

## Unit 4

## Arrays

### Structure:

- 4.1 Introduction
  - Objectives
- 4.2 Formatted Output
- 4.3 Type Casting
- 4.4 Arrays
- 4.5 Language C Preprocessor
  - Macro Expansion
  - File Inclusion
- 4.6 Storage Class
- 4.7 Summary
- 4.8 Terminal Questions
- 4.9 Answer to Self Assessment and Terminal Questions

### 4.1 Introduction

The need for arrays can be best understood by thinking, what if they were not there. If we had to write a program to add three integers, we may declare a, b, c as three integers and display the result as “a+b+c”. What if we had to add 100 numbers only after accepting all of them? We would need 100 different variables with 100 different identifiers with number of variable locations as relative of each other. This approach is very cumbersome and lengthy.

Arrays are the basis for creating any new data structures; understanding of arrays is essential in this field. Using arrays one can declare and define multiple variables of the same type with one identifier. For e.g. `int a[10]` is a declaration of 10 variables which are clustered together.

The format of the program's output is very important for the user. It sometimes may change the meaning also. So we use the format to display the output.

In language C there is pre processor directive through which we can create faster substitution of expression.

The scope of the variable can be defined so that we can change the variables lifetime. It is very much applicable in programming.

### Objectives:

At the end of this unit, you will be able to:

- explain arrays and their usage in programming languages.
- describe formatting of programming output and type casting .
- explain variable's scope and lifetime.
- state the pre-processor directives .

## 4.2 Formatted Output

The output can be formatted according to our requirement. There are several types of formats which are given below:

Format	Meaning
"%wd"	Minimum field width occupied for output, where w is the width of display space and d is for decimal number (scan format)
"%-wd"	Same as the above but the member will left justified.
"%0d"	The leading space occupied by zero.
"%w.pf"	Minimum field width (w) occupied for output and p is the number of decimal places. If %10.2f is defined then w=10 is the total width and p=2 is the display digits after decimal point and '.' occupies one place. So the view will be: 7(before decimal)+1(decimal)+2(after decimal point) If <b>w.p</b> is ignored then it becomes unformatted output.
"%-w.pf "	Same as the above but the number will be left justified.
"%w.pe"	Same as the above but display with exponent of the digit, where w is the width of float type. The number will be right justified. P indicates the number of decimal place. The width will be counted as (before decimal place +1(for point) +p).

**Table 4.1**

**Example 1:**

```
float a=14734.8845;
```

```
int b=31234;
```

Expression	Output
<code>printf(" %10.2f ",a);</code>	14734.89
<code>printf(" %-10.2f ",a);</code>	14734.89
<code>printf(" %12d ",b);</code>	31234
<code>printf(" %-12d ",b);</code>	31234

One example is given for the use of formatted output.

**Example 2:** Printed few numbers in vertical way

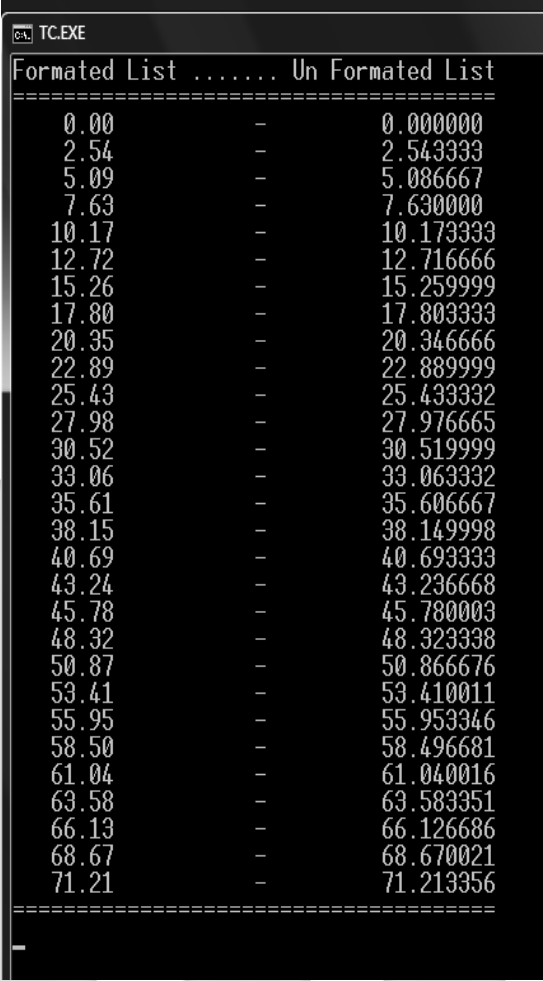
Unformatted way	Formatted way
<pre>void main() {     int i;     printf("12345678901234567890\n");     printf("-----\n");     for(i=1;i&lt;=200;i+=10)         printf("%d %d\n",i,i+5); }</pre>	<pre>void main() {     int i;     printf("12345678901234567890\n");     for(i=1;i&lt;=200;i+=10)         printf("%5d %5d\n",i,i+5); }</pre>
Output	Output
<pre>12345678901234567890 ----- 1 6 11 16 21 26 31 36 41 46 51 56 61 66 71 76</pre>	<pre>12345678901234567890 -----     1    6    11   16    21   26    31   36    41   46    51   56    61   66    71   76</pre>

81 86	81 86
91 96	91 96
101 106	101 106
111 116	111 116
121 126	121 126
131 136	131 136
141 146	141 146
151 156	151 156
161 166	161 166
171 176	171 176
181 186	181 186
191 196	191 196

**Example 3:** The program shows the formatted output

```
void main()
{
    float i,j,a;
    clrscr();
    printf("Formated List ..... Un Formated List\n");
    printf("=====\n");
    for(i=0;i<=2;i=i+0.07)
    {
        a=109*i/3;
        printf(" %7.2f      -      %f \n",a,a);
    }
    printf("=====\n");
}
```

Output:



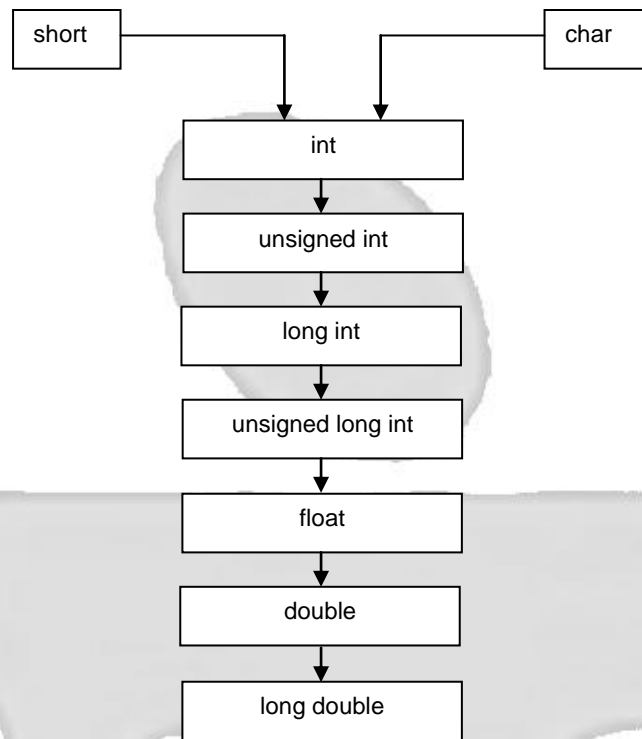
The screenshot shows a window titled 'TC.EXE' with a black background and white text. It displays two columns of floating-point numbers: 'Formatted List' and 'Un Formated List'. The numbers are separated by hyphens. The formatted list shows values with two decimal places, while the unformatted list shows the same values with more decimal places, illustrating the precision of the unformatted output.

Formatted List	Un Formated List
0.00	0.000000
2.54	2.543333
5.09	5.086667
7.63	7.630000
10.17	10.173333
12.72	12.716666
15.26	15.259999
17.80	17.803333
20.35	20.346666
22.89	22.889999
25.43	25.433332
27.98	27.976665
30.52	30.519999
33.06	33.063332
35.61	35.606667
38.15	38.149998
40.69	40.693333
43.24	43.236668
45.78	45.780003
48.32	48.323338
50.87	50.866676
53.41	53.410011
55.95	55.953346
58.50	58.496681
61.04	61.040016
63.58	63.583351
66.13	66.126686
68.67	68.670021
71.21	71.213356

Fig. 4.1

### 4.3 Type Casting

C permits mixing of constants and variables of different types in an expression. C automatically converts any intermediate values to the proper types so that the expression can be evaluated without losing any significance. This auto conversion is called implicit conversion. The water fall model is given for implicit conversion.

**Fig. 4.2**

If we want to convert forcefully to other data type called explicit conversion, the syntax is

variable = (data type) expression;

**Example 4:**

```
x=(int)(a+b);
```

// The result of (a+b) will be converted into integer and stored to x

**4.4 Arrays**

Array is the contiguous allocation of memory having the common name. To identify each element on the array, an index number called subscription is defined. It is numbered from 0 to (n-1) where the array contains n elements. An Array is also known as a subscripted variable.

The basic advantage of an array declaration is that by using a simple expression we can define large number of variables. But normally it cannot be done in normal use. There are a number of dimensions to declare the array variable.

**The following are the points to be noted:**

1. An array is a collection of similar elements.
2. The first element of the array is referred with subscript (array index) 0 (zero) and the subscript of the last element is 1 less than the size of the array.
3. Before using an array, its type and dimension must be declared.
4. Its elements are always stored in contiguous memory locations.
5. Without initialization, array contains garbage value.
6. At the time of initialization of array, dimension is optional.

**General Syntax for Array Declaration:**

<data type> <array\_name> [size of array-1][size-2],...,[size-n];

**Initialization at the time of declaration:**

<Data type> < array name> [ ] = {value -1, value-2, value-3, ..., value-n};

**Example 5:** Declaration of 1 dimension (1-D) array

int a[5];

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

**Example:** Declaration of 1 dimension (1-D) array with initialization

int a[5]={2,4,8,67,22};    or    int a[ ]={2,4,8,67,22};

2	4	8	67	22
a[0]	a[1]	a[2]	a[3]	a[4]

### Character Array

A *string* is a continuous sequence of characters terminated by '\0', the null character. The length of a string is considered to be the number of characters excluding the terminating null character. There is no string type in C, and consequently there are no operators that accept strings as operands.

Instead, strings are stored in arrays whose elements have the type char. The C standard library provides numerous functions to perform basic operations on strings, such as comparing, copying, and concatenating them.

You can initialize arrays of char using string literals. For example, the following two array definitions are equivalent:

```
char str1[30] = "Let's go"; /* String length: 8; array length: 30. */  
char str1[30] = { 'L', 'e', 't', '\'', 's', ' ', 'g', 'o', '\0' };
```

An array holding a string must always be at least one element longer than the string length to accommodate the terminating null character. Thus the array str1 can store strings upto a maximum length of 29. It would be a mistake to define the array with length 8 rather than 30, because then it wouldn't contain the terminating null character.

If you define a character array without an explicit length and initialize it with a string literal, the array created is one element longer than the string length.

An example:

```
char str2[ ] = " to London!"; // String length: 11 (note leading space);  
                        // array length: 12.
```

The following statement uses the standard function `strcat ( )` to append the string in str2 to the string in str1. The array str1 must be large enough to hold all the characters in the concatenated string.

```
strcat (str1, str2);
```



**Example 6:** Declaration of 2-D Array

```
int a[3][5];
```

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

**Example 7:** Declaration of 2-D Array with initialization

```
int a[3][5]={{3,7,9,5,4},{54,9,94,0,100},{85,2,3,5,84}};
```

3	7	9	5	4
54	9	94	0	100
85	2	3	5	84

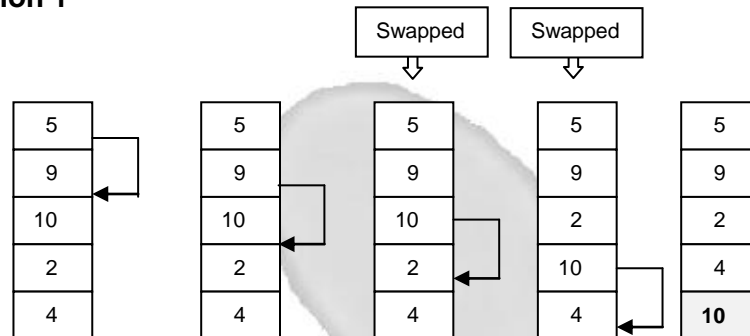
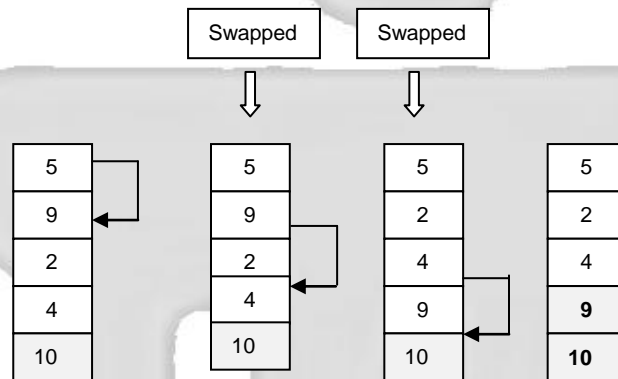
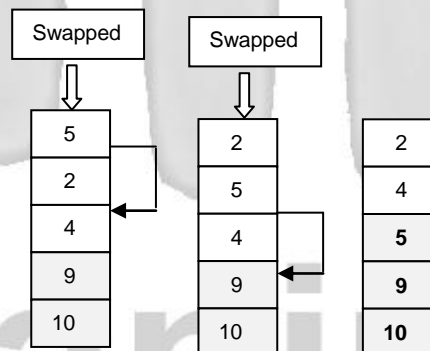
**Example 8:** Declaration of 3D array.

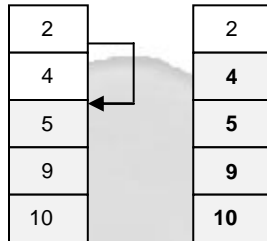
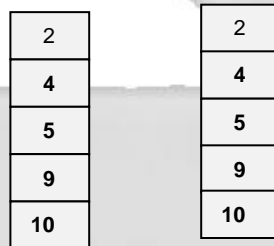
```
int a[2][3][4];
```


**Bubble Sorting**

Sorting is generally understood as a process of re-arranging a given set of objects in a specific order. The purpose of sorting is to facilitate that later search for members of the sorted set. As such it is an almost universally performed, fundamental activity. Objects are sorted in telephone books, in income tax files, in tables of contents, in libraries, in dictionaries, in warehouses and almost everywhere where stored objects have to be searched and retrieved.

There are several sorting algorithms which will be discussed later. Here **Bubble Sort** method is explained.

**Iteration 1****Iteration 2****Iteration 3**

**Iteration 4****Iteration 5****Sorting Complete****Example 9:**

**Q. 1)** Write a program that accepts 10 different numbers into an array and find the sum of odd numbers and even numbers separately.

[File Name: u4q1.c]

```
#define N 10                /*Macro value N can be changed */

void main()
{
    int array[N],i,even_sum=0,odd_sum=0;
    clrscr();                /* To clear the screen*/
    printf("Input %d different numbers \n\n",N);
    for(i=0;i<N;i++)
    {
        printf("Enter Number [%3d] : ",i+1);
        scanf("%d",&array[i]);
    }
    /*end of input */
```

```
for(i=0;i<N;i++)
{
    if(array [i]%2==0)          /* adding to sum separately */
        odd_sum+=array[i];
    else
        even_sum+=array[i];
}
printf("The sum of odd  numbers is = %5d\n",odd_sum);
printf("The sum of even numbers is = %5d\n",even_sum);
getch();
}
```

**Q. 2)** Write a program to keep track information of students having name, roll and marks of two subjects that is entry and display the information. [File Name: u4q2.c]

**Solution:**

```
#include<stdio.h>
main()
{
    char name[30][20],ch,choice,grd;
    int roll[30],m1[30],m2[30],i,n=0,r,tot;
    do{
        clrscr();
        printf("MAIN MENU\n");
        printf("1.....Add Records.\n");
        printf("2.....List Records.\n");
        printf("3.....Exit.\n");
        printf("Your Choice Please [1,2,3] : ");  ch=getch();
        switch(ch)
        {
```

```
case '1':
    i=n;
    do{
        clrscr();
        printf("Add Records\n");
        printf("Enter Students Name : "); fflush(stdin); gets(name[i]);
        printf("Enter Roll      : ");      scanf("%d",&roll[i]);
        printf("Marks in English   : ");      scanf("%d",&m1[i]);
        printf("Marks in Mathematics : ");      scanf("%d",&m2[i]);
        printf("Continue...[Y/N]   : ");      choice=getche(); i++;
        if(choice=='n' || choice == 'N') break;
    }while(1);
    n=i;
    break;
/*****/
case '2':
    clrscr();
    printf("List of Records\n");
    printf(" SL...Roll..Name.....English....Math.....Grade\n");
    for(i=0;i<n;i++)
    {
        tot=m1[i]+m2[i];
        if(tot<60)
            grd='F';
        else if(tot<90)
            grd='3';
        else if(tot<120)
            grd='2';
        else if(tot<150)
```

```
        grd='1';
    else
        grd='*';
    printf("%4d %5d  %-20s %5d %10d
    %10c\n",i+1,roll[i],name[i],m1[i],m2[i],grd);
    }
    getch();
    break;
    /*****/
case '3':
    exit();
default :
    printf("Invalid Choice..."); getch();
} /* End of Switch */
}while(1);
}/* End of main() */
/*****/
```

**Q3)** Write a program to sort the elements in ascending order.

[File Name: u4q3.c]

```
/** Sort 10 random numbers using bubble sort algorithm */
#include <stdio.h>
#include <stdlib.h>
#define SIZE 10
void main()
{
    int array[SIZE],i,j,temp;
    clrscr();
    printf(" ***** BUBBLE SORT *****\n");
    printf(" Unsorted list is as follows \n");
```

```
for(i=0;i<SIZE;i++)          /* Generating Random Numbers */
{
    array[i] = rand() % 100; /* rand() is a library function */
    printf("%4d", array[i]);
}
for (i=0;i<SIZE-1;i++)
{
    for(j=0;j<SIZE-i-1;j++)
    {
        if(array[j]<array[j+1])
        {
            temp=array[j];
            array[j]=array[j+1];
            array[j+1]=temp;
        }
    }
}
printf("\n\n Sorted list is as follows \n");
for(i=0;i<SIZE;i++)
    printf("%4d",array[i]);
getch();
}
```

/\*\*\*\*\*\*

#### 4.5 The Language 'C' pre-processor

There are basically two types of pre-processors. Each of these directives begins with a # symbol. The directives can be placed anywhere in the program but are most often placed at the beginning of the program, before main ( ), or before the beginning of a particular function. The following processor directives are:

1. Macro expansion
2. File inclusion

The conditional compilation and miscellaneous directives are beyond the scope of this book.

#### 4.5.1 Macro Expansion

# define is the macro expansion

##### Rules:

- 1) # must be the first character in the line.
- 2) # define statement must not end with semicolon.
  - 1) The blank between # and define is optional.
  - 2) A blank space is required between # define and symbolic name.
  - 3) The symbolic name should be written by uppercase to identify it; it is just a convention.

##### Macro without argument:

```
# define N          50
# define AND        &&
# define ARANGE     (a>25 && a<50)
# define PRINT      printf("Good Bye")
```

##### Macro with argument:

```
#define AREA(X)      (3.14 * X * X)
#define MAX (a,b)    ((a>b) ? a : b)
#define SQR(X)       (X * X)
#define CUBE(X)      (SQR (X) * X)
```



**Example 10:**

**Q. 4)** Write a program to print the area of circle after inputting the radius using macro. [File Name: u4q4.c]

```
#include <stdio.h>

#define P      printf      /* No argument Macro */
#define AREA(R) (3.14*R*R) /*Argument Macro where R is argument*/

void main()
{
    float radius,circle_area;
    printf("Enter the radius of the circle :");
    scanf("%f",&radius);
    circle_area=AREA(radius); /* Calling Macro */
    P("The Area of Circle is = %f",circle_area);
}/* End of main() */

/*****/
```

**4.5.2 File Inclusion**

This directive causes one file to be included in the program. If I include one header file then all the functions inside the header files can be used in the program. The processor command for file inclusion looks like this:

```
# include <file name>
or
# include "filename"
```

**# include< filename>:** With this command the file will be included and searched from default directory only.

**# include "filename":** With this command the file will be included and searched from default as well as specified path.

**Example 11:** # include <stdio.h>

Or

# include "d:\myfile\userfile.h"

### 4.6 Storage Class

In language C the storage class is mainly used for the scope of the variable within the program. There are four such storage classes given below:

There are four storage classes in 'C'

- a) Automatic Storage Class
- b) Register Storage Class
- c) Static Storage Class
- d) External Storage Class

Storage class	Storage	Default initial value	Scope	Life
AUTOMATIC (Auto)	Memory	An unpredictable value, which is often called a garbage value.	Local to block in which the variable is defined.	Till the control remains within the block in which the variable is defined.
REGISTER (register)	CPU register	Garbage value	Local to the block in which the variable is defined	Till the control remains within the block in which the variable is defined.
STATIC (Static)	Memory	Zero (0)	Local to the block in which the variable is defined	Till the control remains within the block in
EXTERNAL (Extern)	Memory	Zero(0)	Global	As long as the program execution doesn't come to an end.

Table 4.2: Storage Classes

**Example 12:****a) Auto Variable**

```
main()
{
    auto int i=1;
    {
        auto int i=2;
        {
            auto int i=3;
            printf("%d ",i);
        }
        printf("%d ",i);
    }
    printf("%d ",i);
}
```

**Output:** 3 2 1

**b) register variable :**

```
main()
{
    register int i;
    for(i=0; i<5;i++)
        printf("%d ",i);
}
```

**Output:** 0 1 2 3 4

**c) static variable**

```
main()
{
    inct();
    inct();
}
```

```
    inct();  
}
```

```
void inct()  
{  
    static int i=1;  
    printf("%d",i);  
    i=i+1;  
}
```

Output: 1 2 3

**d) External variable:**

```
extern int x=10;  
main()  
{  
    int x=20;  
    printf("%d\n",n);  
    display ( );  
}
```

```
display()  
{  
    printf("%d",x);  
}
```

Output	20
	10

### 4.7 Summary

If we have to write a program to add three integers, we might need to declare a, b, c as three integers and display the result as “a+b+c”. Suppose we want to add 100 numbers, we would need 100 different variables and forming an expression with 100 variables would be a time consuming process. This approach is very cumbersome and lengthy. The concept of array will help to overcome this problem. Arrays are the basis for creating any new data structures; understanding of arrays is essential and becomes the backbone in this field. The format of the program's output is very important for the user. It sometimes may change the meaning also. So we use the format to display the output. In language C there is pre processor directive through we can create faster substitution of expression. This unit covers entire array description, substitution variables, and proper formatting of input and outputs.

#### Self Assessment Question

1. Auto conversion of type is called implicit conversion. (True / False)
2. Array index started from 1 in language C. (True / False)
3. Any array element can be accessed randomly. (True / False)
4. Macro substitution needed #. (True / False)
5. We can't change value of static variable. (True / False)

### 4.8 Terminal Questions

1. Why do we require formatted output and how?
2. Define array. What is the requirement of its dimension?
3. What is a macro? Why is it needed in a program?
4. Give one example for each type of macro.
5. Write a program to sort the elements in ascending order.

6. Write a program that accepts 15 different numbers and find the LCM and HCM.
7. Write a program that counts the frequency of the number.
8. Write a program to add and multiply two matrices.
9. Write a program to count how many numbers are smaller, greater and equal to the checked number.
10. Write a program to print all prime numbers from the array.

#### **4.9 Answer to Self Assessment and Terminal Questions**

##### **Self Assessment Questions**

1. True
2. False
3. False
4. True
5. False

##### **Terminal Questions**

1. Section 4.2
2. Section 4.4
3. Section 4.5
4. Section 4.5.1
5. Section 4.4

# Manipal