UNIT-III ARRAYS AND STRINGS

Contents

Single and Multidimensional Arrays: Array Declaration and Initialization of arrays – Arrays as function arguments. Strings: Initialization and String handling functions. Structure and Union: Definition and Declaration - Nested Structures, Array of Structures, Structure as function arguments, Function that return structure – Union.

ARRAYS

Introduction:

So far we have used only single variable name for storing one data item. If we need to store multiple copies of the same data then it is very difficult for the user. To overcome the difficulty a new data structure is used called arrays.

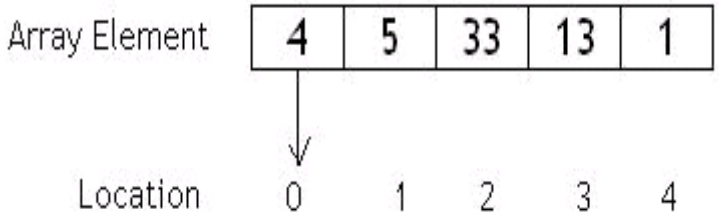
- An array is a linear and homogeneous data structure
- An array permits homogeneous data. It means that similar types of elements are stored contiguously in the memory under one variable name.
- An array can be declared of any standard or custom data type.

Example of an Array:

Suppose we have to store the roll numbers of the 100 students the we have to declare 100 variables named as roll1, roll2, roll3, ……. roll100 which is very difficult job. Concept of C programming arrays is introduced in C which gives the capability to store the 100 roll numbers in the contiguous memory which has 100 blocks and which can be accessed by single variable name.

1. C Programming Arrays is the **Collection of Elements**
2. C Programming Arrays is collection of the Elements of the **same data type.**
3. All Elements are stored in the **Contiguous memory**
4. All elements in the array are accessed using the subscript variable (index).

Pictorial representation of C Programming Arrays



The above array is declared as int a [5];

a[0] = 4; a[1] = 5; a[2] = 33; a[3] = 13; a[4] = 1;

In the above figure 4, 5, 33, 13, 1 are actual data items. 0, 1, 2, 3, 4 are index variables.

Index or Subscript Variable:

1. Individual data items can be accessed by the name of the array and an integer enclosed in square bracket called subscript variable / index

2. Subscript Variables helps us to identify the item number to be accessed in the contiguous memory.

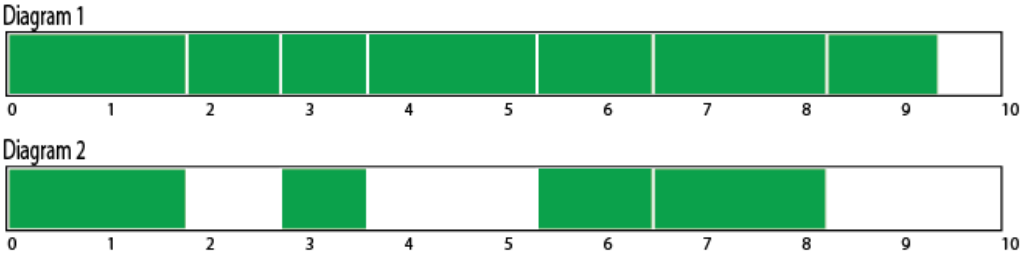
What is Contiguous Memory?

1. When Big Block of memory is reserved or allocated then that memory block is called as Contiguous Memory Block.
2. Alternate meaning of Contiguous Memory is continuous memory.
3. Suppose inside memory we have reserved 1000-1200 memory addresses for special purposes then we can say that these 200 blocks are going to reserve contiguous memory.

Contiguous Memory Allocation

1. Two registers are used while implementing the contiguous memory scheme. These registers are base register and limit register.
2. When OS is executing a process inside the main memory then content of each register are as

Register	Content of register
Base register	Starting address of the memory location where process execution is happening
Limit register	Total amount of memory in bytes consumed by process



Here diagram 1 represents the contiguous allocation of memory and diagram 2 represents non-contiguous allocation of memory.

3. When process try to refer a part of the memory then it will firstly refer the base address from base register and then it will refer relative address of memory location with respect to base address.

How to allocate contiguous memory?

1. Using static array declaration.
2. Using alloc ( ) / malloc ( ) function to allocate big chunk of memory dynamically.

Array Terminologies:

Size: Number of elements or capacity to store elements in an array. It is always mentioned in square brackets [ ].

Type: Refers to data type. It decides which type of element is stored in the array. It is also instructing the compiler to reserve memory according to the data type.

Base: The address of the first element is a base address. The array name itself stores address of the first element.

Index: The array name is used to refer to the array element. For example num[x], num is array and x is index. The value of x begins from 0.The index value is always an integer value.

Range: Value of index of an array varies from lower bound to upper bound. For example in num[100] the range of index is 0 to 99.

Word: It indicates the space required for an element. In each memory location, computer can store a data piece. The space occupation varies from machine to machine. If the size of element is more than word (one byte) then it occupies two successive memory locations. The variables of data type int, float, long need more than one byte in memory.

Characteristics of an array:

1. The declaration int a [5] is nothing but creation of five variables of integer types in memory instead of declaring five variables for five values.
2. All the elements of an array share the same name and they are distinguished from one another with the help of the element number.
3. The element number in an array plays a major role for calling each element.
4. Any particular element of an array can be modified separately without disturbing the other elements.
5. Any element of an array a[ ] can be assigned or equated to another ordinary variable or array variable of its type.
6. Array elements are stored in contiguous memory locations.

Array Declaration:

Array has to be declared before using it in C Program. Array is nothing but the collection of elements of similar data types.

Syntax: <data type> array name [size1][size2].....[sizen];

Syntax Parameter	Significance
Data type	Data Type of Each Element of the array
Array name	Valid variable name

Size	Dimensions of the Array

Array declaration requirements

Requirement	Explanation
Data Type	Data Type specifies the type of the array. We can compute the size required for storing the single cell of array.
Valid Identifier	Valid identifier is any valid variable or name given to the array. Using this identifier name array can be accessed.
Size of Array	It is maximum size that array can have.

What does Array Declaration tell to Compiler?

1. Type of the Array
2. Name of the Array
3. Number of Dimension
4. Number of Elements in Each Dimension

Types of Array

1. **Single Dimensional Array / One Dimensional Array**
2. **Multi Dimensional Array**

Single / One Dimensional Array:

1. Single or One Dimensional array is used to represent and store data in a linear form.
2. Array having only one subscript variable is called **One-Dimensional array**
3. It is also called as **Single Dimensional Array** or **Linear Array**

Single Dimensional Array Declaration and initialization:

Syntax for declaration: <data type> <array name> [size];

Examples for declaration: int iarr[3]; **char** carr[20]; **float** farr[3];

Syntax for initialization: <data type> <array name> [size] = {val1, val2, …, valn};

Examples for initialization:

int iarr[3] = {2, 3, 4};

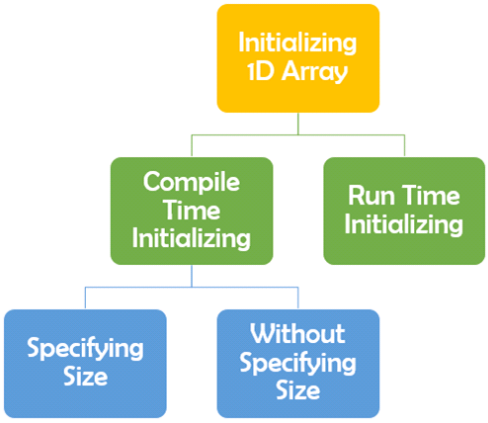
char carr[20] **=** “program”;

float farr[3] = {12.5, 13.5, 14.5};

Different Methods of Initializing 1-D Array

Whenever we declare an array, we initialize that array directly at compile time.

Initializing 1-D Array is called as compiler time initialization if and only if we assign certain set of values to array element before executing program. i.e. at compilation time.



Here we are learning the different ways of compile time initialization of an array.

Ways of Array Initializing 1-D Array:

1. Size is Specified Directly
2. Size is Specified Indirectly

Method 1: Array Size Specified Directly

In this method, we try to specify the Array Size directly.

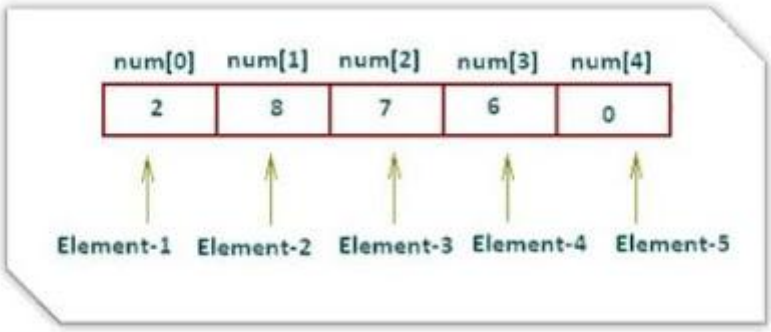
int num [5] = {2,8,7,6,0};

In the above example we have specified the size of array as 5 directly in the initialization statement. Compiler will assign the set of values to particular element of the array.

num[0] = 2; num[1] = 8; num[2] = 7; num[3] = 6; num[4] = 0;

As at the time of compilation all the elements are at specified position So This initialization scheme is Called as "**Compile Time Initialization**".

Graphical Representation:



Method 2: Size Specified Indirectly

In this scheme of compile time Initialization, We do not provide size to an array but instead we provide set of values to the array.

int num[ ] = {2,8,7,6,0};

Explanation:

1. Compiler Counts the Number Of Elements Written Inside Pair of Braces and Determines the Size of An Array.
2. After counting the number of elements inside the braces, The size of array is considered as 5 during complete execution.
3. This type of Initialization Scheme is also Called as "**Compile Time Initialization**"

Example Program

```
#include <stdio.h>
int main()
int num[] = {2,8,7,6,0};
int i;
for (i=0;i<5;i++) {
printf("\n Array Element num [%d] = %d",i, num[i]); }
return 0; }
```

Output:

```
Array Element num[0] = 2
Array Element num[1] = 8
Array Element num[2] = 7
Array Element num[3] = 6
Array Element num[4] = 0
```

Accessing Array

1. We all know that array elements are randomly accessed using the subscript variable.
2. Array can be accessed using array-name and subscript variable written inside pair of square brackets [ ].

Consider the below example of an array

In this example we will be accessing array like this

arr[3] = Forth Element of Array

arr[5] = Sixth Element of Array

whereas elements are assigned to an array using below way

arr[0] = 51; arr[1] = 32; arr[2] = 43; arr[3] = 24; arr[4] = 5; arr[5] =26;

Example Program1: Accessing array

```
#include<stdio.h>
#include<conio.h>
void main()
{
int arr[] = {51,32,43,24,5,26};
int i;
for(i=0; i<=5; i++) {
printf("\nElement at arr[%d] is %d",i,arr[i]);
}
getch();
}
```

```
Output:
Element at arr[0] is 51
Element at arr[1] is 32
Element at arr[2] is 43
Element at arr[3] is 24
Element at arr[4] is 5
Element at arr[5] is 26
```

How a[i] Works?

We have following array which is declared like int arr[] = { 51,32,43,24,5,26};

As we have elements in an array, so we have track of base address of an array. Below things are important to access an array.

Expression	Description	Example
arr	It returns the base address of an array	Consider 2000
*arr	It gives zeroth element of an array	51

Expression	Description	Example
*(arr+0)	It also gives zeroth element of an array	51
*(arr+1)	It gives first element of an array	32

So whenever we tried accessing array using arr[i] then it returns an element at the location*(arr + i)

Accessing array a[i] means retrieving element from address (a + i).

Example Program2: Accessing array

```
#include<stdio.h>
#include<conio.h>
void main()
{
int arr[] = {51,32,43,24,5,26};
int i;
for(i=0; i<=5; i++) {
printf("\n%d %d %d %d",arr[i],*(i+arr),*(arr+i),i[arr]);
}
getch();
}
```

Output:

```
51 51 51 51
32 32 32 32
43 43 43 43
24 24 24 24
5 5 5 5
26 26 26 26
```

Operations with One Dimensional Array

1. Deletion – Involves deleting specified elements form an array.
2. Insertion – Used to insert an element at a specified position in an array.
3. Searching – An array element can be searched. The process of seeking specific elements in an array is called searching.

4. Merging – The elements of two arrays are merged into a single one.
5. Sorting – Arranging elements in a specific order either in ascending or in descending order.

Example Programs:

1. C Program for deletion of an element from the specified location from an Array

```
#include<stdio.h>
int main() {
int arr[30], num, i, loc;
printf("\nEnter no of elements:");
scanf("%d", &num);
//Read elements in an array
printf("\nEnter %d elements :", num);
for (i = 0; i < num; i++) {
scanf("%d", &arr[i]);    }
//Read the location
printf("\nLocation of the element to be deleted :");
scanf("%d", &loc);
/* loop for the deletion  */
while (loc < num) {
arr[loc - 1] = arr[loc];
loc++;    }
num--;  // No of elements reduced by 1
//Print Array
for (i = 0; i < num; i++)
printf("\n %d", arr[i]);
return (0);
}
```

Output:

```
Enter no of elements: 5
Enter 5 elements: 3 4 1 7 8
```

```
Location of the element to be deleted: 3
3 4 7 8
```

2. C Program to delete duplicate elements from an array

```
int main() {
int arr[20], i, j, k, size;
printf("\nEnter array size: ");
scanf("%d", &size);
printf("\nAccept Numbers: ");
for (i = 0; i < size; i++)
scanf("%d", &arr[i]);
printf("\nArray with Unique list: ");
for (i = 0; i < size; i++) {
for (j = i + 1; j < size;) {
if (arr[j] == arr[i]) {
for (k = j; k < size; k++) {
arr[k] = arr[k + 1]; }
size--; }
else
j++; }
}
for (i = 0; i < size; i++) {
printf("%d ", arr[i]); }
return (0);
}
```

Output:

```
Enter array size: 5
Accept Numbers: 1 3 4 5 3
Array with Unique list: 1 3 4 5
```

3. C Program to insert an element in an array

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

```
int arr[30], element, num, i, location;
printf("\nEnter no of elements:");
scanf("%d", &num);
for (i = 0; i < num; i++) {
scanf("%d", &arr[i]); }
printf("\nEnter the element to be inserted:");
scanf("%d", &element);
printf("\nEnter the location");
scanf("%d", &location);
//Create space at the specified location
for (i = num; i >= location; i--) {
arr[i] = arr[i - 1];   }
num++;
arr[location - 1] = element;
//Print out the result of insertion
for (i = 0; i < num; i++)
printf("n %d", arr[i]);
return (0);
}
Output:
Enter no of elements: 5
1 2 3 4 5
Enter the element to be inserted: 6
Enter the location: 2
1 6 2 3 4 5
```

4. C Program to search an element in an array

```
#include<stdio.h>
int main() {
int a[30], ele, num, i;
printf("\nEnter no of elements:");
scanf("%d", &num);
```

```
printf("\nEnter the values :");
for (i = 0; i < num; i++) {
scanf("%d", &a[i]);    }
//Read the element to be searched
printf("\nEnter the elements to be searched :");
scanf("%d", &ele);
//Search starts from the zeroth location
i = 0;
while (i < num && ele != a[i]) {
i++;   }
//If i < num then Match found
if (i < num) {
printf("Number found at the location = %d", i + 1);
}
else {
printf("Number not found"); }
return (0);
}
```

Output:

```
Enter no of elements: 5
11 22 33 44 55
Enter the elements to be searched: 44
Number found at the location = 4
```

5. C Program to copy all elements of an array into another array

```
#include<stdio.h>
int main() {
int arr1[30], arr2[30], i, num;
printf("\nEnter no of elements:");
scanf("%d", &num);
//Accepting values into Array
printf("\nEnter the values:");
```

```
for (i = 0; i < num; i++) {
scanf("%d", &arr1[i]);    }
/* Copying data from array 'a' to array 'b */
for (i = 0; i < num; i++) {
arr2[i] = arr1[i];   }
//Printing of all elements of array
printf("The copied array is:");
for (i = 0; i < num; i++)
printf("\narr2[%d] = %d", i, arr2[i]);
return (0);
}
```

Output:

```
Enter no of elements: 5
Enter the values: 11 22 33 44 55
The copied array is: 11 22 33 44 55
```

6. C program to merge two arrays in C Programming

```
#include<stdio.h>
int main() {
int arr1[30], arr2[30], res[60];
int i, j, k, n1, n2;
printf("\nEnter no of elements in 1st array:");
scanf("%d", &n1);
for (i = 0; i < n1; i++) {
scanf("%d", &arr1[i]); }
printf("\nEnter no of elements in 2nd array:");
scanf("%d", &n2);
for (i = 0; i < n2; i++) {
scanf("%d", &arr2[i]); }
i = 0;
j = 0;
k = 0;
```

```
// Merging starts
while (i < n1 && j < n2) {
if (arr1[i] <= arr2[j]) {
res[k] = arr1[i];
i++;
k++; }
else {
res[k] = arr2[j];
k++;
j++; }
}
/*Some elements in array 'arr1' are still remaining where as the array
'arr2' is exhausted*/
while (i < n1) {
res[k] = arr1[i];
i++;
k++; }
/*Some elements in array 'arr2' are still remaining where as the array
'arr1' is exhausted */
while (j < n2) {
res[k] = arr2[j];
k++;
j++; }
//Displaying elements of array 'res'
printf("\nMerged array is:");
for (i = 0; i < n1 + n2; i++)
printf("%d ", res[i]);
return (0);
}
Enter no of elements in 1st array: 4
11 22 33 44
```

```
Enter no of elements in 2nd array: 3
10 40 80
Merged array is: 10 11 22 33 40 44 80
```

Programs for Practice

1 C Program to display array elements with addresses
2 C Program for Reading and printing Array Elements
3 C Program to calculate Addition of All Elements in Array
4 C Program to find Smallest Element in Array
5 C Program to find Largest Element in Array
6 C Program to reversing an Array Elements

1. C Program to display array elements with addresses

```
#include<stdio.h>
#include<stdlib.h>
#define size 10
int main() {
int a[3] = { 11, 22, 33 };
printf("\n a[0],value=%d : address=%u", a[0], &a[0]);
printf("\n a[1],value=%d : address=%u", a[1], &a[1]);
printf("\n a[2],value=%d : address=%u", a[2], &a[2]);
return (0);
}
```

Output:

```
a[0],value=11 : address=2358832
a[1],value=22 : address=2358836
a[2],value=33 : address=2358840
```

2. C Program for Reading and printing Array Elements

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

```
int i, arr[50], num;
printf("\nEnter no of elements :");
scanf("%d", &num);
//Reading values into Array
printf("\nEnter the values :");
for (i = 0; i < num; i++) {
scanf("%d", &arr[i]); }
//Printing of all elements of array
for (i = 0; i < num; i++) {
printf("\narr[%d] = %d", i, arr[i]); }
return (0);
}
```

Output:

```
Enter no of elements : 5
Enter the values : 10 20 30 40 50
arr[0] = 10
arr[1] = 20
arr[2] = 30
arr[3] = 40
arr[4] = 50
```

3. C Program to calculate addition of all elements in an array

```
#include<stdio.h>
int main() {
int i, arr[50], sum, num;
printf("\nEnter no of elements :");
scanf("%d", &num);
//Reading values into Array
printf("\nEnter the values :");
for (i = 0; i < num; i++)
scanf("%d", &arr[i]);
//Computation of total
```

```
sum = 0;
for (i = 0; i < num; i++)
sum = sum + arr[i];
//Printing of all elements of array
for (i = 0; i < num; i++)
printf("\na[%d]=%d", i, arr[i]);
//Printing of total
printf("\nSum=%d", sum);
return (0);
}
```

Output:

```
Enter no of elements : 3
Enter the values : 11 22 33
a[0]=11
a[1]=22
a[2]=33
Sum=66
```

4. C Program to find smallest element in an array

```
#include<stdio.h>
int main() {
int a[30], i, num, smallest;
printf("\nEnter no of elements :");
scanf("%d", &num);
//Read n elements in an array
for (i = 0; i < num; i++)
scanf("%d", &a[i]);
//Consider first element as smallest
smallest = a[0];
for (i = 0; i < num; i++) {
if (a[i] < smallest) {
smallest = a[i]; } }
```

```
// Print out the Result
printf("\nSmallest Element : %d", smallest);
return (0);
}
```

Output:

```
Enter no of elements : 5
11 44 22 55 99
Smallest Element : 11
```

5. C Program to find largest element in an array

```
#include<stdio.h>
int main() {
int a[30], i, num, largest;
printf("\nEnter no of elements :");
scanf("%d", &num);
//Read n elements in an array
for (i = 0; i < num; i++)
scanf("%d", &a[i]);
//Consider first element as largest
largest = a[0];
for (i = 0; i < num; i++) {
if (a[i] > largest) {
largest = a[i]; } }
// Print out the Result
printf("\nLargest Element : %d", largest);
return (0);
}
```

Output:

```
Enter no of elements : 5
11 55 33 77 22
Largest Element : 77
```

6. C Program to reverse an array elements in an array

```
#include<stdio.h>
int main() {
int arr[30], i, j, num, temp;
printf("\nEnter no of elements : ");
scanf("%d", &num);
//Read elements in an array
for (i = 0; i < num; i++) {
scanf("%d", &arr[i]);    }
j = i - 1;   // j will Point to last Element
i = 0;       // i will be pointing to first element
while (i < j) {
temp = arr[i];
arr[i] = arr[j];
arr[j] = temp;
i++;             // increment i
j--;          // decrement j
}
//Print out the Result of Insertion
printf("\nResult after reversal : ");
for (i = 0; i < num; i++) {
printf("%d \t", arr[i]); }
return (0);
}
```

Output:

```
Enter no of elements : 5
11 22 33 44 55
Result after reversal : 55 44 33 22 11
```

Multi Dimensional Array:

1. Array having more than one subscript variable is called Multi-Dimensional array.
2. Multi Dimensional Array is also called as **Matrix**.

Syntax: <data type> <array name> [row subscript][column subscript];

Example: Two Dimensional Arrays

Declaration: Char name[50][20];

Initialization:

```
int a[3][3] = { 1, 2, 3
                5, 6, 7
                8, 9, 0};
```

In the above example we are declaring 2D array which has 2 dimensions. First dimension will refer the row and 2nd dimension will refer the column.

Example: Three Dimensional Arrays

Declaration: Char name[80][20][40];

The following information are given by the compiler after the declaration

Example	Type	Array Name	Dimension No.	No. of Elements in Each Dimension
1	integer	roll	1	10
2	character	name	2	80 and 20
3	character	name	3	80 and 20 and 40

Two Dimensional Arrays:

1. Two Dimensional Array requires **Two Subscript Variables**
2. Two Dimensional Array stores the values in the form of matrix.
3. One Subscript Variable denotes the "**Row**" of a matrix.
4. Another Subscript Variable denotes the "**Column**" of a matrix.

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Declaration and use of 2D Arrays:

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

```
for(i=0;i<row,i++)
for(j=0;j<col,j++) {
printf("%d",a[i][j]); }
```

Meaning of Two Dimensional Arrays:

1. Matrix is having 3 rows (i takes value from 0 to 2)
2. Matrix is having 4 Columns (j takes value from 0 to 3)
3. Above Matrix 3×4 matrix will have 12 blocks having 3 rows & 4 columns.
4. Name of 2-D array is '**a**' and each block is identified by the row & column number.
5. Row number and Column Number Starts from 0.

Two-Dimensional Arrays: Summary with Sample Example:

Summary Point	**Explanation**
No of Subscript Variables Required	2
Declaration	a[3][4]
No of Rows	3
No of Columns	4
No of Cells	12
No of for loops required to iterate	2

Memory Representation:

1. 2-D arrays are stored in contiguous memory location **row wise**.
2. 3 X 3 Array is shown below in the first Diagram.
3. Consider **3×3 Array is stored in Contiguous memory** location which starts from 4000.
4. Array element **a[0][0]** will be stored at address **4000** again **a[0][1]** will be stored to next memory location i.e. Elements stored row-wise
5. After **Elements of First Row are stored** in appropriate memory locations, elements of next row get their corresponding memory locations.

	Col 0	Col 1	Col 2
Row 0	1	2	3
Row 1	4	5	6
Row 2	7	8	9

6. This is integer array so each element requires 2 bytes of memory.

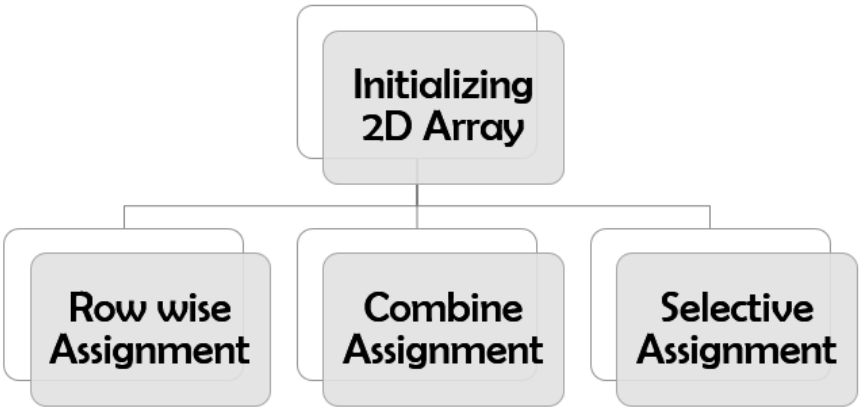
Basic Memory Address Calculation:

`a[0][1] = a[0][0] + Size of Data Type`

Element	Memory Location
a[0][0]	4000
a[0][1]	4002
a[0][2]	4004
a[1][0]	4006
a[1][1]	4008
a[1][2]	4010
a[2][0]	4012
a[2][1]	4014
a[2][2]	4016

1 4000	2 4002	3 4004
4 4006	5 4008	6 4010
7 4012	8 4014	9 4016

Initializing 2D Array



Method 1: Initializing all Elements row wise

For initializing 2D Array we need to assign values to each element of an array using the below syntax.

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

Example Program

```
#include<stdio.h>
int main()
{
int i, j;
int a[3][2] = { { 1, 4 }, { 5, 2 }, { 6, 5 } };
for (i = 0; i < 3; i++) {
for (j = 0; j < 2; j++) {
printf("%d ", a[i][j]); }
printf("\n"); }
return 0;
}
```

Output:

```
1 4
5 2
6 5
```

We have declared an array of size 3 X 2, it contains overall 6 elements.

```
Row 1:  {1, 4},
Row 2:  {5, 2},
Row 3:  {6, 5}
We have initialized each row independently
a[0][0] = 1
a[0][1] = 4
```

Method 2: Combine and Initializing 2D Array

Initialize all Array elements but initialization is much straight forward. All values are assigned sequentially and row-wise

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

Example Program:

```
#include <stdio.h>
int main() {
int i, j;
int a[3][2] = { 1, 4, 5, 2, 6, 5 };
for (i = 0; i < 3; i++) {
for (j = 0; j < 2; j++) {
printf("%d ", a[i][j]);  }
printf("\n"); }
return 0;
}
```

Output:

```
1 4
5 2
6 5
```

Method 3: Some Elements could be initialized

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

Now we have again going with the way 1 but we are removing some of the elements from the array. Uninitialized elements will get default 0 value. In this case we have declared and initialized 2-D array like this

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

```
int main() {
int i, j;
int a[3][2] = { { 1 }, { 5, 2 }, { 6 }};
for (i = 0; i < 3; i++) {
for (j = 0; j < 2; j++) {
printf("%d ", a[i][j]); }
printf("\n"); }
return 0;
}
Output:
1 0
5 2
6 0
```

Accessing 2D Array Elements:

1. To access every 2D array we requires **2 Subscript variables**.
2. i – Refers the **Row number**
3. j – Refers **Column Number**
4. a[1][0] refers element belonging to **first row and zeroth column**

Example Program: Accept & Print 2×2 Matrix from user

```
#include<stdio.h>
int main() {
int i, j, a[3][3];
// i : For Counting Rows
// j : For Counting Columns
for (i = 0; i < 3; i++) {
for (j = 0; j < 3; j++) {
printf("\nEnter the a[%d][%d] = ", i, j);
scanf("%d", &a[i][j]); } }
//Print array elements
for (i = 0; i < 3; i++) {
for (j = 0; j < 3; j++) {
```

```
printf("%d\t", a[i][j]); }
printf("\n"); }
return (0);
}
```

How it Works?

1. For Every value of row Subscript , the **column Subscript incremented from 0 to n-1 columns**
2. i.e. For Zeroth row it will accept zeroth, first, second column (a[0][0], a[0][1], a[0][2]) elements
3. In **Next Iteration** Row number will be incremented by 1 and the **column number again initialized to 0.**
4. **Accessing 2-D Array:** a[i][j] → Element From i^{th} Row and j^{th} Column

Example programs for practice:

1. C Program for addition of two matrices

2. C Program to find inverse of 3 X # Matrix

3. C Program to Multiply two 3 X 3 Matrices

4. C Program to check whether matrix is magic square or not?

1. C Program for addition of two matrices

```
#include<stdio.h>
int main() {
int i, j, mat1[10][10], mat2[10][10], mat3[10][10];
int row1, col1, row2, col2;
printf("\nEnter the number of Rows of Mat1 : ");
scanf("%d", &row1);
printf("\nEnter the number of Cols of Mat1 : ");
scanf("%d", &col1);
printf("\nEnter the number of Rows of Mat2 : ");
scanf("%d", &row2);
printf("\nEnter the number of Columns of Mat2 : ");
```

```
scanf("%d", &col2);
/* before accepting the Elements Check if no of rows and columns of
both matrices is equal */
if (row1 != row2 || col1 != col2) {
printf("\nOrder of two matrices is not same ");
exit(0); }
//Accept the Elements in Matrix 1
for (i = 0; i < row1; i++) {
for (j = 0; j < col1; j++) {
printf("Enter the Element a[%d][%d] : ", i, j);
scanf("%d", &mat1[i][j]); } }
//Accept the Elements in Matrix 2
for (i = 0; i < row2; i++)
for (j = 0; j < col2; j++) {
printf("Enter the Element b[%d][%d] : ", i, j);
scanf("%d", &mat2[i][j]); }
//Addition of two matrices
for (i = 0; i < row1; i++)
for (j = 0; j < col1; j++) {
mat3[i][j] = mat1[i][j] + mat2[i][j];}
//Print out the Resultant Matrix
printf("\nThe Addition of two Matrices is : \n");
for (i = 0; i < row1; i++) {
for (j = 0; j < col1; j++) {
printf("%d\t", mat3[i][j]); }
printf("\n"); }
return (0);
}
```

Output:

```
Enter the number of Rows of Mat1 : 3
Enter the number of Columns of Mat1 : 3
```

```
Enter the number of Rows of Mat2 : 3
Enter the number of Columns of Mat2 : 3
Enter the Element a[0][0] : 1
Enter the Element a[0][1] : 2
Enter the Element a[0][2] : 3
Enter the Element a[1][0] : 2
Enter the Element a[1][1] : 1
Enter the Element a[1][2] : 1
Enter the Element a[2][0] : 1
Enter the Element a[2][1] : 2
Enter the Element a[2][2] : 1
Enter the Element b[0][0] : 1
Enter the Element b[0][1] : 2
Enter the Element b[0][2] : 3
Enter the Element b[1][0] : 2
Enter the Element b[1][1] : 1
Enter the Element b[1][2] : 1
Enter the Element b[2][0] : 1
Enter the Element b[2][1] : 2
Enter the Element b[2][2] : 1
The Addition of two Matrices is :
2 4 6
4 2 2
2 4 2
```

2. C Program to find inverse of 3 X 3 Matrix

```
#include<stdio.h>
void reduction(float a[][6], int size, int pivot, int col) {
int i, j;
float factor;
factor = a[pivot][col];
for (i = 0; i < 2 * size; i++) {
```

```
a[pivot][i] /= factor; }
for (i = 0; i < size; i++) {
if (i != pivot) {
factor = a[i][col];
for (j = 0; j < 2 * size; j++) {
a[i][j] = a[i][j] - a[pivot][j] * factor; } } } }
void main() {
float matrix[3][6];
int i, j;
for (i = 0; i < 3; i++) {
for (j = 0; j < 6; j++) {
if (j == i + 3) {
matrix[i][j] = 1;}
else {
matrix[i][j] = 0; } } }
printf("\nEnter a 3 X 3 Matrix :");
for (i = 0; i < 3; i++) {
for (j = 0; j < 3; j++) {
scanf("%f", &matrix[i][j]); } }
for (i = 0; i < 3; i++) {
reduction(matrix, 3, i, i); }
printf("\nInverse Matrix");
for (i = 0; i < 3; i++) {
printf("\n");
for (j = 0; j < 3; j++) {
printf("%8.3f", matrix[i][j + 3]); } } }
```

Output:

```
Enter a 3 X 3 Matrix
1 3 1
1 1 2
2 3 4
```

```
Inverse Matrix
   2.000   9.000  -5.000
   0.000  -2.000   1.000
  -1.000  -3.000   2.000
```

3. C Program to Multiply two 3 X 3 Matrices

```
#include<stdio.h>
int main() {
int a[10][10], b[10][10], c[10][10], i, j, k;
int sum = 0;
printf("\nEnter First Matrix : ");
for (i = 0; i < 3; i++) {
for (j = 0; j < 3; j++) {
scanf("%d", &a[i][j]);  } }
printf("\nEnter Second Matrix :");
for (i = 0; i < 3; i++) {
for (j = 0; j < 3; j++) {
scanf("%d", &b[i][j]);  } }
printf("The First Matrix is : \n");
for (i = 0; i < 3; i++) {
for (j = 0; j < 3; j++) {
printf(" %d ", a[i][j]); }
printf("\n"); }
printf("The Second Matrix is : \n");
for (i = 0; i < 3; i++) {
for (j = 0; j < 3; j++) {
printf(" %d ", b[i][j]); }
printf("\n"); }
//Multiplication Logic
for (i = 0; i <= 2; i++) {
for (j = 0; j <= 2; j++) {
sum = 0;
```

```
for (k = 0; k <= 2; k++) {
sum = sum + a[i][k] * b[k][j]; }
c[i][j] = sum; } }
printf("\nMultiplication Of Two Matrices : \n");
for (i = 0; i < 3; i++) {
for (j = 0; j < 3; j++) {
printf(" %d ", c[i][j]); }
printf("\n"); }
return (0);
}
```

Output:

```
Enter First Matrix :
1    1    1
1    1    1
1    1    1
Enter Second Matrix :
2    2    2
2    2    2
2    2    2
The First Matrix is :
1    1    1
1    1    1
1    1    1
The Second Matrix is :
2    2    2
2    2    2
2    2    2
Multiplication Of Two Matrices :
6    6    6
6    6    6
6    6    6
```

Multiplication is possible if and only if

i. No. of Columns of Matrix 1 = No of Columns of Matrix 2

ii. Resultant Matrix will be of Dimension – c [No. of Rows of Mat1][No. of Columns of Mat2]

4. C Program to check whether matrix is magic square or not?

What is Magic Square?

1. A magic square is a simple mathematical **game developed during the 1500.**
2. Square is divided into **equal number of rows and columns.**
3. Start filling each square with the number from 1 to num (**where num = No of Rows X No of Columns**)
4. You can only **use a number once.**
5. Fill each square so that the **sum of each row is the same as the sum of each column.**
6. In the example shown here, **the sum of each row is 15,** and the sum of each column is also 15.
7. In this Example: The numbers from 1 through 9 is used only once. This is called a **magic square.**

```
#include<stdio.h>
#include<conio.h>
int main() {
int size = 3;
int matrix[3][3]; // = {{4,9,2},{3,5,7},{8,1,6}};
int row, column = 0;
int sum, sum1, sum2;
int flag = 0;
printf("\nEnter matrix : ");
for (row = 0; row < size; row++) {
for (column = 0; column < size; column++)
scanf("%d", &matrix[row][column]); }
printf("Entered matrix is : \n");
for (row = 0; row < size; row++) {
```

```
printf("\n");
for (column = 0; column < size; column++) {
printf("\t%d", matrix[row][column]); } }
//For diagonal elements
sum = 0;
for (row = 0; row < size; row++) {
for (column = 0; column < size; column++) {
if (row == column)
sum = sum + matrix[row][column]; } }
//For Rows
for (row = 0; row < size; row++) {
sum1 = 0;
for (column = 0; column < size; column++) {
sum1 = sum1 + matrix[row][column]; }
if (sum == sum1)
flag = 1;
else {
flag = 0;
break; } }
//For Columns
for (row = 0; row < size; row++) {
sum2 = 0;
for (column = 0; column < size; column++) {
sum2 = sum2 + matrix[column][row]; }
if (sum == sum2)
flag = 1;
else {
flag = 0;
break; } }
if (flag == 1)
printf("\nMagic square");
```

```
else
printf("\nNo Magic square");
return 0;
}
```

Output:

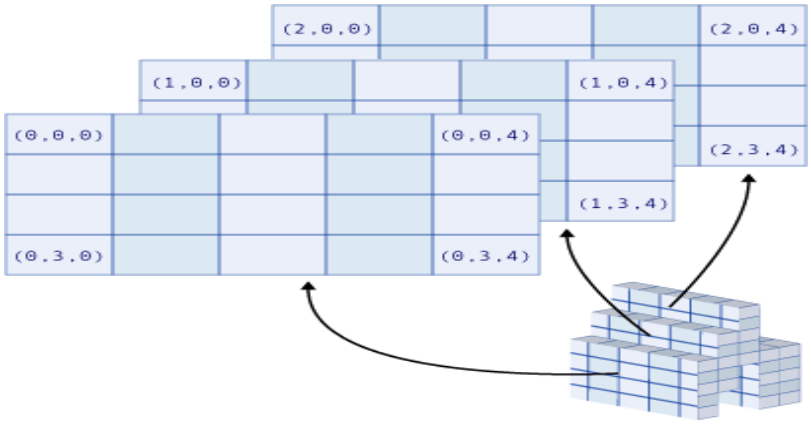
```
Enter matrix : 4 9 2 3 5 7 8 1 6
Entered matrix is :
 4 9 2
 3 5 7
 8 1 6
Magic square
Sum of Row1 = Sum of Row2 [Sum of All Rows must be same]
Sum of Col1 = Sum of Col2 [Sum of All Cols must be same]
Sum of Left Diagonal = Sum of Right Diagonal
```

Limitations of Arrays:

Array is very useful which stores multiple data under single name with same data type. Following are some listed limitations of Array in C Programming.

A. Static Data

1. Array is **Static data** Structure
2. Memory Allocated during **Compile time**.
3. Once Memory is allocated at Compile Time it cannot be changed during **Run-time**



B. Can hold data belonging to same Data types

1. Elements belonging to **different data types** cannot be stored in array because array data structure can hold data belonging to same data type.
2. **Example** : Character and Integer values can be stored inside separate array but cannot be stored in single array

C. Inserting data in an array is difficult

1. **Inserting element** is very difficult because before inserting element in an array we have to create empty space by shifting other elements one position ahead.
2. This operation is faster if the array size is smaller, but same operation will be more and more time consuming and non-efficient in case of array with large size.

D. Deletion Operation is difficult

1. Deletion is not easy because the elements are stored in contiguous memory location.
2. Like insertion operation , we have to delete element from the array and after deletion empty space will be created and thus we need to fill the space by moving elements up in the array.

E. Bound Checking

1. If we specify the size of array as 'N' then we can access elements up to 'N-1' but in C if we try to access elements after 'N-1' i.e. Nth element or N+1th element then we does not get any error message.
2. Process of checking the extreme limit of array is called Bound Checking and C does not perform **Bound Checking.**
3. If the array range exceeds then we will get garbage value as result.

F. Shortage of Memory

1. Array is Static data structure. Memory can be allocated at compile time only Thus if after executing program we need more space for storing additional information then we cannot allocate additional space at run time.
2. **Shortage of Memory** , if we don't know the size of memory in advance

G. Wastage of Memory

1. **Wastage of Memory**, if array of large size is defined

Applications of Arrays:

Array is used for different verities of applications. Array is used to store the data or values of same data type. Below are the some of the applications of array –

A. Stores Elements of Same Data Type

Array is used to store the number of elements belonging to same data type.

int arr[30];

Above array is used to store the integer numbers in an array.

arr[0] = 10;

arr[1] = 20;

arr[2] = 30;

arr[3] = 40;

arr[4] = 50;

Similarly if we declare the character array then it can hold only character. So in short character array can store character variables while floating array stores only floating numbers.

B. Array Used for maintaining multiple variable names using single name

Suppose we need to store 5 roll numbers of students then without declaration of array we need to declare following –

int roll1, roll2, roll3, roll4, roll5;

1. Now in order to get roll number of first student we need to access roll1.
2. Guess if we need to store roll numbers of 100 students then what will be the procedure.
3. Maintaining all the variables and remembering all these things is very difficult.

Consider the Array **int** roll[5]; Here we are using array which can store multiple values and we have to remember just single variable name.

C. Array can be used for Sorting Elements

We can store elements to be sorted in an array and then by using different sorting technique we can sort the elements.

Different Sorting Techniques are:

1. Bubble Sort
2. Insertion Sort
3. Selection Sort
4. Bucket Sort

D. Array can perform Matrix Operation

Matrix operations can be performed using the array. We can use 2-D array to store the matrix. Matrix can be multi dimensional.

E. Array can be used in CPU Scheduling

CPU Scheduling is generally managed by Queue. Queue can be managed and implemented using the array. Array may be allocated dynamically i.e at run time. [Animation will Explain more about Round Robin Scheduling Algorithm | Video Animation]

F. Array can be used in Recursive Function

When the function calls another function or the same function again then the current values are stores onto the stack and those values will be retrieving when control comes back. This is similar operation like stack.

Arrays as Function arguments:

Passing array to function:

Array can be passed to function by two ways:

1. **Pass Entire array**
2. **Pass Array element by element**

1. Pass Entire array

- Here entire array can be passed as a argument to function.
- Function gets **complete access** to the original array.
- While passing entire array address of first element is passed to function, any changes made inside function, directly **affects the Original value**.
- Function Passing method : **"Pass by Address"**

2. Pass Array element by element

- Here individual elements are passed to function as argument.
- Duplicate **carbon copy of Original variable** is passed to function.
- So any changes made inside function **do not affect the original value.**
- Function doesn't get complete access to the original array element.
- Function passing method is **"Pass by Value"**

Passing entire array to function:

- Parameter Passing Scheme : **Pass by Reference**
- Pass name of array as function parameter.
- Name contains the base address i.e. (Address of 0th element)
- Array values are updated in function.
- Values are reflected inside main function also.

Example Program #1:

```
#include<stdio.h>
#include<conio.h>
void fun(int arr[ ])
{
int i;
for(i=0;i< 5;i++)
 arr[i] = arr[i] + 10;
```

```
}
void main( )
{
int arr[5],i;
clrscr();
printf("\nEnter the array elements : ");
for(i=0;i< 5;i++)
scanf("%d",&arr[i]);
printf("\nPassing entire array .....");
fun(arr);  // Pass only name of array
for(i=0;i< 5;i++)
 printf("\nAfter Function call a[%d] : %d",i,arr[i]);
getch();
}
```

Output :

```
Enter the array elements : 1 2 3 4 5
Passing entire array .....
After Function call a[0] : 11
After Function call a[1] : 12
After Function call a[2] : 13
After Function call a[3] : 14
After Function call a[4] : 15
```

Passing Entire 1-D Array to Function in C Programming:

- Array is passed to function completely.
- Parameter Passing Method : **Pass by Reference**
- It is Also Called "**Pass by Address**"
- Original Copy is Passed to Function
- Function Body can modify **Original Value.**

Example Program #2:

```
#include<stdio.h>
#include<conio.h>
void modify(int b[3]);
void main()
{
int arr[3] = {1,2,3};
modify(arr);
```

```
for(i=0;i<3;i++)
printf("%d",arr[i]);
getch();
}
void modify(int a[3])
{
int i;
for(i=0;i<3;i++)
a[i] = a[i]*a[i];
}
```

Output:

```
1 4 9
```

Here “arr” is same as “a” because Base Address of Array “arr” is stored in Array “a”

Alternate Way of Writing Function Header:

void modify(int a[3]) **OR** void modify(int *a)

Passing Entire 2D Array to Function in C Programming:

Example Program #3:

```
#include<stdio.h>
void Function(int c[2][2]);
int main(){
int c[2][2],i,j;
printf("Enter 4 numbers:\n");
for(i=0;i<2;++i)
for(j=0;j<2;++j){
scanf("%d",&c[i][j]); }
Function(c);   /* passing multi-dimensional array to function */
return 0;
}
void Function(int c[2][2])
{
/* Instead to above line, void Function(int c[][2]) is also valid */
int i,j;
printf("Displaying:\n");
```

```
for(i=0;i<2;++i)
for(j=0;j<2;++j)
printf("%d\n",c[i][j]);
}
```

Output:

```
Enter 4 numbers:
2
3
4
5
Displaying:
2
3
4
5
```

Passing array element by element to function:

1. Individual element is passed to function using **Pass By Value** parameter passing scheme
2. An original Array element remains same as Actual Element is never passed to Function. Thus function body cannot modify **Original Value.**
3. Suppose we have declared an array 'arr[5]' then its individual elements are arr[0],arr[1]…arr[4]. Thus we need 5 function calls to pass complete array to a function.

Consider an array **int** arr[5] = {11, 22, 33, 44, 55};

Iteration	Element Passed to Function	Value of Element
1	arr[0]	11
2	arr[1]	22
3	arr[2]	33
4	arr[3]	44
5	arr[4]	55

Example Program #1:

```
#include< stdio.h>
#include< conio.h>
void fun(int num)
```

```
{
printf("\nElement : %d",num);
}
void main() {
int arr[5],i;
clrscr();
printf("\nEnter the array elements : ");
for(i=0;i< 5;i++)
scanf("%d",&arr[i]);
printf("\nPassing array element by element.....");
for(i=0;i< 5;i++)
fun(arr[i]);
getch();
}
```

Output:

```
Enter the array elements : 1 2 3 4 5
Passing array element by element.....
Element : 1
Element : 2
Element : 3
Element : 4
Element : 5
```

Disadvantage of this Scheme:

1. This type of scheme in which we are calling the function again and again but with **different array element is too much time consuming**. In this scheme we need to call function by pushing the current status into the system stack.
2. It is better to pass complete array to the function so that we can save some system time required for pushing and popping.
3. We can also pass the address of the individual array element to function so that function can modify the original copy of the parameter directly.

Example Program #2: Passing 1-D Array Element by Element to function

```
#include<stdio.h>
void show(int b);
void main() {
int arr[3] = {1,2,3};
int i;
for(i=0;i<3;i++)
show(arr[i]);
}
void show(int x)
{
printf("%d ",x);
}
```

Output:

```
1 2 3
```

STRINGS

- A string is a sequence of character enclosed with in double quotes (" ") but ends with \0. The compiler puts \0 at the end of string to specify the end of the string.
- To get a value of string variable we can use the two different types of formats.

Using scanf() function as: **scanf("%s", string variable);**

Using gets() function as : **gets(string variable);**

STRING HANDLING FUNCTIONS

C library supports a large number of string handling functions. Those functions are stored under the header file **string.h** in the program.

Let us see about some of the string handling functions.

(i) strlen() function

strlen() is used to return the length of the string , that means counts the number of characters present in a string.

Syntax

integer variable = strlen (string variable);

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
char str[20];
int strlength;
clrscr();
printf("Enter String:");
gets(str);
strlength=strlen(str);
printf("Given String Length Is: %d", strlength);
getch();
}
```

Output:

```
Enter String
Welcome
Given String Length Is:7
```

(ii) strcat() function

The strcat() is used to concatenate two strings. The second string will be appended to the end of the first string. This process is called concatenation.

Syntax

strcat (StringVariable1, StringVariable 2);

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
char str1[20],str2[20];
clrscr();
printf("Enter First String:");
```

```
scanf(“%s”,str1);
printf(“Enter Second String:”);
scanf(“%s”,str2);
printf(“ Concatenation String is:%s”, strcat(str1,str2));
getch();
}
```

Output:

```
Enter First String
Good
Enter Second String
Morning
Concatenation String is: GoodMorning
```

(iii) strcmp() function

strcmp() function is used to compare two strings. strcmp() function does a case sensitive comparison between two strings. The two strings are compared character by character until there is a mismatch or end of one of the strings is reached (whichever occurs first). If the two strings are identical, strcmp( ) returns a value zero. If they’re not, it returns the numeric difference between the ASCII values of the first non-matching pairs of characters.

Syntax

strcmp(StringVariable1, StringVariable2);

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
char str1[20], str2[20];
int res;
clrscr();
printf(“Enter First String:”);
scanf(“%s”,str1);
printf(“Enter Second String:”);
scanf(“%s”,str2);
```

```
res = strcmp(str1,str2);
printf(“ Compare String  Result is:%d”,res);
getch();
}
```

Output:

```
Enter First String
Good
Enter Second String
Good
Compare String Result is: 0
```

(iv) strcmpi() function

strcmpi() function is used to compare two strings. strcmpi() function is not case sensitive.

Syntax

strcmpi(StringVariable1, StringVariable2);

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
char str1[20], str2[20];
int res;
clrscr();
printf(“Enter First String:”);
scanf(“%s”,str1);
printf(“Enter Second String:”);
scanf(“%s”,str2);
res = strcmpi(str1,str2);
printf(“ Compare String  Result is:%d”,res);
getch();
}
```

Output:

```
Enter First String
WELCOME
Enter Second String
welcome
Compare String Result is: 0
```

(v) strcpy() function:

strcpy() function is used to copy one string to another. strcpy() function copy the contents of second string to first string.

Syntax

strcpy(StringVariable1, StringVariable2);

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
char str1[20], str2[20];
int res;
clrscr();
printf("Enter First String:");
scanf("%s",str1);
printf("Enter Second String:");
scanf("%s",str2);
strcpy(str1,str2)
printf(" First String is:%s",str1);
printf(" Second String is:%s",str2);
getch();
}
```

Output:

```
Enter First String
Hello
```

```
Enter Second String
welcome
First String is: welcome
Second String is: welcome
```

(vi) strlwr () function:

This function converts all characters in a given string from uppercase to lowercase letter.

Syntax

strlwr(StringVariable);

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
char str[20];
clrscr();
printf("Enter String:");
gets(str);
printf("Lowercase String : %s", strlwr(str));
getch();
}
```

Output:

```
Enter String
WELCOME
Lowercase String :    welcome
```

(vii) strrev() function:

strrev() function is used to reverse characters in a given string.

Syntax

strrev(StringVariable);

Example:

```
#include<stdio.h>
#include<conio.h>
```

```
void main()
{
char str[20];
clrscr();
printf("Enter String:");
gets(str);
printf("Reverse String : %s", strrev(str));
getch();
}
```

Output:

```
Enter String
WELCOME
Reverse String :    emoclew
```

(viii) strupr() function:

strupr() function is used to convert all characters in a given string from lower case to uppercase letter.

Syntax

strupr(Stringvariable);

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
char str[20];
clrscr();
printf("Enter String:");
gets(str);
printf("Uppercase String : %s", strupr(str));
getch();
}
```

Output:

```
Enter String
welcome
Uppercase String :    WELCOME
```

STRUCTURES

Arrays are used for storing a group of SIMILAR data items. In order to store a group of data items, we need structures. Structure is a constructed data type for packing different types of data that are logically related. The structure is analogous to the “record” of a database. Structures are used for organizing complex data in a simple and meaningful way.

Example for structures:

Student	:	regno, student_name, age, address
Book	:	bookid, bookname, author, price, edition, publisher, year
Employee	:	employeeid, employee_name, age, sex, dateofbirth, basicpay
Customer	:	custid, cust_name, cust_address, cust_phone

Structure Definition

Structures are defined first and then it is used for declaring structure variables. Let us see how to define a structure using simple example given below:

```
struct book
{
      int bookid;
      char bookname[20];
      char author[20];
      float price;
      int year;
      int  pages;
      char publisher[25];
};
```

The keyword “struct” is used for declaring a structure. In this example, book is the name of the structure or the structure tag that is defined by the struct keyword. The book structure has six fields and they are known as structure elements or structure members. Remember each structure member may be of a different data type. The structure tag name or the structure name can be used to declare variables of the structure data type.

The syntax for structure definition is given below:

```
struct tagname

{

      Data_type  member1;

      Data_type  member2;

      …….

      …….

};
```

Note:

1. To mark the completion of the template, semicolon is used at the end of the template.
2. Each structure member is declared in a separate line.

Declaring Structure Variables

First, the structure format is defined. Then the variables can be declared of that structure type. A structure can be declared in the same way as the variables are declared. There are two ways for declaring a structure variable.

1) Declaration of structure variable at the time of defining the structure (i.e structure definition and structure variable declaration are combined)

```
struct book
{
            int bookid;
            char bookname[20];
            char author[20];
            float price;
            int year;
            int  pages;
            char publisher[25];
```

```
}  b1,b2,b3;
```

The b1, b2, and b3 are structure variables of type struct book.

2) Declaration of structure variable after defining the structure

```
struct book
{
            int bookid;
            char bookname[20];
            char author[20];
            float price;
            int year;
            int pages;
            char publisher[25];
};

struct book b1, b2, b3;
```

NOTE:

- Structure tag name is optional.

E.g.

```
struct
{
int bookid;
char bookname[20];
char author[20];
float price;
int year;
int pages;
char publisher[25];
}b1, b2, b3;
```

Declaration of structure variable at a later time is not possible with this type of declaration. It is a drawback in this method. So the second method can be preferred.

- Structure members are not variables. They don't occupy memory until they are associated with a structure variable.

Accessing Structure Members

There are many ways for storing values into structure variables. The members of a structure can be accessed using a “dot operator” or “period operator”.

E.g. b1.author -> b1 is a structure variable and author is a structure member.

Syntax

STRUCTURE_Variable.STRUCTURE_Members

The different ways for storing values into structure variable is given below:

Method 1: Using Simple Assignment Statement

```
b1.pages = 786;

b1.price = 786.50;
```

Method 2: Using strcpy function

```
strcpy(b1.title, “Programming in C”);

strcpy(b1.author, “John”);
```

Method 3: Using scanf function

```
scanf(“%s \n”, b1.title);

scanf(“%d \n”, &b1.pages);
```

Example

```
#include<stdio.h>

#include<conio.h>

struct book

      {

      int bookid;

      char bookname[20];

      char author[20];
      float price;
      int year;
      int pages;
      char publisher[25];
```

```
        };
        struct book b1, b2, b3;
    main()
        {
        struct book b1;
        clrscr();
        printf("Enter the Book Id:  ");
        scanf("%d", &b1.bookid);
        printf("Enter the Book Name:  ");
        scanf("%s", b1.bookname);
        printf("Enter the Author Name:  ");
        scanf("%s", b1.author);
        printf("Enter the Price:  ");
        scanf("%f", &b1.price);
        printf("Enter the Year:  ");
        scanf("%d", &b1.year);
        printf("Enter the Total No. of Pages:  ");
        scanf("%d", &b1.pages);
        printf("Enter the Publisher Name:  ");
        scanf("%s", b1.publisher);
        printf("%d %s %d %f %d %d %s", b1.bookid, b1.bookname,
      b1.author, b1.price, b1.year, b1.pages, b1.publisher);
        getch();
        }
```

Output

```
        Enter the Book Id:  786
        Enter the Book Name:  Programming
        Enter the Author Name:  John
        Enter the Price:  123.50
        Enter the Year:  2015
        Enter the Total No. of Pages:  649
        Enter the Publisher Name:  Tata McGraw
        786   Programming     2118  123.500000      2015  649   Tata
```

Structure Initialization

Like variables, structures can also be initialized at the compile time.

Example

```
main()
   {
     struct
     {
          int  rollno;
          int attendance;
     }
   s1={786, 98};
   }
```

The above example assigns 786 to the rollno and 98 to the attendance.

Structure variable can be initialized outside the function also.

Example

```
   main()
   {
     struct student
     {
          int  rollno;
          int attendance;
     };
     struct student s1={786, 98};
     struct student s2={123, 97};
   }
```

Note:

Individual structure members cannot be initialized within the template. Initialization is possible only with the declaration of structure members.

Nested Structures or Structures within Structures

Structures can also be nested. i.e A structure can be defined inside another structure.

Example

```
   struct  employee
   {
     int empid;
     char empname[20];
```

```
  int basicpay;
  int da;
  int hra;
  int cca;
} e1;
```

In the above structure, salary details can be grouped together and defined as a separate structure.

Example

```
struct  employee
{
  int empid;
  char empname[20];
              struct
  {
        int basicpay;
        int da;
        int hra;
        int cca;
  } salary;
} e1;
```

The structure employee contains a member named salary which itself is another structure that contains four structure members. The members inside salary structure can be referred as below:

```
e1.salary.basicpay
e1.salary.da;
e1.salary.hra;
e1.salary.cca;
```

However, the inner structure member cannot be accessed without the inner structure variable.

Example

```
e1.basicpay
e1.da
e1.hra
e1.cca
```

are invalid statements

Moreover, when the inner structure variable is used, it must refer to its inner structure member. If it doesn't refer to the inner structure member then it will be considered as an error.

Example

```
e1.salary
```
(salary is not referring to any inner structure member. Hence it is wrong)

Note: C permits 15 levels of nesting and C99 permits 63 levels of nesting.

Array of Structures

A Structure variable can hold information of one particular record. For example, single record of student or employee. Suppose, if multiple records are to be maintained, it is impractical to create multiple structure variables. It is like the relationship between a variable and an array. Why do we go for an array? Because we don't want to declare multiple variables and it is practically impossible. Assume that you want to store 1000 values. Do you declare 1000 variables like a1, a2, a3…. Upto a1000? Is it easy to maintain such code ? Is it a good coding? No. It is not. Therefore, we go for Arrays. With a single name, with a single variable, we can store 1000 values. Similarly, to store 1000 records, we cannot declare 1000 structure variables. But we need "Array of Structures".

An array of structure is a group of structure elements under the same structure variables.

```
struct student s1[1000];
```

The above code creates 1000 elements of structure type student. Each element will be structure data type called student. The values can be stored into the array of structures as follows:

```
s1[0].student_age = 19;
```

Example

```
#include<stdio.h>
#include<conio.h>
struct book
{
        int bookid;
```

```
        char bookname[20];
        char author[20];
};


Struct b1[5];

main()
{
int i;
clrscr();
for (i=0;i<5;i++)
{
printf("Enter the Book Id:  ");
scanf("%d", &b1[i].bookid);
printf("Enter the Book Name:  ");
scanf("%s", b1[i].bookname);
printf("Enter the Author Name:  ");
scanf("%s", b1[i].author);
}
for (i=0;i<5;i++)
{
printf("%d \t %s \t %s \n", b1[i].bookid, b1[i].bookname,
b1[i].author);
}
getch();
}
```

Output:

```
        Enter the Book Id:  786
        Enter the Book Name:  Programming
        Enter the Author Name:  Dennis Ritchie
        Enter the Book Id:  101
        Enter the Book Name:  Java Complete Reference
        Enter the Author Name:  Herbert Schildt
        Enter the Book Id:  008
        Enter the Book Name:  Computer Graphics
```

```
Enter the Author Name:   Hearn and Baker


786   Programming   Dennis Ritchie
101   Java Complete Reference   Herbert Schildt
008   Computer Graphics   Hearn and Baker
```

Structure as Function Argument

Example

```
struct sample
{
      int no;
      float avg;
} a;
void main( )
{
      a.no=75;
      a.avg=90.25;
      fun(a);
}

void fun(struct sample p)
{
      printf("The no is=%d   Average is %f",p.no , p.avg);
}
```

Output

```
The no is 75    Average is 90.25
```

Function that returns Structure

The members of a structure can be passed to a function. If a structure is to be passed to a called function , we can use any one of the following method.

Method 1 :- Individual member of the structure is passed as an actual argument of the function call. The actual arguments are treated independently. This method is not suitable if a structure is very large structure.

Method 2:- Entire structure is passed to the called function. Since the structure declared as the argument of the function, it is local to the function only. The members are valid for the

function only. Hence if any modification done on any member of the structure , it is not reflected in the original structure.

Method 3 :- Pointers can be used for passing the structure to a user defined function. When the pointers are used , the address of the structure is copied to the function. Hence if any modification done on any member of the structure , it is reflected in the original structure.

```
Return data type  function name ( structured variable )

Structured Data type for the structured variable;

 {
    Local Variable declaration;
    Statement 1;
    Statement 2;
    --------------
    -------------
   Statement n;
   }
```

Example :

```
#include <stdio.h>
struct st
{
char name[20];
int  no;
int marks;
};
int  main( )
{
   struct st x ,y;
   int res;
   printf("\n Enter the First Record");
   scanf("%s%d%d",x.name,&x.no,&x.marks);
   printf("\n Enter the Second Record");
   scanf("%s%d%d",y.name,&y.no,&y.marks);
   res = compare ( x , y );
   if (res == 1)
         printf("\n First student has got the Highest Marks");
   else
         printf("\n Second  student has got the Highest Marks");
}
compare ( struct st st1 , struct st st2)
{
if (st1.marks > st2. marks  )
  return ( 1 );
else
```

```
  return ( 0 );
}
```

In the above example , x and y are the structures sent from the main ( ) function as the actual parameter to the formal parameters st1 and st2 of the function compare ( ).

Example program (1) – passing structure to function

```
#include<stdio.h>
#include<conio.h>
//-------------------------------------
struct Example
{
  int num1;
  int num2;
}s[3];
//-------------------------------------
void accept(struct Example *sptr)
{
  printf("\nEnter num1 : ");
  scanf("%d",&sptr->num1);
  printf("\nEnter num2 : ");
  scanf("%d",&sptr->num2);
}
//-------------------------------------
void print(struct Example *sptr)
{
  printf("\nNum1 : %d",sptr->num1);
  printf("\nNum2 : %d",sptr->num2);
}
//-------------------------------------
void main()
{
int i;
clrscr();
for(i=0;i<3;i++)
accept(&s[i]);

for(i=0;i<3;i++)
print(&s[i]);

getch();
}
```

Output :

```
Enter num1 : 10
Enter num2 : 20
```

```
Enter num1 : 30
Enter num2 : 40
Enter num1 : 50
Enter num2 : 60
Num1 : 10
Num2 : 20
Num1 : 30
Num2 : 40
Num1 : 50
Num2 : 60
```

Example program (2) – passing structure to function in C by value:

In this program, the whole structure is passed to another function by value. It means the whole structure is passed to another function with all members and their values. So, this structure can be accessed from called function. This concept is very useful while writing very big programs in C.

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

struct student
{
            int id;
            char name[20];
            float percentage;
};

void func(struct student record);

void  main()
{
            struct student record;

            record.id=1;
            strcpy(record.name, "Raju");
            record.percentage = 86.5;

            func(record);
            getch();
}

void func(struct student record)
{
            printf(" Id is: %d \n", record.id);
            printf(" Name is: %s \n", record.name);
```

```
            printf(" Percentage is: %f \n", record.percentage);
}
```

Output:

```
Id is: 1
Name is: Raju
Percentage is: 86.500000
```

Example program (3) Passing structure by value

A structure variable can be passed to the function as an argument as normal variable. If structure is passed by value, change made in structure variable in function definition does not reflect in original structure variable in calling function.

Write a C program to create a structure student, containing name and roll. Ask user the name and roll of a student in main function. Pass this structure to a function and display the information in that function.

```
#include <stdio.h>
struct student
{
    char name[50];
    int roll;
};
void Display(struct student stu);
/* function prototype should be below to the structure declaration
otherwise compiler shows error */
     int main()
{
    struct student s1;
    printf("Enter student's name: ");
    scanf("%s",&s1.name);
    printf("Enter roll number:");
    scanf("%d",&s1.roll);
    Display(s1);   // passing structure variable s1 as argument
    return 0;
}
void Display(struct student stu){
  printf("Output\nName: %s",stu.name);
  printf("\nRoll: %d",stu.roll);
}
```

Output

```
Enter student's name: Kevin Amla
Enter roll number: 149
Output
```

Example program (4) – Passing structure to function in C by address:

In this program, the whole structure is passed to another function by address. It means only the address of the structure is passed to another function. The whole structure is not passed to another function with all members and their values. So, this structure can be accessed from called function by its address.

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

struct student
{
           int id;
           char name[20];
           float percentage;
};

void func(struct student *record);

void  main()
{
          struct student record;

          record.id=1;
          strcpy(record.name, "Raju");
          record.percentage = 86.5;

          func(&record);
          getch();
}

void func(struct student *record)
{
          printf(" Id is: %d \n", record->id);
          printf(" Name is: %s \n", record->name);
          printf(" Percentage is: %f \n", record->percentage);
}
```

Output:

```
Id is: 1
Name is: Raju
Percentage is: 86.500000
```

Example program (5) Passing structure by reference

The address location of structure variable is passed to function while passing it by reference. If structure is passed by reference, change made in structure variable in function definition reflects in original structure variable in the calling function.

Write a C program to add two distances(feet-inch system) entered by user. To solve this program, make a structure. Pass two structure variable (containing distance in feet and inch) to add function by reference and display the result in main function without returning it.

```
#include <stdio.h>
struct distance
{
    int feet;
    float inch;
};
void Add(struct distance d1,struct distance d2, struct distance *d3);
int main()
{
    struct distance dist1, dist2, dist3;
    printf("First distance\n");
    printf("Enter feet: ");
    scanf("%d",&dist1.feet);
    printf("Enter inch: ");
    scanf("%f",&dist1.inch);
    printf("Second distance\n");
    printf("Enter feet: ");
    scanf("%d",&dist2.feet);
    printf("Enter inch: ");
    scanf("%f",&dist2.inch);
    Add(dist1, dist2, &dist3);

/*passing structure variables dist1 and dist2 by value whereas passing
structure variable dist3 by reference */
    printf("\nSum of distances = %d\'-%.1f\"",dist3.feet, dist3.inch);
    return 0;
}
void Add(struct distance d1,struct distance d2, struct distance *d3)
{
/* Adding distances d1 and d2 and storing it in d3 */
     d3->feet=d1.feet+d2.feet;
     d3->inch=d1.inch+d2.inch;
     if (d3->inch>=12) {     /* if inch is greater or equal to 12,
converting it to feet. */
         d3->inch-=12;
         ++d3->feet;
    }
}
```

Output

```
First distance
Enter feet: 12
Enter inch: 6.8
```

```
Second distance
Enter feet: 5
Enter inch: 7.5
Sum of distances = 18'-2.3"
```

Explanation

In this program, structure variables *dist1* and *dist2* are passed by value (because value of *dist1* and *dist2* does not need to be displayed in main function) and *dist3* is passed by reference ,i.e, address of *dist3* (&dist3) is passed as an argument. Thus, the structure pointer variable *d3* points to the address of *dist3*. If any change is made in *d3* variable, effect of it is seed in *dist3* variable in main function.

Example program(6) to declare a structure variable as global in C:

Structure variables also can be declared as global variables as we declare other variables in C. So, When a structure variable is declared as global, then it is visible to all the functions in a program. In this scenario, we don't need to pass the structure to any function separately.

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

struct student
{
            int id;
            char name[20];
            float percentage;
};
struct student record; // Global declaration of structure

void structure_demo();

int main()
{
            record.id=1;
            strcpy(record.name, "Raju");
            record.percentage = 86.5;

            structure_demo();
            return 0;
}

void structure_demo()
{
            printf(" Id is: %d \n", record.id);
            printf(" Name is: %s \n", record.name);
            printf(" Percentage is: %f \n", record.percentage);
}
```

Output:

```
Id is: 1
Name is: Raju
Percentage is: 86.500000
```

Example program(7)Passing Array of Structure to Function in C Programming

Array of Structure can be passed to function as a Parameter.function can also return Structure as return type.Structure can be passed as follow

Example :

```
#include<stdio.h>
#include<conio.h>
//-------------------------------------
struct Example
{
  int num1;
  int num2;
}s[3];
//-------------------------------------
void accept(struct Example sptr[],int n)
{
  int i;
  for(i=0;i<n;i++)
  {
  printf("\nEnter num1 : ");
  scanf("%d",&sptr[i].num1);
  printf("\nEnter num2 : ");
  scanf("%d",&sptr[i].num2);
  }
}
//-------------------------------------
void print(struct Example sptr[],int n)
{
  int i;
  for(i=0;i<n;i++)
  {
  printf("\nNum1 : %d",sptr[i].num1);
  printf("\nNum2 : %d",sptr[i].num2);
  }
}
//-------------------------------------
void main()
{
int i;
```

```
clrscr();
accept(s,3);
print(s,3);
getch();
}
Output :
Enter num1 : 10
Enter num2 : 20
Enter num1 : 30
Enter num2 : 40
Enter num1 : 50
Enter num2 : 60
Num1 : 10
Num2 : 20
Num1 : 30
Num2 : 40
Num1 : 50
Num2 : 60
```

Explanation :

Inside main structure and size of structure array is passed. When reference (i.e ampersand) is not specified in main , so this passing is simple pass by value. Elements can be accessed by using dot [.] operator

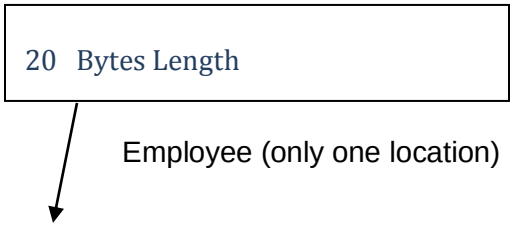
Union

The concept of Union is borrowed from structures and the formats are also same. The distinction between them is in terms of storage. In structures , each member is stored in its own location but in Union , all the members are sharing the same location. Though Union consists of more than one members , only one member can be used at a particular time. The size of the cell allocated for an Union variable depends upon the size of any member within Union occupying more no:- of bytes. The syntax is the same as structures but we use the keyword **union** instead of **struct.**

Example:- the employee record is declared and processed as follows

```
union  emp
    {
        char name[20];
        int  eno;
        float salary;
     } employee;
```

where employee is the union variable which consists of the member name,no and salary. The compiler allocates only one cell for the union variable as



Location / Cell for name,no and salary

20 bytes cell can be shared by all the members because the member name is occupying the highest no:- of bytes. At a particular time we can handle only one member.To access the members of an union , we have to use the same format of structures.

Example program for C union:

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

union student
{
            char name[20];
            char subject[20];
            float percentage;
};

int main()
{
    union student record1;
    union student record2;

    // assigning values to record1 union variable
       strcpy(record1.name, "Raju");
       strcpy(record1.subject, "Maths");
       record1.percentage = 86.50;

       printf("Union record1 values example\n");
       printf(" Name       : %s \n", record1.name);
       printf(" Subject    : %s \n", record1.subject);
       printf(" Percentage : %f \n\n", record1.percentage);

    // assigning values to record2 union variable
       printf("Union record2 values example\n");
       strcpy(record2.name, "Mani");
       printf(" Name       : %s \n", record2.name);

       strcpy(record2.subject, "Physics");
       printf(" Subject    : %s \n", record2.subject);
```

```
        record2.percentage = 99.50;
        printf(" Percentage : %f \n", record2.percentage);
        return 0;
}
```

Output:

```
Union record1 values example
Name :
Subject :
Percentage : 86.500000;
Union record2 values example
Name : Mani
Subject : Physics
Percentage : 99.500000
```

Explanation for above C union program:

There are 2 union variables declared in this program to understand the difference in accessing values of union members.

Record1 union variable:

- "Raju" is assigned to union member "record1.name" . The memory location name is "record1.name" and the value stored in this location is "Raju".
- Then, "Maths" is assigned to union member "record1.subject". Now, memory location name is changed to "record1.subject" with the value "Maths" (Union can hold only one member at a time).
- Then, "86.50" is assigned to union member "record1.percentage". Now, memory location name is changed to "record1.percentage" with value "86.50".
- Like this, name and value of union member is replaced every time on the common storage space.
- So, we can always access only one union member for which value is assigned at last. We can't access other member values.
- So, only "record1.percentage" value is displayed in output. "record1.name" and "record1.percentage" are empty.

Record2 union variable:

- If we want to access all member values using union, we have to access the member before assigning values to other members as shown in record2 union variable in this program.
- Each union members are accessed in record2 example immediately after assigning values to them.
- If we don't access them before assigning values to other member, member name and value will be over written by other member as all members are using same memory.
- We can't access all members in union at same time but structure can do that.

Example program – Another way of declaring C union:

In this program, union variable "record" is declared while declaring union itself as shown in the below program.

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

```
union student
{
            char name[20];
            char subject[20];
            float percentage;
}record;

int main()
{

            strcpy(record.name, "Raju");
            strcpy(record.subject, "Maths");
            record.percentage = 86.50;

            printf(" Name       : %s \n", record.name);
            printf(" Subject    : %s \n", record.subject);
            printf(" Percentage : %f \n", record.percentage);
            return 0;
}
```

Output:

```
Name :
Subject :
Percentage : 86.500000
```

Note:

We can access only one member of union at a time. We can't access all member values at the same time in union. But, structure can access all member values at the same time. This is because, Union allocates one common storage space for all its members. Where as Structure allocates storage space for all its members separately.

Difference between structure and union in C:

S.no	C Structure	C Union
1	Structure allocates storage space for all its members separately.	Union allocates one common storage space for all its members. Union finds that which of its member needs high storage space over other members and allocates that much space
2	Structure occupies higher memory space.	Union occupies lower memory space over structure.
3	We can access all members of structure at a time.	We can access only one member of union at a time.
4	Structure example: `struct student`	Union example: `union student`

	{
	int mark;
	char name[6];
	double average;
	};
		{
		int mark;
		char name[6];
		double average;
		};

5	For above structure, memory allocation will be like below. int mark – 2B char name[6] – 6B double average – 8B Total memory allocation = 2+6+8 = 16 Bytes	For above union, only 8 bytes of memory will be allocated since double data type will occupy maximum space of memory over other data types. Total memory allocation = 8 Bytes

Assignment Question

1. Create a structure to store the employee number, name, department and basic salary. Create a array of structure to accept and display the values of 10 employees.

PRACTICE QUESTIONS

Programs for Practice:

1) Write a C program to initialize an array using functions.
2) Write a C program to interchange array elements of two arrays using functions.
3) Write a C program to pass an array containing age of person to a function. This function should find average age and display the average age in main function.
4) Write a c program to check whether a given string is a palindrome or not
5) What would be the output of the following programs:

```
main( )
    {
  char  c[2] = "A" ;
  printf ( "\n%c", c[0] ) ;
  printf ( "\n%s", c ) ;
```

```
      }
main( )
{
   char str1[ ] = { 'H', 'e', 'l', 'l', 'o'  } ;
   char str2[ ] = "Hello" ;
   printf ( "\n%s", str1 ) ;
   printf ( "\n%s", str2 ) ;
  }
```

a)
```
main(  )
   {
   printf ( 5 + "Good Morning " ) ;
     }
```

6) Point out the errors,if any,in the following programs

(a)
```
main( )
     {
          char *str1 = "United" ;
          char *str2 = "Front" ;
          char *str3 ;
          str3 = strcat ( str1, str2 ) ;
          printf ( "\n%s", str3 ) ;
}
```

7) Which is more appropriate for reading in a multi-word string?

` gets( ) printf( ) scanf( ) puts( )

8) If the string "Alice in wonder land" is feed to the following

scanf( ) statement, what will be the contents of the arrays

str1, str2, str3 and str4 ?

```
scanf ( "%s%s%s%s%s", str1, str2, str3, str4 ) ;
```

9) Fill in the blanks:

a. "A" is a _____ while 'A' is a _____.

b. A string is terminated by a _____ character, which is written as _____.

c. The array char name [10] can consist of a maximum of _____ characters.

1. Write a C program to initialize an array using functions.

```
#include<stdio.h>
int main()
int k, c(), d[ ]={c(),c(),c(),c(),c()};
printf("\nArray d[] elements are:");
for(k=0;k<5;k++)
printf("%2d",d[k]);
return(0);
}
c()
{
int m,n;
m++;
printf("\nEnter number d[%d] : ",m);
scanf("%d",&n);
return(n);
}
```

Output:

```
Enter Number d[1] : 20
Enter Number d[2] : 30
Enter Number d[3] : 40
Enter Number d[4] : 50
Enter Number d[5] : 60
```

Array d[] elements are: 20 30 40 50 60

2. **Write a C program to interchange array elements of two arrays using functions.**

```
#include<stdio.h>
#include<conio.h>
void main()
{
int read();
void change(int*,int*);
int x,a[5],b[5];
clrscr();
printf("Enter 10 Numbers :");
for(x=0;x<10;x++)
{
if(x<5)
a[x]=read();
else
b[x-5]=read();
}
printf("\nArray A & B");
for(x=0;x<5;x++)
{
printf("\n%7d%8d",a[x],b[x]);
change(&a[x],&b[x]);
}
printf("\nNow A & B");
for(x=0;x<5;x++)
{
printf("\n%7d%8d",a[x],b[x]);
}
}
```

```
int read()
{
int x;
scanf(“%d”,&x);
return(x);
}
void change(int *a,int *b)
{
int *k;
*a=*a+*b;
*b=*a-*b;
*a=*a-*b;
}
```

Output:

```
Enter 10 Numbers:
0 1 2 3 4 5 6 7 8 9
Array A & B
0  5
1  6
2  7
3  8
4  9
Now A & B
5  0
6  1
7  2
8  3
9  4
```

3. **Write a C program to pass an array containing age of person to a function. This function should find average age and display the average age in main function.**

```
#include <stdio.h>
float average(float a[]);
int main(){
float avg, c[]={23.4, 55, 22.6, 3, 40.5, 18};
avg=average(c);   /* Only name of array is passed as argument. */
printf("Average age=%.2f",avg);
return 0;
}
float average(float a[])
{
int i;
float avg, sum=0.0;
for(i=0;i<6;++i){
sum+=a[i];
}
avg =(sum/6);
return avg;
}
```

Output:

```
Average age=27.08
```