Function (An independent, reusable module)

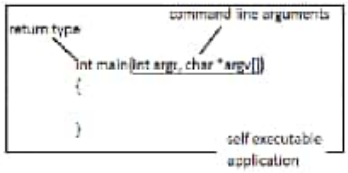
- The function allow us to form the **module** and which **attachable detachable with any application.**
- **Function is self- contained (independent), reusable block of code that may takes parameters and may return a value.**
- **It is not application specific and can be attach with any application as required.**
- Any application developer takes **services of functions (module) any number of times.**
- Benefits of function
  - **Faster development.**
  - **Several programmers can work on individual programs at the same time.**
  - **Easy debugging and maintenance.**
  - **Easy to understand as each module works independently towards a goal.**
  - **Less code has to be written.**
  - **The scoping of variables can easily be controlled.**
  - **Modules can be re-used, eliminating the need to retype the code many times**

- **Functions are two type.**
  - **Library function**
    - These are those functions that are **precompile** such **sqrt, pow, clrscr, getch, printf**, **scanf** etc. the technology have more **than 5000 library functions**.
  - **User define function:**
    - The **C** technology allow us to create function according to need of application for application developer.
    - Syntax;

      [Return Type] FunctionName([prameter lis
      {
      Statements;
      }

e.g.

e.g.



Write a user define function that takes integer parameter and return integer.

```
//file name is square.c
int sqr(int n) // definition of function
{
        int s = n * n;
        return s;
}
```

```
//executable application named test.c
#include "square.c"
main()
{
        int num , s ;
        printf("\nEnter a number");
        scanf("%d",&num);
        s = sqr(num);
        printf("\nsquare = %d",s);
}
```

- A C/C++ program is collection of functions but program execution begin **main()** function.
- **Once a function is defined it can be call any number time.**
- **Return type is optional** to use, if we do not specify it then technology specify as **int.**

  e.g.

```
//square.c
sqr(int n) // definition of function
{
    int s = n * n;
    return s;
}
```

- **Parameter types of function are optional** and if we do not provide it technology make is **int**

  e.g.

```
//square.c
int sqr(n) // definition of function
{
    int s = n * n;
    return s;
}
```

- **Parameter**
  - Two functions can communicate with each other by sharing their local information.
  - Function sharing their local information by using the concept parameter.
  - It is often known as argument.
  - There are two type of parameters
    - **Actual parameter**
      - **Actual parameters are those they sending the time of calling the function.**

```
//executable application named test.c
#include "sqr.c"
main()
{
   int num , sqr ;
   printf("\nEnter a number");
   scanf("%d",&num);
   sqr=square(num);
   printf("\nsquare = %d",sqr);
}
```

actual parameter

- **formal parameter**
  - **Formal parameters define while defining the function.**

```
//square.c
int sqr(int n) // definition of function
{
    int s = n * n;
    return s;
}
```

Formal parameter

- **Function prototype**
  - It is declaration statement that declares information to compiler so compiler makes translation possible.
  - It tells about function name, their parameters list and return type of function.

    float square(float n); // function prototype
  - If function is define after function call then have to place function prototype before function call so compiler binds function definition with function call accordingly but in case of integer or char type parameter it is not required in C language.

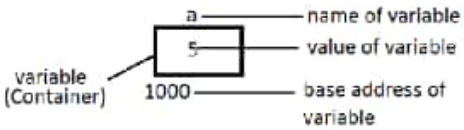
    e.g.

```
main()
{
        float square(float n); // function prototype
        float  num , sqr ;
        printf("\nEnter a number");
        scanf("%f",&num);
        sqr=square(num); // function call
        printf("\nsquare = %f",sqr);
}
float square(float n) // definition of function
{
        float s = n * n;
        return s;
}
```


- If function is define in a program after function call there is no need to place function call before calling.

Pointer:

- It is a special variable that allow us to store address of element of "C/C++" language such variable, function, array, structure, union, file, pointer etc.
- When we want to perform remote work on the element of "C/ C++" element then and then only we pointer.
- Several kind of pointers we in the industry
  - Pointer
  - Pointer to pointer …
  - Pointer to function or Function pointer
  - Pointer to array
  - Array of pointers
  - Pointer to structure
  - Pointer to union
  - NULL pointer
  - Dangling pointer
  - Generic pointer
  - Smart pointer etc.



e.g.

int a = 5;

It tells the compiler to

1. reserve the memory for int type variable
2. assign the name "a" to it by making the entry in symbol table.
3. assign 5 value into it.

- Sometimes it is required to store base address of the element such as variable etc. then we create pointer variable.
  - Syntax:
    <data type> *ptr;

e.g.

int *ptr;
float *ptr;

- If you want to store address of "X" type variable then we have to create pointer of type "X".

e.g.

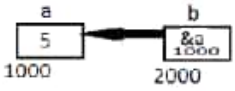
int a = 5 , b ;

b = &a; — b is not capable to store address of variable becuase b is not defined as pointer

e.g.

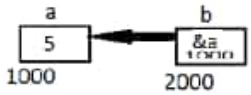
int a = 5 , * b;

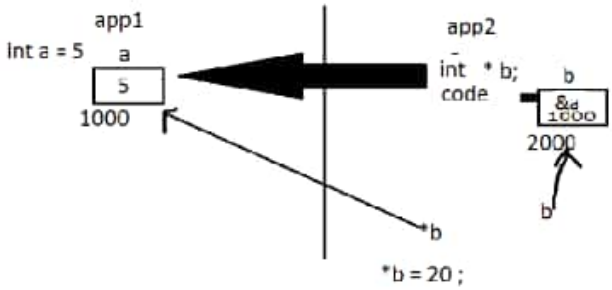
b= &a; OK



- Indirection Operator (*) :
  - When we want to perform any action on that element which is pointed by the pointer use indirection operator.

e.g.
int a = 5 , * b;
b= &a; OK





e.g.

```
void pointer_test(float *b)
{
        ++*b;
}

main()
{
        float a = 5 ;
        pointer_test(&a);
        printf("\na = %f",a);

}
```

Define a function that takes radius as parameter & calculate area and circumference of circle and both return to the caller of the function.

```
void area_cir(float radius, float *a, float *c)
{
    *a = 3.14 * radius * radius;
    *c = 2 * 3.14 * radius;
}

main()
{
    float r, area , cir;
    printf("\nEnter radius");
    scanf("%f",&r);
    area_cir(r, &area, &cir);
    printf("\narea = %f",area);
    printf("\ncircumference = %f",cir);
}
```

formal parameter

radius	*a	*c
1.0	&area 2000	&cir 3000

actual paramters

r	area	cir
1.0	[illegible]	[illegible]
1000	2000	3000

```
Enter radius
1.0
area = 3.14
circumference = 6.24
```

Difference between call by value and call by address

1. In call by value, if we made any change in the formal parameter actual parameter does not affect whereas in call by address, if we made any change in formal parameter (pointer based parameters) will affect actual parameter.
2. While passing value we can use name of variable or constant. In call by address we cannot use constant value.
3. In call by value, Xerox of value will be send but in call by address, the address of element will be sending which is unsafe.