UNIT III

Arrays : Single and Multidimensional Arrays-– Array as Function Arguments, Strings: String Handling Functions, Structure: Nested Structures – Array of Structures – Structure as Function Argument–Function that Returns Structure, Union.

Algorithms: Sum of array elements- Removal of duplicates from an array-Finding the K^{th} smallest element.

Array

The array is the simplest data structure where each data element can be randomly accessed by using its index number.

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.

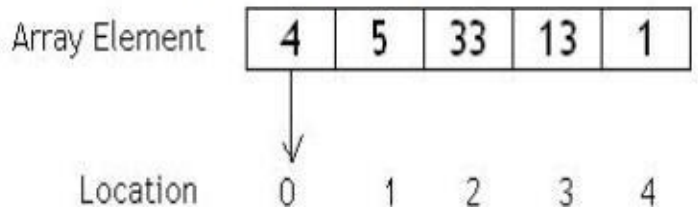
- An array is a linear and homogeneous data structure
- An array permits homogeneous data. It means that similar types of elements are stored contiguously in the memory under one variable name.
- An array can be declared of any standard or custom data type.

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?

a. When Big Block of memory is reserved or allocated then that memory block is called as Contiguous Memory Block.
b. Alternate meaning of Contiguous Memory is continuous memory.
c. 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.

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: array name [size1][size2] ....[sizen];

Where Data type ➔ Data Type of Each Element of the array and Data Type specifies the type of the array. We can compute the size required for storing the single cell of array,

Array name ➔ Valid identifier is any valid variable or name given to the array. Using this identifier name array can be accessed.

Size ➔ Dimensions of the Array, It is maximum size that array can have.

What does Array Declaration tell to Compiler?

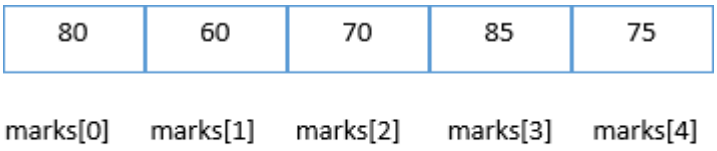
1. Type of the Array
2. Name of the Array
3. Number of Dimension
4. Number of Elements in Each Dimension

Initialization of Array

Initialization means assigning initial variable to an array. The simplest way to initialize an array is by using the index of each element. We can initialize each element of the array by using the index. Consider the following example.

```
int a[10] = { 10, 9, 7, 101, 23, 44, 12, 78, 34, 23};
marks[0]=80;//initialization of array
marks[1]=60;
marks[2]=70;
marks[3]=85;
marks[4]=75;
```

where marks is the name of the array.



Initialization of Array

Example Program for Array Intialization :

```
#include<stdio.h>
int main()
{
int i=0;
```

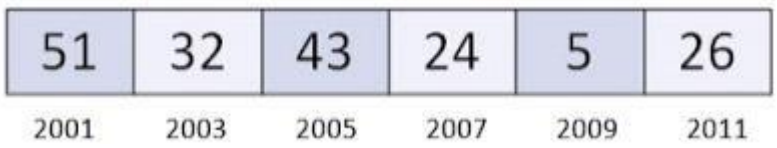
```
int marks[5];//declaration of array
marks[0]=80;//initialization of array
marks[1]=60;
marks[2]=70;
marks[3]=85;
marks[4]=75;
//traversal of array
for(i=0;i<5;i++){
printf("%d \n",marks[i]);
}//end of for loop
return 0;
}
```

Output

```
80
60
70
85
75
```

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;

TYPES OF ARRAY:

1. One dimensional array
2. Two dimensional array or multi dimensional array.

ONE DIMENSIONAL ARRAY:

In this we deal with only one dimension when storing data in the memory.

Example:

```
// Program to find the average of n (n < 10) numbers using arrays
#include <stdio.h>
int main()
{
int marks[10], i, n, sum = 0, average;
printf("Enter n: ");
scanf("%d", &n);
for(i=0; i<n; ++i)
{
printf("Enter number%d: ",i+1);
scanf("%d", &marks[i]);
sum += marks[i];
}
average = sum/n;
printf("Average = %d", average);
return 0;
}
```

Output:

```
Enter n: 5
Enter number1: 45
```

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 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

4. C Program to find largest element in an array

```
#include 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

5. 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

TWO DIMENSIONAL ARRAY:

The two-dimensional array can be defined as an array of arrays.
The 2D array is organized as matrices which can be represented as the collection of rows and columns.

DECLARING A 2D ARRAY:

arr[rows][column];

	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]

INITIALIZING TWO-DIMENSIONAL ARRAYS:

Two dimensional arrays may be initialized by specifying bracketed values for each row. Following is an array with 3 rows and each row has 4 columns.

int a[2][[2] = { {1,2}, {3,4} }; Where a[2][2] means there are two rows and two columns, in the nested brackets, the first set denotes values in the first row and next set in the second row.

as in the matrix 0,0 0,1 positions as in row and column.

0, 1 1, 1

values will appear as 1 2 according to the positions.

3 4

Example:

```
#include<stdio.h>
int main()
{
int i=0,j=0;
int arr[4][3]={{1,2,3},{2,3,4},{3,4,5},{4,5,6}};
//traversing 2D array
 for(i=0;i<4;i++) // since row size is 4
{
 for(j=0;j<3;j++) //since column size is 3
{
   printf("arr[%d] [%d] = %d \n",i,j,arr[i][j]);
 }//end of j
}//end of i
```

```
return 0;
}
```

Output:

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

ARRAYS AS FUNCTIONS:

- A single array element or an entire array can be passed to a function.
- This can be done for both one-dimensional array or a multi-dimensional array.

PASSING ONE-DIMENSIONAL ARRAY IN FUNCTION

Single element of an array can be passed in similar manner as passing variable to a function.

Example:

```
#include <stdio.h>
int main()
{
int ageArray[] = { 2, 3, 4 };
display(ageArray[2]); //Passing array element ageArray[2] only.
return 0;
}
void display(int age)
{
printf("%d", age);
}
```

STRINGS

- The string can be defined as the one-dimensional array of characters terminated by a null ('\0').
- The character array or the string is used to manipulate text such as word or sentences.
- Each character in the array occupies one byte of memory, and the last character must always be 0.
- The termination character ('\0') is important in a string since it is the only way to identify where the string ends.
- When we define a string as char s[10], the character s[10] is implicitly initialized with the null in the memory.

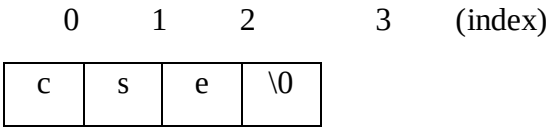
There are two ways to declare a string in c language.

1. By char array
2. By string literal

Let's see the example of declaring string by char array

char ch[5]= {‘c’,’s’,’e’};

As we know, array index starts from 0, so it will be represented as in the figure given below.



While declaring string, size is not mandatory. So we can write the above code as given below:

Char ch[]= {‘c’,’s’,’e’};

We can also define the string by the string literal in C language. For example:

char ch[]= “cse”;

In such case, '\0' will be appended at the end of the string by the compiler.

Example:

```
#include<stdio.h>
#include <string.h>
int main(){
  char ch[11]= {‘c’,’s’,’e’};
   char ch2[11]="cse";
   printf("Char Array Value is: %s\n", ch);
 printf("String Literal Value is: %s\n", ch2);
 return 0;
}
```

Output:

Char Array Value is: cse
String Literal Value is:cse

TRAVERSING STRING

Traversing the string is one of the most important aspects in any of the programming languages. We may need to manipulate a very large text which can be done by traversing the text. Traversing string is somewhat different from the traversing an integer array. We need to know the length of the array to traverse an integer array, whereas we may use the null character in the case of string to identify the end the string and terminate the loop.

Hence, there are two ways to traverse a string.

- By using the length of string
- By using the null character.

USING THE LENGTH OF STRING

Let's see an example of counting the number of vowels in a string.

```
#include<stdio.h>
void main ()
{
   char s[11] = "javatpoint";
   int i = 0;
   int count = 0;
   while(i<11)
   {
      if(s[i]=='a' || s[i] == 'e' || s[i] == 'i' || s[i] == 'u' || s[i] == 'o')
      {
         count ++;
      }
      i++;
   }
   printf("The number of vowels is %d",count);
}
```

Output:
The number of vowels is 4

USING THE NULL CHARACTER

Let's see the same example of counting the number of vowels by using the null character.

```
#include<stdio.h>
void main ()
{
   char s[11] = "javatpoint";
   int i = 0;
   int count = 0;
   while(s[i] != NULL)
   {
      if(s[i]=='a' || s[i] == 'e' || s[i] == 'i' || s[i] == 'u' || s[i] == 'o')
      {
         count ++;
      }
      i++;
   }
   printf("The number of vowels is %d",count);
}
```

Output:

The number of vowels is 4

C GETS() AND PUTS() FUNCTIONS

The gets() and puts() are declared in the header file stdio.h. Both the functions are involved in the input/output operations of the strings.

C GETS() FUNCTION

The gets() function enables the user to enter some characters followed by the enter key. All the characters entered by the user get stored in a character array. The null character is added to the array to make it a string. The gets() allows the user to enter the space-separated strings. It returns the string entered by the user.

DECLARATION

char[] gets(char[]);

C PUTS() FUNCTION

The puts() function is very much similar to printf() function. The puts() function is used to print the string on the console which is previously read by using gets() or scanf() function. The puts() function returns an integer value representing the number of characters being printed on the console. Since, it prints an additional newline character with the string, which moves the cursor to the new line on the console, the integer value returned by puts() will always be equal to the number of characters present in the string plus 1.

DECLARATION

int puts(char[])

Example:

```
#include<stdio.h>
#include <string.h>
int main(){
char name[50];
printf("Enter your name: ");
gets(name); //reads string from user
printf("Your name is: ");
puts(name);  //displays  string
return 0;
}
```

Output:

```
Enter your name: abc
Your name is: abc
```

STRING FUNCTIONS

There are many important string functions defined in "string.h" library.

S.No	Function	Description
1.	strlen(string_name)	returns the length of string name.
2.	strcpy(destination, source)	copies the contents of source string to destination string.
3.	strcat(first_string, second_string)	concats or joins first string with second string. The result of the string is stored in first string.
4.	strcmp(first_string, second_string)	compares the first string with second string. If both strings are same, it returns 0.
5.	strrev(string)	returns reverse string.
6.	strlwr(string)	returns string characters in lowercase.
7.	strupr(string)	returns string characters in uppercase.

STRING LENGTH: STRLEN() FUNCTION

The strlen() function returns the length of the given string. It doesn't count null character '\0'

Example:

```
#include<stdio.h>
#include <string.h>
int main(){
char ch[20]= {'c', 's', 'e', '\0'};    };
 printf("Length of string is: %d",strlen(ch));
 return 0;
}
```

Output:

```
Length of string is: 3
```

COPY STRING: STRCPY()

The strcpy(destination, source) function copies the source string in destination.

Example:

```
#include<stdio.h>
#include <string.h>
int main(){
 char ch[5]={'c', 's', 'e', '\0'};
   char ch2[20];
   strcpy(ch2,ch);
```

```
   printf("Value of second string is: %s",ch2);
 return 0;
}
```

Output:

Value of second string is: cse

CONCATENATION: STRCAT()

The strcat(first_string, second_string) function concatenates two strings and result is returned to first_string.

Example:

```
#include<stdio.h>
#include <string.h>
int main(){
  char ch[10]={'h', 'e', 'l', 'l', 'o', '\0'};
   char ch2[10]={'c', '\0'};
   strcat(ch,ch2);
   printf("Value of first string is: %s",ch);
 return 0;
}
```

Output:
Value of first string is: helloc

COMPARE STRING: STRCMP()

The strcmp(first_string, second_string) function compares two string and returns 0 if both strings are equal.

Here, we are using *gets()* function which reads string from the console.

Example:

```
#include<stdio.h>
#include <string.h>
int main(){
  char str1[20],str2[20];
  printf("Enter 1st string: ");
  gets(str1);//reads string from console
  printf("Enter 2nd string: ");
  gets(str2);
  if(strcmp(str1,str2)==0)
     printf("Strings are equal");
  else
     printf("Strings are not equal");
```

```
 return 0;
}
```

Output:
Enter 1st string: hello
Enter 2nd string: hello
Strings are equal

REVERSE STRING: STRREV()

The strrev(string) function returns reverse of the given string. Let's see a simple example of strrev() function.

Example:

```
#include<stdio.h>

#include <string.h>
int main(){
  char str[20];
  printf("Enter string: ");
  gets(str);//reads string from console
  printf("String is:  %s",str);
  printf("\nReverse String is: %s",strrev(str));
 return 0;
}
```

Output:
Enter string: cse
String is: cse
Reverse String is: esc

UPPERCASE: STRUPR()

The strupr(string) function returns string characters in uppercase. Let's see a simple example of strupr() function.

Example:

```
#include<stdio.h>
#include <string.h>
int main(){
  char str[20];
  printf("Enter string: ");
  gets(str);//reads string from console
  printf("String is: %s",str);
  printf("\nUpper String is: %s",strupr(str));
 return 0;
}
```

Output:
Enter string: cse
String is: cse
Upper String is: CSE

LOWERCASE: STRLWR()

The strlwr(string) function returns string characters in lowercase. Let's see a simple example of strlwr() function.

Example:

```
#include<stdio.h>
#include <string.h>
int main(){
  char str[20];
  printf("Enter string: ");
  gets(str);//reads string from console
  printf("String is: %s",str);
  printf("\nLower String is: %s",strlwr(str));
 return 0;
}
```

Output:

Enter string: CSE
String is: CSE
Lower String is: cse

STRUCTURE

- Structure in c is a user-defined data type that enables us to store the collection of different data types.
- Each element of a structure is called a member.
- The **struct** keyword is used to define the structure.

Let's see the **SYNTAX** to define the structure in c.

```
struct structure_name
{
   data_type member1;
   data_type member2;
   .
   .
   data_type memeberN;
};
```

Let's see the **example** to define a structure for an entity employee in c.

```
struct employee
{ int id;
   char name[20];
   float salary;
};
```

Here, **struct** is the keyword; **employee** is the name of the structure; **id**, **name**, and **salary** are the members or fields of the structure.

DECLARING STRUCTURE VARIABLE

We can declare a variable for the structure so that we can access the member of the structure easily. There are two ways to declare structure variable:

1. By struct keyword within main() function
2. By declaring a variable at the time of defining the structure.

1st way:

Let's see the example to declare the structure variable by struct keyword. It should be declared within the main function.

struct employee e1, e2;

The variables e1 and e2 can be used to access the values stored in the structure. Here, e1 and e2 can be treated in the same way as the objects in C++ and Java.

2nd way:

Let's see another way to declare variable at the time of defining the structure.

```
struct employee
{ int id;
   char name[50];
   float salary;
}e1,e2;
```

Comparing both approach to see which is good:

- If number of variables are not fixed, use the 1st approach. It provides you the flexibility to declare the structure variable many times.
- If no. of variables are fixed, use 2nd approach. It saves your code to declare a variable in main() function.

ACCESSING MEMBERS OF THE STRUCTURE

There are two ways to access structure members:

1. By . (member or dot operator)
2. By -> (structure pointer operator)

Let's see the code to access the *id* member of e*1* variable by. (member) operator then access is done like this e1.id

Example:

```
#include<stdio.h>
#include <string.h>
struct employee
{ int id;
   char name[50];
}e1; //declaring e1 variable for structure
int main( )
{
  //store first employee information
  e1.id=101;
  strcpy(e1.name, "abc");//copying string into char array
  //printing first employee information
  printf( "employee 1 id : %d\n", e1.id);
  printf( "employee 1 name : %s\n", e1.name);
return 0;
}
```

Output:

```
employee 1 id : 101
employee 1 name : abc
```

NESTED STRUCTURE IN C

- C provides us the feature of nesting one structure within another structure by using which, complex data types are created.
- For example, we may need to store the address of an entity employee in a structure.
- The attribute address may also have the subparts as street number, city, state, and pin code.
- Hence, to store the address of the employee, we need to store the address of the employee into a separate structure and nest the structure address into the structure employee.

Consider the following program as an exapmle.

```
#include<stdio.h>
struct address
{
   char city[20];
   int pin;
```

```
   char phone[14];
};
struct employee
{
   char name[20];
   struct address add;
};
void main ()
{
   struct employee emp;
   printf("Enter employee information?\n");
   scanf("%s %s %d %s",emp.name,emp.add.city, &emp.add.pin, emp.add.phone);
   printf("Printing the employee information. ..\n");
   printf("name: %s\nCity: %s\nPincode: %d\nPhone: %s",emp.name,emp.add.city,emp.add.
pin,emp.add.phone;}
```

Output:

Enter employee information?

Arun

Delhi

110001

1234567890

Printing the employee information....

name: Arun

City: Delhi

Pincode: 110001

Phone: 1234567890

The structure can be nested in the following ways.

1. By separate structure
2. By Embedded structure

SEPARATE STRUCTURE

Here, we create two structures, but the dependent structure should be used inside the main structure as a member. Consider the following example.

```
struct Date
{
```

```
  int dd;
  int mm;
  int yyyy;
};
struct Employee
{
  int id;
  char name[20];
  struct Date doj;
}emp1;
```

As you can see, doj (date of joining) is the variable of type Date. Here doj is used as a member in Employee structure. In this way, we can use Date structure in many structures.

EMBEDDED STRUCTURE

The embedded structure enables us to declare the structure inside the structure. Hence, it requires less line of codes but it can not be used in multiple data structures. Consider the following example.

```
struct Employee
{
  int id;
  char name[20];
  struct Date
   {
     int dd;
     int mm;
     int yyyy;
   }doj;
}emp1;
```

ACCESSING NESTED STRUCTURE

We can access the member of the nested structure by Outer_Structure.Nested_Structure.member as given below:

1. e1.doj.dd
2. e1.doj.mm
3. e1.doj.yyyy

Nested Structure example

```
#include <stdio.h>
#include <string.h>
struct Employee
{
  int id;
  char name[20];
  struct Date
```

```
    {
      int dd;
      int mm;
      int yyyy;
    }doj;
}e1;
int main( )
{
   //storing employee information
   e1.id=101;
   strcpy(e1.name, "Sonoo Jaiswal");//copying string into char array
   e1.doj.dd=10;
   e1.doj.mm=11;
   e1.doj.yyyy=2014;
  //printing first employee information
   printf( "employee id : %d\n", e1.id);
   printf( "employee name : %s\n", e1.name);
   printf( "employee date of joining (dd/mm/yyyy) : %d/%d/%d\n", e1.doj.dd,e1.doj.mm,e1.d
oj.yyyy);
   return 0;
}
```

Output:

```
employee id : 101
employee name : Sonoo Jaiswal
employee date of joining (dd/mm/yyyy) : 10/11/2014
```

ARRAY OF STRUCTURES

WHY USE AN ARRAY OF STRUCTURES?

Consider a case, where we need to store the data of 5 students. We can store it by using the structure as given below.

Example:

```
#include<stdio.h>
struct student
{
    char name[20];
    int id;
    float marks;
};
void main()
{
    struct student s1,s2,s3;
    printf("Enter the name, id, and marks of student 1 ");
    scanf("%s %d %f",s1.name,&s1.id,&s1.marks);
    printf("Enter the name, id, and marks of student 2 ");
    scanf("%s %d %f",s2.name,&s2.id,&s2.marks);
```

```
    printf("Enter the name, id, and marks of student 3 ");
    scanf("%s %d %f",s3.name,&s3.id,&s3.marks);
    printf("Printing the details. ..\n");
    printf("%s %d %f\n",s1.name,s1.id,s1.marks);
    printf("%s %d %f\n",s2.name,s2.id,s2.marks);
    printf("%s %d %f\n",s3.name,s3.id,s3.marks);
}
```

Output:

```
Enter the name, id, and marks of student 1 James 90 90
Enter the name, id, and marks of student 2 Adoms 90 90
Enter the name, id, and marks of student 3 Nick 90 90
Printing the details....
James 90 90.000000
Adoms 90 90.000000
Nick 90 90.000000
```

- In the above program, we have stored data of 3 students in the structure.
- However, the complexity of the program will be increased if there are 20 students.
- In that case, we will have to declare 20 different structure variables and store them one by one.
- This will always be tough since we will have to declare a variable every time we add a student.
- Remembering the name of all the variables is also a very tricky task.
- However, c enables us to declare an array of structures by using which, we can avoid declaring the different structure variables; instead we can make a collection containing all the structures that store the information of different entities.

ARRAY OF STRUCTURES IN C

An array of structres in C can be defined as the collection of multiple structures variables where each variable contains information about different entities. The array of structures in C are used to store information about multiple entities of different data types. The array of structures is also known as the collection of structures.

Let's see an example of an array of structures that stores information of 5 students and prints it.

```
#include<stdio.h>
#include <string.h>
struct student{
int rollno;
char name[10];
};
int main(){
int i;
struct student st[5];
printf("Enter Records of 5 students");
for(i=0;i<5;i++){
printf("\nEnter Rollno:");
```

```
scanf("%d",&st[i].rollno);
printf("\nEnter Name:");
scanf("%s",&st[i].name);
}
printf("\nStudent Information List:");
for(i=0;i<5;i++){
printf("\nRollno:%d, Name:%s",st[i].rollno,st[i].name);
}
  return 0;
}
```

Output:

```
Enter Records of 5 students
Enter Rollno:1
Enter Name:Sonoo
Enter Rollno:2
Enter Name:Ratan
Enter Rollno:3
Enter Name:Vimal
Enter Rollno:4
Enter Name:James
Enter Rollno:5
Enter Name:Sarfraz

Student Information List:
Rollno:1,  Name:Sonoo
Rollno:2,   Name:Ratan
Rollno:3,  Name:Vimal
Rollno:4,  Name:James
Rollno:5, Name:Sarfraz
```

STRUCTURE IN FUNCTIONS

A structure can be passed to any function from main function or from any sub function.

Structure definition will be available within the function only.

It won't be available to other functions unless it is passed to those functions by value or by address (reference).

Else, we have to declare structure variable as global variable. That means, structure variable should be declared outside the main function. So, this structure will be visible to all the functions in a C program.

PASSING STRUCTURE TO FUNCTION IN C:

It can be done in below 3 ways.

1. Passing structure to a function by value
2. Passing structure to a function by address(reference)
3. No need to pass a structure – Declare structure variable as global

EXAMPLE PROGRAM – 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.

Example:

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

struct student
{
        int id;
        char name[20];
        float percentage;
};
void func(struct student record);

int main()
{
        struct student record;
        record.id=1;
        strcpy(record.name, "Raju");
        record.percentage = 86.5;
        func(record);
        return 0;
}

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);
}
```

Ouput:

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

EXAMPLE PROGRAM – 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);

int main()
{
       struct student record;

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

       func(&record);
       return 0;
}

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 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
```

FUNCTION RETURNING STRUCTURE

Structure is user-defined data type, like built-in data types structure can be return from function.

Example:

```
#include<stdio.h>
    struct Employee
    {
        int Id;
        char Name[25];
        int Age;
        long Salary;
    };
    Employee Input();          //Statement 1
    void main()
    {
        struct Employee Emp;
```

```
    Emp = Input();
    printf("\n\nEmployee Id : %d",Emp.Id);
    printf("\nEmployee Name : %s",Emp.Name);
    printf("\nEmployee Age : %d",Emp.Age);
    printf("\nEmployee Salary : %ld",Emp.Salary);
}
Employee Input()
{
    struct Employee E;
        printf("\nEnter Employee Id : ");
        scanf("%d",&E.Id);
        printf("\nEnter Employee Name : ");
        scanf("%s",&E.Name);
        printf("\nEnter Employee Age : ");
        scanf("%d",&E.Age);
        printf("\nEnter Employee Salary : ");
        scanf("%ld",&E.Salary);
    return E;           //Statement  2
}
```

Output :

```
Enter Employee Id : 10
Enter Employee Name : Ajay
Enter Employee Age : 25
Enter Employee Salary : 15000


Employee Id : 10
Employee Name : Ajay
Employee Age : 25
Employee Salary : 15000
```

In the above example, statement 1 is declaring Input() with return type Employee. As we know structure is user-defined data type and structure name acts as our new user-defined data type, therefore we use structure name as function return type. Input() have local

variable E of Employee type. After getting values from user statement 2 returns E to the calling function and display the values.

UNION:

- C Union is also like structure, i.e. collection of different data types which are grouped together. Each element in a union is called member.

- Union and structure in C are same in concepts, except allocating memory for their members.
- Structure allocates storage space for all its members separately.
- Whereas, **Union allocates one common storage space for all its members**
- 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.
- Many union variables can be created in a program and memory will be allocated for each union variable separately.

Below table will help you how to form a C union, declare a union, initializing and accessing the members of the union.

Using normal variable	Using pointer variable

Syntax:
union tag_name
{
data type var_name1;
data type var_name2;
data type var_name3;
};
	Syntax:
	union tag_name
	{
	data type var_name1;
	data type var_name2;
	data type var_name3;
	};

Example:
union student
{
int mark;
char name[10];
float average;
};
	Example:
	union student
	{
	int mark;
	char name[10];
	float average;
	};

Declaring union using normal variable: union student report;	**Declaring union using pointer variable:** union student *report, rep;

Initializing union using normal variable: union student report = {100, “Mani”, 99.5};	**Initializing union using pointer variable:** union student rep = {100, “Mani”, 99.5}; report = &rep;
Accessing union members using normal variable: report.mark; report.name; report.average;	**Accessing union members using pointer variable:** report -> mark; report -> name; report -> average;

Example:

```
#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
```

Algorithms: Sum of array elements

Algorithm:
This program should give an insight of how to parse (read) array. We shall use a loop and sum up all values of the array.

Algorithm
Let's first see what should be the step-by-step procedure of this program −

```
START
  Step 1 → Take an array A and define its values
  Step 2 → Loop for each value of A
  Step 3 → Add each element to 'sum' variable
  Step 4 → After the loop finishes, display 'sum'
STOP
```

```
#include <stdio.h>

int main() {
  int array[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 0};
  int sum, loop;

  sum = 0;

  for(loop = 9; loop >= 0; loop--) {
    sum = sum + array[loop];
  }

  printf("Sum of array is %d.", sum);

  return 0;
}
```

The output should look like this −

```
Sum of array is 45.
```

Removal of duplicates from an array

SCSA1104-Problem Solving Techniques with C and C++

Program to remove duplicate elements in an array (sorted and unsorted array cases) is discussed here. Given an array, all the duplicate elements of the array are removed.

For example, consider the array

Input: arr = {1, 2, 3, 4, 4}

Output: arr = {1, 2, 3, 4}

Algorithm to remove duplicate elements in an array (sorted array)

- Input the number of elements of the array.
- Input the array elements.
- Repeat from i = 1 to n
- if (arr[i] != arr[i+1])
- temp[j++] = arr[i]
- temp[j++] = arr[n-1]
- Repeat from i = 1 to j
- arr[i] = temp[i]
- return j.

program

```
/* C program to remove duplicate elements in an array */
#include<stdio.h>
// Code without the usage of pointers
int remove_duplicate_elements(int arr[], int n)
{

if (n==0 || n==1)
return n;

int temp[n];

int j = 0;
int i;
for (i=0; i<n-1; i++)
if (arr[i] != arr[i+1])
temp[j++] = arr[i];
temp[j++] = arr[n-1];

for (i=0; i<j; i++)
arr[i] = temp[i];

return j;
}


int main()
{
int n;
scanf("%d",&n);
int arr[n];
int i;
for(i = 0; i < n; i++)
{
scanf("%d",&arr[i]);
}

n = remove_duplicate_elements(arr, n);
```

```
for (i=0; i<n; i++)
printf("%d ",arr[i]);

return 0;
}
```

Finding the K^{th} smallest element.

Algorithm

- Input the number of elements of the array.
- Input the array elements.
- Repeat from i = 0 to length-1
- Repeat from j=0 to length-1
- array[j]>array[j+1]
  temp=array[j];
  array[j]=array[j+1];
  array[j+1]=temp;

- Print the sorted array
- Enter the k^{th} smallest element
- Print the smallest element.

```
#include<stdio.h>
#include<conio.h>
int main()
{
  int array[100],length,i,j,temp,n;
   printf("Enter The length Of The Array ");
  scanf("%d",&length);
  printf("\nEnter The Numbers:");
  for(i=0;i<length;i++)
   {
    scanf("%d",&array[i]);
    }
  for(i=0;i<length;i++)
    {
     for(j=0;j<length-1;j++)
      {
       if(array[j]>array[j+1])
        {
         temp=array[j];
         array[j]=array[j+1];
         array[j+1]=temp;
         }
        }
      }
   printf("The new array is:");
   for(i=0;i<length;i++)
   {
    printf(" %d ",array[i]);
    }
   printf("\nEnter Which Smallest Number You want");
   scanf("%d",&n);
```

```
    printf("\nThe %d th smallest number is: %d",n,array[n-1]);
    getch();
    return 0;

}
```

SCSA1104-Problem Solving Techniques with C and C++