a verb in nature.

3. Parameter list : when the function is called for usage, there might be some data that need to be carried over to perform the task. So a parameter list is provided to hold data passed in local variables of function. The parameter list consist of formal parameters.

Function body : It holds the statements to perform the task for which the function is composed.

Function Prototype :

A function prototype tells the compiler the name of the function, the type of data returned by the function, the number of parameters the function expects to receive, the types of the parameters and the order in which these parameters are expected.

Like any variable in a C program it is necessary to write the prototype or declare a function before its use. It informs the compiler that the function would be referenced at a stage in the program. ***The general form of function Prototype is :***

Return_type function – name (argument list);

Example : void function – name ( ); is a function prototype or declaration. Here void specifies that this function does not return any value, and the empty parentheses indicate that it takes no parameters (arguments).

When we place the function prototype above all the functions (including main( )), it is known as a global prototype. A prototype declared in global environment is available for all the functions in the program.

When we place the function prototype inside the definition of another function (i.e., in the local declaration section), the prototype is known as a local prototype. Such declarations are primarily used by the functions containing them.

It is a good programming style to have global prototypes for adding flexibility and enhancing documentation. Function prototypes are not mandatory (compulsory) in C. These are desirable, however, because these help in error checking between the calls to a function and the corresponding function definition.

Example : int sum(int a, int b, int c) ;

Parameter Passing :

Parameter passing methods are the ways in which parameters are transfered between functions when one function calls another. Basically, C provides two parameter passing methods- pass-by-value and pass-by-reference.

Pass by Value :

In this type, value of actual arguments are passed to the formal arguments and the operation is done on the formal argument. Any change made in the formal argument does not affect the actual arguments because formal arguments are photocopy of actual arguments.

Hence, when the function is called by the call by value method, it does not effect the actual contents of the actual arguments. Changes made in the formal arguments are local to the block of the called function. Once control returns back to the calling function, the changes made vanish.

Important points regarding pass by value mechanism :

1. The actual arguments can be constants, variables or expressions.
2. When the control is transferred from the calling function to the called function, the memory for formal arguments and local variables is allocated, values of the actual arguments are substituted in the corresponding formal arguments, and the statements in the function body are executed.
3. As soon as the called function, finishes its execution, the memory allocated for it is deallocated i.e., the values of formal arguments and local variables are destroyed, and finally the control is transferred back to the calling function.
4. Any change made to the formal arguments will have no effect on actual arguments, since the function will only be using the local copy of the arguments.

/*Program to send values by Call by Value */

```
#include <stdio.h>
#include <conio.h>
void main( )
{
    int x, y;
    change(int, int);
    clrscr( );
    printf("Enter values of X and Y");
    scanf("%d%d", &x, &y);
    change(x, y);
    printf("\n In main( ) X = %d  Y = %d", x, y);
    getch( );
}
```

```
change(int a, int b)
{
        int k;
        k = a;
        a = b;
        b = k;
   printf("\n In change( ) X = %d  Y = %d", a, b);
}
```

Pass by Reference :

In pass by reference, the addresses of the actual arguments is used in the function call. In this way the addresses of the actual arguments are passed to the function. When control is transferred to the called function, the addresses of the actual arguments are substituted to corresponding formal arguments and the body of the function is executed. The formal arguments are declared as pointers to types that match the data types of the actual arguments. This approach is of practical importance while passing arrays and structures among functions, and also for passing back more than one value to the calling functions.

Important point regarding Pass by Reference mechanism :

1. The actual arguments can only be variables.
2. When the control is transferred to the calling function to the called function, the memory for formal arguments and local variables is allocated, addresses of the actual arguments are substituted in the corresponding formal arguments, and the statements in the function body are executed.
3. As soon as the called function finishes its execution, the memory allocated for it is deallocated i.e. the values of formal argument and local variables are destroyed, and finally the control is transferred back to the calling function.
4. Any change made to the formal arguments will have immediate effect on actual arguments, since the function will be working on actual arguments through pointers.

/*Program to send a value by reference to the user-defined function */

```
#include <stdio.h>
#include <conio.h>
void main( )
{  int x, y, change(int *, int *);
   clrscr( );
   printf("\n Enter values of X and Y");
   scanf("%d%d", &x, &y);
   change(&x, &y);
   printf("\n In main( ) X = %d  Y = %d", x, y);
   getch( );
}
change(int *a, int *b)
{
       int *k;
       *k = *a;
       *a = *b;
       *b = *k;
   printf("\n In change( ) X=%d  Y=%d", *a, *b);
   return;
}
```

Q12(b). What do you know about bitwise operation? Explain about some bitwise operators by providing the examples for each?

Ans. Bitwise operators :

[Please Refer to Q2 Unit-V, Page-67]

Q13(a). What are the commonly used input functions in 'C'? Write their syntax and explain the purpose of each?

Ans. Input functions used in C :

Input functions are the functions through which data is entered into the computer. These functions can feed the data from the standard input device like keyboard to the computer. Input functions used in C programming languages are :

(i) getchar( ) : This function is used to input a single character from the keyboard. This function can only store character type data and no other datatype can be stored through this function. This function will echo the character on the screen while inputting. This is buffered function. The general syntax is as :

Example : char non;
mn = getchar();

(ii) getch( ) : This function is similar to the getchar() function. But this function doesn't echo the character on the screen. It stores the character in the computer's memory. The general syntax is :

Example : v = getch( );

(iii) gets( ) : This function in C programming language can input a complete character string from the keyboard to the computer's memory. This function can include blanks and special character in the string. Its syntax is :

Example :
```
char nm[80];
gets (nm);
```

(iv) scanf( ) : scanf( ) function can input every type of data. To input data through this function different codes knows as control codes or format codes are used to input data of different types. These codes are:

%d – integer data
%f – float data
%lf – long float data
%ld – long integer
%c – for single character
%s – // string of characters
%u – unsigned integer etc.

Example :
```
char nm[15];
int a;
float b;
scanf("%s %d %f", nm, &a, &b);
```

To read the string we don't need to use '&' (ampersand sign), whereas it is necessary to use this sign with all other variables of other data types.

Q13(b). Define Auto and Register variables in the content of 'C'. What is the basic difference between these two variables?

Ans: Auto Variables (Storage Class) :

It is the default storage class for all local variables. These are the most common. All the variables defined in a code block are auto by default. Auto variables are initialized to undefined values until a valid assignment (of that type). The keyword indicates that the variable can only be used within the current block since the variable will be automatically created & destroyed as it is needed. This also means that auto variables cannot be global.

Example :
```
void a (void)
{
    int i; // equivalent to auto int i
}
```

Register storage class :

Register is used to define local variables that should be stored in a register instead of RAM. This means that the variable has a maximum size equal to the register size (usually one word) and cannot have the unary '&' operator applied to it (as it doesn't have a memory location)

Example :
```
{
    register int miles;
}
```

Register should only be used for variables that require quick access such as counters. It should also be noted that defining 'register' does not mean that the variable will be stored in a register. It means that it might be stored in a register depending upon hardware and implementation restrictions.