➢ FUNCTIONS

3.1 Types of functions
3.2 Types of library functions

- ♦ string functions

strcpy(), strncpy(), strcat(), strncat(), strchr(), strrchr(), strcmp(), strncmp(), strspn(), strcspn(), strlen(), strpbrk(), strstr(), strtok()

- ♦ Mathematical Functions

Acos(), asin(), atan(), ceil(), cos(), div(), exp(), fabs(), floor(), fmod(), log(), modf(), pow(), sin(), sqrt()

- ♦ Date and time functions

clock(), difftime(), mktime(), time(),asctime(),ctime(), gmtime(), localtime(), strftime()

- ♦ I/O Formatting Functions

printf(), scanf(), getc(), getchar(), gets(), putc(),putchar(), puts(), ungetc()

- ♦ Miscellaneous Functions

delay(), clrscr(), clearer(), errno(), isalnum(), isalpha(), iscntrl(), isdigit(), isgraph(), islower(), isprint(), isspace(), isupper(), isxdigit(), toupper(), tolower().

- ♦ Standard Library functions

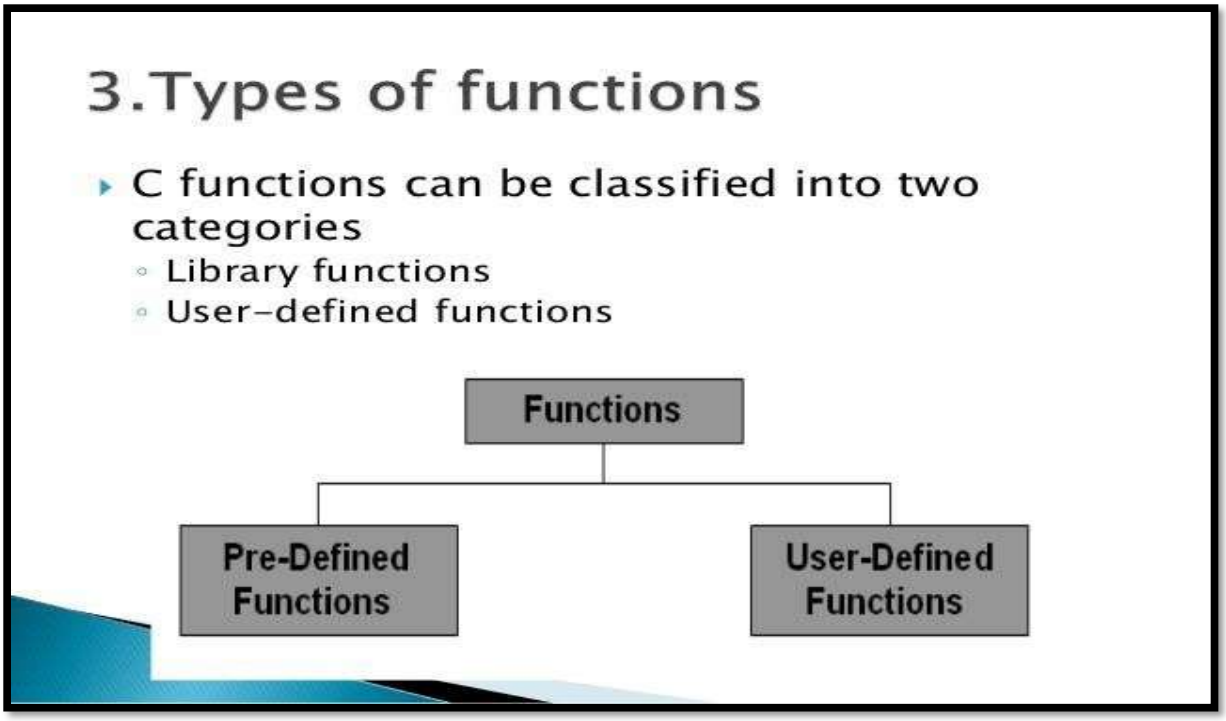
abs(), atof(), atol(), exit(), free(), labs(), qsort(), rand(), strtoul(), srand()

- ♦ Memory Allocation functions

malloc(), realloc(), calloc()

3.3 Types of user defined functions
3.4 Function call by value
3.5 Function call by reference
3.6 Recursion
3.7 Storage class
3.8 Passing and returning values.

Q-1 Explain Types of functions in C.



Detail :-

A function is a self-contained block of statements that perform a task of Some kind. C functions can be classified into two categories:

1. Library Functions.
2. User-defined functions

1. Library Functions:

C provides library functions for performing some operations. These functions are present in the c library and they are predefined for example. Sqrt() is a mathematical library function which is used for finding the square root of any number The function scanf and printf() are input and output library function similarly we have strcmp() and strlen() for string manipulations.

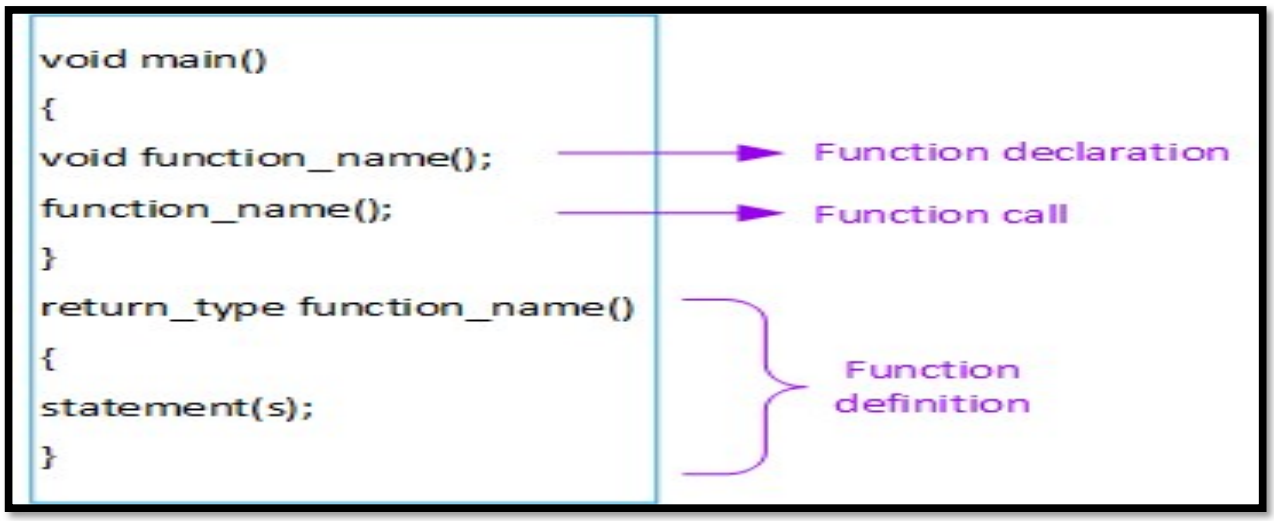
To use a library function we have to include some header file using the preprocessor directive #include. Example: to use input and output function like printf() and scanf() we have to include stdio.h, for math library function we have to include math.h for string library string.h should be included.

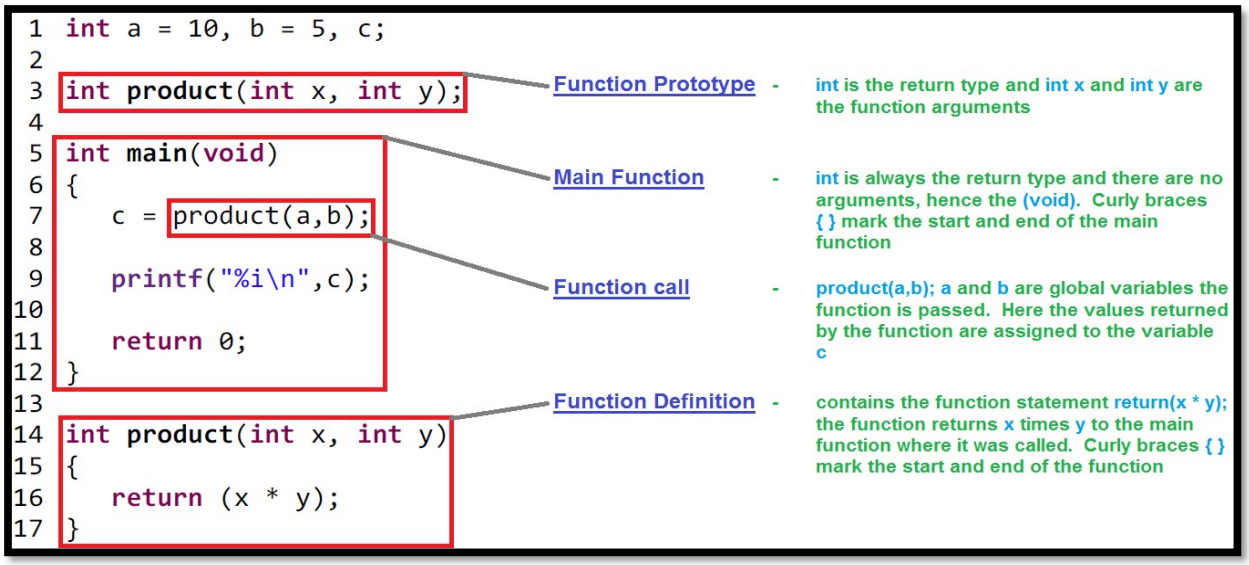
2. User-defined Functions:

User defined function (UDF) is an independent program module with group of statements and that function can be called and use whenever it is required. To make programming easier, we break larger program into smaller sub-programs to perform a specific well defined task. These sub-programs are called functions. To create and use UDF following 3 things should be kept in mind:

- Function Declaration
- Function Definition
- Function Call

Detail :-





- **Function Declaration (Function Prototype):**

The calling program should declare the function before it is used. This is known as function declaration or function prototype. A function declaration contains four elements

1. Function type
2. Function name
3. parameters list
4. terminating semicolon

Syntax:

function_type function_name(parameter list);

Example :

int add(int, int);

Function declaration is similar to function header except the terminating semicolon

- **Function Definition:**

The function definition consists of the whole description and code of a function. It tells that what the function is doing and what is

The input output for that. A function is called by simply writing the name of the function followed by the argument list inside the parenthesis. There are six elements of function definition:

1. Function type
2. Function name
3. Parameters list (arguments)
4. Local variable declaration
5. Function statements
6. return statement

Syntax:

```
function_type function_name(parameter list)
{
    local variable declaration;
    function statement 1;
    function statement 2;
      .
      .
      .
    return statement;
}
```

Example :

```
int add(int x, int y)
{
    int z;
    z = x + y;
    return(z);
}
```

The elements of function definition are categorized in two parts:

1. Function header
2. Function body

1. Function header: A function header contains three elements of function definition with function type, function name and parameter list.
   Example:

int add(int x,int y) is known as function header.

2. Function body: A function body contains last three elements of function definition with local variable declaration, function statements and a return statement. In the above example:

```
int z;
z=x+y;          Function body
return z;
```

- **Function call:**

A function can be called by using the function name and list of actual parameters.

Example :

```
main()
{
    c=add(10,5);//function call
    printf("%d",c);
}
```

In the above example, when a function calls statement encounter in program then the control is transferred to the function add(). After this add function is executed line by line and the answer is return to the function call using the return statement.

C String Functions:

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

String Function: strcpy, strncpy, strcat, strncat, strchr, strrchr, strcmp, strncmp, strspn, strcspn, strlen, strpbrk, strstr, strtok

Strcpy:

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

Example: C strcpy()

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

void main()
{
  char str1[20] = "C programming";
  char str2[20];

  // copying str1 to str2
  strcpy(str2, str1);

  printf("copy string is %s",str2); // C programming

  getch();
}
```

Strncpy:

- strncpy( ) function copies portion of contents of one string into another string.

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

```
void main( )
{
  char source[ ] = "Radha
  Ranpara" ;
  char target[20];
  printf ( "\nsource string = %s", source ) ;
  printf ( "\ntarget string = %s", target ) ;
  strncpy ( target, source, 5 ) ;
  printf ( "\ntarget string after strcpy( ) = %s", target ) ;
  getch();
}
```

Strcat:

concats or joins first string with second string. The result of the string is stored in first string.

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

Strncat:

The C library function **char *strncat(char *dest, const char *src, size_t n)** appends the string pointed to by **src** to the end of the string pointed to by **dest** up to **n** characters long.

```
#include<stdio.h>
#include <string.h>
int main(){
 char ch[10]="Radha ";
  char ch2[10]="soni";
  strncat(ch,ch2,3);
```

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

strcmp()

The `strcmp()` compares two strings character by character. If the strings are equal, the function returns 0.

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

int main() {
  char str1[] = "abcd", str2[] = "abCd", str3[] = "abcd";
  int result;

  // comparing strings str1 and str2
  result = strcmp(str1, str2);
  printf("strcmp(str1, str2) = %d\n", result);

  // comparing strings str1 and str3
  result = strcmp(str1, str3);
  printf("strcmp(str1, str3) = %d\n", result);

  return 0;
}
```

strncmp()

The C strncmp function is a String Function used to compare two strings. Or it checks whether those two strings are equal or not. The strncmp function uses the third argument to limit the comparison**.** Instead of comparing the whole string, it means you can compare the first four characters, or five characters, etc

```
#include <stdio.h>
#include<string.h>
int main()
{
  char str1[50] = "abcdef";
  char str2[50] = "abcd";
  char str3[] =  "ghi";
  int i, j, k;
  i = strncmp(str1, str2, 4);
  printf("\n The Comparison of str1 and str2 Strings = %d", i);
```

```
    j = strncmp(str1, str2, 6);
    printf("\n The Comparison of str1 and str2 Strings = %d", j);
    k = strncmp(str1, str3, 3);
    printf("\n The Comparison of str1 and str3 = %d", k);
}
```

strlen()

The strlen() function returns the length of the given string.

```
#include<stdio.h>

#include <string.h>

void main()

{

char ch[20]="radha";

   printf("Length of string is: %d",strlen(ch));

 return 0;
```

```
}
```

Strspn():

The C library function **size_t strspn(const char *str1, const char *str2)** calculates the length of the initial segment of **str1** which consists entirely of characters in **str2**.

```
#include <stdio.h>
#include <string.h>
#include<conio.h>
void main () {
   int len;
    char str1[] = "ABCDEFG019874";
    char str2[] = "ABCD";

   len = strspn(str1, str2);

   printf("Length of initial segment matching %d\n", len );

   getch();
}
```

Strcspn():

The `strcspn()` function finds the first occurrence of a character in *string1* that belongs to the set of characters that is specified by *string2*. Null characters are not considered in the search.

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

void main ()
```

```
{
  int len;
   char str1[] = "ABCDEF4960910";
   char str2[] = "013";

  len = strcspn(str1, str2);

  printf("First matched character is at %d\n", len );

  getch();
}
```

Strstr():

Return pointer to the first occurrence of a string.

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

void main ()
 {

    char str[] = "hello this is c programming";
    char *ptr = strstr(str,"this");

   printf("%s",ptr );

   getch();
}
```

Strtok():string tokenization

Split the string.

```
#include <stdio.h>
#include <string.h>
void main()
```

```
{
   char name[]="hii-world-hello";
   char *ptr=strtok(name,"-");
   while(ptr!=NULL)
   {
      printf("%s\n",ptr);
      ptr=strtok(NULL,"-");
   }
}
```

Output:

```
hii
world
hello
```

Strchr():

Return pointer to the first occurrence of a character.

```
#include <stdio.h>
#include<string.h>
void main()
{
   char str[20]="hello world";
   char *ch=strchr(str,'l');

   printf("\n%d",ch);//ch memory address return
   printf("\n%s",ch);//first occur string.

   getch();
}
```

Strrchr():

Return pointer to the last occurrence of a character.

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

```
void main()
{
   char str[20]="hello world";
   char *ch=strrchr(str,'l');

   printf("\n%s",ch);//first occur string.

   getch();
}
```

Strpbrk():

The function strpbrk() is used to find the first character of first string and matches it to any character of second string. It returns NULL, if no matches are found otherwise, it returns a pointer to the character of first string that matches to the character of second string.

```
#include <stdio.h>
#include <string.h>
int main () {
   const char s1[] = "AASDFAFS3123123";
   const char s2[] = "0123456789";
   char *result=strpbrk(s1,s2);
   printf(" %s\n", result);
   return(0);
}
```

Output:3123123

⍰ Header File:- <math.h>

- **MATHS FUNCTIONS :-**

Detail :-

(1) abs() :-

- This functions is used to return absolute value of any given number.

Syntax :- abs (<number>)
Example :- abs (-90)

(2) sqrt() :-

- This functions is used to return square root of any given number.

Syntax :- sqrt (<number>)
Example :- sqrt (100)

(3) ceil() :-

- This functions is used to return or round up given value upto near by maximum value of given number.

Syntax :- ceil (<number>)
Example :- ceil(-14.55)

(4) floor() :-

- This functions is used to return or round up given value upto near by minimum value fo given number.

Syntax :- floor (<number>)
Example :- floor(-14.55)

(5) div() :-

- This functions is used to return reminder of any given number.

Syntax :- div (<number>,<divisible by>)
Example :- div(10,3)

(6) exp() :-

- This functions is used to return exponent value of any given number in which e raised to value of n.

Syntax :- exp (<number>)
Example :- exp(5)

(7)pow() :-

- This functions is used to return the value calculated by given number and its power.

Syntax :- pow (<number>,<raised to>)
Example :- pow(2,3)

(8)pow() :-

- This functions is used to return natural logarithm of any given number.

Syntax :- log(<number>)
Example :- log(3.123)

(9) modf() :-

- This functions is used to return integer part and floating point part of any given number.

Syntax :- modf(<number>,<int.part>)
Example :- modf(3.00000,3)

(10) fabs() :-

- This functions is used to return absolute value of floating point number.

Syntax :- fabs(<number>)
Example :- fabs(-23.40)

(11)sin() :-

- This functions is used to calculate sine value of given number.

Syntax :- sin(<number>)
Example :- result = sine(param number * pi)/180;

(12)cos() :-

- This functions is used to calculate cosine value of any given number.

It return the value in the range from -1 to +1;

Syntax :- cos(<number>)
Example :- result = cos(number * pi)/180;

(13) asin() :- [arc = curve]

- Asin() is known as arc sin function.
- It used to calculate value of arc sine in the range from -1 to +1.

Syntax :- asin(<number>)
Example :- result = asin(number)*180.0/PI;

(14) acos() :-

- This function is used to calculate principle value of any number in the range from 0 to 1.

Syntax :- acos(<number>)
Example :- acos(number)*180.0/1;

(15) atan() :-

- This function is known as arc tangent function.
- It is used to calculate principal value of any given number in the range [-n/2 , +n/2].

Syntax :- atan(<number>)
Example :- acos(number)*180.0/1;

(16) fmod() :-

- This function is used to calculate floating point reminder.
- You can calculate value like , x/y and it give you reminder.

Syntax :- fmod(<number>,<divisible by>)
Example :- fmod(5.3,2);

Date & Time Functions: clock, difftime, mktime, time, asctime, ctime, gmtime, localtime, strftime

Header file <time.h>

Function	Description
clock()	This function is used to get the date & time
time()	This function is used to get current system time as structure
difftime()	This function is used to get the difference between two given times
strftime()	This function is used to modify the actual time format
mktime()	This function interprets tm structure as calendar time
localtime() gmtime()	This function shares the tm structure that contains date and time information
ctime()	This function is used to return string that contains date and time information
asctime()	Tm structure contents are interpreted by this function as calendar time. This time is converted into string.

It describes three time-related data types.

clock_t: clock_t represents the date as an integer which is a part of the calendar time.

time_t: time_t represents the clock time as an integer which is a part of the calendar time.

struct tm: struct tm holds the date and time which contains:

```
struct tm {
   int tm_sec;         /* seconds,  range 0 to 59          */
   int tm_min;         /* minutes, range 0 to 59           */
   int tm_hour;        /* hours, range 0 to 23             */
   int tm_mday;        /* day of the month, range 1 to 31  */
   int tm_mon;         /* month, range 0 to 11             */
   int tm_year;        /* The number of years since 1970   */
   int tm_wday;        /* day of the week, range 0 to 6    */
   int tm_yday;        /* day in the year, range 0 to 365  */
};
```

Example:1

```
#include <stdio.h>
#include <string.h>
#include <time.h>
int main () {
  struct tm t;
  t.tm_sec    = 10;
  t.tm_min    = 10;
  t.tm_hour  = 6;
  t.tm_mday  = 25;
  t.tm_mon    = 2;
  t.tm_year  = 89;
  t.tm_wday = 6;
  puts(asctime(&t));
  return(0);
}
```

Output: Sat Mar 25 06:10:10 1989

Example 2:

```
#include <stdio.h>
#include<conio.h>
#include  <time.h>
int main()
{
   time_t seconds;
   seconds=time(NULL);
   printf("seconds since 1st jan 1970=%ld\n",seconds);
   return 0;
}
```

Output: seconds since 1st jan 1970=1664820155

Example:3

```
#include <stdio.h>
#include <string.h>
#include <time.h>
int main ()
{
  time_t mytime;
  time(&mytime);
```

```
    printf("current time = %s",ctime(&mytime));

    return(0);

}
```

Output: current time = Mon Oct 3 18:11:56 2022

I/O Formatting Functions: printf, scanf, getc, getchar, gets, putc, putchar, puts, ungetc

printf():
printf() function is used in a C program to display any value like float, integer, character, string, etc on the console screen. It is a pre-defined function that is already declared in the stdio.h(header file).

scanf():
scanf() function is used in the C program for reading or taking any value from the keyboard by the user, these values can be of any data type like integer, float, character, string, and many more. This function is declared in stdio.h(header file), that's why it is also a pre-defined function. In scanf() function we use &(address-of operator) which is used to store the variable value on the memory location of that variable.

getchar():

A **getchar()** function is a **non-standard** function whose meaning is already defined in the **stdin.h** header file to accept a single input from the user. In other words, it is the C library function that gets a single character (unsigned char) from the stdin. However, the getchar() function is similar to the getc() function, but there is a small difference between the getchar() and getc() function.

putchar():

putchar is a function in the C programming language that writes a single character to the standard output stream, The character to be printed is fed into the function as an argument, and if the writing is successful, the argument character is returned. Otherwise, end-of-file is returned.

The `putchar` function is specified in the C standard library header file stdio.h.

```c
#include  <stdio.h>
#include <conio.h>
void main()
{
    char c;
    printf ("\n Enter a character \n");
    c = getchar(); // get a single character
    printf(" You have passed ");
    putchar(c); // print a single character using putchar
   getch();
}
```

Output:

```
Enter a character
A
 You have passed A
```

gets():

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.

puts():

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.

Example:

```c
#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:

Radha


Your name is: Radha
```

getc() and putc() :

The **getc** and **putc** functions are analogous to **getchar** and **putchar** functions and handle one character at a time.

ungetc():

The **ungetc()** function takes a single character and shoves it back onto an input stream. It is the opposite of the getc() function, which reads a single character from an input stream. Also, ungetc() is an input function, not an output function.

Syntax:

```
int ungetc(int char, FILE *stream)
```

- **STANDARD LIBRARY FUNCTIONS :-**

Header File: :- <stdio.h>

Standard C-Library Functions

<stdlib.h>	
int atoi(s)	Converts string s to an integer
long atol(s)	Converts string s to a long integer.
float atof(s)	Converts string s to a double-precision quantity.
void* calloc(u1,u2)	Allocate memory to an array u1, each of length u2 bytes.
void exit(u)	Closes all files and buffers, and terminate the program.
void free (p)	Free block of memory.
void* malloc (u)	Allocate u bytes of memory.
int rand(void)	Return a random positive integer.
void* realloc(p,u)	Allocate u bytes of new memory to the pointer variable p.
void srand(u)	Initialize the random number generator.
void systerm(s)	Pass command string to the operating system.

Detail :-

1.atol():

- This function is used to convert string argument to long integer value.

Syntax:

```
atol(<string>)
```

Example:

```
Void main()
{
        Char a[20] = “hi friends”;
        long b;
        b= atol(a);
        printf(“string=%s”,longint = %ld”,a,b);
        getch();
}
```

2. atof():

- This function is used to convert string argument to float number.

Syntax:

```
atof(<string>)
```

Example:

```
Void main()
{
        Char a[20] = “1234567”;
        float b;
        clrscr();
        b= atof(a);
        printf(“string=%s”,float = %f”,a,b);
        getch();
}
```

3. exit():

This function is used to terminate calling process immediately.

Syntax:

```
exit(<int>)
```

Example:

```
Void main()
{
        Clrscr();
        Printf("starting of program\n");
        Exit(0);
        Printf("exiting of program\n");
        getch();
}
```

4. labs():

- This function is used to return absolute value of long integer value.

Syntax:

```
labs(<long integer>)
```

Example:

```
Void main()
{
        long int a,b;
        a=labs(65987);
        Printf("%ld\n",a);
        B=labs(-100509);
        Printf("%ld\n",b);
        getch();
}
```

- **5) Free():**
- This function is used to free or de-allocate the memory allocated by calloc , malloc and re-alloc.

Syntax:

Free(<pointer variable>)

Example:

```
Void main()
{
        Char *str;
        Str = (char *) malloc(15);
        Strcpy(str,"tutorial point");
        Printf("string=%s ,address=%u\n",str,str);
        Free(str);
        Getch();
}
```

6. rand():

- This function is used to generate random numbers in the range from 0 to n.

Syntax:

labs(<long integer>)

Example:

```
Void main()
{
        int i,n;
        clrscr();
        printf("enter n");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        { Printf("%d",rand()%50);
        }getch();
        }
```

- **MISCELLANEOUS FUNCTIONS :-**

Header File: :- <stdio.h>

Miscellaneous functions	Description
getenv()	This function gets the current value of the environment variable
setenv()	This function sets the value for environment variable
putenv()	This function modifies the value for environment variable
perror()	Displays most recent error that happened during library function call
rand()	Returns random integer number range from 0 to at least 32767
delay()	Suspends the execution of the program for particular time

Function	Return Type	Use
isalnum(c)	int	Determine if the argument is alphanumeric or not
isalpha(c)	int	Determine if the argument is alphabetic or not
isascii(c)	int	Determine if the argument is ASCII character or not
isdigit(c)	int	Determine if the argument is a decimal digit or not.
toascii(c)	int	Convert value of argument to ASCII
tolower(c)	int	Convert character to lower case
toupper(c)	int	Convert letter to uppercase

Detail :-

1. **delay():<dos.h>**

   - This function is used to suspend execution of program for particular amount of time.

Syntax:

```
delay(<int>)
```

Example:

```
#include<dos.h>

Void main()
{
        printf("This c program will exit in 10 seconds.\n");
        delay(10000);
       getch();
}
```

   - Above c program exits in ten seconds, after the printf function is executed the program waits for 10000 milliseconds or 10 seconds and then program termination occurs.

2. **clrscr():**

   - This function is used to clear the output of the screen.

Syntax:

```
Clrscr();
```

Example:

```
Void main()
{
        Int i=10;
```

```
        Clrscr();
        printf("%d",i);
        getch();
    }
```

3. islower():<ctype.h>

- This function is used to check if entered character is in lowercase or not.

Syntax:

```
islower(character);
```

Example:

```
#include<ctype.h>

Void main()
{
    Char ch = 'A';
    Clrscr();
    If(islower(ch))
        Printf("lowercase");
    Else
        Printf("not in lowercase");

    getch();
}
```

4. isupper():<ctype.h>

- This function is used to check if entered character is in uppercase or not.

Syntax:

```
isupper(character);
```

Example:

```
#include<ctype.h>
```

```
Void main()
{
        Char ch = 'A';
        Clrscr();
        If(isupper(ch))
                Printf("uppercase");
        Else
                Printf("not in uppercase");

        getch();
}
```

5. isspace():<ctype.h>

- This function is used to check if entered character is whitespace character or not.

Syntax:

```
isspace(character);
```

Example:

```
#include<ctype.h>

Void main()
{
        Char ch = ' ';
        Clrscr();
        If(isspace(ch))
                Printf("whitespace ");
        Else
                Printf("not whitespace");

        getch();
}
```

6. isalpha():<ctype.h>

- This function is used to check if entered character is alphabet or not.

Syntax:

```
Isalpha(character);
```

Example:

```
#include<ctype.h>

Void main()
{
        Char ch = 1;
        Clrscr();
        If(isalpha(ch))
                Printf("alphabet");
        Else
                Printf("not alphabet");

        getch();
}
```

7. isprint():<ctype.h>

- This function is used to check if entered character is printable or not.

Syntax:

```
Isalpha(character);
```

Example:

```
#include<ctype.h>

Void main()
{
        Char ch = 1;
        Clrscr();
        If(isprint(ch))
                Printf("printable");
```

```
        else
            Printf("not printable");

        getch();
    }
```

8. isdigit():<ctype.h>

- This function is used to check if entered character is digit or not.

Syntax:

```
Isdigit(character);
```

Example:

```
#include<ctype.h>

Void main()
{
    Char ch = 'a';
    Clrscr();
    If(isdigit(ch))
        Printf("digit");
    else
        Printf("not digit");

    getch();
}
```

9. isalnum():<ctype.h>

- This function is used to check if entered character is alphabet or number.

Syntax:

```
Isalnum(character);
```

Example:

```
#include<ctype.h>
```

```
Void main()
{
        Char ch = 'a';
        Clrscr();
        If(isalnum(ch))
                Printf("alpha-numeric");
        Else
                Printf("not alpha-numeric");

        getch();
}
```

10. **isctrl():<ctype.h>**

- This function is used to check if entered character is control character or not.

Syntax:

```
Isdigit(character);
```

Example:

```
#include<ctype.h>

Void main()
{
        Char ch = '%';
        Clrscr();
        If(isctrl (ch))
                Printf("control character");
        Else
                Printf("not control character");

        getch();
}
```

11. **errno():<ctype.h>**

- This function is used to return error number related to particular object.
- **Note :-** you have to include header file **<errno.h>**

Syntax:

```
errno( );
```

Example:

```
#include<errno.h>

Void main()
{
        File *fp;
        Fp=fopen("hello.txt","r");
        Printf("errno=%d\n",errno());
        getch();
 }
```

12. toupper():<ctype.h>

- This function is used to convert lowercase character into uppercase.

Syntax:

```
toupper(<character>)
```

Example:

```
#include<errno.h>

Void main()
{
        Char ch= 'a';
        Clrscr();
        Int c=toupper(ch);
        Printf("%c",c);
        getch();


 }
```

13. tolower():<ctype.h>

- This function is used to convert uppercase character into lowercase.

Syntax:

```
tolower(<character>)
```

Example:

```
#include<errno.h>

Void main()
{
        Char ch= 'A';
        Clrscr();
        Int c=tolower(ch);
        Printf("%c",c);
        getch();
 }
```

malloc(): (runtime memory allocation)

The **“malloc”** or **“memory allocation”** method in C is used to dynamically allocate a single large block of memory with the specified size. It returns a pointer of type void which can be cast into a pointer of any form. It doesn’t Initialize memory at execution time so that it has initialized each block with the default garbage value initially.

Syntax:

```
ptr = (cast-type*) malloc(size-byte)//ptr pointer variable
for block base address.
```

example:

ptr=(int*)malloc(10);//10 byte block

int=2 byte : 10 byte//5 array

10 byte: 

1. Program to calculate the sum of n numbers entered by the user (MALLOC FUNCTION).

```
#include  <stdio.h>
#include <stdlib.h>
int main() {
  int n, i, *ptr, sum = 0;
  printf("Enter number of elements: ");
  scanf("%d", &n);
  ptr = (int*) malloc(n * sizeof(int));
  // if memory cannot be allocated
  printf("Enter elements: ");
  for(i = 0; i < n; ++i)
  {
```

```
    scanf("%d", ptr + i);
    sum += *(ptr + i);
  }
  printf("Sum = %d", sum);
  // deallocating the memory
  free(ptr);
  return 0;
}
```

calloc() : (runtime memory allocation)

- The name "calloc" stands for contiguous allocation.
- The malloc() function allocates memory and leaves the memory uninitialized, whereas the calloc() function allocates memory and initializes all bits to zero.
- 3,10 byte: 

Syntax:

```
ptr = (castType*)calloc(n, size);
```

example:

1. // Program to calculate the sum of n numbers entered by the user (CALLOC FUNCTION)

```
#include  <stdio.h>

#include <stdlib.h>

int main() {

  int n, i, *ptr, sum = 0;
```

```
    printf("Enter number of elements: ");
    scanf("%d", &n);
    ptr = (int*) calloc(n , sizeof(int));
    // if memory cannot be allocated
    printf("Enter elements: ");
    for(i = 0; i < n; ++i) {
      scanf("%d", ptr + i);
      sum += *(ptr + i);
    }
    printf("Sum = %d", sum);
    // deallocating the memory
    free(ptr);
    return 0;
  }
```

realloc()

realloc() is a function of C library for adding more memory size to already allocated memory blocks.

The purpose of realloc in C is to expand current memory blocks while leaving the original content as it is.

realloc() function helps to reduce the size of previously allocated memory by malloc or calloc functions.

realloc stands for reallocation of memory.

Syntax for realloc in C

```
ptr =(int*) realloc (ptr,newsize);
```

Call by value and Call by reference in C

There are two methods to pass the data into the function in C language, i.e., *call by value* and *call by reference*.

Call by value in C

- In call by value method, the value of the actual parameters is copied into the formal parameters. In other words, we can say that the value of the variable is used in the function call in the call by value method.
- In call by value method, we can not modify the value of the actual parameter by the formal parameter.
- In call by value, different memory is allocated for actual and formal parameters since the value of the actual parameter is copied into the formal parameter.
- The actual parameter is the argument which is used in the function call whereas formal parameter is the argument which is used in the function definition.

Call by Value Example: Swapping the values of the two variables

```
#include <stdio.h>
void swap(int , int); //prototype of the function
int main()
{
   int a = 10;
   int b = 20;
   printf("Before swapping the values in main a = %d, b = %d\n",a,b);
// printing the value of a and b in main
   swap(a,b);
   printf("After swapping values in main a = %d, b = %d\n",a,b); // Th
e value of actual parameters do not change by changing the formal p
arameters in call by value, a = 10, b = 20
}
void swap (int a, int b)
```

```
{
    int temp;
    temp = a;
    a=b;
    b=temp;
    printf("After swapping values in function a = %d, b = %d\n",a,b); //
Formal parameters, a = 20, b = 10
}
```

Output:

```
Before swapping the values in main a = 10, b = 20
After swapping values in function a = 20, b = 10
After swapping values in main a = 10, b = 20
```

Call by reference in C

- In call by reference, the address of the variable is passed into the function call as the actual parameter.
- The value of the actual parameters can be modified by changing the formal parameters since the address of the actual parameters is passed.
- In call by reference, the memory allocation is similar for both formal parameters and actual parameters. All the operations in the function are performed on the value stored at the address of the actual parameters, and the modified value gets stored at the same address.

Call by reference Example: Swapping the values of the two variables

```
#include <stdio.h>

#include<conio.h>

    void swap(int *, int *); //prototype of the function
    int main()
    {
        int a = 10;
        int b = 20;
```

```
    printf("Before swapping the values in main a = %d, b = %d\n",a,b);
// printing the value of a and b in main
    swap(&a,&b);
    printf("After swapping values in main a = %d, b = %d\n",a,b); // Th
e values of actual parameters do change in call by reference, a = 10, b
 = 20
}
void swap (int *a, int *b)
{
    int temp;
    temp = *a;
    *a=*b;
    *b=temp;
    printf("After swapping values in function a = %d, b = %d\n",*a,*b);
// Formal parameters, a = 20, b = 10
}
```

Output

```
Before swapping the values in main a = 10, b = 20
After swapping values in function a = 20, b = 10
After swapping values in main a = 20, b = 10
```

Difference between call by value and call by reference in c

No.	Call by value	Call by reference
1	A copy of the value is passed into the function	An address of value is passed into the function
2	Changes made inside the function is limited to the function only. The values of the actual parameters do not change by changing the formal parameters.	Changes made inside the function validate outside of the function also. The values of the actual parameters do change by changing the formal parameters.
3	Actual and formal arguments are created at the different memory location	Actual and formal arguments are created at the same memory location

- Difference:-

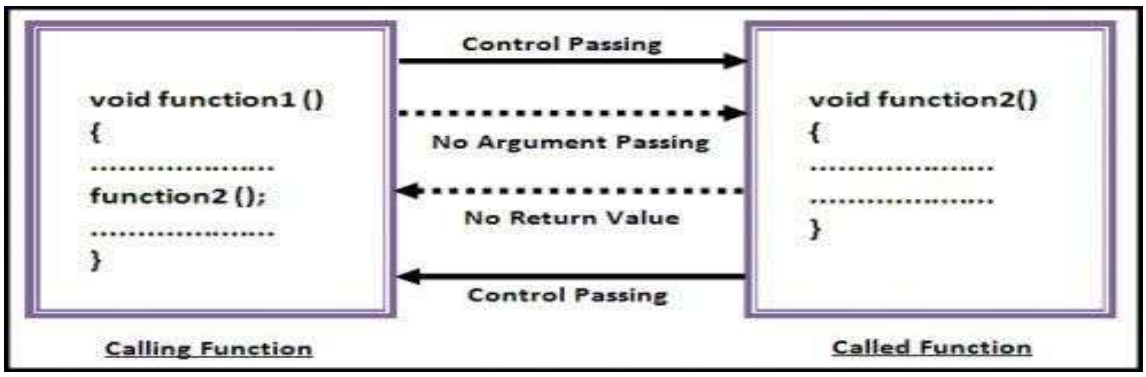
Calloc	Malloc
1. we need to split and pass the memory as we want.	1. In malloc, how much memory we want we can pass it in a one shot.
2.example : p = calloc(5 ,sizeof(int));	2.example : p = malloc(5*sizeof(int));
3.Function return void *	3.Function return void *
4.Returns starting address but before it allocating to us, it will zero it. means privious garbadge value memory clear	4.Returns starting address
5.Calloc is slower than malloc	5.malloc is faster than calloc
6.It is safe as compare to malloc	6.It is not safe as much as calloc

UDF

There are 4 types of UDF:

1. Function with no arguments and with no return value
2. Function with no arguments and with return value
3. Function with arguments and no return value
4. Function with arguments and with return value

1. Function with no arguments and with no return value:



No Way Data Communication

This type of UDF does not receive any values from the calling function as well as it does not return any value. The calling function does not receive any values from the called function that means there is no transfer of data between the calling and called function. Only the control passes to the called function.

Example:

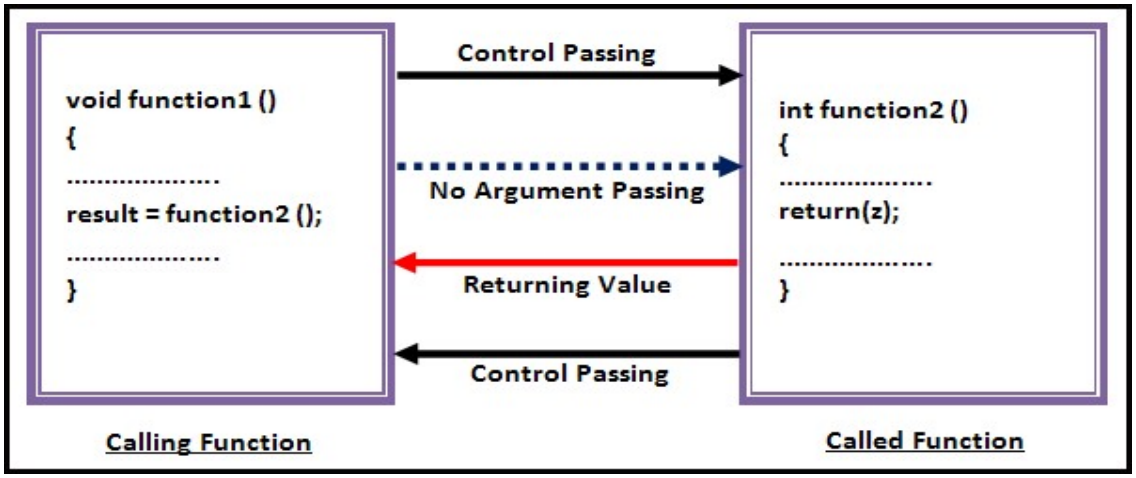
```
#include<stdio.h>
#include<conio.h>

void print(); // Function Declaration

void main()
{
        print();        // Function call
        getch();
}
void print()  // Function Definition
{
```

Output:

Hello

```
        printf("\n Hello");
}
```

2. Function with no arguments and with return value:



One Way Data Communication

This type of UDF does not have any arguments so called function does not receive any data from the calling function but it has return type so calling function receive the data from the called function. So, in this type of UDF, one way data communication takes place.

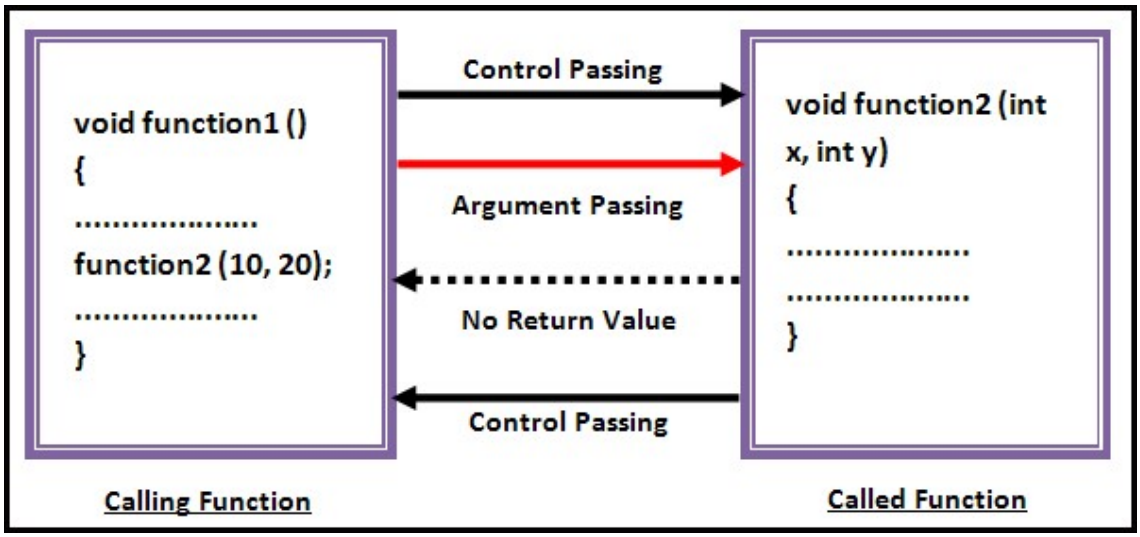
Example:

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

int add();      // Function Declaration

void main()
{
        int x;
        x=add(); // Function call
```

Output:
enter the first number 5
enter the second number 5
answer=10

```
        printf("\n answer=%d",x);
        getch();
}
int add()  // Function Definition
{
        int a,b,c;
        printf("\n enter the first number");
        scanf("%d",&a);
        printf("\n enter the second number");
        scanf("%d",&b);
        c=a+b;
        return c;
}
```

3. Function with arguments and with no return value:



One Way Data Communication

This type of UDF have arguments so called function receives data from the calling function but it does not have return type so calling function does

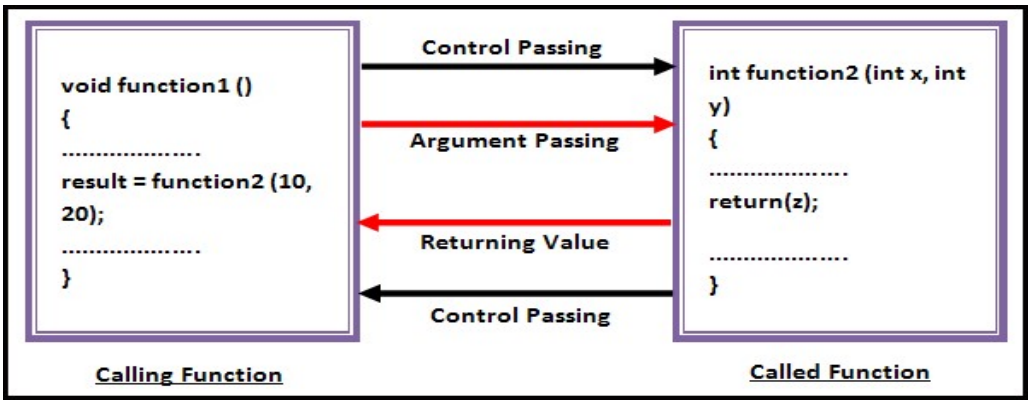
not receive the data from the called function. So, in this type of UDF, one way data communication takes place.

Example:

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

void add(int, int); // Function Declaration

void main()
{
        int x,y;
        printf("\n enter x:");
        scanf("%d",&x);
        printf("\n enter y:");
        scanf("%d",&y);
        add(x,y); // Function call
        getch();
}

void add(int a, int b)  // Function Definition
{
        int c;
        c=a+b;
        printf("\n addition=%d",c);
}
```

Output:
enter x: 5
enter y: 6
addition=11

4. Function with arguments and with return value:



Two Way Data Communication

This type of UDF have both arguments and return value so called function receives the data from calling function and calling function also receives the data from the called function so in this type of UDF, two way communication takes place.

Example:

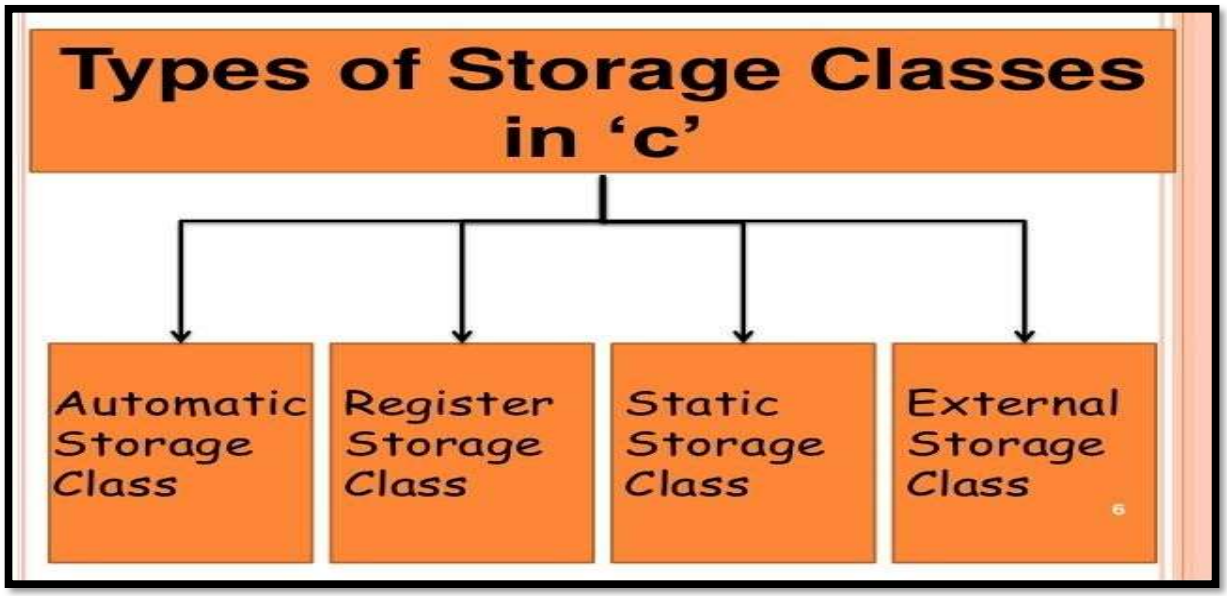
```
#include<stdio.h>
#include<conio.h>

int add(int, int);      // Function Declaration

void main()
{
        int x,y,z;
        printf("\n enter x:");
        scanf("%d",&x);
        printf("\n enter y:");
        scanf("%d",&y);
        z=add(x,y); // Function call
        printf("\n addition=%d",z);
        getch();
}


int add(int a, int b)  // Function Definition
{
        int c;
        c=a+b;
        return c;

}
```

Output:
enter x: 5
enter y: 6
addition=11

- **Explain Storage Class in detail.**

Detail :-

Storage class determines the scope (region of the program in which a variable is available for use.) and lifetime (duration in which a variable exist in the memory during execution of a program.). Storage class tells:



Storage Class	Declaration Location	Scope (Visibility)	Lifetime (Alive)
auto	Inside a function/block	Within the function/block	Until the function/block completes
register	Inside a function/block	Within the function/block	Until the function/block completes
extern	Outside all functions	Entire file plus other files where the variable is declared as extern	Until the program terminates
static (local)	Inside a function/block	Within the function/block	Until the program terminates
static (global)	Outside all functions	Entire file in which it is declared	Until the program terminates

- Where the variable is stored.
- Initial value of the variable.
- Scope of the variable. Scope specifies the part of the program which a variable is accessed.
- Life of the variable.

There are four types of storage class:

1. Automatic Storage class
2. Register Storage class
3. Static Storage class
4. External Storage Class

1. Automatic Storage Class:

The variables that are local to the function are known as automatic variables i.e. declared within the function. It is the default storage class for the variables declared in a function.

- Keyword : auto
- Storage Location : Main memory
- Initial Value : Garbage Value
- Life : Local (function in which it is declared).
- Scope : Local to the block in which variable is declared.

Example:

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

void main()
{
        auto int i=10;
        clrscr();
        {
      auto int i=20;
       printf("\n\t %d",i);
      printf("\n\n\t %d",i);
}
      getch();}
```

Output:

20

10

2. Register Storage Class:

Register variables are declared inside the function. When the variables are required very often during the program, they may be declared as register variable. Register access is faster than memory access. Generally, the register variables are used as loop counters which are used multiple times in the program. Register variables are not applicable for arrays, structures or pointers.

- Keyword : register
- Storage Location : CPU Register
- Initial Value : Garbage
- Life : Local to the block in which variable is declared.
- Scope : Local to the block.

Example:

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

void main()
{
        register int i;
        for(i=1;i<10;i++)
        printf("%d",i);
        getch();
}
```

Output:

1 2 3 4 5 6 7 8 9 10

3. Static Storage Class:

Static storage class can be used only if we want the value of a variable to persist between different function calls. If the control comes back to the same function again, the static variables have the same values they had last time around. By default, global variables are static variables.

- Keyword : static
- Storage Location : Main memory
- Initial Value: Zero and can be initialize only once.
- Life : depends on function calls and the whole application or program.
- Scope : Local to the block.

Example:

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

void print();

void main()
{
}
void print()
{
clrscr();
print();
print();
getch();
static int k;
printf("\n the value of static variable is %d",k);
k++;
}
```

Output:

value of static variable is 0
value of static variable is 1

4. External Storage Class:

The variables of external storage class can be referred to as 'global or external variables.' They are declared outside the functions and can be invoked at anywhere in a program. The only drawback of declaring the variables as extern is it wastes lot of memory because these variables remain active throughout the life of the program.

- Keyword : extern
- Storage Location : Main memory
- Initial Value : Zero
- Life : Until the program ends.
- Scope : Global to the program.

Example:

```
#include <stdio.h>


#include <conio.h>

extern int i=10;
void main()
{
        int i=20;
        void show(void);
        clrscr();
        printf("\n\t %d",i);
        show();
        getch();
}
void show(void)
{
        printf("\n\n\t %d",i);
}
```

Output:

20

10

ONE - MARK QUESTIONS

- What is function?
- Give Full Form of UDF.
- List out Types of UDF.
- Which Operator is used to declare Pointer Variable?
- List out different Memory Allocation Functions.
- Write down Syntax for Realloc()

TWO - MARK QUESTIONS

- Explain any two String Functions with Example.
- Explain any two Maths Functions with Example.
- Explain any two Date & Time Functions with Example.

THREE - MARK QUESTIONS

- Explain Recursion with suitabale Example.
- Explain any three I/O Formatting functions with Example.
- Explain Free() , Exit() ,atof() ,atol() with Example.
- Give difference between Malloc() and Calloc().
- Give difference between call value and call by reference.

FIVE - MARK QUESTIONS

- Explain any six Miscellaneous Functions with Example.
- Explain Pointer with Suitable Example.
- Explain Call by Value and Call by Reference with Example.
- Write note on different Storage Classes.
- Explain User Define Function with Example.