Unit 3 Functions

functions: we have used main(), printf, scanf and etc in a program to perform some particular task.

→ if a program contains a few lines of code then it is easy to understand and debug & test the program

→ if a program contains a huge number of lines, it is difficult to test and, debug and maintaining the program

→ if a large program is sub-divided into smaller or some functions parts and may be independently coded then it is easy to maintain the program

→ These independently coded parts are known as sub programes. In 'C' language such sub programs are refered to as functions

Function:
It is a self contained block of statements used to perform a particular task

→ Types of functions: These are of 2 types:

1) Library functions

These are pre defined functions and are available in 'c' library

eg: printf(), scanf(), pow(), sqrt()

2) User-defined functions:

These are defined by the user

eg:- xyz(), abc(), sum(), avg(), total()

When 'C' program is executed operating system starts its execution by calling the main( ), after that we can call any function from any other function.

Note

If a function abc( ) is calling from the function xyz( ) then xyz( ) is calling function
abc( ) is called function

Advantages of functions.

1) The lengthy complex program can be reduced by dividing the smaller parts.
2) Testing and Debugging becomes easy
3) easy to isolate the errors.
4). reusability of the existing code (we can write the code once and can use any number of times)

* Structure of 'C' function (or called function or function definition)

Syntax

```
Returntype functionname(parameter list)      function header
{
    Local declaration;
    Executable stmt 1;
               stmt 2;
    - - - - - - - - - -
    - - - - - - - - - -
    Executable stmt n;
    Return stmt;
}
```

where Return type is any datatype, function name is a user defined name & parameter list is containing the variables or arguments.

eg.
```
int mul(int a, int b)
{
    int c;
    c = a*b;
    return(c);
}
```
it returns(c) ie the product of a and b to the calling function

Note:- the last two stmts can be combined into a single stmt ie return(a*b)

*Characteristics of functions:

→ A function may have any number of arguments that depends on the requirement of the function.

→ It may or may not have a return value and the default return value from a function is integer.

→ If a function doesnot return any value then it should be declared as type of void.

→ If a function returns non-integer value. then mention the return type.

*In order to use user defined functions :-

we need to esstablish 3 elements.

1) function declaration (or) prototype
2) function definition
3) function call.

function [illegible]

1) function declaration or prototype

like any other variable function should be declared before its usage

syntax

```
Returntype   functionname(paratmeter list);
        eg int  sum(int,int);
                  or
            int  sum(int a, int b);
```

Note : parameter name is optional inthe declaration

2) * function header: first line of the called function or function definition is called as function header

3) function call:

It is an operational statement which sends the control to the called function.

Syntax:

```
function name(parameter list);

  eg. sum(a,b);        square(x);
      mul (x,y);
```

* Actual parameters : parameters inside the function call are called as actual parameters

* formal parameters : parameters inside the called function are called as formal parameter

Q) Write a 'c' program to find out square of a given number using functions

program:-

```
#include<stdio.h>
# include < conio.h>
void main()
{
 void square(int); // prototype (or) Declaration.
   int x;
   clrscr();
   printf("\n enter the value of x");
   scanf ("%d", &x);
   square(x);      -> Actual parameter        x = 5
   getch();                                   calling function
}
void square (int x)    -> formal parameter
{                      -> function header
   int s;
   s = x * x;                                 fn body  s = 5*5
   printf("s = %d", s);                       =>S=25
}                                             =>S=25
```

* Catagiries of functions based on arguments and return value (or) User-defined types :-

-> These are of 4 types.

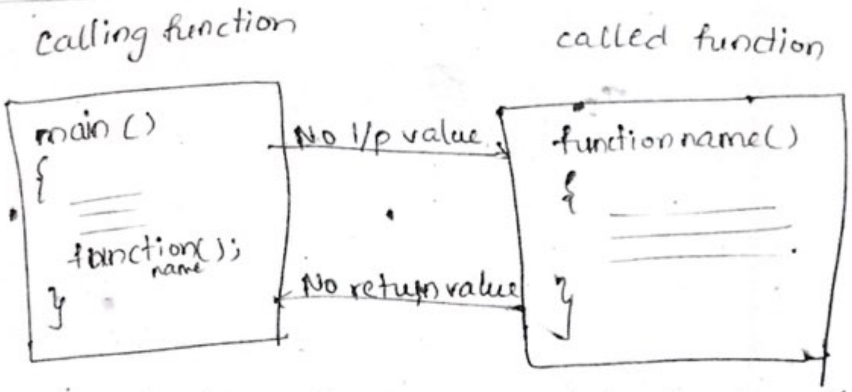
(1) A function without arguments and without return value

2) A function without arguments and with return value

(3) A function with argument and without return value

ii) A function with argument and with return value

I) A funiction without argument and without return value;



→ calling function does not pass any data to the called function in the same way called function doesn't return any value to the calling function

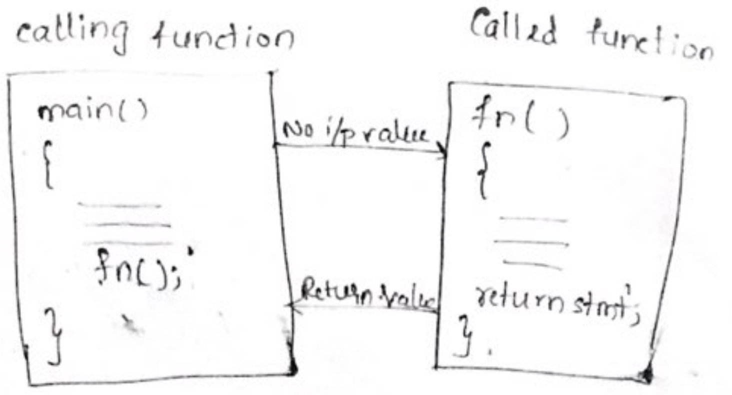
eg: program:

```
# include<stdio.h>
# include <conio.h>
void main()
{
  void sum();
  clrscr();
  sum();
  getch();
}
void sum()
{
 int a,b,c;
 printf("enter the value of a,b");
 scanf("%d%d",&a,&b);
 c=a+b;
 printf("c=%d", c);
}
```

void - function

any value

2) A function without arguments and with return value

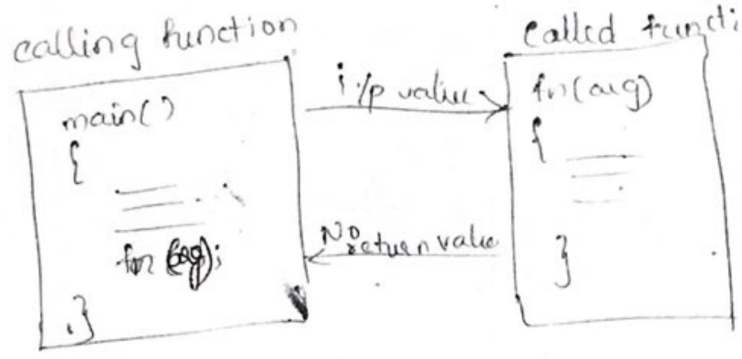


→ the calling function does not pass any data to the called function but the called function return the value to the calling function

eg: program:

```
#include<stdio.h>
#include<conio.h>
voidmain()
{
   int sum();
   int c;
   clrscr();
  c=sum();
  printf("c=%d", c);
  getch();
}
.int sum()
{
  int a,b,c;
  printf("enter the value of a,b");
  scanf("%d %d", &a &b);
  c=a+b;
  return(c);
}
```

3) with arguments without return value.

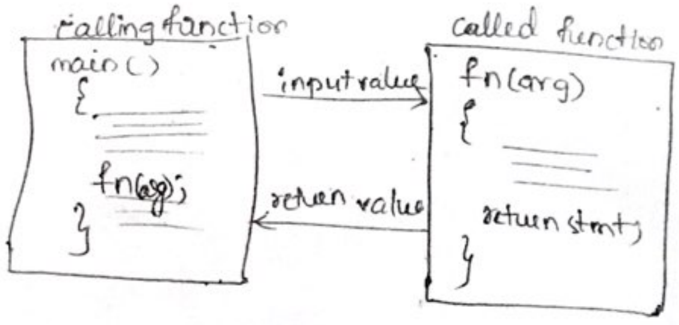


→ calling function calls or pass the data to called function but called function does not return value to calling function

eg program

```
#include<stdio.h>
#include <conio.h>
void main()
{
 void sum(int, int);
 int a,b;
 clrscr();
printf("\n enter a,b values");
scanf("%d %d",&a,&b);
  sum(a, b);
  getch();
}
void sum(int a, int b)
{
  int c;
  c = a+b;
  printf("sum=%d", c);
}
```

4) A function with argument and with return value



1 the calling function pass the data to called function and similarly called function return the value to the calling function

{program
{
```
    #include<stdio.h>
    # include<conio.h>
    void main()
    {
     int sum(int,int);
     int a,b;
     clrscr();
     printf("enter the value of a,b");
     scanf("%d %d",&a,&b);
    c=sum(a,b);
    printf("sum=%d", c);
    getch();
    }
    int sum(int a, int b);
    {
     int ~~c~~;e
     e=a+b;
     ~~print~~
     return(e);
    }
```

b = 6

d = 5

e = 6

f = 11

c=11

(8) Write a 'c' program to perform the operations (1) Sum, difference, multiplication, division using function

program

```
#include<stdio.h>
#include<conio.h>
void main()
{
  int sum(int,int);
  int diff (int,int);
  int mul (int,int);
  int div (int, int);
  int a,b,s,d,p,q;
  clrscr();
  printf("\n enter the value of a,b");
  scanf ("%d %d",&a,&b);
  s = sum (a,b);
  d = diff(a,b);
  p = mul(a,b);
  q = div (a,b);
  printf("sum=%d", s);
  printf("diff= %d", d);
  printf("mul= %d",p);
  printf("\ndiv=%d",q);
  getch();
}
int sum(int a, int b)
 {
  int s; [illegible]
  s=a+b;
  return(s);
 }
~~int diff~~
```

```
int diff (int a, int b)
{
    int d ;
    d = a-b ;
    return(d);
}
int mul (int a, int b)
{
    int p ;
    p = a*b;
   return (p);
}
int div (int a, int b)
{
   int q ;
    q = a/b ;
    return(q);
}
```

*Scope of a variable:

Scope of a variable determines over what region the variable value is available for use

→ depending on the place of declaration of a variable these are divided into 3 types

1) Global variable (public / external)
2) Local variable (private / internal)
3) Block variable

1) Global variable:- the variable which is declared outside of all the functions is called global variable the scope of global variable is available throughout the program (in all the functions)

eg program

```
# include<stdio.h>
# include <conio.h>
int  a=5 , b=10;
void main()
{
  clrscr();
  printf("\n a = %d", a);
  printf ("\n b=%d" b);
  getch();
}
```

output : a = 5
b=10

eg -2

```
# include<stdio.h>
#include<conio.h>
int a=5 , b=10;
void main()
{
   void fun();
   clrscr();
   printf("\n a= %d", a);
   printf("\n b = %d", b);
   fun();
   print f("\n %d", b);
   getch();
}
void fun( ).
 {
   printf("\n a= %d", a);
 }
```

Output a=5
b=10
a = 5
10

2) local variable : The variable which is declared inside a function is called local variable.
scope : the value of variable is available inside the function in which it is declared

eg: program

```
#include<stdio.h>
#include<conio.h>
void main()
{
  int a=10, b=20;
  clrscr();
  printf("\n a=%d",a)
  printf("\n b=%d",b)
  getch();
}
```

output
```
a=10
b=20
```

eg-2
```
#include<stdio.h>
#include<conio.h>
void main()
{
  void fun();
  int a=10, b=20;
  clrscr();
  printf("\n a=%d",a);
  printf("\n b=%d",b);
  fun();
  printf("\n %d",b);
  getch();
}
void fun()
{
  printf("\na=%d",a);
}
```

out put
```
a=10
b=20
[redacted]
b=20
```

3) Block variable: The variable which is declared inside a block ({ ,}) is called block variable

scope: the variable is availble inside the block in which It is declared.

eg: program

```
 #include<stdio.h>
 # include <conio.h>
 void main()
 {   int i=20;
     {
     int j=10;
     printf(" j=%d",j);
     }
 }
```

o/p: j=10

* If one variable is declared with same name in more than one block that variable value is available to that corresponding block.

```
eg: #include<stdio.h>
    # include <conio.h>
    void main()
    {
      {
        int i=10;
        printf("\n%d", i);
      }
      {
        int i=20;
        printf("\n%d",i);
      }
    getch();
    }
```

o/p: i:20

ex ~~#include~~

* if the variables are declared in multiple blocks then the variable in the ~~inner~~ most block is executed first.

eg:
```
void main()
{
    int i=4;
    {
      int i=5;
      {
        int i=6;
        clrscr();
        printf("\n i=%d", i);
      }
      printf("\n i=%d", i+1);
    }
    printf("\n i=%d", i-1);
    getch();
}
```

```
output  i=6
        i=6
        i=3
```

Week-6(a)-1

5) write a 'c' program to find out the factorial of a given number using function (non-recursive functi

program:
```
#include<stdio.h>
#include <conio.h>
void main()
{
  long fact(int), f;
  int n,i;
  clrscr();
  printf("\n enter the n value:");
  scanf("%d", &n);
  f=fact(n);
  printf("\n factorial of %d=%ld", n,f);
  getch();
}
```

```
long fact (int n)
{   int f=1;
    for (i=1; i<=n; i++)
      {
         f=f*i;
      }
    return(f);
}
```

n=5
1<=5(T) => f=1*1=1
2<=5(T) => f=1*2=2
(non recursive function)
3<=5(T) => f=2*3=6
4<=5(T) => f=6*4=24
5<=5(T) => f=24*5=120
6<=5(F)
But [illegible] functions we mention f=1

6(a) - 2

* Recursion: the process of defining something in terms of itself

Recursive function: A function ~~that~~ which calls itself again in the body of the function

```
eg: main( )
    {
        ________
        ________
        printf("\n this is an example of recursion");
        main();
    }
```

the above program causes to infinite loop to overcome this problem it is necessary to write if statement in every recursive function.

```
eg: #include<stdio.h>
    #include<conio.h>
    void main()
    {
        long fact (int),f;
        int n;
        clrscr();
        printf ("\n enter value of n");
        scanf ("%d", &n);
    f=fact(n);
    printf("\n factorial of %d=%ld", n,f);
   getch();
}
long fact(int n)
 {
    int f=1;                    n=5
    if(n==1)                    5* fact(4);
    return 1;                  5*4*fact(3);
    else                      5*4*3*fact(2);
     f=n* fact(n-1);         5*4*3*2*fact(1);
    return(f);               f =5*4*3*2*1
 }
```

* Storage classes

These are provided some information of the variable to the user

1) Where would be a variable is stored in memory or register

2) what is the lifetime of the variable (how long it exist)

3) what is the scope of the variable.

4) The initial or default value of the variable

There are four types of storage classes

1) Automatic ~~Store~~ variables

2) external variable

3) Static variable

4) Register variable

1) Automatic variables: These are represented by the keyword "auto", they are created when function is called and destroyed when function is exited automatically. The name auto

→ These are private or local to a function because of this property, these are also called as local or private or internal variables

~~eg program~~

1) Default value: Garbage value
2) Scope: inside the function in which it is declared.
3) lifetime: till the end of the called function
4) storage location: stack

eg program

```
void main()
{
    int a=10;                         output: a=20
    clrscr();
    {
     auto int a=20;
     printf("%d",a);
    }
    getch();
}
```

eg2

```
void main()                     → void fun()
{                                  {
 void fun();                        auto int a=20;
 int a=10;                          printf("\n%d",a);
 clrscr();                         }
 printf("\n%d",a);
 fun();                           output: 10
 printf("\n%d",a);                        20
 getch();                                 10
}
```

~~2) External Variable~~
eg:(3)

```
main( )
{
void fun( );
int a=10;
clrscr();
printf("%d",a);                    output   10
fun();                                      20
printf("%d",a);                             10
getch();
}
void fun()
{
~~auto~~ int a=20;
printf("%d",a);
a++;
}
```

2) External variables: The variables which are declared outside of all the function are external vaiable and represented by the keyword "extern", the value the vaiable is available throughout program.

eg: program

```
int a=10;
void main()
{
  int b=20;                        output: 10
  clrscr();                                20
  {
    int c=20;
    printf("\n %d",a);
  }
  printf("\n %d",b);
  getch();
}
```

eg-3)

```
extern int a=10;
    main()
    {
     void fun();
     clrscr();
     printf("\n%d",a);
     fun();
     printf("\n%d",a);
     getch();
    }
    void fun()
    {
     printf("\n%d", a);
     a++;
    }
```

```
output   10
         10
         11
```

1) default value = zero
2) scope : inside all the function (global) which are declared in the programe
3) lifetime : till the end of program
4) Storage location : memory

3) static storage class : The variables which are declared using static storage class are called static variables. These are initialized only once in the program
1) default value = zero
2) scope : inside the function (in b/w function calls) and throughout the program (Global.)
↓
say only global

3) lifetime: inside the function (local) and throughout the program

4) Storage location = memory

* Static variable should be initialized, if it is not initialized compiler automatically initializes with zero.

eg: program:
```
void main()
{
    inc();      → function declaration
    clrscr();     Here, there is no
    inc();        return type is used.
    inc();
    inc();                      output :: 15
    getch();                              20
}                                         25
inc()
{
    static int x=10;
    x= x+5;
    printf("\n%d", x);
}
```

eg-②
```
main()
{
    void check();
    clrscr();
    check();
    check();        Output:    5
    check();
    getch();                   10
}                              15
void check()
{
    static int c=0;
    c=c+5;
    printf("\n%d", c);
}
```

eg: main()

```
{
void check();
clrscr();
check();
check();
check();
getch();
}
void ~~auto~~ check()
{
 auto int c=0;
 c=c+5;
 printf("\n%d", c);
}
```

Output:

```
5
5
5
```

4) Register variable : The variables which are declared with register storage class are called register variables. and represented by keyword register

eg: register int a=10;

1) default value = 0
2) scope : inside a function
3) lifetime : till the end of the function
4) Storage location : CPU registers

→ The difference b/w register variable and other variables is storage location. CPU registers are allocated for register variables where as for all remaining variables the space is allocated in memory.

The value is stored in register is much faster than the value stored in memory.

→ By using register variables we can decrease acc
time.
eg: loop variables and index.

* Arrays and functions :-

There are two ways to pass an array as argume
the funct

1) Sending an entire array as an argument to a func
2) Sending individual elements of an array as argument to the function

1) sending an entire array as an argument :-

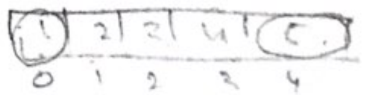
eg:
```
void main( )
{
 void  display(int [ ]);
 int a[5], i ;
 clrscr ( );
 printf("\n enter array elements ");
 for (i=0 ; i<5 ; i++)
  {
   scanf("%d", &a[i]);
  }
 display (a); // → sending entrie array.
 getch( );
}
void display(int a[5])
{
  int i;
  printf("\n array elements are");
  for (i=0; i<5 ;i++)
  { printf("%d\t", a[i]);
  }
}
```

2) Sending individual elements as argument to functions:

if we want to send individual elements as arguments then send the ~~array~~ elements along with its subscript

program

```
void main()
{
  void display(int,int)
  int a[5], i,a,b,
  clrscr();
  printf("\n enter array elements");
  for(i=0; i<5; i++)
  {
   scanf("%d", &a[i]);
  }
  display(a[0], a[4]);
  getch();
}
void display(int a, int b)
 {
  printf("first element is %d", a);
  printf("last element is %d", b);
 }
```



*** Headers files: (or) Standard functions.**

→ The files which contains pre-defined or standards (or) inbuilt functions are called headers file

eg: stdio.h
conio.h
math.h
stdlib.h
ctype.h
string.h
alloc.h
process.h