable inside main() as it only exists inside func().

3.5.2 Global variables (External variable)

- These are the variables which are defined before all functions in global areaof the program.
- Memory is allocated only once to these variables and initialized to zero.
- These variables can be accessed by any function and are alive and active throughout the program.
- Memory is deallocated when program execution is over.

Example: Global Variable

```
#include <stdio.h>
void display();
int n = 5;   // global variable
int main()
{
    ++n;
    display();
    return 0;
}
void display()
{
    ++n;
    printf("n = %d", n);
}
```

Output: n = 7

3.5.3 Static variables

- The variables that are declared using the keyword static are called static

variables.

- The static variables can be declared outside the function and inside thefunction. They have the characteristics of both local and global variables.
- Static can also be defined within a function.

Ex:

static int a,b;

Example Program to illustrate static variables

```
#include <stdio.h>

void display();
int main()
{
   display();
   display();
}
void display()
{
   static int c = 1;
   c += 5;   //  c = c + 5
   printf("%d  ",c);
}
```

OUTPUT:

```
6  11
```

- ✓ During the first function call, the value of ‘c’ is initialized to 1. Its value is increased by 5.
- ✓ Now, the value of ‘c’ is 6, which is printed on the screen.
- ✓ During the second function call, ‘c’ is not initialized to 1 again. It's because ‘c’ is a static variable.
- ✓ The value ‘c’ is increased by 5. Now, its value will be 11, which is printed on the screen.

3.5.4 Register variables

ii. Any variables declared with the qualifier register is called a register variable.

iii. This declaration instructs the compiler that the variable under use is to bestored in one of the registers but not in main memory.

iv. Register access is much faster compared to memory access. Ex: register int a;

Example 1: (Register Variables)

```
#include <stdio.h>
int main()
{
register int a; // variable a is allocated memory in the
                   CPU register. The initial default value of a
                   is 0.
printf("%d",a);
}
```

OUTPUT:

0

Example 2: (Register Variables)

```
#include <stdio.h>
int main()
{
   register int count = 0;
    while (count < 5)
    {
      printf("Count: %d\n", count);
      count++;
   }
   return 0;
}
```

OUTPUT:

Count: 0
Count: 1
Count: 2
Count: 3
Count: