*

4. Functions

Syntax :-

```
returntype <fun name>(datatype arg1, datatype arg..
{
  local variable declaration
  - - - - - - - - -
  - - - - - - - - - - -
  - - - - - - - - - - -
  [return variable(value)],
}
```

① Function defination
② Function call
③ Function Prototype
→ These are must and should.

→ There are '4 ways' to implement 'functions'.

① "Function with argument & with returntype."
② "Function without argument & with returntype"
③ "Function with argument & without returntype"
④ "Function without argument & without returntype"

* Example program;

```
void sum( ) → Function Definition.
{
  int a, b, sum=0;
  printf("Enter two numbers:");
  scanf("%d %d", &a, &b);
  sum=a+b;
  printf("sum=%d\n", sum);
}
void main( )
{
  sum( ),   } Function call
  sum( ),   }
  printf("Hello");
  sum( ),   → Function call
```

* **Advantages of Functions:-**

(1) Reusability.

(2) It's easy to debug the program.

(3) Better Memory utilization.

(4) Reduce the complexity of program.

1] Function with argument & with returntype :-

```
void main ( )
{
   int a, b, s;
   printf ("Enter two integers \n");
   scanf ("%d %d", &a, &b);
  s=addition (a,b).
   printf ("sum of a&b = %d," s);
}
int addition (int x, int y)
{
   return x+y;
}
```

3] Function without returntype & with argument :-

```
void main( )
{
  int a, b, s;
  printf ("Enter two integers \n");
  scanf ("%d %d", &a, &b);
   ~~s=~~ substraction (a,b);
}
void substraction (int p, int q)
{
  printf ("substraction = %d", p-q);
}
```

3] Function without argument & with returntype :-

```
Void main( )
{
  int a,b;
  printf("multiplication =%d", multi( ));
}

int multi( )
{
  int x,y;
  printf("Enter two integers for mult \n");
  scanf("%d %d", &x, &y);
  return x*y;
}
```

4] Function without argument & without returntype:-

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

void division( )
{
  int a,b;
  printf("Enter two integers \n");
  scanf("%d %d", &a, &b);
  printf("division= %f", a/b);
}
```

* **Note:-**

→ Protocols from 4 examples :-

```
int addition (int, int);
void substraction (int, int);
int multi ( );
void division( );
```

* Recursion :-

→ A Function which calls itself again and again based on condition is called 'Recursion'.

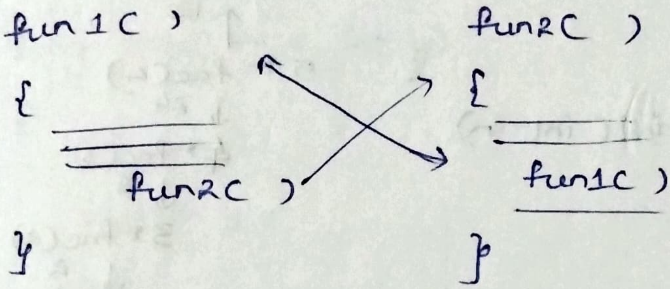
→ Here the condition is called 'Base condition'.

→ There are 2 types of Recursions.

① 'Indirect Recursion.'

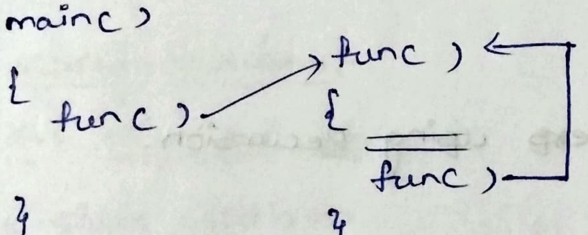
② 'Direct Recursion.'

① Indirect Recursion :-



```
fun1( )                    fun2( )
{                          {
  ______                     ______
  ______                     fun1( )
  fun2( )                    ______
}                          }
```

② Direct Recursion :-



```
main( )          func( )
{                {
  func( )          ______
}                  func( )
                 }
```

* Advantages of Recursion :-

① It is one type of Technic.

② Recursion is ~~veryful~~ very useful in the Data structures.

* Disadvantages :-

① It needs to maintain & follow the stack.

② It is more complex than while & for loops.

③ Program overhead occurs.

* Ex:- Write a program to find factorial of a number using Recursion.

A) Factorial :- n! →

5! → 5×4×3×2×1.

```
int main( )
{
    int num;
    printf("Enter a no to find factorial : ");
    scanf(" %d", &num);
    printf(" factorial = %ld", factorial(num));
}

long int factorial( int n)
{  if (n ==0)
      return 1;
   else
      return   n× factorial (n-1).
}
```

5
↑ 120
5 * fac(4)
↓ 24
~~4~~ * fac(3)
↓ 6
3 * fac(2)
↓ 2
~~2~~ * fac(1)
↓ 1
1 * fac(0)
1

* Ex:- sum of numbe~~rs~~ using Recursion.

```
A) int main( )
{
    int num;
    printf(" Enter a number: ");
    scanf (" %d", &num);
    printf (" sum of numbers = %d", sum(num));
}
int sum (int n)
{
   if (n==0)
     return n;
   else
     return n+sum(n-1);
}
```

5
↑ 10
5 + sum(4)
↓ 10
4 + sum(3)
6
↓
3 + sum(2)
↓ 3
2 + sum(1);
↓ 1
1 + sum(0)
0

* Ex:- power of a number [x^y]:-

A)
```
int main( )
{ int b, p;
  printf("Enter base no=");
  scanf("%d", &b);
  printf("Enter power value=");
  scanf("%d", &p);
  printf(Exp(b,p));
         32
}

long exp( int x, int y)
              2      2
{
  if (y==0)
    return 1;
  else
    return x * exp(x, y-1);
           2     (2, 4)
}
```

```
2* exp(2,4)
   ↓    16
  2* exp(2,3)
      ↓    8
    2*exp(2,2)
       ↓   4
      2*exp(2,1)
         ↓  2
        2*exp(2,0)
               1
```

* ~~storage classes~~:-

* Ex:- Fibonacci series :-

A)
```
#include <stdio.h>
int main( )
{
  int n, i, r;
  printf("Enter a no :");
  scanf("%d", &n);
  for (i=0; i<n; i++)
  {
    printf(fib(i));
  }
}
int fib(int a)
{
  if (a==0)
     return 0;
  else if (a==1)
     return 1;
  else
     return fib(a-2)+fib(a-1); }
```

0, 1, 1, 2, 3, 5, 8__

→ n=10

0<10 return 0

1<10 return 1

2<10 (2-2)+(2-1)

0+1 =1

3<10 (3-2)+(3-1)

fib(1)+fib(2)=3

fib(1)+0

* Ex:- G.C.D of a Number.

A)
```
#include <stdio.h>
int main( )
{
  int x,y;
  printf("Enter a no:");
  scanf("%d", x);
  printf("Enter a no:");
  scanf("%d",y);
  if (x>y).
  pritf("x is dividend");          du=x; dr=y;
  else
   printf("y is dividend");        du=y; dr=x;
  gcd (x,y);
}
int gcd(int x, inty)
{
  int r=x%y;
  if (r==0)
     return y;
  else
     return gcd(y,r);
}.
```

* storage classes:-

→ Indicates capability & scope of the program [variable] & Life of the variable

→ Four types.

1. Auto
2. Extern
3. static
4. Register.

*

storage class	Scope	Default value	Memory Allocation	Lifetime (Access).
① Automatic variables	With in Block	Garbage values	In the RAM	Block.
② Extern [Global variables]	Out of the file	zero	In the RAM	Entire program.
③ static	With in Block	zero	In the RAM	Entire program.
④ Register.	local with in Block	Garbage values	C.P.U.	Block.

* Pre Processor Directives :-

→ It is used as text replacement tool.

→ Always starts with '#'

→ It reduces the complexity of program.

Ex :- # include

define

# undef

if def

ifndef

if

else

elif

endif

# pragma

err

include :-

Ex :- ① # include is used as Headerfile, #include <stdio.h>

② We can also use like this #include "First.c".

#define :-

Ex :- ① #define · SIZE 50

int arr[SIZE].

② # define PRINT Printf("Hai Welcome");

PRINT.

```
③ # define AREA(a)  (3.1415 * a * a).
  void main( )
  {
      float r=3.2, area;
      area = AREA(r);
  }
```

Ex:- #undef:-

```
① #undef NULL
   # define NULL 0
```

#if def :-

```
Ex:- ~~##~~ if def NULL   — True
      # undef NULL
      # define NULL 0
    # endif
```

#elif :-

→ For multiple conditions.

pragma :-

→ It is a 'speeial purpose Direetive' & used to turn on or off some features.