Chapter-2

Programming in 'C' Language

2.1 Array

When we have to make a list of 10 students then we have to declare 10 variables. If the number of students increases then variables will be increased and the program will be complex. In the 'C' language, the Array is used to solve this problem.

Array is a variable which can collect similar types of data. We have studied that a variable can store only one value at a time. But some time, it is required where we use similar and related data. It is typical to store all values in the different variables. So, a variable can store all values. In this group, every value indicate by its index value. The type of data can be char, int, float, double etc. If there are 50 students in the class then we have to use an array named marks which can store all marks and it is known as marks[50]. The number 50 is written in the square bracket([ ]),that it shows, it is a group of 50 elements. Every element recognized by their index value. For example marks[8] means the number is stored at position eight.

Example :

The ten elements of an array P[10] is as follows :

P[0], P[1], P[2], . . . P[8], P[9]

2.1.1 Array Declaration

An array must be declared before it used in the program as same as the other variables, three thing are declared in an array declaration :

(i) Type of an Array

(ii) Name of an array

(iii) Capacity of an array (Number of Elements)

When we declare an array then the memory will be allotted according to the size of the array. An array declares as follows :

< Type of array > < Name of array > < size >

Example :

int marks[50] ;

Here, int marks[50] tells us that the size of marks is 50 and it can store integer only.

2.1.2 Processing On Array

The processing on an array, is done on doing processing on each element. It is required a loop for processing on an array. When a loop executes it processed all element.

Example :

```
for ( i = 0; i<10; i++)
    scanf("%d",&A[i]) ;
```

In the above statements, it takes 10 inputs for an array. It will be clear from the following example :

Program 1 : Write a 'C' language program to arrange 10 numbers.

```
#include<stdio.h>
#include<conio.h>
/*It is C program to sort 10 numbers using selection sort. */
int main()
{
    int A[10];
    int i,j,temp;
    printf("\nEnter Ten Values :\n");
    for(i=0;i<10;i++)
        scanf("%d",&A[i]));
    /* Sorting Algoritham */
    for(i=0;i<9;i++)
        for(j=i+1;j<10;j++)
```

```
            if(A[i]) > A[j])
            {
                temp   = A[i];
                A[i] = A[j];
                A[j] = temp;
            }
    printf("\nThe Sorted Data are :\n");
    for(i = 0; i<10;i++)
        printf("%d  ", A[i]);
    getch();
    return 0;
}
```

Result:

Enter Ten Values :

5 19 45 63 7 4 78 12 89 56

The Sorted Data are :

4 5 7 12 19 45 56 63 78 89

2.1.3 Type of Array :

Array are two types :

1. One Dimensional Array
2. Multi Dimensional Array

1. One Dimensional Array

The one dimensional array has one row or one column. For example , there are 50 students in the class and uses one dimensional array to store their marks. If the array name is Marks then the values will be :

Marks [0] = 77

Marks [1] = 55

Marks [2] = 67

Marks [3] = 65

Marks [4] = 89

:

:

:

Marks [47] = 78

Marks [48] = 48

Marks [49] = 52

2. Multi Dimensional Array

It is required most of the time when the data can not be stored in the single dimensional array. For example, if there are 20 students in a class and store marks of five subject of each student then we require a table having 20 row and 5 column.The multi dimensional array is a kind of array which have two or more dimension. It is written as follows :

<Type of array><Name of array>[Max size 1] [Max size 2].....[Max size n]

If it is two dimensional array then two square bracket ([ ][ ])will be attached.

Example :

In the array A[5][5], its first element A[0][0], second element is A[0][1] and similarly last element is A[4][4]. It has 5 row and 5 column. It is shown in figure 3.1.

	0	1	2	3	4
0	9	6	16	24	61
1	16	72	95	97	55
2	49	57	40	45	25
3	29	64	5	18	2
4	10	12	98	59	26

Figure 3.1 : Array A[5][5]

2.1.4 Initialization of Multi Dimensional Array

The initialization of multi dimensional array is similar to one dimensional array.

Example :

int MAT[3][3]={0,1,2,3,4,5,6,7,8,};

In The above initialization, the row and column are not shown separately. The above array can also be initialized in row and column format.

int MAT[3][3]={ {0,1,2,} {3,4,5,} {6,7,8,}};

The values of first row will be enclosed in first bracket. The value of second and third row will be enclosed in second and third bracket respectively. If the total values in the bracket is less than the size of an array then remaining values will be automatically initialized by zero.

Example :

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

The capacity of the array is 9 and there are 6 values in the bracket. So, remaining three value will be zero.

The values of an array will be as following :

MAT[0][0]=1 MAT[0][1]=2 MAT[0][2]=3

MAT[1][0]=4 MAT[1][1]=5 MAT[1][2]=6

MAT[2][0]=0 MAT[2][0]=0 MAT[2][2]=0

Program 2 : Write a 'C' language program to multiply two 3 x 3 matrices.

```
#include<stdio.h>

#include<conio.h>

main()

{

    /*It is C program to Multiply two 3 x 3 Matrix*/

    /*Array C is initialized by Zero*/
```

```
    int i,j,k,A[3][3],B[3][3],C[3][3]={0};
    /* Read Matrix A & B*/
    printf("\nEnter Matrix A :\n");
    for(i=0; i<3; i++)
        for(j=0; j<3; j++)
            scanf("%d",&A[i][j]);
        printf("Enter Matrix B :\n");
    for(i=0; i<3; i++)
        for(j=0; j<3; j++)
            scanf("%d",&B[i][j]);
    /* Multiply Two Matrix A And B*/
    for(i=0; i<3; i++)
        for(j=0; j<3; j++)
            for(k=0; k<3; k++)
                C[i][j] += A[i][k] * B[k][j];
        printf("\nMultiply Matrix C :\n");
    for(i=0; i<3; i++)
    {
        for(j=0; j<3; j++)
            printf("%d  ",C[i][j]);
        printf("\n");
    }
    getch();
}
```

Result :

Enter Matrix A :

1 2 3

3 2 1

6 8 4

Enter Matrix A :

1 4 5

6 2 1

5 3 8

Multiply Matrix C :

28 17 31

20 19 25

74 52 70

Program 3 : Write a 'C' language program to arrange 10 numbers.

```
#include<stdio.h>
#include<conio.h>
/*It is C++ program to sort 10 numbers using Bubble sort.*/
int main()
{
    /*Array A declare with initialization.*/
    int a[10]={5,4,89,12,78,6,19,45,63,7};
    int i,j,temp;
    /* Sorting Algorithm*/
    for(i=0;i<9;i++)
```

```
        for(j=0;j<9-i;j++)
            if(a[j] > a[j+1])
            {
                    temp = a[j];
                    a[j] = a[j+1];
                    a[j+1] = temp;
            }
    printf("\nThe Sorted Data are :\n");
    for(i = 0; i<10;i++)
        printf("%d ",a[i]);
    getch();
}
```

Result:

The Sorted Data are :

4 5 6 7 12 19 45 63 78 89

2.2 String

String is a group of characters. A string is defined as an array of character in the 'C' language. The group of character enclosed within double quotation then it is called string constant.

Example :

```
" C is better than other language." /* It is string constant*/
```

2.2.1 Declaration and Initialization of String Variable

A string is defined as an Array of char. String is defined as follows :

```
char <string name> [size]
```

The size is show the number of character in the string.

Example :

```
char name[20];

char address[30];
```

In the above example, the array can store 20 and 30 characters. But it can store 19 and 29 characters because last character of the string should be a NULL character ('\0'). NULL char is the indication of end of string.

Here. it is remembered that if we are processed whole string at a time then the NULL character automatically attached at the end of string. If we are processed a string char by char then programmer must be remembered to store NULL char at the end of string. Because in this case the compiler donot store NULL char. A string can be initialized as follows :

```
name[20]="RAMSHANKAER";

name[20]={'R','A','M','S','H','A','N','K','E','R', '\0'};
```

In the above initialization, we have remembered that if a string is used to initialize as a string constant then there is no need to assign NULL char otherwise a NULL character must be stored at the end of string. When we initialize a string then there is no need to write size in the square bracket.

Example :

```
name[]="RAM SHANKER";
```

In the above example, we omit the size of array. The size of array will be fixed according to the size of string constant. In the above example, the size of array will be 12.

2.2.2 Input a String

There are many methods to read or input a string. A string can be read through scanf( ) function.

Example :

```
char name[20];

scanf("%s", name);
```

The ampersand (&) operator is not used with string variables because the name of string itself denotes the address of the string. In the above method, it can read a string upto the blank space. In the above scanf( ) function, gives an input "Ram Shanker" then it stores "Ram" in the string variable. If we want to solve the above problem then we have

to change the control string for the string.

Example :

```
scanf("%[^\n]", Name);
```

It read the string before the new line character ('\n'). If we give an input and press Enter key then "Ram Shanker" will be stored in the string. We can also read a string char by char.

Example :

```
int i = - 1;

char name[20], ch;

while(ch!='\n')

{

    ch=getchr();

    name[i]=ch;

}

name[i]='\0' /*store NULL Char at end.*/
```

The above code can be written as follows with the help of for loop.

```
for(i=0; (name[i] = getchar())!='\n'; i+ +);

name[i]='\0';
```

In the above method, we can input a string upto the new line character. A library function (gets( )) is also available for inputting a string. It is also read a string upto the new line character.

Example :

```
gets(name);
```

2.2.3 Print a string

A string can be printed with the help of function printf( ) and puts( ).

Example :

```
printf("%s", name);
```

O R

```
gets(name);
```

In the above both methods, the string will be print upto NULL character.

2.2.4 Library Function for string

'C' language includes library function for strings. Some of the important functions are shown in the table 3.1.

Table 3.1 : String Functions(string.h)

Function	Purpose
strcat( )	Concatenates two strings
strcmp( )	Compare two strings
strcpy( )	Copy one string over another string
strlen( )	Find the length of a string

strcat( ) Function :

The syntax of strcat( ) function is :

```
strcat(string1, string2);
```

The string1 and string2 are character array. When the function strcat() will execute then string2 will append at the end of string1. The string2 remain unchanged.

Example :

```
char st1[10]="Computer";
char st2[10]=" System";
        strcat (st1, st2);
```

After execution, the string will be :

```
st1="Computer System";
st2="System";
```

The string st2 append at the end of string st1. It is remembered that first argument must be a variable.

strcmp( ) Function

The syntax of strcmp( ) function is :

```
n = strcmp(string1, string2);
```

It compares two strings 'string1' and 'string2' and gives the following result :

string1 == string2 The value of n will be zero.

string1 < string2 The value of n will be negative.

string2 > string2 The value of n will be positive.

Example :

```
strcmp("this","that");
```

In the above example, the string "this" is bigger because its first two characters are same and third character 'i' is bigger than character 'a' (ASCII value of i is 105 and a is 97). A string called a bigger string if it ASCII value is higher then other string. And it return the difference of the ASCII values. In the above example, it returns value 8(105- 97 = 8).

Example :

```
strcmp ("Kapil","Anish");          return –7
strcmp ("Mohan","Mohan");          return zero
strcmp ("Anish","Anil");           return +7
```

The strcmpi() function is also used for comparing two strings which ignore the case (upper and lower) of the string means both strings are same. The strcmp( ) function make difference in both cases.

strcpy( ) Function

The syntax of strcpy( ) function is :

```
strcpy(string1, string2);
```

The string1 must be a variable and string2 will be variable or constant. The characters of the string2 will be copied into the string1.

Example :

```
strcpy (State, "Rajasthan");
```

It will copy the string constant "Rajasthan" into the variable state.

strlen( ) Function

The syntax of strcpy( ) function is :

```
n = strlen(String);
```

,

It stores the string length (total character) in the variable 'n'. String may be variable or constant.

Example :

```
n = strlen ("Rajasthan");
```

The value of n will be 10. It also count the NULL character.

2.2.5 Two Demensional String Array

If we want to store more than one string in an array then we have to use two dimensional array. It is declared as follows :

```
char name [10][20];
```

In the above array, it can store 10 string and their length may be 20.

This string can be read as follows :

```
for (i=  0; i < 10; i + + )
        scanf ("%s", name[i]);
/* or gets (name[i]);*/
```

And print as follows :

```
for (i=  0; i < 10; i + + )
        printf ("%s", name[i]);
/* puts (name[i]);*/
```

Program 4 : Write a program to find whether a given string is Palindrome or not.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
    int m,n,flag=1;
    char  str[80];
    clrscr();
    printf("\nEnter a String :");
    gets(str);
    m=0;
    n= strlen(str) - 1;
    while(m <= n)
    {
        if(str[m] != str[n])
        {
            flag = 0;
            break;
        }
        m++;
        n--;
    }
    if(flag==1)
```

```
        printf("\nString is Palindrome");
    else
        printf("\nString is not Palindrome");
    getch();
    return 0;
}
```

2.3 Function

It is typical to write a large (complex) program. So, a 'C' language program is divided into small parts. These small parts are completed a process. These all parts are combined in a sequence that it solves a complex problem.

All high level languages have this provision that the name is given to the block and written separately, when we require to call the block we can call it number of times. This independent block is called subprogram or function.

Different program blocks are made for every problem.These blocks are called function. Definition of the function can be written as follows :

" A function is a self contained program segment that is used for some specific well defined task ."

Function has important roll in the 'C' language. We have already used scanf( ), printf( ) functions through main( ) function. In the 'C' language, functions are divided into two parts.

2.3.1 main( ) function :

All 'C' program start their execution from the main( ) function. The main() function define as follows:

```
main()
{
    statements;
}
```

This function is perfectly valid in 'C' language because main( ) does not return any value but new compiler return int type value. If we donot want to return a value then

we have to use void key-word before the main( ) function.

Example :

```
void  main()
{
    statements;
}
```

Therefore, explicitly defines main( ) as matching one of the following prototypes.

```
int main()
void main()
int main (int argc, char * argv[])
void main (int argc, char * argv[] )
```

If we are not used int or void keyword then it will return integer value. Similarly, if the function does not specify return type value then it also return integer type value.

2.3.2 Advantages of Function

1. The function is divided into small parts so, the large program can be written easily.
2. A program can read and understand easily using function.
3. We can add a new function in the program, which can easily change the program.
4. The complex problem can easily be solved by the function.
5. The error can be found out easily by the function.
6. All statements, which are repeated in the program, are written in the form of function so, that it is not required to write repeatedly in main function. It reduces the size of the program.
7. The complex program can easily be understood by using functions.
8. In the 'C' language, function is a subprogram which is use for special purpose.

In the 'C' language, a program has one or more functions. In the every program, the main( ) function must write once and once only. The function will not return any value

to the user. There are many function which are already available in 'C' language. For example, scanf(), pow( ) etc. Every function returns a value at a time. A user define function is called using in the main( ) function. When a function calls in a main function the control flow leave the current instruction and transfer to the function where it is written. After execution of the function again it return to the instruction where control flow leave the instruction. We can call a function several times in the main( ) function.

2.3.3 Types of Functions

Functions are divided into two parts :

1. Library Functions

2. User Defined Functions

1. Library Functions :

The function have defined in the language, are called library function. They can use directly in the program. This facility is available in all high level language. It is very useful to collect general function in the library. This is helpful to make large program.The library function are shown in table 2.2. Library function - printf, scanf, sqrt, calloc, malloc, starcat & many more.

Table 2.2 Library Function

S. No.	Name of Function	Purpose of Function	Header file name
1.	getchar()	returns the next character typed on the keyboard.	stdio.h
2.	putchar()	outputs a single character to the screen.	stdio.h
3.	printf()	outputs a single character to the screen.	stdio.h
4.	scanf()	returns the next character typed on the keyboard.	stdio.h
5.	isdigit()	returns non-0 if arg is digit 0 to 9	ctype.h
6.	isalpha()	returns non-0 if arg is a letter of the alphabet	ctype.h
7.	isalnum()	returns non-0 if arg is a letter or digit	ctype.h
8.	islower()	returns non-0 if arg is lowercase letter	ctype.h
9.	isupper()	returns non-0 if arg is uppercase letter	ctype.h
10.	acos()	returns arc cosine of arg	math.h
11.	sqrt()	returns square root of num	math.h
12.	fabs()	returns absolute value of num	math.h

13.	asin()	returns arc sine of arg	math.h
14.	atan()	returns arc tangent of arg	math.h
15.	cos()	returns cosine of arg	math.h
16.	exp()	returns natural logarithim e	math.h

2. User Defined Functions

All type of library functions is not presented in the language which fulfil the requirement of the user. Many times it is required that we have to execute group of statement number of times and this type of function is not available in the library. Then we are required User Define Functions. The functions, which are written according to the requirement of the user, are called Use Defined Function (UDF). In a program, one or more user define function can be made. The User define function can be written before or after the main( ) function.

2.3.4 Declaration of Function and Definition a function

In the 'C' language, a user define function can be define as follows :

Return type Function Name (List of arguments)

{

Local Variabals

function body

Expressions

}

Function name is similar to an identifier. When a function calls, it executes according to the statements are written in the function and it returns a value. This value must have a data type. So, at the time of declaration, we define the return value type which tell us the type of value to be return by the return expression. if we are not defined the return type value then it return integer type value. A function can use more than one return statement and it is not necessary to write the return statement at the end of the function, it can be written anywhere in the function according to the requirement.

Program 5 : Write a 'C' program to find out smaller number out of two numbers using function.

#include<stdio.h>

#include<conio.h>

```
/* It is C program to illustrate Function*/
/* It is use to find biggest number between two numbers*/
main() /* Main function */
{
      int A,B,big; /*Local variable for main function*/
      printf("\n Enter two number :");
      scanf("%d  %d",&A,&B);
      big = fun_big(A,B); /*Function Call By Value*/
      printf("The biggest number : %d",big);
      getch();
}
/* Function return an integer value */
int fun_big(int C,int D)
{
      if( C > D )
            return(C);
      else
            return(D);
}
```

Result :

```
Enter two number : 45 85
The biggest number : 85
Enter two number : 105 85
The biggest number : 105
```

First, it executes main( ) function. When the control flow executes call statement (fun_big( )) in the main( ) function then it will execute the statements which are written in it. When the control transfers from main( ) function to user define function then the values written with the name of the function in the bracket also copied in the variables of the user define function.

For example, when it complete the execution of the function then the control return back to the main( ) function and executes the remaining statements in the main( ) function.

The function where we call the user define function are called "Calling Function" and the user define function which calls are called "Called Function".

2.3.5 Actual Parameters and Formal/Dummy Parameters

When we call user define function in the main( ) function then the values are written with the name of function in the bracket are called actual parameter. The actual parameter can be variables, constants, and addresses.

At the time of function definition the arguments are written in bracket, are called Formal/Dummy Parameter.

When a user define function call then the actual parameters will be copied in the formal parameters. It is remembered at the time of calling a function that the number of actual parameters must be equal to the formal parameters.

Example :

```
int a ;

main ( )

{

    int x, y;

    int z;

    printf("Enter the first number\n");

    scanf("%d",&x);

    printf("Enter the second number\n");

    scanf("%d",&y);

    z = mul (x,y); /*Actual parameters (x, y)*/
```

```
    printf("%d\n",z);
}
int mul(int a, int b)/* Formal parameter*/
{
    int c;
    c = a * b;
    return(c) ;
}
```

The values at that time are a = x and b = y.

2.3.6 Calling a Function

We call a function in the main( ) function by its name. It should be remembered that the function name would be written right side of the expression. There are two methods to call a function.

1. Call by Value
2. Call by Address

1. Call by Value

When we pass the variable or constants as arguments at the time of calling to the function then this type of calling is called 'call by value'.

Example :

```
main ( )
{
    int mark1,mark2,mark3;
    float d;
    printf("Enter the first marks\n");
    scanf("%d",&mark1);
```

```
    printf(Enter the second marks\n");
    scanf("%d",&mark2);
    printf("Enter the third marks\n");
    scanf("%d",&marks3);
    d=average(mark1, mark2, mark3); /*function call*/
    printf("Average of three marks is %f\n",d);
}
float average(int x, int y, int z)
{
    float f;
    f = (x + y + z)/3;
    return (f);
}
```

In this example, it is sending variables at the time of function calling.

2. Call by Reference

At the time of function calling, we pass the addresses of the variable in place of variables or constants then it is called 'Call by Reference'. Here, we have to remember that the formal argument must be a pointer type variable to store the address of the actual parameters. A simple variable cannot store the address of a variable.

Example :

```
main ()
{
    int a,b;
    float d;
    printf("Enter the first number\n");
    scanf("%d",&a);
```

```
    printf("Enter the second number\n");
    scanf("%d",&b);
    d=average(&a,&b);
    printf("Average of two number is %f\n",d);
}
float average(int*x, int *y)
{
    float f;
    f = (*x + *y)/2
    return (f)
}
```

Here, the address of a and b will pass to the function through avg(&a,&b).

Let 1000 and 1050 are addresses, where the address of x and y will be 1000 and 1050 respectively. If we write int x, int y in place of int *x, int *y then it will be wrong because simple variables x and y can not store the addresses.

2.3.7 Categories of Functions

Functions are divided into three different categories according to their way of passing arguments to the function.

1. Function with No Arguments and No Return Values
2. Function with Arguments but No Return Values
3. Function with Arguments and Return Value

1. Function with No Arguments and No Return Values :

In this types of functions, we do not pass any argument and the function does not return any value. In this type of category, the control go and come from main( ) function to user define function and user define function to main( ) function. They have no data with them. This type of category is used to transfer the control from one place to the other place.

Example :

```
main ( )
{
    printf("You are in Main program\n");
    average();
}
void average( )
{
    float f;
    printf("Enter the first marks\n");
    scanf("%d",&x);
    printf(Enter the second marks\n");
    scanf("%d",&y);
    printf("Enter the third marks\n");
    scanf("%d",&z);
        f = (x + y + z)/3;
        printf("Average of three marks is %f\n",d);
    }
```

2. Function with Argument But No Return Values

In this category, we send the arguments at the time of calling from the main( ) function but function does not return any value. We read the data in the main( ) function and send them to the user define function later.

In this category, we have to remember that when the arguments pass from the main( ) function then the data type of the actual arguments should be same to the formal arguments. All actual arguments will be copied into the formal parameters one by one.

Example :

```
main ( )
    {
        int mark1,mark2,mark3;
        float d;
        printf("Enter the first marks\n");
        scanf("%d",&mark1);
        printf(Enter the second marks\n");
        scanf("%d",&mark2);
        printf("Enter the third marks\n");
        scanf("%d",&marks3);
        average(a,b,c);
    }
    void average(int x, int y, int z)
    {
        float f;
        f = (x + y + z)/3;
        printf("Average of three marks is %f\n",f);
    }
```

Whenever, the function average( ) are called then control transfer to the user define function and the values of a and b will be copied into the variable x and y (x = a and y = b) respectively. When the user define function is completed then control will return to the main( ) function without return a value after completing its execution.

3. Function with Arguments and Return Value

In this category, the actual arguments are passed and stored in the formal arguments at the time of the calling of user define function. When the execution of the user define function completed then control return to the main( ) function with a return value. When all values of user define function will be used later in the main( ) function then this cat-

egory will be used. Here, the return statement is used to get a value from user define function. The return statement returns a value at a time. So, we can say that whenever we call a function then it returns a single value on each call. The return statement must write here and it return a float value automatically.

Example :

```
main ( )
{
    int mark1,mark2,mark3;
    float d;
    printf("Enter the first marks\n");
    scanf("%d",&mark1);
    printf("Enter the second marks\n");
    scanf("%d",&mark2);
    printf("Enter the third marks\n");
    scanf("%d",&marks3);
    d=average(marks1, marks2, marks3);
    printf("Average of three marks is %f\n",d);
}
float average(int x, int y, int z)
{
    float f;
    f = (x + y + z)/3;
    return (f);
}
```

2.4 Scope rules

We are using the variable in the function, where are these variables can be used

in the program? It is decided by the scope of rules. The variables used in the program are two types.

1. Local variable
2. Global variable

1. Local variable

These variables are declared in the function. The variables declared in the header of the function also called local variables. The boundary of the local variable is the boundary of function. So, the local variable works within the function only.

Example :

```
#include<stdio.h>

main( )

{

    int a=5, b=20; /*Local variables*/

    printf("%d", b);

    fun( );

}

fun( )

{

int b=10;\* Local variable*/

printf("%d", b);

/*printf("%d", a); uncommented cause and error*/

}
```

output:

20 10

Here, three variables are declared. Variable a and b are declared in main( ) function and variable b is declared in fun( ) function. Both b are different from each other.

The function can be used within the boundary of the function where it is declared. For example, variable a cannot be used within the function fun( ).

2. Global Variable

These variables are declared at fixed place of the program and they are visible in whole program. The global variables are declared after the header files.

Example :

```
#include<stdio.h>

int a, b, c:/* Global declaration of vaiable*/

main()

{

    a=10; b=20; c=30;

    fun1();

    printf("a=%d  b=%d  c=%d", a , b, c);

}

funl()

{

    printf("\na=%d  b=%d  c=%d", a, b, c);

    a + = 5;

    c + = 10;

    return;

}
```

output:

```
    a=10  b=20   c=30

    a=15  b=20   c=40
```

The value of a, b and c can be changed in any function in the program. Here, it is remembered that if a variable declares as a global and local variable with a same name

then the local variable will be used in the function and global variable will not use in the function. The global variable will be used where local variable is not declared with the same name.

Example :

```
#include<stdio.h>

int a=10, b=20;

main()

{

    a=20; c=30;

    printf("a=%d b=%d c=%d",a, b,c);

}
```

output:

a=20, b=20, c=30

Here, it prints the value of local variable 'a' because the precedence of local variable is higher than global variable.

2.5 Storage classes type

There is two ways to characterize variables.

1. Data Type
2. Storage class

Data type refer to the type of data stores in the variable and storage class refer to the scope and life time of the variable with in the program.

There are four different type of storage class :

(i) automatic

(ii) external

(iii) static and

(iv) register

They are declared by keyword auto, extern, static and register respectively.

Example :

```
auto int x, y, z;

extern float r1, r2;

register char ch1;

static int c, d;
```

The storage class keyword is specified a particular storage class must be placed at the beginning of the variable declaration. We have already read about local and global in the paragraph 3.4 of this chapter.

2.5.1 Automatic variables

Automatic variables are declared within the function and these are local variable to the function. Their scope is within the function only and its lifetime starts with the function calling and ends with the termination of the function. A variable can be declared in the different function with the same name. They are independent to each other. The automatic variables are required a keyword auto for declaration of the variables.

Example :

```
auto int A;
```

If the storage class is not written in the declaration statement then it makes the variable of automatic type (Default storage class is Automatic).

Example :

```
#include<stdio.h>

#include<conio.h>

main()

{

    /*Function main() scope start here*/

    auto int total1=10;
```

```
    auto int total2=30;
    printf("\ntotal1 = %d",total1);
    fun();
    printf("\ntotal1 = %d",total1);
}/*Function main() scope End here*/
fun()
{
    /*Function fun() scope start here*/
    auto int total1=30;
    printf("\ntotal1 = %d",total1);
    /*printf("total2 = %d",total2);
    uncommented to get an error*/
}/*Function fun() scope End here*/
```

Result :

```
total1 = 10
total1 = 30
total1 = 10
```

In the above program, the variable total1 is declared in both function main( ) and fun( ). Both total1 are independent to each other and the value of total1of main( ) function will not be changed before or after the calling of function fun( ) and the variable total2 can not be used within the function fun( ). It can be used within the main( ) function only.

2.5.2 External variable

External variables are not related to a single function as automatic variables. The external variable is declared in a function, all function can be accessed the variable which are written after the declared function. Their scope extends from point of declaration through the remainder of the program. They are using as the global variables. When we call a function, where it declares then its lifetime starts and end with the termination of the program. All function can be used the variable after declaration it. The decalaration of

variable uses extern keyword.

Example :

```
extern int A, B, C;

extern char D, E;
```

2.5.3 Static variables

The Static variables are also declared in the functions and therefore have the same scope as automatic variables. The Static variables are local to the function but they retain their value through out the life of the program. The first call of the function (where the static variable declares) declares the static variables and it initailize by zero if it is not initailized. It is lifetime start with the first execution of the function and died with the termination of the program. The static variables retain their last value and will use when it calls again. The declaration of variable uses static keyword.

Example :

```
#include<stdio.h>

main()

{

    int I;

    for (I = 1; I <= 10; I ++);

    fun();

}

void fun();

{

    static int K;

    printf("%d", K);

    K++;

}
```

The function fun( ) is called ten times in main() function by a loop. When we call

the function first time the initial value of K will be zero (i.e. K = 0). And it print the value of K is zero and value of K will be increased by one. When the function is called second time then the last value of K will print (in first call) 1 and the value of K will be increased by one. When the function is called third time then the last value of K will print (in second call) 2 and the value of K will be increased by one. And it will prints ten values of the variable K. The output will be as follows :

0 1 2 3 4 5 6 7 8 9

2.5.4 Register variable

In the above three types of variables are stored at the memory address. But the register storage class variables store the value in the CPU registers. Mostly, those variables are declared which are used extensively in the program. This is helpful to reduce the execution time of the program.

Example :

```
register int A, B, C;

register cahr x, y, z;
```

These variables are local to the function and work as the automatic variables. Moreover declaring certain variables to be register variables does not guarantee that they will actually be treated as register variables. The declaration will be valid only if the requested register space is available if it is not so, then the variable will be treated as automatic variable. The summary of storage classes as shown in Table 2.3.

Table 2.3

Class Type	Storage Space	Default initial Value	Scope	Life Time
Automatic	Memory	Garbage Value	Local to the function in which the Variable is defined.	Till the control remains within the function in which the variable is defined.
External	Memory	Zero	At the decla-	After declaration

			ration point to entire program.	to termination of the program.
Static	Memory	Zero	Local to the function in which the Variable is defined.	Value of the variable persist between different function call.
Register	CPU register	Garbage Value	Local to the function in which the Variable is defined.	Till the control remains within the function in which the variable is defined.

2.6 Arrays and Functions

If we want to pass an array from main( ) function to user define function then we have to take name of the array as an argument. And also take another argument as refer to the size of the array.

Example :

```
average (a,n);
```

Here, average is the name of user define function. a and n are name and size of an array. The function will de declared as follows :

```
float average (float arr[ ], int n1);
```

When, average(a,n) statement executes in the main( ) function then the control will transfer from main( ) function to user define function with its arguments. This will be float arr [] = a and n1 = n.

The bracket ([ ]) with the name of an array arr show that arr is an array and n1 is the size of the array. Here, it is not required to write the size of an array in the bracket because it would decide by the actual parameters.

Example :

```
float average (float arr [],int n1);

main()

{

    float a[40],r ;

    int n;

    printf("Enter the size of the array\n");

    scanf("%d",&n);

    printf("Enter the element of the array\n");

    for(i=0;i<n;i++)

    {

        scanf("%d",&a[i]);

    }

    r=average(a,n);

    printf("The average of the element is%f",r);

}

float average(float arr[], int n1);

{

    int i, sum = 1;

    float ave;

    for(i=0;i<n1;i++)

    {
```

```
        sum = sum + arr[i];
    }
    ave = sum/n1;
    return(ave);
}
```

In the above example, we suppose that the maximum size of the array is 40. But we store 10 (n = 10) values in the array. When average(a,n) executes in main( ) function and its mean average(a,10) then control will transfer with both values to executes the function. Whenever, control goes to the function then it stores all values of array a[ ] into the array arr[ ] and value of n1 equal to n. Then the function will execute according to the values and result will return to the main( ) function

2.7 Recursion

When a function calls itself until some specified condition has been satisfied, this processing called recursion. To write a function recursively then we have to remember two things :

1. The function must be call itself.

2. The function must have a condition which stop the recursion on satisfaction (Termination condition).

Generally, functions are called in the another function. The main( ) function is exception, it cannot be called anywhere in the program. The facilities available in many programming languages that a function calls itself, this is called recursive function and process is called recursion. A function calls a other function than the program control return back to the calling function. But function calls it self then it will come to the same function. It is dangerous that it may not be terminated from the function. In these conditions, program will run and if there is no resources available for computer it terminates abnormally and control will out from the program. It should be remembered that any function must have a termination condition. Otherwise it will cause an error message.

Program 6 : Write a function to findout the factorial of a given number using recursion function.

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

```
/* It is C program to calculate the factorial of N*/
main() /* Main function */
{
      int N,f1; /*Local variable for main function*/
      int fact(int N);/* Function Prototype*/
      printf("\n Enter a number :");
      scanf("%d",&N);
      f1 = fact(N); /* Function Call By Value*/
      printf("The Factorial of %d is = %d",N,f1);
      getch();
}
/* Function return an integer value */
int fact(int K)  /* Function Defination */
{
      int i,f2=1; /*Local variable for fact function*/
      for(i=1;i<=K;i++)
            f2 = f2 * i;
      return(f2);
}
```

Result :

```
Enter a number : 5
The Factorial of 5 is = 120
Enter a number : 7
The Factorial of 7 is = 5040
```

Let number = 5

In the first term it return(5 * fact(4))

In the second term it return (5 * 4 * fact(3))

In the third term it return (5 * 4 * 3 * fact(2))

In the fourth term it return (5 * 4 * 3 * 2 * fact(1))

In the fifth and last term it return (5 * 4 * 3 * 2 * 1)

Program 7 : Similarly, we also understand the recursion using another recursive function.

Let findout the value of x^y using successive multiplication.

```
int power(int a, int b);
main()
{
      int x,y,z;
      printf("Enter the Number\n");
      scanf("%d\n",&x);
      printf("Enter the power\n");
      scanf("%d\n",&y);
      z = power(x,y);
      printf("The result is\n");
      printf("%d\n",z);
}
int power(int a, int b)
{
      if (b==1)
            return a;
```

```
    else
        return(a*power(a,b-1));
}
```

Program 8 : Write a program to findout the n^{th} fibbonacci number. #include<stdio.h>

```
#include<conio.h>
/* It is C program to calculate Nth Fibonacci Number*/
/* Fibonacci Number Start with 1,1,2,3,5,8,13,21,34,55*/
main()
{
    int N,f1;
    int fib(int N);/* Function Prototype*/
    printf("\nEnter a number :");
    scanf("%d",&N);
    f1 = fib(N); /* Function Call By Value*/
    printf("The Nth Fibonacci Number is = %d",f1);
    getch();
}
/* Function return an integer value */
int fib(int K)  /* Function Definition */
{
    int i,f1=1,f2=1,f3; /*Local variable for fact function*/
    for(i=1;i<=K-2;i++)
    {
        f3 = f1 + f2;
```

```
            f1 = f2;
            f2 = f3;
        }
        return(f3);
}
```

Result :

Enter a number : 5

The Nth Fibonacci Number is = 5

Enter a number : 10

The Nth Fibonacci Number is = 55

Program 9 : Write a function to solve the quardatic equation $ax^2+bx+c=0$.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
main()
{
      void Q_EQ(int p,int q,int r);
      int a,b,c;
      printf("Enter the value of a,b and c :");
      scanf("%d %d %d",&a,&b,&c);
      Q_EQ(a,b,c);
      getch();
}
void Q_EQ(int p,int q,int r)
```

```
{
    float r1,r2,D;
    D = q * q - 4.0 * p * r;
    if(D < 0)
        printf("Roots are imaginary");
    else
    {
        if(D == 0)
        {
            printf("Roots are Equals\n");
            r1 = r2 = - q / (2.0 * p);
        }
        else
        {
            r1 = (- q - sqrt(D)) / (2.0 * p);
            r2 = (- q + sqrt(D)) / (2.0 * p);
        }
        printf("\nRoots are : %f %f", r1,r2);
    }
}
```

Program 10 : Write a function 'prime' which return 1 if the argument is prime otherwise it return 0.

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

```
main()
{
    int prime(int p);
    int flag,N;
    printf("\nEnter the value of N :");
    scanf("%d",&N);
    flag = prime(N);
    if(flag == 1)
        printf("\nNumber is prime");
    else
        printf("\nNumber is not prime");
    getch();
}
int prime(int p)
{
    int i,r1;
    r1 = (int)sqrt(p);
    for(i=2; i<=r1; i++)
        if(p % i == 0)
            return(0);
    return(1);
}
```

2.8 Pointer

Pointer is a powerful tool. Every variable stores in the primary memory that is called the address of the variable. Every variable has a memory address.

A pointer is a variable that stores the address of the data item (Address of the data in memory). The pointers are used to make the good and complex program in the 'C' language.

Every variable allocates the memory according to the type of the data. The char type data stores in 1 Byte, int type data store in 2 Bytes and float type data store in 4 Bytes. There is one memory address for a group of Bytes and that is stored in the pointer variable associate with the group. For example : int type data stores in 2 Bytes and there addresses are FAB and FAC respectively the address of the variable will be FAB. This is the address of first memory cell.

Let d is a variable, which represent integer data type. The compiler allocates the memory according to the type of data where it stores its value. It can easily accessible. Let we store value 10 in the variable d.

K is a pointer type variable, it can store address only. So that , K stores the address of the memory 4500 (Address of variable d).

There are two operators perform the above work in the 'C' language. The * operator is used to declare a pointer variable and it is also used to retrieve the value from the address stored in the pointer variable. This is also called value operator and the & operator is used to retrieve the memory address of any variable. For example we want to know the address of the variable d then we have to write &d. So, we write the following statement to store the address of d in pointer variable.

K = &d;

According to the above description *K and d shows the same value.

Program 1 : Write a program to add five integer number using pointers.

```
#indclude<stdio.h>

#include<conio.h>

#include<alloc.h>

main( )

{

    int I, *A,sum=0, B;

    A = &B;
```

```
    for(I=1; I <+ = 5, I++)
    {
        printf("Enter a number:");
        scanf("%d", A);
/*don't use ampersand sign with pointer variable.*/
        sum=sum + *A;
    }
    printf("\nSum of Five Numbers = %d', sum);
    getch( );
}
```

2.8.1 Pointer Expression

The pointer expression is the expressions, which uses pointer variables. If P1 and P2 are two pointer variables. They are declared and initialized.

1. Assignment Expressions :

These expressions are used to assigned values, are called assignment expressions.

Example :

```
P2++;
--P1;
P1 = P1 + 1;
*P2 = *p2 + 1
```

First three expressions are used to increase or decrease the memory address stored in pointer variables. And fourth expression increases the value stored at memory location stored in variable P2.

2. Arithmetic Expressions :

```
P2 = P2 + 1;
```

P2 = P1 + 1;

In the pointer variable, a memory location can be increase or decrease. The following arithmetic operations are not valid on pointers :

P1 / P2 Two pointer can be subtract if they are indicate same type

P1 / 50 of elements. Generally, two pointers can subtract when

P1 + P2 only they are indicated the elements of an array.

P1 * P2

P1 * 17

3. Relational Expression

The relational operators are used on pointer variables.

Example :

P2 > P1

P1 == P2

P2 >= P1

P1 != P2

2.8.2 Advantage of pointers

1. The pointers efficiently control an array.
2. The pointer can pass the information to the function, it automatically change the value of variables which are passed by the calling statement.
3. They speedup the execution of the program.
4. Pointer also provides an alternative way to access individual array elements.
5. The variable outside the function can be accessed with the help of pointers.

2.9 Dynamic Memory Allocation

Here, the memory can be assigned two types :

(1) Static memory assignment

(2) Dynamic memory assignment

1. Static memory is assigned before the execution of the program. It is used in array and it cannot be changed at the time of execution.

2. Dynamic memory can be assigned the memory after the execution of the program. It is assigned according to requirement of the programmer. The function available in 'C' are used to assign dynamic memory are shown in the table 4.1

Table 2.4

Function	Return value	Purpose
malloc (size)	Pointer void	Allocate required size of block.
calloc (size)	Pointer void	Allocate required size of block which divided into equal size of parts and each part is initialized by zero.
free(pointer)	No value	Free previous allocate memory.
realloc (size)	Pointer void	Modify the size of memory.

molloc( ) Function

The block of memory may be allocated using the function malloc(). The malloc() function is allocated a block of specified size and its type will be void pointer, we can change the void type memory in another data type.

The malloc function can be return as follows :

Str = malloc (size in byte)

If we change the memory type which is allotted by malloc function (if str stores the address of int type data) then malloc will be written as follows :

Str=(int*)malloc(size in byte)

Example :

```
int *integer;

integer=(int *) malloc (sizeof(int) * 10);
```

The above statement allocates 20 Bytes memory block. A integer number stores in 2 Bytes and return a integer type pointer.

Example :

```
char *cptr;

cptr=(char *) malloc(20);
```

It is also allocate 20 Bytes and return char type pointer.

calloc( ) Fuction

The memory can be allocated by the calloc() function. It divided the memory in the same size of blocks. It is written as follows :

```
Str = (type cast*) calloc(n, sizeof(element));
```

The above statement allocate configures space for same size of n blocks with same size and each block store zero value.

Example :

```
int *cptr,

cptr = (int *) calloc(10, size of (int));
```

It allocate 10 blocks of 2 Bytes and each block stores value zero.

Releasing Memory

When we does not use the memory block or no longer need the memory in the program then we may release that block of memory for future use to allocate memory of any other data. It uses free( ) function for the above purpose. It is written as follows :

```
free(Str);
```

Str is a pointer which had assigned the memory by the function calloc( ) or malloc( ).

Altering the size of memory block

The realloc( ) function can increase or decrease the memory block.

Example :

Allocate memory by function malloc( )

```
Str = malloc(size)
```

Then its size can be increased or decreased by the function realloc( ).

Str=realloc(ptr, new_size)

The memory will be allocated of new size.

Note : if there is not enough memory to allocate then malloc( ) or calloc( ) function return the NULL value.

2.10 Pointer and Array

We have read that the space for array is reserved at the time of declaration of the array. In this space we can store the elements of the array. The base address of the array is the address of the first element of the array.

Let an array declares as follows :

int A[5] = {7, 9, 15, 12, 14};

It is also assumed that the base address of the array is 2000. Because integer data, stores in 2 Bytes. So, the five elements of the array will store as follows :

We have known that the name of an array also shows the base address of the array. So, A and &A[0] shows the base address of an array and elements of an array stores in the continuous memory location.

Similarly, pointer variable also stores the base address of the memory block. So, an array and a pointer variable can be interchanged.

Example :

```
int*ptr;

int A[5] = {7, 9, 15, 12, 14};

ptr = A    /* ptr = &A[0]*/
```

In the above example, ptr and A both are worked as an array. Both can be used as pointer notation or traditional way.

For example the value of ptr[1] is 9 and the value of *(A + 1) will also be 9. It can be understood by the following example.

Program 12

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

```
#include<conio.h>
main()
{
    int *ptr,I;
    int A[5] = {7, 9, 1, 2, 4};
    ptr = A;
    clrscr();
    for(I=0;I<+5;I++)
    {
        printf("\nI = %d ptr[I] = %d",I,ptr[I]);
        printf("   *(A+I) = %d &ptr[I] = %x",*(A+I),&ptr[I]);
        printf("   (A+I) = %x",ptr[I]);
    }
    getch();
}
```

The output will be following :

```
I = 0    ptr[I] = 7    *(A+I) = 7    &ptr[I] = ffca    (A+I) = 7
I = 1    ptr[I] = 9    *(A+I) = 9    &ptr[I] = ffcc    (A+I) = 9
I = 2    ptr[I] = 1    *(A+I) = 1    &ptr[I] = ffce    (A+I) = 1
I = 3    ptr[I] = 2    *(A+I) = 2    &ptr[I] = ffd0    (A+I) = 2
I = 4    ptr[I] = 4    *(A+I) = 4    &ptr[I] = ffd2    (A+I) = 4
```

In the above program, the pointer variable is written in form of a traditional array and an array variable is written in form of pointer type. Which is shown that a one dimensional array can be interchange with pointer variable. Here, it is necessary to remember that if we add 1 in the pointer variable then it increases 2 Bytes. It is depended upon the data stored in the variable. If it is long int then we add 1 in the pointer variable

then it increases four Bytes.

Program 13 : Write a 'C' program to arrange 10 integer numbers using selection sort.

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
#include<string.h>
/*It is C program to sort 10 Numbers. */
int main()
{
    int *A;
    /* Array A declared as pointer type.*/
    int i,j,temp;
    A = (int *)malloc(20);
    printf("\nEnter Ten Numbers :\n");
    for(i=0;i<+10;i++)
        scanf("%d",(A+i));
    /* Sorting Algoritham */
    for(i=0;i<+9;i++)
        for(j=i+1;j<+10;j++)
            if(*(A+i) > *(A+j))
            {
                temp = *(A+i);
                *(A+i) = *(A+j);
                *(A+j) = temp;
```

```
                }
        printf("\nThe Sorted Numers are :\n");
        for(i = 0; i<+10;i++)
              printf("%d\n", *(A+i));
        getch();
        return 0;
}
```

Result:

Enter Ten Numbers :

20 10 40 60 50 30 80 70 90 75

The Sorted Numbers are :

10 20 30 40 50 60 70 75 80 90

Program 14 : Write a program to add two strings using pointers.

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
#include<string.h>
main()
{
      char *s1,*s2,*s3;
      int n1,i,j;
      /*Memory Allocation */
      printf("\nEnter the length of first string  : ");
      scanf("%d",&n1);
```

```
    s1 = (char *)malloc(n1+1);
    printf("\nEnter the length of second string : ");
    scanf("%d",&n1);
    s2 = (char *)malloc(n1+1);
    s3 = (char *)malloc(strlen(s1) + strlen(s2) -1);
    printf("\nEnter Two strings :\n");
    scanf("%s %s",s1,s2);
    /* Algorithm */
    strcpy(s3,s1);
    i=strlen(s3);
    j=0;
    while(s2[j] != '\0')
    s3[i++] = s2[j++];
    s3[i] = '\0';
    printf("Strings are : \n %s %s %s",s1,s2,s3);
    getch();
}
```

2.11 Structures

We have ready read about an array which have similar types of elements. But the different type of data can be written in the structure. So, a structure can use different types of data int, float, double, char and array according to a user. Every element in the structure are called member. If we have to store Name, Basic, Address and HRA of 40 employees then we have read that array is the best method. But Name, Basic, Address and HRA are stored in different array. It can make easy to use structures. It declares a structure named Employe and their members will be Name, Basic, address and HRA.We can store the information of 40 employees to make an array of the structure.

2.11.1 Declaration of Structure :

The structure declaration is more difficult than an array declaration. Because every element have to be declared in the structure. The general term, the composition of structure may be define as follows:

```
struct tag
{
        <data type> member1;
        <data type> member2, member3;
        ……….
.....
        ……….
        <data type> membern;
}
```

In the above declaration, struct is a keyword, tag is a name that identifies structures of this type. And member1, member2, member3 .....member n are an individual member declaration. The individual members can be ordinary variables, pointers, array or other structures. The same type of members can be specified in a single statement. It is declared in the above example.

```
<datatype> member2, member3;
```

Example :

Define an employee structure and the members are as following:

name, basic, address, HRA

```
struct employee
        {
                char    name[20];
                int     basic;
                char    address[25];
                int     HRA;
```

```
};
```

The above structure employee is called the user define structure. Now we can declare multiple copies of employee structure.

Example :

```
struct employee emp1, emp2;
```

The emp1and emp2 are similar to the structure employee. The above two statements can be combined in a single statement.

Example :

```
struct tag
{
        <data type> member 1
        <data type> member2
        . . . . . .
        . . . . . .
        data type member n;
}vaiable1, variable2;
```

In this case, it is not necessary to write tag in the declaration. The above statement is written as follows.

```
struct employee
{
        char    name[20];
        int             basic;
        char    address[25];
        int             HRA;
}emp1, emp2;
```

In the above example, we are used simple variables and array variables.

Example :

```
struct employee
{
        char    name[20];
        int             basic;
        char    address[25];
        int             HRA;
}emp1, emp2;
```

We can declare a structure member with in a structure.

Example :

```
struct  date
{
        int     day;
        int     month;
        int     year;
};
struct employee
        {
        char    name[20];
        int             basic;
        char    address[25];
        int             HRA;
        struct  date DOB;
```

```
}emp1;
```

In the above example DOB member is a date type structure. This is written with in the employee structure.

We can initialize a structure when it declares.

Example :

```
struct employee emp1 = {"Sangeeta Gupta", 8000, "153, Arya Nagar", 1200, 25, 01, 1976};
```

First value will be stored in the first member of structure, second value will be stored in the second member of the structure and so on.

2.11.2 Memory Map

The different memory block will be allocated for different members continuously in the memory. The size of the structure is equal to the sum of the each member size. The memory map of the employee structure is as follows :

The memory will use in the structure :

20 + 2 +25 + 2 + 2 + 2 + 2 = 55 bytes

The size of structrue can be calculated by a sizeof operator.

Example :

```
sizeof(employee);
```

output : 55

2.11.3 Processing a Structure

The member of structure is process individually. The structure member can be accessed by writting following statement :

<Name of structure> l <Member of structure>

It uses the . (dot) operator. Please see the following example :

```
struct employee
{
        char name[20];
```

```
        int     basic;
        char address[25];
        int HRA;
        struct date DOB;
    }emp1;
```

The member of emp1 will be accessed as follows.

```
emp1.basic = 8000
emp1.HRA = 1200
strcpy(emp1.name, "Yashika");
```

Similarly, we can read and write all members of the structure. For reading of the member of the structure :

```
scanf("%d", &emp1.basic);
scanf("%d", &emp1.HRA);
scanf("%s", emp1.name);
```

For writing of the member of the structure :

```
printf("%d", emp1.basic);
printf("%d", emp1.HRA);
printf("%s", emp1.name);
```

If a structure is declared with in a structure then the . (dot) operator is used to access the sub member of the structure. It is written as follows :

<Name of struct> l <Member of struct> l <Sub-member of struct>

In the above example, DOB member declares as date type structure which is declared as follows.

```
struct date
{
```

```
        int     day;
        int     month;
        int     year;
    };
```

The member of DOB access as follows :

```
    emp1.DOB.day= 25
    emp1.DOB.Month = 01
    emp1.DOB.Year = 1976,
```

If we declare two variables of a structure then we can store the value of a variable into the another variable using a assignment operator.

Example :

```
    struct student emp1={"Sunil Methi", 12000, "Alwar", 1200, 24, 11, 1968},
emp2;
```

If we use the assignment operator then the statement will be written as follows :

```
    emp2 = emp1;
```

The value of member of emp1 will be copied into the member of emp2. There is no need to copy from member to member for the same structure.

2.12 Structure and Array

We can declare the array as the member of the structure. Which is done in the above examples.

We can also declare structure of array. If we want to store the data of 100 employee then we declare a structure of array. It is declared as follows :

```
struct employee
    {
        char        name[20];
        int         basic;
```

```
        char        address[25];
        int         HRA;
        struct  date DOB;
    };
struct employee emp1[100];
```

In the above example, emp1 is declared as an array of structure. It is accessed as follows :

Name of structure[subscript Value]lMember of structure

Example :

```
emp1[0].basic = 12000;
emp1[1].basic = 8000;
```

2.13 Structure and Function

We can pass the member of structure as the simple variable.

Example :

```
Calculate(emp1.basic,emp1.HRA);
```

Basic and HRA are the member of emp1. We can pass the whole structure at a time. For example emp1 is declared as employee structrue then

```
Calculate(emp1);  /* calling */
```

Example :

```
void Calculate (struct employee Temp)
{
    Statements;
}
```

All member of emp1 will be copied into members of temp.

Program 15 : Declare a employee structure and it members are as follows

name, address, basic pay, HRA and DA and store their values and calculate Gross Pay.

```
#include<stdio.h>
#include<conio.h>
main()
{
        struct employee
        {
                char name[20];
                char address[25];
                int basic_pay;
                float da,hra;
                float gross_pay;
        };
        struct employee e1;
        /* Read structure*/
        printf("\nEnter Employee Data :");
        printf("\nName      : ");
        gets(e1.name);
        printf("Address   : ");
        gets(e1.address);
        printf("Basic Pay : ");
        scanf("%d",&e1.basic_pay);
        printf("DA(%)     : ");
        scanf("%f",&e1.da);
```

```
        printf("HRA(%)    : ");
        scanf("%f",&e1.hra);
        /* Gross Pay Calculation*/
        e1.gross_pay = e1.basic_pay + e1.basic_pay * e1.da/100 +
                                        e1.basic_pay * e1.hra/100;
        /* Printing */
        printf("\nEmployee Data :");
        printf("\nName      : %s",e1.name);
        printf("\nAddress   : %s",e1.address);
        printf("\nBasic Pay : %d",e1.basic_pay);
        printf("\nDA        : %5.2f%",e1.da);
        printf("\nHRA       : %5.2f%",e1.hra);
        printf("\nGross Pay : %8.2f",e1.gross_pay);
        getch();
}
```

Results:

```
        Enter Employee Data :
        Name      : Sunil Methi
        Address   : 153, Arya Nagar, Alwar.
        Basic Pay : 9375
        DA(%)     : 43
        HRA(%)    : 15
        Employee Data :
```

Name : Sunil Methi

Address : 153, Arya Nagar, Alwar.

Basic Pay : 9375

DA : 43.00%

HRA : 15.00%

Gross Pay : 14812.50

Program 16 : Write a program to store student name, roll_no, and marks obtained in four subjects of a class in a structure and print name, roll_no, total marks obtained by every students.

```
#include<stdio.h>

#include<conio.h>

main()

{

        int i,j,tot;

        struct

        {

                char    name[20];

                int     roll_no;

                int     sub1,sub2,sub3,sub4;

        }student[5]; /* In this case tag not required*/

        /* Read structure*/

        for(i=0; i<+=4; i++)

        {

                printf("\nEnter Student %d Data :",i+1);

                printf("\nName        : ");
```

```
                scanf("  %[^\n]",student[i].name);
                printf("Roll Number    : ");
                scanf("%d",&student[i].roll_no);
                printf("Marks Subject 1 : ");
                scanf("%d",&student[i].sub1);
                printf("Marks Subject 2 : ");
                scanf("%d",&student[i].sub2);
                printf("Marks Subject 3 : ");
                scanf("%d",&student[i].sub3);
                printf("Marks Subject 4 : ");
                scanf("%d",&student[i].sub4);
        }
        /* Printing */
        for(i=0; i<+=4; i++)
        {
                printf("\nName            : %s",student[i].name);
                printf("\nRoll Number    : %d",student[i].roll_no);
                tot=student[i].sub1+student[i].sub2
   +student[i].sub3+student[i].sub4;
                printf("\nTotal Marks    : %d",tot);
        }
        getch();
}
```

Important Points

1. Arrays are two types. One dimensional array and two dimensional array.
2. The part of the program, which is group of statements, and known by the separate name, called function.
3. The main ( ) function is also a user define function and it is must be presented in the program.
4. The arguments define at the time of function declaration are called formal /dummy parameter.
5. When a function calls, the arguments written with the calling statement is called actual parameters.
6. The function can call two types :

   (i) call by value

   (ii) call by reference
7. When a function calls itself, this process is called recursion.
8. The function must have a condition, which stop the recursion on satisfaction (Termination condition).
9. Pointer stores the address of a simple variable.
10. Dynamic memory can be assigned by the pointer variables.
11. The calloc, malloc are used to allocate the memory.
12. The free function is used to release the memory.
13. We can collect the different types of data using the structure.
14. The member of the structure are accessed by the . (dot) operator.
15. We can make a link list using structure.

EXERCISE

Objective Type Questions :

1. The group of similar types data are called

   (a) Array (b) Function

   (c) String (d) None of these

2. How many elements are in the array float arr[3][2] :

   (a) 2 (b) 3

   (c) 6 (d) 9

3. Which is false statement :

   (a) Global variable can be used anywhere in the program.

   (b) Auto variables are declared in the main( ) function.

   (c) Local variables are not work within the function or block.

   (d) Local variable can be declared in the different function with the same name.

4. Which is the correct statement to call a function int add(int x) :

   (a) add(); (b) add(x);

   (c) add(int x); (d) int add (int x);

5. How many types a function can call :

   (a) 2 (b) 1

   (c) 3 (d) 4

6. How many types of function are :

   (a) 1 (b) 2

   (c) 4 (d) 3

7. The value stores in the pointer variable

   (a) Integer Value (b) Any Value

(c) Address of the another variable (e) None of these

8. int B = 10;

int A =&B;

printf("%d", B) will print.

(a) 10 (b) Address of variable A

(c) Address of variable B (d) Print according to program

9. int*A;

A= (int *)malloc(sizeof(int)*10);

printf("%d",A); will print.

(a) Address of variable A (b) Address of first element of an Array

(c) First value of an array (d) Not print any value

10. The member of a structure can be

(a) Pointer variable (b) Integer Variable

(c) Floating variable (d) All of these

11. Which symbol is used to access the member of structures.

(a) . (dot) (b) *

(c) ® (d) &

12. a=1011

b=1111 and

x = a & b then value of x -

(a) 10 (b) 11 (c) 12 (d) 13

13. If nay Union have int float and double data type then how much momory for this Union.

(a) 2 bytes (b) 4 bytes (c) 10 bytes (d) 8 bytes

Very Short Answer Type Questions :

1. Where is a return statement written?
2. How many types are arguments?
3. What are the local variables?
4. Which header file includes the '\0' character?
5. How can you declare a one dimensional array?
6. How is declared a pointer variables?
7. int A;

   How is print the address of a variable A?
8. How is release the memory by the function free()?
9. How is declared a structure the member?
10. How is written the statement to initialize the member of the structure?
11. which file contains the malloc() function?
12. How we decleare member of Function?
13. How we initialize the member of Function?
14. How we allocate bits for a variable?
15. Which keyword is use to create new data type?

Short Answer Type Questions :

1. Which is the local variable?
2. Write a example to declare a two dimensional array.
3. How is a function declares?
4. How many types are functions in the 'C' language?
5. How is a array declared?

6. Write two condition of the recursion.

7. What is pointer variable?

8. int *A={10, 20, 30};

   If the base address of A is 2000 the write all addresses.

9. How are an array declared by a pointer type variable.

10. What is structure? Explain.

11. Write student structure whose member are name, address and roll_No.

12. How we declare bit field to store 0 or 1 for male or female?

13. Write an example of .....data.

Eassy Type Questions :

1. Write advantages of prointers.

2. How with declare Arry by pointer? explain with example.

3. Store 10 name, address and phone number using structure. agange these names according to Alphabat. Write a programe to print the this order.

4. write a programe to read one string and count the frequency of Alphbet.

5. How with declare pointers to the function. explain with example.

6. Write structure for following members showing bit field.

   i. be male or female

   ii. have one of the eight different hobbies.

   iii. be single, married, divorcded or widowed.

7. What is drived data type? how can be written drived data type of 12 months?

Answer Key

1. a 2. c 3. b 4. c 5. a 6. b 7. b 8. a 9. b

10. d 11. a 12. b 13.d