Programming in C

By

Dr Ramkumar Krishnamoorthy

kramdharma@gmail.com

Fundamentals of C

Unit II

Contents

Branching
Looping
Goto, Continue
Arrays
Strings

Advanced Features
Formatted Input and Output
Getch()
Getche()
Gets()

Unit II – Chapter I

Array

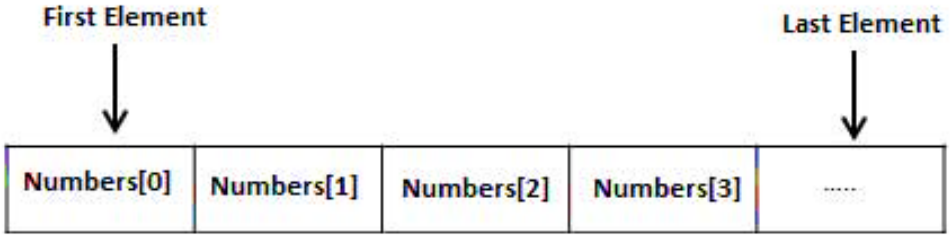
Arrays a kind of data structure that can **store a fixed-size sequential collection of elements of the same type**.

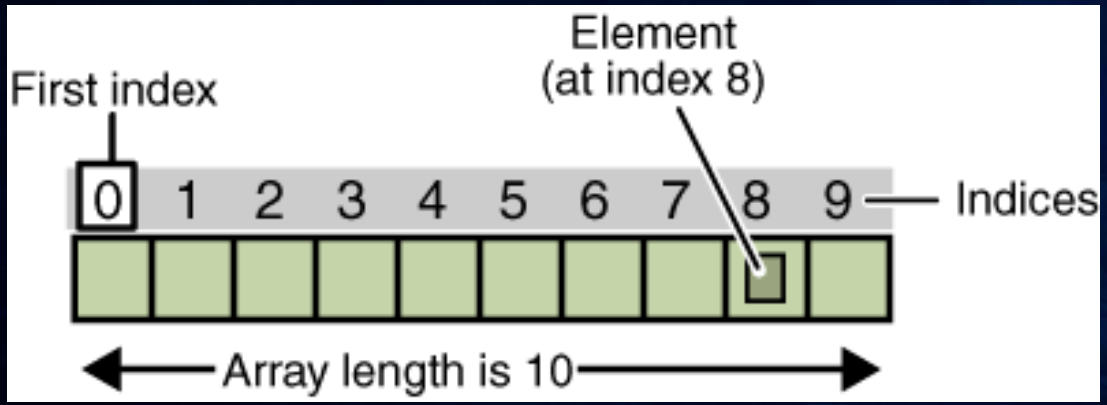
Instead of declaring individual variables, such as number0, number1, ..., and number99, users declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables.

A specific element in an array is accessed by an index.

Array

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





Array – Example

Arrays in C

Data_Type Array_Name [Array_Size]

float Student [7]

	1	2	3	4	5	6	7
	Float Value	Float Value	Float Value	Float Value	Float Value	Float Value	Float Value
INDEX →	Student [0]	Student [1]	Student [2]	Student [3]	Student [4]	Student [5]	Student [6]
BASE ADDRESS →	1000	1008	1016	1024	1032	1040	1048

Contiguous Memory Allocation
(Size of Float is 8 Byte)

Array

Initialization of Array in Compile Time

```c
#include<stdio.h>
int main()
{
    //integer array
    int student[5] = {60, 70, 65, 80, 85};
    //float array
    float student[5] = {60.0, 75.5, 80.0, 67.0, 83.5};
    return 0;
}
```

Array

Initialization of Array in Run Time

```c
#include<stdio.h>
int main()
{
    int n;                          //size of array
    scanf("%d",&n);
    int student[n];                 //array
    for(int i=0; i<n; i++)
        scanf("%d",&student[i]);
    return 0;
}
```

Example for One Dimensional Array

```c
#include<stdio.h>
int main()
{
    int n;                                          //size of array
    scanf("%d",&n);
    int student[n];                                 //decleration of array
    int average, sum=0;
    for(int i=0; i<n; i++)
    {
        scanf("%d",&student[i]);                    //initialization of array
    }
    student[2] = student[2] + 7;                    //accessing 3rd element
    student[4] = student[4] + 7;                    //accessing 5th element

    for(int i=0; i<n; i++)
    {
        sum  = sum + student[i];                    //accessing elements via loop
    }
    average = sum / n;                              //calculating average

    for(int i=0; i<n; i++)
    {
        printf("Marks of student %d : %d\n", i+1,student[i]);
    }
    printf("Sum is : %d\n",sum);
    printf("Average is : %d",average);
    return 0;
}
```

Output

```
5
60 70 83 65 88
Marks of student 1 : 60
Marks of student 2 : 70
Marks of student 3 : 90
Marks of student 4 : 65
Marks of student 5 : 95
Sum is : 380
Average is : 76
```

Array Declaration Example

```
int student [5] ;
```

Student - 1	Student - 2	Student - 3	Student - 4	Student - 5
60	70	83	65	88
student [0]	student [1]	student [2]	student [3]	student [4]

student[2] = student[2] + 7

Student - 3		Student - 3		
90	=	83	+	7
student [2]		student [2]		

student[4] = student[4] + 7

Student - 5		Student - 5		
95	=	88	+	7
student [4]		student [4]		

Array Example with Memory Allocation

```c
#include<stdio.h>
int main()
{
    int student[5] = {60, 70, 83, 65, 88};
    printf("Size of integer in this compiler is %d\n",sizeof(int));
    for(int i=0 ;i<5 ;i++)
    {
        printf("Address student[%d] is %d\n", i, &student[i]);
    }
    return 0;
}
```

The program given below proves that the Array follows contagious memory allocation

Output

```
Size of integer in this compiler is 4
Address student[0] is 6422280
Address student[1] is 6422284
Address student[2] is 6422288
Address student[3] is 6422292
Address student[4] is 6422296
```

Types in Arrays

Arrays are of different types -

i. **One-dimensional array :** A structured collection of elements, all of the same type, that is given a single name. Each element is accessed by an index that indicates the component's position within the collection.

ii. **Two-dimensional array :** A two dimensional array is a collection of elements, all of the same type, structured in two dimensions. Each element is accessed by a pair of indices that represent the element's position in each dimension.

iii. **Multi-dimensional array :** A collection of elements, all of the same type, ordered on N dimensions ($N \geq 1$). Each element is accessed by N indices, each of which represents the element's position within that dimension.

One Dimensional Array

Definition

One-dimensional array : A structured collection of components, all of the same type, that is given a single name. Each component (array element) is accessed by an index that indicates the component's position within the collection.

The general form of array declaration is :

Data_type *Array-Name* [**SIZE**];

```
float   height[5];
int     testscore[10];
```

height

height[0]	5.6
height[1]	4.8
height[2]	6.9
height[3]	4.3
height[4]	3.4

testscore

testscore[0]	23
testscore[1]	45
testscore[2]	90
testscore[3]	39
testscore[4]	87
testscore[5]	65
testscore[6]	34
testscore[7]	80
testscore[8]	78
testscore[9]	98

One Dimensional Array

Array Initialization

A special feature of arrays is that they can be initialized when they are declared. This is done by following the array name and dimension with an equal sign, followed by a pair of braces. These braces contain a series of constant values separated by commas.

Example int marks[5]= { 56, 67, 15, 98, 12 };

In this declaration, *marks[0]* is initialized to 56, *marks[1]* is initialized to 67 and so on. There must be at least one initial value between the braces. If you specify too many initial values, an error message will be displayed. If you specify too few, the remaining array elements are initialized to zero.

Example int marks[5]= { 56 };

In this declaration, *marks[0]* is initialized to 56, *marks[1]* to *marks[4]* are initialized to 0.

Example One Dimensional Array

Program A C program to read and print a one-dimensional array.

```
#include <stdio.h>
main()
{
      int   a[10];      /*  max of elements in the array is 10 */
      int   i,n;

      printf("\nEnter number of elements in  the array ");
      scanf("%d",&n);

      /*    Input array elements                    */
      printf("\nEnter the array elements\n");
      for(i=0   ;   i<n   ;   i++)
            scanf("%d",&a[i]);

      /*    printing array elements */
      printf("\nArray elements are\n");
      for(i=0   ;    i<n   ; i++)
            printf("\t%d",a[i]);
}
```

OUTPUT

```
Enter number of elements in  the array  5
Enter the array elements
23
55
28
45
63
Array elements are
23      55      28      45      63
```

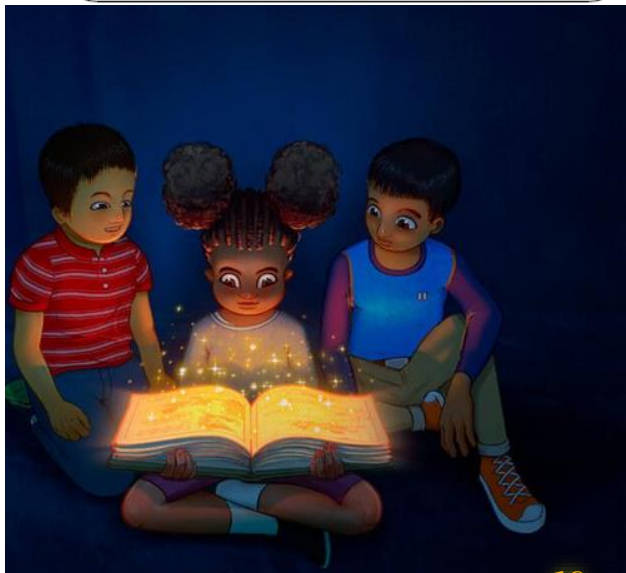
Largest Element Searching

Program A C program to find the largest element in an array and position of it's occurrence.

```
#include <stdio.h>
main()
{
      int   a[100];
      int   largest,position,num,index;
      printf("Enter number of elements in the array ");
      scanf("%d",&num);
      for( index=0; index <num ;index++)
            scanf("%d",&a[index]);

      largest=a[0];   /* Assign first element as largest      */
      position = 0;   *      Position of largest element      */
      for( index = 1 ;      index<num ;       index++)
      {
            if(largest < a[index])
           {
                 largest = a[index];
                 position = index ;
           }
      }

      printf("Largest element is      %d \n",largest);
      printf("Position in the array %d\n",position+1);
}
```

OUTPUT

```
Enter number of elements in the array 6
8 45 2 56 433 5
Largest element in the array is 433
Largest element's position in the array 5
```



Maximum and Minimum in the Array

Program Write a program to find the maximum and minimum elements in an array having N elements.

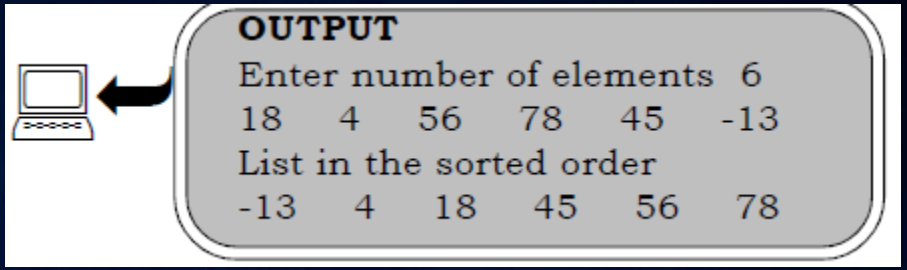
```
#include <stdio.h>
main()
{
        int   a[100];          /*      Array declaration      */
        int   i,n,max,mini;
        printf("Enter number of elements in the array ");
        scanf("%d",&n);
        for(  i=0  ;  i<n  ;  i++)
              scanf("%d",&a[i] );
        max =a[0];
        mini=a[0];
        for( i=1  ;  i<n ;  i++)
        {
              if(max < a[i] ) max=a[i];
              if(mini > a[i] )  mini=a[i];
        }
        printf("\nMaximum element in the array  is %d",max);
        printf("\nMinimum element in the array is %d", mini);
}
```

OUTPUT

```
Enter number of elements in the array  5
34     62     13     3     67
Maximum element in the array  is  67
Minimum element in the array is    3
```

Example for Sorting

```
#include <stdio.h>
main()
{
        int a[100];
        int n,i,j,t;
        printf("Enter number of elements ");
        scanf("%d",&n);
        for(i=0 ; i<n ; i++)
        scanf("%d",&a[i]);
        for(i=1 ; i<n ; i++)/* Pass number */
        for(j=0 ; j<n-i ; j++)/* Comparison */
                if(a[j] > a[j+1])
                {
                        t=a[j];
                        a[j]=a[j+1];
                        a[j+1]=t;
                }
        printf("\nList in the sorted order\n");
        for(i=0;i<n;i++)
        printf("\t%d",a[i]);
}
```



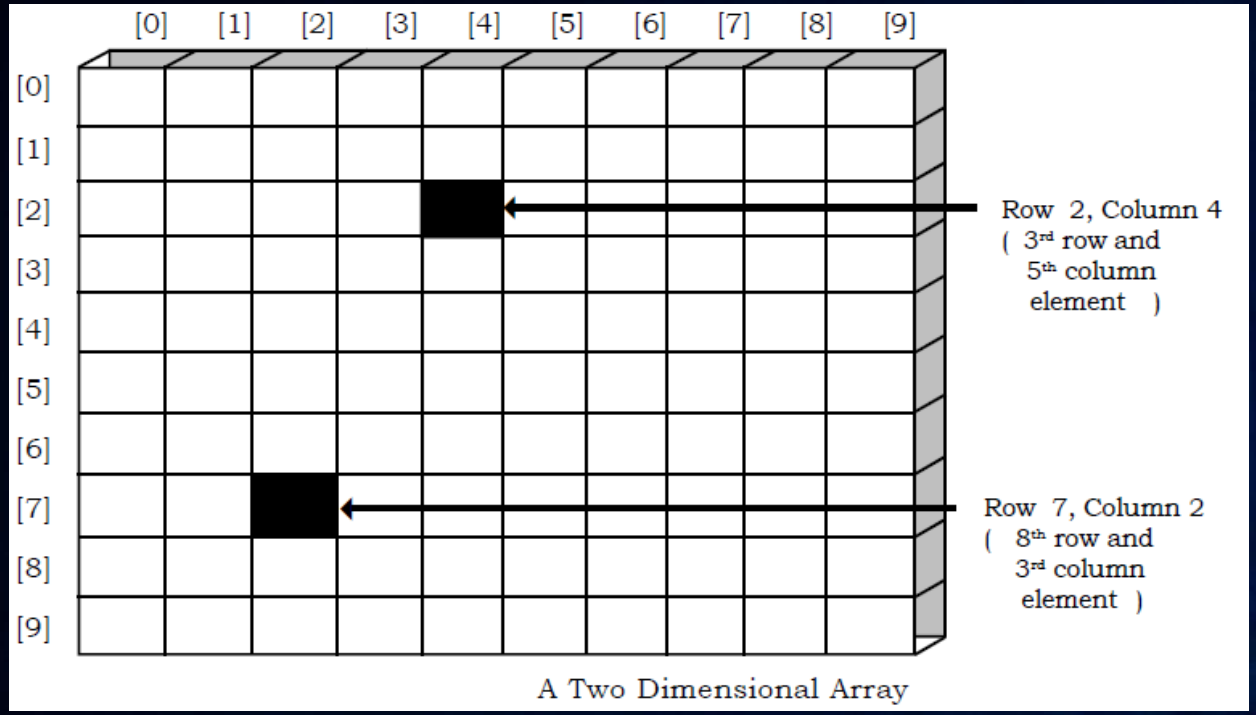
Two Dimensional Array

- A two dimensional array is used to represent elements in a table with rows and columns, provided each element in the table is of
- the same type. An element in a two-dimensional array is accessed by specifying the row and column indicies of the item in an array.

Two-dimensional arrays are called **matrices** in mathematics and **tables** in business applications; hence two-dimensional arrays are sometimes called **matrix arrays**.

Definition
Two-dimensional array : A collection of components, all of the same type and same name structured in two dimensions. Each component (array element) is accessed by a pair of indices that representats the component's position in each dimension.

Two Dimensional Array Structure



A Two Dimensional Array

Two Dimensional Array

Declaration Syntax

Data type arrayname [row size][cols size]

Declaration Example

int array [5][5];

Data_type *array_name* [**ROW SIZE**][**COLUMN SIZE**];

Example Consider the following declaration

int a[10][10];

a is a two dimensional array which contains 10 rows and 10 columns.

Two Dimensional Array

Example

```
int     matrix[2][3]={1,2,3,4,5,6};
```

matrix is a two dimensional array which contains 2 rows and 3 columns and these assignments would be

```
matrix[0][0] = 1    matrix[0][1] = 2    matrix[0][2] = 3
matrix[1][0] = 4    matrix[1][1] = 5    matrix[1][2] = 6
```

Accessing the Elements Two Dimensional Array

Example Consider the following declaration :

```
int  a[3][3];
```

(i) The following **for** loops would read in 9 numbers and store them in the array **a**.

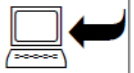
```
for(  row=0 ;   row < 3;  row++)
      for(column=0;  column<3  ; column++)
                  scanf("%d",&a[row][column] );
```

(ii) The following **for** loops would print matrix elements in the matrix format.

```
for( row=0 ;   row < 3;  row++)
{
      for( column=0 ;  column < 3  ;  column++)
                  printf("\t%d",a[row][column] );
            printf("\n");
}
```

Example Matrix Display

```
#include <stdio.h>
main()
{
        int     a[10][10];
        int     n,i,j;
        printf("Enter order of the matrix ");
        scanf("%d",&n);

        printf("Enter matrix elements "); /*     Row major reading    */
        for(i=0  ;   i<n   ;   i++)       /*     Row number           */
                for(j=0 ;    j<n ;    j++)       /*     Column number */
                        scanf("%d",&a[i][j]);


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



OUTPUT

Enter order of the matrix 3
Enter matrix elements
4 5 7
2 3 6
1 3 5
Matrix A is
4 5 7
2 3 6
1 3 5

Example Matrix Transpose

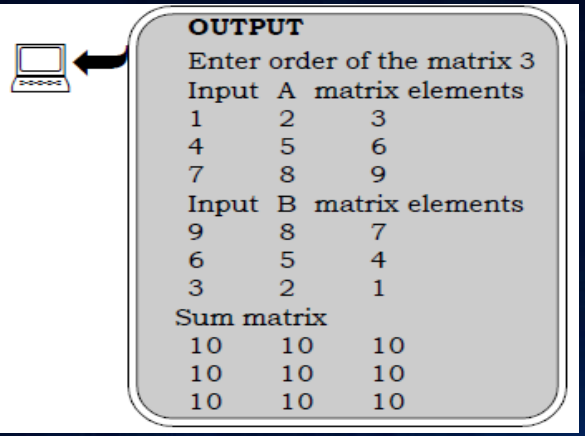
```
#include <stdio.h>
main()
{
        int     a[10][10],b[10][10]; /*      Array declaration     */
        int     n,i,j;
        printf("Enter order of the matrix ");
        scanf("%d  %d",&n,&m);
        printf("Enter matrix elements ");
        for(i = 0;    i<n;     i++)
             for(j = 0;    j<m;      j++)
             {
               scanf("%d",&a[i][j]);
               b[j][i]=a[i][j];
              }
        printf("Matrix  A\n");
        for(i=0;  i<n;   i++)
        {
               for(j = 0;  j<m;  j++)
               printf("%d\t",a[i][j]);
               printf("\n");
        }
        printf("Transpose  of A\n");
        for(i=0;    i<m;     i++)
        {
               for(j=0    ;    j<n;     j++)
               printf("%d\t",b[i][j]);
               printf("\n");
        }
}
```

OUTPUT

```
Enter order of the matrix
2 3
Enter matrix elements
1       2       3
4       5       6
Matrix  A
1       2       3
4       5       6
Transpose of A
1       4
2       5
3       6
```

Example Matrix Addition

```
#include <stdio.h>
main()
{
        int    a[10][10],b[10][10],c[10][10],i,j,n;

        printf("Enter order of the matrix ");
        scanf("%d",&n);

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

        printf("\nInput  B  matrix elements\n");
        for(i=0;    i<n;     i++)
                for(j=0;     j<n;    j++)
               scanf("%d",&b[i][j]);

        for(i=0;     i<n;     i++)
            for(j=0;    j<n;    j++)
              c[i][j]=a[i][j]+b[i][j];

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



OUTPUT

```
Enter order of the matrix 3
Input  A  matrix elements
1       2       3
4       5       6
7       8       9
Input  B  matrix elements
9       8       7
6       5       4
3       2       1
Sum matrix
 10      10      10
 10      10      10
 10      10      10
```

Matrix Multiplication

```
#include <stdio.h>
main()
{
        int a[10][10],b[10][10],c[10][10],i,j,n;
        printf("Enter order of the matrix ");
        scanf("%d",&n);
        printf("\nInput A matrix elements\n");
        for(i=0; i<n; i++)
        for(j=0; j<n; j++)
                scanf("%d",&a[i][j]);
        printf("\nInput B matrix elements\n");
        for(i=0; i<n; i++)
        for(j=0; j<n; j++)
                scanf("%d",&b[i][j]);
       /* Find the product of two matrices */
        for(i=0; i<n; i++)
        for(j=0; j<n; j++)
        {
                c[i][j]=0;
                for(k=0; k<n; k++)
                c[i][j] = c[i][j] + a[i][k]*b[k][j];
        }
        printf("\nProduct matrix \n");
        for(i=0; i<n; i++)
        {
        for(j=0; j<n; j++)
        printf("%d\t",c[i][j]);
        printf("\n");
        }
}
```

OUTPUT

```
Enter order of the matrix 3
Input  A  matrix elements
1       2       3
4       5       6
7       8       9
Input  B  matrix elements
9       8       7
6       5       4
3       2       1
Product matrix
30      24      18
84      69      54
138     114     70
```

Multi Dimensional Array (More than One Dimension

Definition

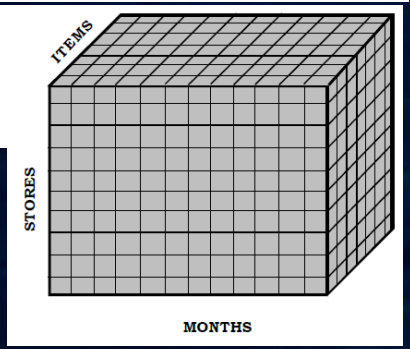
Multidimensional array : Collection of elements, all of the same type and same name, ordered on N dimensions (N≥1). Each element is accessed by N indicies, each of which represents the element's position within that dimension.

The general form of the multidimensional array is :

type array_name[s_1][s_2][s_3].....[s_m];

where $\mathbf{s}_i$ is the size of the i^{th} dimension.

```
#define  NUM_ITEMS    7
#define  NUM_STORES  10
int      sales[ NUM_STORES ] [ 12  ][ NUM_ITEMS ];
```



Strings

Definition

String : A string is a sequence of characters.

Syntax : `char variable_name[ size ];`

where ***variable_name*** is a string variable and
size is the number of characters in the string.

Example Consider the following string declaration:

char name[12];

When the string "Programming" is assigned to a string variable **name,** the characters are stored in the memory as follows:

char name [12] = "Programming";

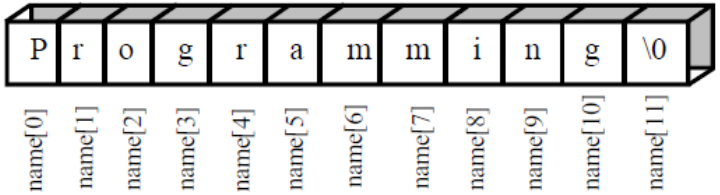
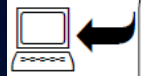


Figure Memory representation of a string

```c
#include<stdio.h>
void main ()
{
    char s[20];
    printf("Enter the string?");
    scanf("%s",s);
    printf("You entered %s",s);
}
```

Strings

```
#include <stdio.h>
main()
{
        char    name[20];
        printf("Enter a string\n");
        gets(name);
        printf("\nEntered string was %s", name);
}
```



Output 1

```
Enter a string
Srinagar
Entered string was Srinagar
```

Output 2

```
Enter a string
Sri nagar
Entered string was Sri nagar
```

String Operations

- Length (number of characters in a string)
- Concatenation (Adding two or more strings)
- Comparing two strings
- Substring (Extract substring from the given string)
- Copy (copies one string over another)

NOTE

☞ To use all the string operations in C you must include **string.h** library header file in the program.

String Operations

Sr.No.	Function & Purpose
1	**strcpy(s1, s2);** Copies string s2 into string s1.
2	**strcat(s1, s2);** Concatenates string s2 onto the end of string s1.
3	**strlen(s1);** Returns the length of string s1.
4	**strcmp(s1, s2);** Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.
5	**strchr(s1, ch);** Returns a pointer to the first occurrence of character ch in string s1.
6	**strstr(s1, s2);** Returns a pointer to the first occurrence of string s2 in string s1.

String Operations

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

int main () {

   char str1[12] = "Hello";
   char str2[12] = "World";
   char str3[12];
   int  len ;

   /* copy str1 into str3 */
   strcpy(str3, str1);
   printf("strcpy( str3, str1) :  %s\n", str3 );

   /* concatenates str1 and str2 */
   strcat( str1, str2);
   printf("strcat( str1, str2):   %s\n", str1 );

   /* total lenghth of str1 after concatenation */
   len = strlen(str1);
   printf("strlen(str1) :  %d\n", len );

   return 0;
}
```

```
strcpy( str3, str1) :  Hello
strcat( str1, str2):   HelloWorld
strlen(str1) :  10
```

String Operations

The function `strncmp` compares between two strings, returning the number 0 if they are equal, or a different number if they are different. The arguments are the two strings to be compared, and the maximum comparison length. There is also an unsafe version of this function called `strcmp`, but it is not recommended to use it. For example:

```
char * name = "John";

if (strncmp(name, "John", 4) == 0) {
    printf("Hello, John!\n");
} else {
    printf("You are not John. Go away.\n");
}
```

String Operations

```
#include<string.h>
void main()
{
        char name[30];
        char *proverb="KNOWLEDGE IS POWER";
        char *chocolate="Dairy Milk";
        char *milk="Aavin";
        char *s1="Ram";
        char *s2="Raj";
        char *s3="Hai";
        char *s4="Hai";
        int len;
        clrscr();
        printf("Enter the name:\n");
        gets(name);
        len=strlen(name);

        printf("Length string:%d\n\n",len);
        printf("Upper case string:%s\n\n",strupr(name));
        printf("Lower case string:%s\n\n",strlwr(name));
        printf("Reverse string:%s\n\n",strrev(proverb));
        printf("Copied string:%s\n\n",strcpy(chocolate,milk));
        printf("Concatenation string:%s\n\n",strcat(s1,s2));
        printf("Comparison string:%s\n\n",strcmp(s3,s4));
        getch();
}
```

Output:

```
Enter the name
sai ram
Length string:6
Upper case string:SAI RAM
Lower case string:sai ram
Reverse string:REWOP SI EGDELWONK
Copied string: Aavin
Concatenation string: RamRaj
Comparison string:0
```

String Length and Concatenation Example

```c
#include <stdio.h>
#include <string.h>
#include <conio.h>
void main()
{
        char a[30];
        int length;
        clrscr();

        printf("Enter a string to calculate its length\n");
        gets(a);

        length = strlen(a);
        printf("Length of the string = %d\n", length);

        getch();
}
```

```
Enter a string to calculate its length: Jain University
Length of the string = 15
```

```c
#include <stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
        char s1[20];
        char s2[20];
        clrscr();
        printf("Enter the first string : ");
        scanf("%s", s1);
        printf("\nEnter the second string :");
        scanf("%s",s2);
        strcat(s1,s2);
        printf("The concatenated string is : %s",s1);
        getch();
}
```

```
Enter the first string :
Programming In C
Enter the second string :
I Can eat C
The concatenated string is :
Programming In C I Can eat C
```

String Operations

FUNCTION	DESCRIPTION
strcat	Adds the characters of one string to another.
strchr	Returns the position of a specified character in the string.
strcmp	Compares two strings.
strcmpi	Compare two strings; not case sensitive.
strcpy	Copies one string or string literal, to another.
strcspn	Returns the position of a character in the string from a specified character set.
strlen	Calculates the string length.
strlwr	Converts a string to lowercase.
strncat	Appends specified characters from one string to another.
strncmp	Compares specified characters of two strings.
strncpy	Copies specified characters from one string to another.
strnset	Changes specified characters in a string to another character.
strrev	Reverses the characters in a string.
strstr	Finds one string within another.
strupr	Converts a string to uppercase.

Arrays of Strings

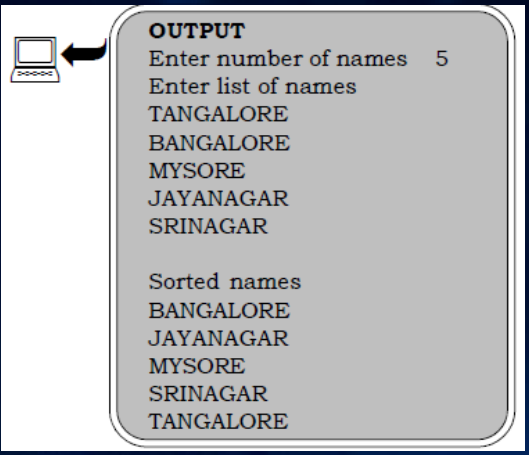
```
char    name[8][20];
```

declares an array of 8 strings, each of which can hold maximum of 20 characters. An array of strings appears in memory as shown in Figure

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
name [0]	D	R	U	V	A	\0														
name [1]	G	U	L	N	D	A	R	\0												
name [2]	S	H	I	V	A	J	I	\0												
name [3]	C	H	I	N	T	A	N	\0												
name [4]	M	Y	S	O	R	E	\0													
name [5]	B	A	N	G	A	L	O	R	E	\0										
name [6]	K	A	R	A	N	\0														
name [7]	H	A	R	I	S	H	\0													

Arrays of Strings (Sorting Names)

```
#include<stdio.h>
#include <string.h>
main()
{
        char   name[10][20];
        char   tempname[20];
        int    n,i,j;
        printf("Enter number of names ");
        scanf("%d",&n);
        printf("Enter list of names\n ");
        for(i=0;i<n;i++)
              gets(name[i]);
        for(i=1;i<n;i++)
              for(j=0;j<n-i;j++)
              if(strcmp(name[j],name[j+1]) > 0)
              {
                    strcpy(tempname,name[j]);
                    strcpy(name[j],name[j+1]);
                    strcpy(name[j+1],tempname);
              }
        printf("\nSorted names");
        for(i=0;i<n;i++)printf("\n%s",name[i]);
}
```



OUTPUT

```
Enter number of names    5
Enter list of names
TANGALORE
BANGALORE
MYSORE
JAYANAGAR
SRINAGAR

Sorted names
BANGALORE
JAYANAGAR
MYSORE
SRINAGAR
TANGALORE
```

Advanced Features

- Converting one datatype into another is known as **type casting** or, **type-conversion**.
- For example, if user wants to store a 'long' value into a simple integer then user can type cast 'long' to 'int'.
- It has two types
    - Implicit Type Conversion
    - Explicit Type Conversion

Advanced Features

1. Implicit Type Conversion

When the type conversion is performed automatically by the compiler without programmers intervention, such type of conversion is known as implicit type conversion or type promotion.

```
int x;
for(x=97; x<=122; x++)
{
        printf("%c", x);
}
```

Advanced Features

2. Explicit Type Conversion

The type conversion performed by the programmer by posing the data type of the expression of specific type is known as explicit type conversion. The explicit type conversion is also known as type casting.

Type casting in c is done in the following form:

```
(data_type)expression;
```

where, data_type is any valid c data type, and expression may be constant, variable or expression.

For example,

```
int x;
for(x=97; x<=122; x++)
{
        printf("%c", (char)x);
}
```

Syntax (type_name) expression

Ex 1: int x = 20;

float y = (float) x;

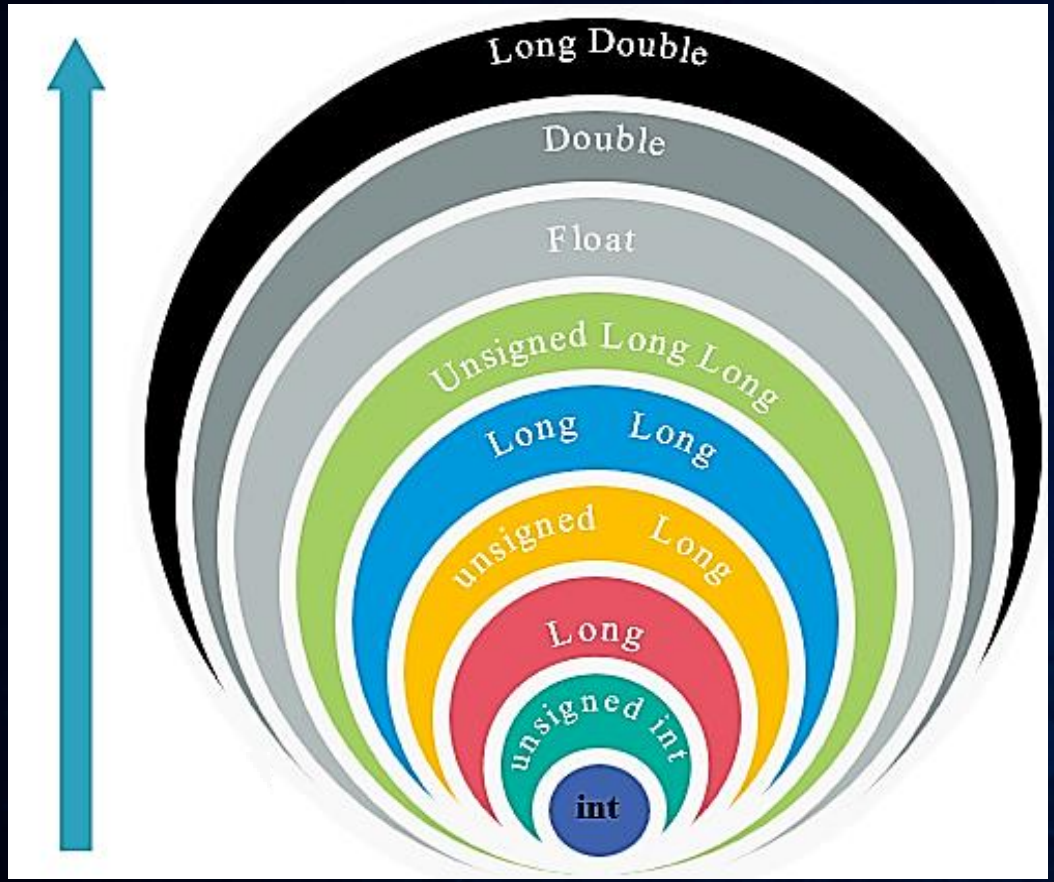
Here, the output is 20.0

Ex 2: float k = 25.90;

int z = (float) k;

Here, the output is 25

Advanced Features



```c
#include<stdio.h>
int main()
{
    float a = 1.2;
    //int b  = a; //Compiler will throw a
    int b = (int)a + 1;
    printf("Value of a is %f\n", a);
    printf("Value of b is %d\n",b);
    return 0;
}
```

```
Value of a is 1.200000
Value of b is 2
```

Console Features

C language provide us console input/output functions. As the name says, the console input/output functions allow us to -

- Read the input from the keyboard by the user accessing the console.
- Display the output to the user at the console.

Note : These input and output values could be of *any primitive data type*.

There are two kinds of console input/output functions -

- Formatted input/output functions.
- Unformatted input/output functions.

Formatted Input and Output Functions

- Formatted console input/output functions are used to take one or more inputs from the user at console and it also allows us to display one or multiple values in the output to the user at the console.
- Some of the most important formatted console input/output functions are -

Functions	Description
scanf()	This function is used to read *one or multiple* inputs from the user at the console.
printf()	This function is used to display *one or multiple* values in the output to the user at the console.
sscanf()	This function is used to read the characters from a string and stores them in variables.
sprintf()	This function is used to read the values stored in different variables and store these values in a character array.

Unformatted Input and Output Functions

- Unformatted console input/output functions are used to read a single input from the user at console and it also allows us to display the value in the output to the user at the console.
- Some of the most important formatted console input/output functions are -

Functions	Description
getch()	Reads a *single* character from the user at the console, ***without*** *echoing it.*
getche()	Reads a *single* character from the user at the console, *and echoing it.*
getchar()	Reads a *single* character from the user at the console, *and echoing it*, but needs an **Enter** key to be pressed at the end.
gets()	Reads a *single* string entered by the user at the console.
puts()	Displays a *single* string's value at the console.
putch()	Displays a *single* character value at the console.
putchar()	Displays a *single* character value at the console.

Input: getch()
getche()
getchar()
gets()

Output: putch()
putchar()
puts()

Input Functions

getch() C Program

getch.c

```
#include <stdio.h> //header file section
#include <conio.h>
int main()
{
printf("\nHello, press any alphanumeric character to exit ");
getch();
return 0;
}
```

```
Hello, press any alphanumeric character to exit
```

getchar() C Program

getchar.c

```
#include <stdio.h> //header file section
#include <conio.h>
int main()
{
char c;
printf("Enter a character : ");
c = getchar();
printf("\nEntered character : %c ", c);
return 0;
}
```

```
Enter a character : y
Entered character : y
```

Input Functions

getche() C Program

getche.c

```
#include <stdio.h> //header file section
#include <conio.h>
int main()
{
printf("\nHello, press any alphanumeric character or symbol to exit \n ");
getche();
return 0;
}
```

```
Hello, press any alphanumeric character or symbol to exit
k
```

gets() C Program

gets.c

```
#include <stdio.h> //header file section
#include <conio.h>
int main()
{
char c[25];
printf("Enter a string : ");
gets(c);
printf("\n%s is awesome ",c);
return 0;
}
```

```
Enter a string : Randy Orton
Randy Orton is awesome
```

Output Functions

putch() C Program

putch.c

```
#include <stdio.h> //header file section
#include <conio.h>
int main()
{
char c;
printf("Press any key to continue\n ");
c = getch();
printf("input : ");
putch(c);
return 0;
}
```

```
Press any key to continue
input : d
```

putchar() C Program

putchar.c

```
#include <stdio.h> //header file section
#include <conio.h>
int main()
{
char c = 'K';
putchar(c);
return 0;
}
```

```
K
```

Output Functions

puts() C Program

puts.c

```
#include <stdio.h> //header file section
#include <conio.h>
int main()
{
char c[25];
printf("Enter your Name : ");
gets(c);
puts(c);
return 0;
}
```

```
Enter your Name: john
john
```

Credits

- https://prepinsta.com/c-program/introduction-to-arrays/
- https://developerinsider.co/type-casting-c-programming/
- https://www.freecodecamp.org/news/format-specifiers-in-c/
- https://www.decodejava.com/c-unformatted-input-output-functions.htm
- https://www.2braces.com/c-programming/c-formatted-io-functions

Roadmap for Unit II

1. Branching Statements, Looping Statements Nested
2. Arrays One Dimensional and Two/Multi Dimensional
3. Strings and its Functions
4. Advanced Features, Type Casting and Type Conversion
5. Formatted and Unformatted Input and Output
6. Summary

Roadmap to Programming in C

Unit I
Computer Based Problem Solvingand Overview of C

1

Unit – II
Branching, Looping, Arrays, Strings

2

Unit III
Functions and Recursions

3

Unit IV
Dynamic Data Structures, Pointers, Union

4

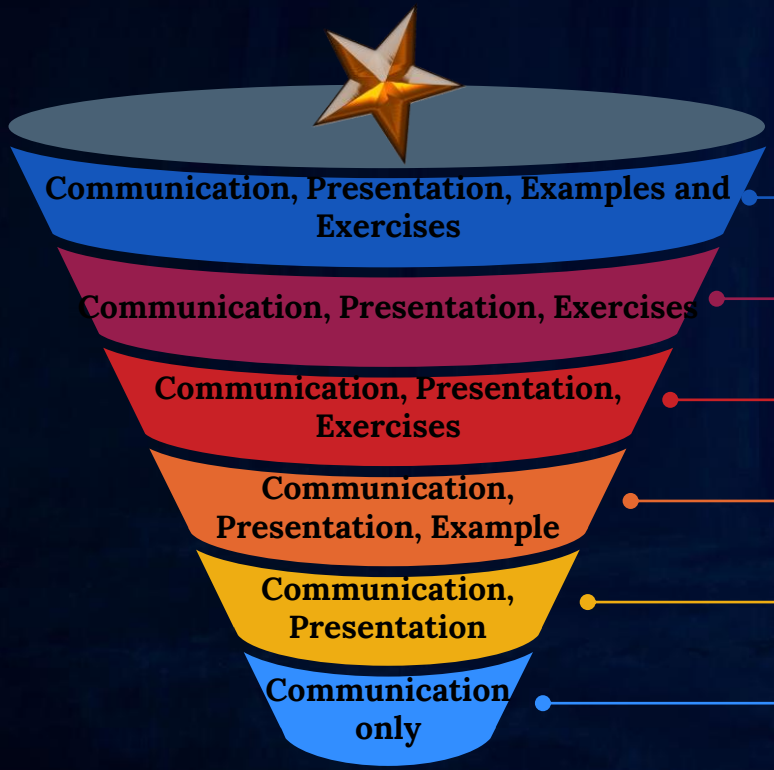
Unit V
File Handling, Preprocessor, Standard Library and Header Files

5

6

Revision

Feedback – Rating Star



$90

$80

$60

$50

$40

$30

Thanks!

Any questions?

You can find me at:

- kramdharma@gmail.com