

# Unit 2

## Topic: Function

# Functions: Definition & Advantages

A function is a **block of code** defined to perform a particular task. {.....}

**C has 2 categories of function:**

1. Library functions

2. User defined functions

There are many **advantages** of using functions:

Length of the source program is reduced.

Easy to locate and debug errors.

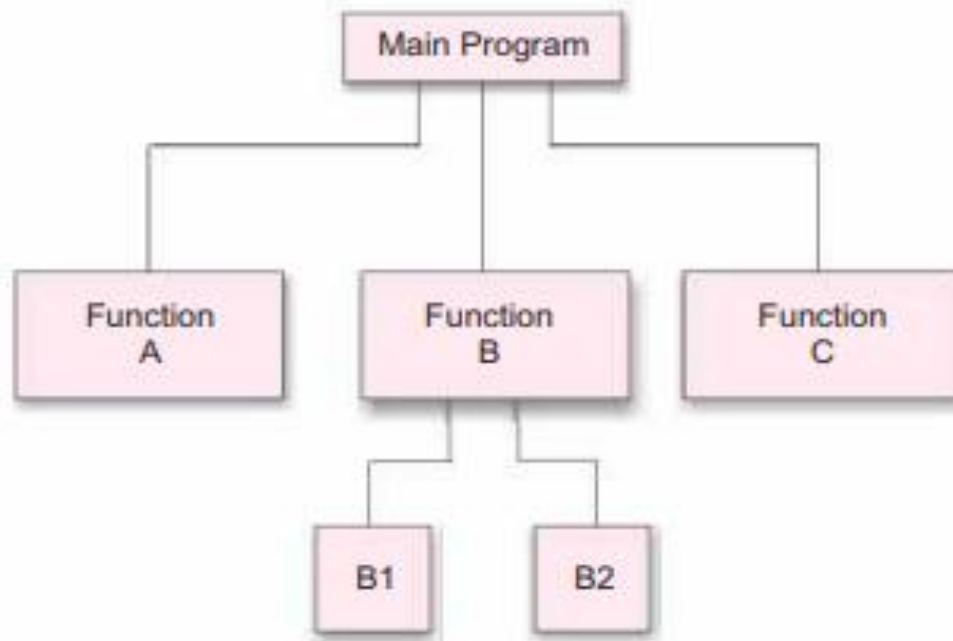
Saves debugging time

A function can be used by many other programs.

It facilitates top to down modular approach.

Promotes reusability of code.

# Top –down modular approach



**Fig. 9.1** *Top-down modular programming using functions*

# Multi Function Program Structure

```
void main()
```

```
{ .....  
  function 1(); // fun call  
  .....function 2();  
  ..... function 1();  
  ...}
```

```
void function 2()
```

```
{  
  .....  
  function 3();  
  .....  
}
```

```
...}
```

```
void function 2()
```

```
{
```

```
.....
```

```
function 3();
```

```
.....
```

```
}
```

```
void function 1()
```

```
{ ..... }
```

```
void function 3() // function definition
```

```
{ ..... }
```

# Tips about function

- A C program is a collection of one or more functions.
- If a C program contains only one function, it must be `main()` .
- If a C program contains more than one function, then one (and only one) of these functions must be `main()` .
- There is no limit on the number of functions that might be present in a C program.
- Each function in a program is called in the sequence specified by the function calls in `main()` .
- After each function has done its thing, control returns to `main()` . When `main()` runs out of statements and function calls, the program ends.

# Tips about function

- (a) Program execution always begins with `main()` . Except for this fact, all C functions enjoy a state of perfect equality. No precedence, no priorities, nobody is nobody's boss.
- (b) Program execution always begins with `main()`. Every function gets called directly or indirectly from `main()` . In other words, the `main()` function drives other functions.
- (c) Any function can be called from any other function. Even `main()` can be called from other functions.
- (d) A function can be called any number of times.
- (e) The order in which the functions are defined in a program and the order in which they get called need not necessarily be same.
- (f) A function cannot be defined in another function.

# Elements of User Defined Functions

**Function Declaration**

**Function Call**

**Function Definition      // body define**

**Function definition includes the following:**

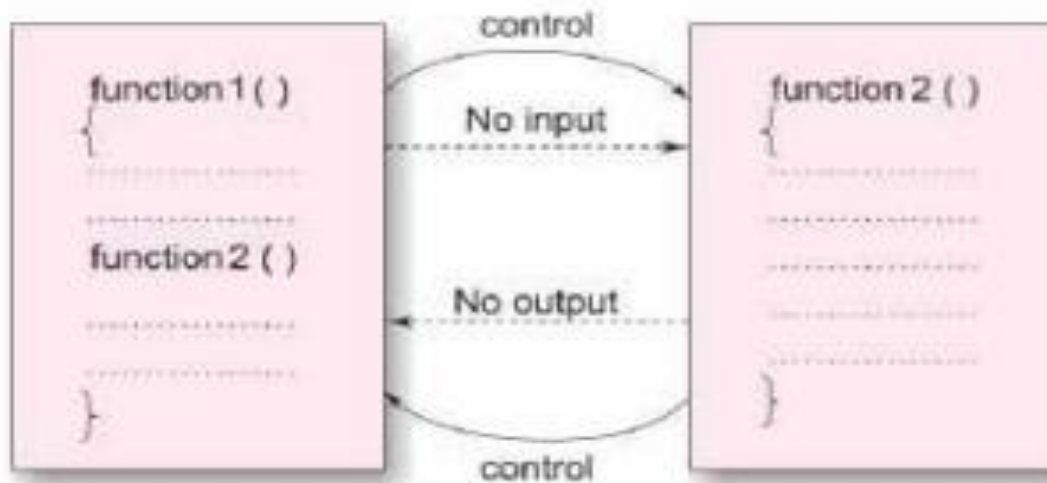
- 1. Function name
  - 2. Function Type
  - 3. List of Parameters
  - 4. Local Variable declarations
  - 5. Function statements
  - 6. A return statement
- Function header
- Function Body

## Type of function

1. Function with no argument and no return
2. Function with argument and no return
3. Function with argument and return
4. Function with no argument and return



# Two way data communication between function



# Function with no arguments and no return value

```
#include<stdio.h>

void add();                      // function declaration

void main()
{
    add();                      //function call
}

void add()
{
    int a,b,c;
    printf("enter numbers to be added");
    scanf("%d %d", &a, &b); // 10 34
    c=a+b; // 10 ,34
    printf("result=%d",c);
}
```

//function definition

## Function with Arguments and no return value

```
void add(int c,int d);           // function declaration
void main()                      //function arguments / parameters
{
    int a,b;
    printf("enter numbers to be added");
    scanf("%d %d", &a, &b); // 10 34
    add(a,b); // function call ,add(10,34) // a and b Actual Argument
}
void add(int c, int d) // (10,34) // c and d Formal Agrument
{
    int e;

    e=c+d; // 10 ,34
    printf("result=%d",e);
}
```

## Tips

1. Type , order and number of actual and formal arguments must always be same.
2. It is not compulsory to use variable names in the prototype declaration.  
void sum (int , int);
3. No separate return statement was necessary to send control back
4. If the value of formal argument changed in the function the corresponding change does not take place in the calling function

# Function with Argument & Return Statement

```
float add(float,float);
```

// function declaration

```
void main()
```

```
{
```

```
    float a,b,s;
```

```
    printf("enter numbers to be added");
```

```
    scanf("%f %f", &a, &b); // 10 34
```

```
    s=add(a,b); // function call ,add(10,34) a,b actual argument
```

```
    printf("Sum=%f",s);
```

```
}
```

```
float add(float c, float d) // (10,34)
```

//return type of a function //formal

```
{
```

```
float e;
```

```
e=a+b; // 10 ,34
```

```
return e;
```

```
}
```

add function definition

main  
function  
definition

## return statement

1. On executing the return statement , it immediately transfer the control back to the calling function
2. It returns the value present in the parentheses after return to the calling function
3. There is no restriction on the number of return statements in a function but A function can return only one value at a time. For eg: to find largest of two numbers

## Function with no argument with return

```
int get_num(void);  
void main()  
{  
    int m=get_num();  
    printf("%d",m);  
}  
int get_num(void)  
{  
    int n;  
    scanf("%d",&n);  
    return(n);  
}
```

## Question

- 1)WAP to calculate simple interest using functions.
- 2)WAP to check whether a number is an Armstrong number or not using functions with return value.
- 3)WAP to calculate factorial of a number using functions with return statement.



## Recursion: Definition & Conditions

Recursion is a process in which a function **calls itself**.


Necessary Conditions for recursion:

**Base case / Stopping condition**

**Call to itself**

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

```
void fun_a()           // recursive fun  
{  
    .....  
    fun_a();  
}
```



## Recursion example

```
factorial(int n)
{
    int fact;
    if (n==1)
        return(1);
    else
        fact = n*factorial(n-1);
    return(fact);
}
```

## Question

WAP to print the terms of Fibonacci series using recursion.

## Call by Value Vs Call by reference

In call by value, the **values** of the function arguments are passed during function call while in call by reference, the **reference/address** of the function arguments are passed during function call.

In call by value, changes are made to the **local copies** of the arguments, while in call by reference, **changes are made at the address values**.

In call by value, old values are retained. In call by reference, old values are updated with new values.

In call by value, changes are temporary but in call by reference, changes made are permanent.

## Example of Call by Reference

Output: 30

```
main()
{
    int x;
    x = 20;
    change(&x); /* call by reference or address */
    printf("%d\n",x);
}
change(int *p)
{
    *p = *p + 10;
}
```

## Swapping of two numbers using call by value & call by reference

```
void swapv (int x, int y);

int main()

{

int a = 10, b = 20;

swapv (a, b);

printf ("a = %d b = %d\n", a, b);

return 0;

}

void swapv (int x, int y)

{

int t;

t = x;

x = y;

y = t;

printf ("x = %d y = %d\n", x, y);

}
```

---

```
void swapr (int *, int *);

int main()

{

int a = 10, b = 20;

swapr (&a, &b);

printf ("a = %d b = %d\n", a, b);

return 0;

}

void swapr (int *x, int *y)

{

int t;

t = *x;

*x = *y;

*y = t;

}
```

## Return more than one value

```
void areaperi (int, float *, float *);

int main()

{

int radius;

float area, perimeter;

printf ("Enter radius of a circle ");

scanf ("%d", &radius);

areaperi (radius, &area, &perimeter);

printf ("Area = %f\n", area);

printf ("Perimeter = %f\n", perimeter);

return 0;

}

void areaperi (int r, float *a, float *p)

{

*a = 3.14 * r * r;

*p = 2 * 3.14 * r;

}
```

## Question

Q Write a function that receives marks of 3 subject and returns average and percentage of these marks



END