Unit 7: Functions

- A function is a subprogram which performs a well-defined task.
- Every C program is a collection of functions.
- There should be at least one function called main( ).
- Every program execution starts with main( ) & main( ) is only one user define function known by Operating System.
- C supports two types of functions
    1. Built-in Library Functions
    2. User Define Functions

1. **Built-in Library Functions:-**
   These functions are provided by C.
   Examples are, printf(), scanf(), pow(), sqrt(), strlen(), puts(),clrscr() etc.
2. **User define functions:-**
   Programmer can define their own functions.

Advantages of functions:-

- Program maintenance & debugging is easy.
- Reusability of code.
- Reduces program development time.
- Work distribution is possible.
- Easy to understand & easy to write.
- Reduces coding size.

User Define Functions:-

- User can define their own functions.

Basic Parts/Components/Elements of functions:-

1. Function Prototype Or Function Declaration.
2. Function Calling.
3. Function Definition.

1. **Function Prototype or Declaration.**
   - Like variables functions are also declared called function prototype.
   - Function should be declared before they are used.
   - We can declare function inside main() or above the main().

Function declaration or prototype is used to tell compiler about following:

1. Return type of function.
2. Number of parameters or arguments.
3. Type of parameters.
4. Name of the function.

Syntax:-

return_type (datatype arg1, datatype arg2, ……..datatype argn);

In above Syntax, we can also eliminates names of arguments see example 2.

Example:-

1. int sum(int a, int b);
2. int sum(int, int);

2. Function Calling.

- Function is called when a function name is followed by a semi comma.
- Whenever function is called, actual parameters of calling functions are copied into formal parameters of called function(if any).
- Number of actual and formal parameters should be same(if any).
- Name of actual and formal parameters may be same or different(if any).
- We can call functions using(also called Parameter Passing)
  i. Call by Value.
  ii. Call by reference.

Syntax:-

function_name(parameter_list);

- Whenever function is called return type is not required.

Example:-

1. add(a,b);
2. add(&a, &b);
   Example 1. Shows it is call by value & 2 is call by reference.

3. Function Definition.

- User define functions definition consists of following two parts.
  i. Function Header.
     - It contains 'return type', 'function name' & 'argument list'
     - Function header is not terminated by a semi colon.
  ii. Function Body
     - Function body should be enclosed in curly braces '{' & '}'.
     - It contains executable statements along with return value, if any.
     - These executable statements performs well define task.

Syntax:-

return_type function_name (argument_list){

//function body.

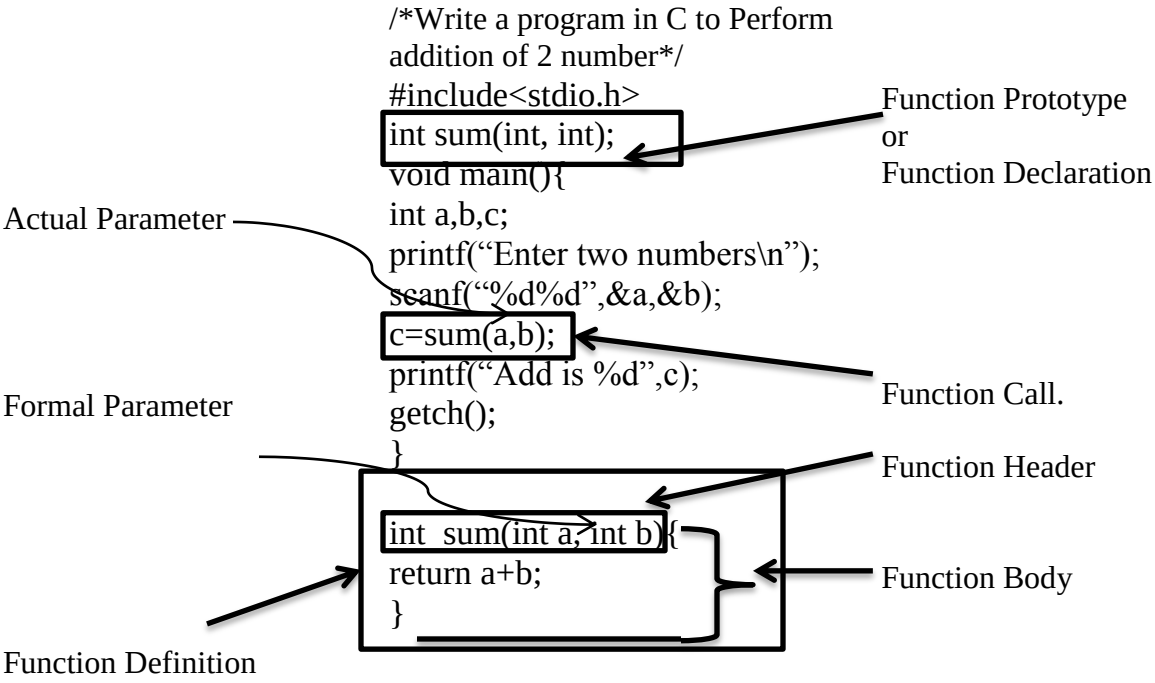
}

Example:-

Function Header… → sum(int a, int b){

Function Body… →
    int add;
    add=b+a;
    printf("addition is %d",add);
}

Programming Example to Show use of User define Function



Call by Value

- Whenever parameters/arguments are passed by value, the actual parameter in calling function are copied into corresponding formal parameters of called function.
- With call by value, changes made to formal parameters of called function have no effect on the values of actual parameters in calling function.
- Multiple values cannot be return…

Programming Example to show use parameter passing technique call by value.

/* C program to exchange(swap) values of two variable using call by Value */

```
#include<stdio.h>
void swap(int,int);
int main( ){
int n1,n2;
printf("Enter two number:");
scanf("%d%d",&n1,&n2);
printf("In main() b4 function call n1=%d n2=%d\n",n1,n2);
swap(n1,n2);
printf("In main() after function call n1=%d n2=%d",n1,n2);
return 0;
}
```

```
void swap(int n1,int n2){

        int temp;
        printf("In Swap( ) B4 swapping n1=%d n2=%d\n",n1,n2);
                temp=n1;
                n1=n2;
                n2=temp;
        printf("In swap( ) after swapping n1=%d n2=%d\n",n1,n2);
        }
```

Output

```
Enter two number:5 9
In main() b4 function call n1=5 n2=9
In Swap( ) B4 swapping n1=5 n2=9
In swap( ) after swapping n1=9 n2=5
In main() after function call n1=5 n2=9
```

Call by Reference or Call by Address:-

- Whenever the parameters/arguments are passed by reference the address of actual parameter in calling function are copied into formal parameter of called function.
- In call by reference, changes made to the formal parameter in called function have effect on the values of actual parameter in the calling function.
- Function can return more than one value at a time.(even function may not have return type.)

Programming example to show use of call by reference.

/* write a program to perform multiplication of two numbers using call by reference*/

```
#include<stdio.h>
void mulref(int*,int*);
main(){
int n1,n2;
printf("Enter two number:");
scanf("%d%d",&n1,&n2);
mulref(&n1,&n2);
}
void mulref(int *n1,int *n2){
int mul;
mul=*n1 *  *n2;
printf("Multiplication is %d",mul);
}
```

Output

```
Enter two number:8 9
Multiplication is 72
```

Local & Global Variables:-

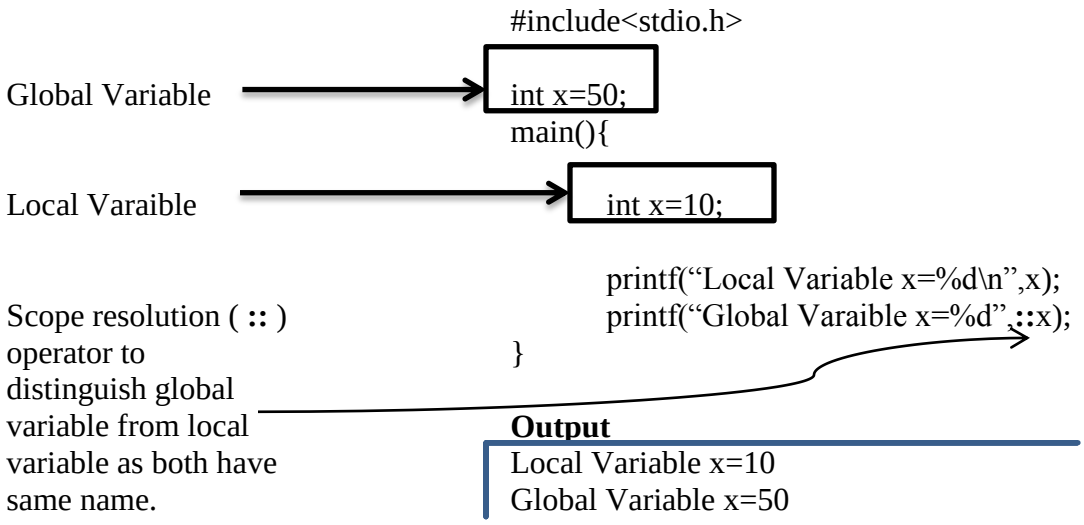
Local Variable:-

- A variable declared within function body is called as local variable.
- Local variables are referenced by the statements that are inside the body of function in which the variables are declared.
- Local variables get activated upon entry of control to the block/body & deactivated or destroyed upon exit.
- We can give same name to local variables in different functions or block.

Global Variable:-

- A variable which is declared outside of all functions is called a global variable.
- Global variables are referenced by any functions available in program.
- Global variable may however be placed anywhere in the program, prior to their first use, provided they are defined outside(above) the function, in this case the content of variables are available for use only by those functions of the program occurring after their declaration.
- If in any program local variable & global variable name is same then scope resolution (**::**) operator is used to distinguish global variable from local variable.

Programming example to show use of local & global variable.

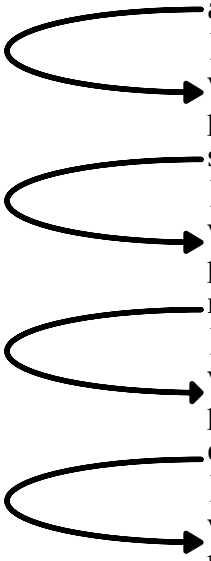


Nesting of Functions:

In c we can also have nesting of functions means one function can call second function & second function can call third function and so on.

Programming Example to show use of Nesting of Functions…

- In following example, from main( ) function we have called add() function then from add() function we have called sub(), and from sub(), mul() is called & finally div() is called from mul()… the curve arrow shows working of nesting function.



```
#include<stdio.h>
void add(int,int);
void sub(int,int);
void mul(int,int);
void div(int,int);
void main(){
int n1,n2;
printf("Enter two no for add sub mul & div:");
scanf("%d%d",&n1.&n2);
add(n1,n2);
}
void add(int n1,int n2){
printf("Addition is %d\n",n1+n2);
sub(n1,n2);
}
void sub(int n1,int n2){
printf("Subtraction is %d\n",n1-n2);
mul(n1,n2);
}
void mul(int n1,int n2){
printf("Multiplication is %d\n",n1*n2);
div(n1,n2);
}
void div(int n1,int n2){
printf("Division is %d",n1/n2);
}
```

Output

```
Enter two no for add sub mul & div: 20 8
Addition is 28
Subtraction is 12
Multiplication is 160
Division is 2
```

/* Write a Program to Find value of $Y=X^n$ Consider X & Y as float & n as integer.* /

```
#include<stdio.h>
float  power(float, int);
void main( ){
float y,x;
int n;
printf("Enter value of x & n\n");
scanf("%f%d",&x,&n);
y=power(x,n);
printf("%g to power %d is %g",x,n,y);
}
float power(float x, int n){
int i;
float y=1.0;
for(i=1;i<=n;i++)
y=y*x;
return y;
}
```

Recursive Function or Recursion:→

- Recursive function or recursion is a function which contains a call to itself.
- Works similar to Divide and conquer.
- Recursion must contains one exit condition that can be satisfied, otherwise the recursive function will call itself repeatedly **until the runtime stack overflows.**
- Example:-

```
#include<stdio.h>
main( ){
printf("I will stop\n");
main( );
}
```

Output

```
I will stop
I will stop
I will stop
:
:
I will stop
```

- In above example, main( ) is called within main( ), means it is recursion or recursive function call.
- As exit condition is not available in above program that's why function will call itself repeatedly until runtime stack overflow.
- That means recursion works on stack area of runtime memory & it will stop whenever stack overflow

Recursion is of two types:-

1. **Direct Recursion**
   - Calling same function within same function recursively.
   - Example:-

```
main( ){
printf("I will stop\n");
main( );
}
```

2. **Indirect Recursion**
   - Calling second function from first function and second and second function have call to first function, it is indirect recursion.
   - Example:-

```
#include<stdio.h>
void msg( );
void main( ){
        printf("I am in main( )\n");
        msg( );
}
void msg( ){
        printf("I am in msg( )\n");
        main( );
}
```

/* Write a program using recursion to perform the summation of given number.*/

```
#include<stdio.h>
int sum(int);
main( ){
        int n,s;
        printf(“Enter value of n:”);
        scanf("%d",&n);
        s=sum(n);
        printf("Sum of series is %d",s);
}
int sum(int n){
        if(n==1)
                return 1;
        else
                return(n+sum(n-1));
}
```

Output

```
Enter value of n:11

Sum of series is 66
```

/* Write a program to calculate the factorial of an integer using recursion.*/

```
#include<stdio.h>
int fact(int);
main( ){
        int n,f;
        printf(“Enter number:”);
        scanf("%d",&n);
        f=fact(n);
        printf("Factorial of %d is %d",n,f);
}
int fact(int n){
        if(n==1)
                return 1;
        else
                return(n*fact(n-1));
}
```

Output

```
Enter number:5

Factorial of 5 is 120
```

Differentiate between

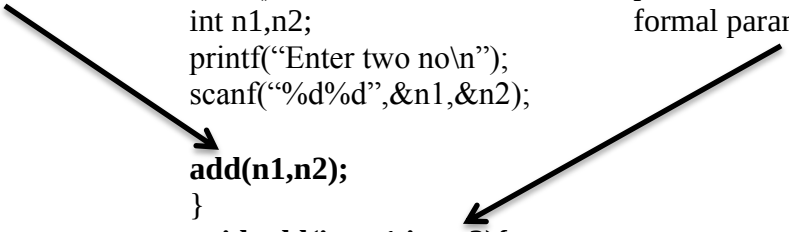
	Recursion	**Iteration**
1.	A function contains a call to itself.	Iterations take place in loop.
2.	Recursion must have at least one exit condition; otherwise recursion will call itself repeatedly until the runtime **stack overflows.**	After satisfying condition iteration will be stop; otherwise loop will never stop i.e. infinite loop.

3.	Example:-
	main(){
	printf("It is example of recursion");
	main()
	}
	…it will stop whenever runtime stack will overflow.
		Example:-
		main(){
		int i;
		for(i=1;i<=10;i++)
		printf("Iteration no %d",i);
		}
		…this for loop will have 10 iterations.

****

	Actual Parameters		**Formal Parameters**
1.	With Actual parameters we can call a function	*Programming Example to show difference between actual and formal parameters*	Whenever function is called, the actual parameter values are copied into formal parameters of a function.

2.	n1 & n2 are actual parameters & these are used to call function add().	#include<stdio.h>
		void add(int,int);
		main(){
		int n1,n2;
		printf("Enter two no\n");
		scanf("%d%d",&n1,&n2);
		add(n1,n2);
		}
		void add(int n1,int n2){
		printf("Addition is %d",n1+n2);
		}
			Here n1 & n2 are formal parameters, value of actual parameter will be copied into formal parameter.



- The number of actual and formal parameter must be equal.
- The names of actual and formal parameter may be same or different.