→ "main()" function is a user defined function.

* Function -
Function is a group of statement that together perform a task. Every 'C' program has at least one function, which is 'main()'

* There are two types of function -
i) Library function (Predefined function).
ii) User defined function

i) Library function -
Those function which come along with the compiler and are present in the disk. These function can't be modified, it can only read and can be used.

* Every ~~program~~ library function has a header file.

* There are a total of 15 header file in 'c'.

* Printf(), scanf(), clrscr(), strcpy() etc are example of library function.

ii) User defined function -
The user defined functions defined by the user according to its requirment.
syntax -
return type name of fn. (data type)

Date ____
Page ____

* Elements of user defined function-

i) Function declaration - (Prototype)

function-return data type   function name User word   argument or parameter data type

Eg:- int sum(int a, int b);

A function declaration tells the compiler about three things-

a) name of the function
b) number and type of argument received by the func.
c) and the type of value returned by the function

* The function declaration always terminated by the semicolon.

ii) Function definition -

It consists of the whole description and code of the function and it tells about what function is doing, what are its input and what are its output.

→ It is made up of two parts-
a) Function header (without semicolon)
b) Function body

Syntax -

```
return type  function name(arguments)   } Function
                                          header
{
    . . . ____
    - . . . .
    - . . .                  /* Function body */
    return value;
}
```

Date
Page

* The return type denotes the type of the value that function will return and it is optional and if it is omitted, it is assumed to be 'int' by default.
* All function by default return int.

* The argument of function definition are known as formal arguments.

iii) Function call-
When the function get called by the calling function then that is called function call.

* The argument that are used inside the function call are called actual argument. These are the original value.

→ These three things are represented like-

```
#include <stdio.h>
    int fun( int, int, int);   // function declaration
    void main()  // calling function
       {
         statements;
         fun(arg1, arg2, arg3);   // fuction call
       }
    int fun(int1 , int2 , int3)  // function definition
      {
         local variable declaration;
          statments;
         return value;
      }
```

Date
Page

* Summation of two values using function -

```
# include < stdio.h>
 int sum (int a1, int a2);
 ~~m~~void main()
 {
   int a, b;
  printf ("Enter two numbers");
  scanf (" %d %d", &a, &b);
  int s = sum(a, b);
  printf (" Sum is :" %d", s);
 }
 int sum (int x, int y)
   {
     int z = x + y ~~f~~;
     return z;
   }
```

* Program to find value of a power number -

```
# include <stdio.h>
~~void main()~~  int power (int, int);
          void main ()
          { int  n, p, ans;
          printf (" Enter no. and its power");
          scanf (" %d %d" &n, &p);
            ans = power (n, p);
            print (" %d", ans);
          }
          int power (int n, int p) -
          {   int ans = 1, i;
          for (i = 1; i < = p; i++)
                ans = ans * n;
            return (ans);
          }
```

Date
Page

* Category of function based on argument and return

i) Function with no argument and no return value.

syntax -

```
void functionname (void);
```

Eg:- 1)
```
# include <stdio.h>
void sum (void);
int main()
  {
     sum () ;

  }
void sum()
  { int a = 5 , b = 7 ;
      s = a+b ;
     printf (" S= %d", s);
  }
```

2)
```
# include <stdio.h>
void AK();
int main()
 {
   AK();
  printf ("in main");
 }
void Ak ()
  { printf (" Come on");

  }
```

Output- Come on

ii) Function with no argument but return -

syntax -

int ~~na~~ function name (void);

Eg:- 1)

```
#include <stdio.h>
int fun(void);
int main()
 {  int r; ~~su~~
    r = fun();
 }
int fun()
  {
   return (exp);
  }
```

"Here called function is independent and are initialized. The value aren't passed by the calling fun."

ii)

```
#include <stdio.h>
  int sum(void);
{ void main()
    int s;
     s = sum();
  printf("sum=%d", s);
}
int sum()
  {
   int a,b, sum=0;
   ~~s = a+b~~
   ~~return~~
   printf("Enter a & b");
    scanf("%d %d",&a,&b);
     sum = a+b;
     return sum;
  }
```

Date ______
Page ______

iii) Function with argument but no return value :-

syntax

    void function name (int, int);

Eg:-

```
# include <stdio.h>
void ~~main~~ fun( int, int );
int main()
  {
    int (a, b);
  }
void fun( int x, int y);
  {
      statements ;
  }
```

"Here the function has argum
so the calling function send
data to the called fn.
but called fn. doesn't
return value"

*

```
# include <stdio.h>
void fun( int, int[], char[] );
int main()
  {
    int a = 20;
   int ar[5] = { 10, 20, 30, 40, 50 };
    char str[30] = {"Beingpro"};
     fun( a, & ar[0] , &str[0] );
  }
 void fun ( int a,  int * ar, char * str )
     { int i ;
      printf (" Value of a is %d \n", a);
     for ( i = 0;  i < 5 ;  i++)
        {
         printf (" Value of ar[%d] is %d \n", i, ar[i]);
        }
     printf ("\n Value of str is %s \n", str);
  }
```

Date
Page

iv) Function with argument and return value -

syntax

int functionname (int , int) ;

(Here the calling fn. has the argument to pass the called function and the called function returned value to the calling fn.)

Eg:-

```
#include <stdio.h>
int fun(int);
void main()
{
    int a, num;
    printf (" Enter value\n");
    scanf (" %d", &a);
    num = fun(a);
}
int fun ( int x )
 {
    ++x;
    return x;
 }
```

Date ____
Page ____

* **Call by value and call by reference -**

These are two way through which we can pass the arguments to the function such as call by value and call by reference.

i) Call by value -
In the call by value copy of the actual argument is passed to the formal argument and the operation is done on formal argument.

* When the function is called by 'call by value', method, it doesn't affect content of the actual argument.

* Changes made to formal argument are local to block of called function so when the control back to calling function the changes made is vanish.

Eg:-
```
#include <stdio.h>
void fun(int, int);
void main()
{ int x=5, y=7;
    fun(x,y);
  printf("x=%d,y=%d", x, y);
void fun(int x, int y)
  {
    x=7;
    y=5;
  printf(x=%d, y=%d", x,y)
  }
}
```

output
```
x=7, y=5
x=5, y=7
```

ii) Call by reference -(Address)

In the case of call by reference, we passed the address or reference of the variable instead of passing the value of variable. So, function operate on address of the variable rather than value.

* In 'call by reference' method, the formal argument is alter to the actual argument, ~~it means~~

Eg :-

```
#include <stdio.h>
void fun(int*, int*);
void main();
{ int x=5, y=7;
   fun(&x,&y);
   printf("x=%d, y=%d", x, y);
}
void fun(int *x, int *y)
   { *x = 7;
     *y = 5;
   printf("x=%d, y=%d",*x, *y);
}
```

output

```
x=7, y=5
x=7, y=5
```

Date
Page

* **Recursion**-
The process of calling a function by itself again and again is called recursion and that function calls itself is called recursive function.

* In recursion calling function and called function are same.

Eg:-
```
# include <stdio.h>
void display (int n)
  { if(n<1)
      return;
    else
     { ~~printf("%d", n);~~
        display (n-1);
      printf ("%d", n);
     }
  }
  void main()
    { int n=3;
      display(n);
    }
```

output - 1, 2, 3

Q

```
# include <stdio.h>
int sum (intx)
{ int s = 0;
   if (x == 1)
      return x;
   else
   S = x + sum(x-1);
      return s;
void main()
{ int a;

   a = sum(5);
   printf("%d", a);
```

output

```
5+4+3+2+1
= 15
```

Q Factorial of a no. using recursive function -

```
#include <stdio.h>
int fact ( int n);
void main()
{ int n,f;
  printf("\n Input a no = ");
   scanf ("%d", &n);
    f = fact(n);
 printf ("\n factorial of %d = %d", n, f);
}
int fact(int n)
 { int f;
   if (n == 1)
    return 1;
   else
    {
     f = n * fact(n-1);
       return(f);
    }
}
```

Date
Page

2nd method using recursion

```
# include <stdio.h>
void fact (int n, int f)   // function
                              definition
  { if (n>=1)
       { f=f*n;
           n--;
         fact (n,f);
       }
     else
       printf(" Factorial= %d",f);
  }
int main ()
 {      int n;
      printf("Enter any number");
        scanf("%d",&n);
         fact (n,1);       // function calling
 }
```

3rd method
Using ~~direct~~ recursion

```
# include <stdio.h>
int fun1 (int);
int fun 2( int);
void main()
  { printf("%d, fun1(5));
  }
int fun1(int n)
  { if(n<=1) return 1;
  else
     return n*fun2(n-1);
  }
int fun2 (int n)
  { if (n<=1) return 1;
  else
     return n*fun1(n-1);
  }
```

Date ____
Page ____

* Passing array to function -
In this type of function, there is an array in the place of parameter and its value is passed at the time of calling.

Eg:-
```
# include <stdio.h>
void ~~main~~ sum(int ar[5])   // function definition
   { int  s = 0 ;
     for (int i = 0; i < 5 ; i++ );    // sum of array
        s = s + ar[i];
     printf(" Total sum of element = %d", s);
   }
 int main()
   { int x[5] = {10, 20, 30, 40, 50};
        sum(x);          // function calling
   }
```

i)
```
#include <stdio.h>
 float largest (float a[], int n);
 int main()
   {
     float value [4] = { 2.5, -4, 1.2, 3.67 };
     printf("%f \n", largest (value, 4) );
   }
 float largest(float a[], int n)
   {  int i;
      float max;
      max = a[0];
     for (i = 1; i < n ; i++)
        if (max < = a[i])
          max = a[i];
   }  return (max);
```