CHAPTER 3
Functions and Pointers

What is a Function?

- A function groups a number of program statements into a unit and gives it a name.

OR

- **A function is a group of statements that perform a particular task.**
- Every C program contains one or more functions.
- Every C program must contain one main () function, because program execution always begins with main().
- A function can be invoked or called from other parts of the program.
- A function may or may not return a value.
- A function can return only one value.
- A function may or may not take arguments.
-

Why to use function?

- Important reason to use function is to reduce program size.
- Programs written using functions are compact & easy to understand.

1. Let's study function with an example,

```
#include<stdio.h>
#include<conio.h>
void func1();
void main()
{
clrscr();
func1();
printf("Hi\n");
func1();
printf("Hello\n");
func1();
printf("Bye\n");
func1();
getch();
}

void func1()
{
int i;
for(i=0;i<=9;i++)
printf("*");
printf("\n");
}
```

OUTPUT

```
**********
Hi
**********
Hello
**********
Bye
**********
```

What components are necessary to add a function to the program?

There are three:

1) Function Prototype.
2) Calls to the function.
3) And the function definition.

1) Function prototype:-

In above program the function prototype is written as,

```
void func1();
```

It tells the compiler that a function **func1()** is coming up later in the program.
The keyword void tells that the function **func1()** does not return value and empty parentheses indicates that **func1()** takes no arguments.

2) Calls to the function:-

A function can be invoked or called from other parts of the program. In above program function **func1()** is called/invoked four times from main() function. And each of the call looks like this,

```
func1();
```

That means to call the function we need function name followed by parentheses. The call is terminated by semicolon.

When the function **func1()** is called, control is transferred to the first statement in the function body. The other statements in the function body are then executed and when the closing brace is met, control return to the statement following the function call. See figure **1**. below.

3) The function definition:-

The body of the function **func1()** is surrounded by braces.
The function definition contains the actual code for the function. In the above program function definition of **func1()** is looks like this,

```
void func1()          //declarator
{
int i;
for(i=0;i<=9;i++)     //function body
printf("*");
printf("\n");
}
```

The definition consists of a line called declarator followed by body of the function.
The declarator must agree with the function prototype. That means it must use the same function name, must have same return type and must have same argument type in the same order (if there are arguments).
In function definition of **func1(),** void of declarator must match with void of function prototype and empty parentheses must math with empty parentheses of function prototype.

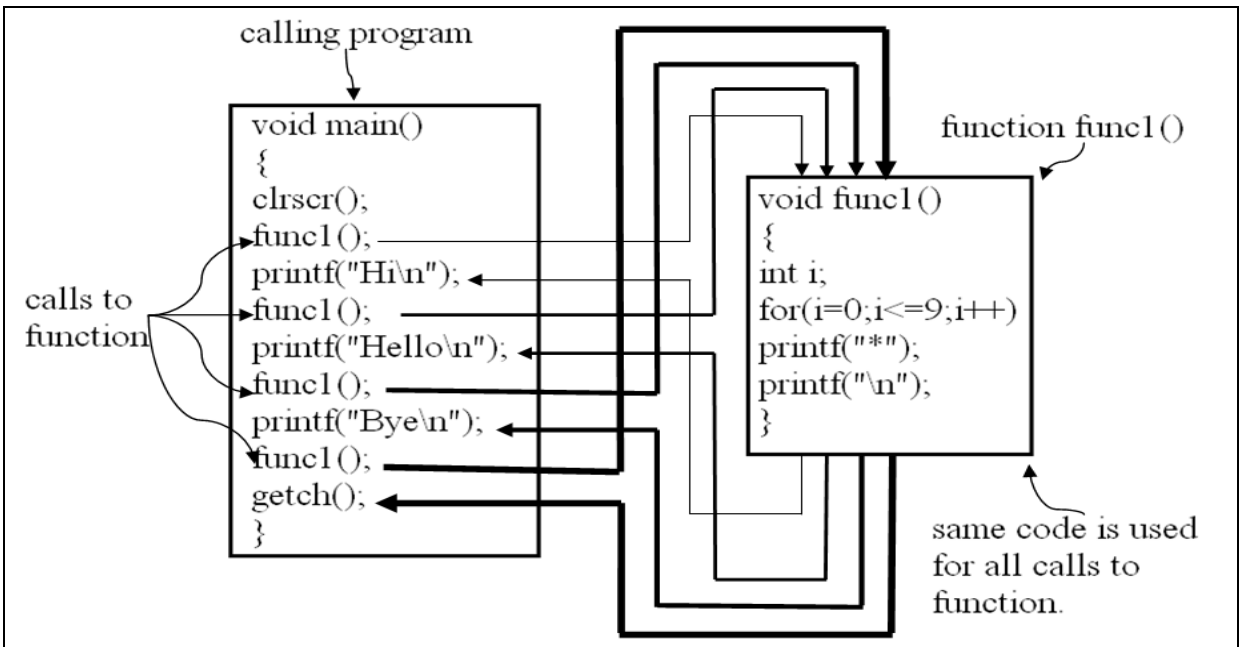


Fig1:- Flow of control to function.

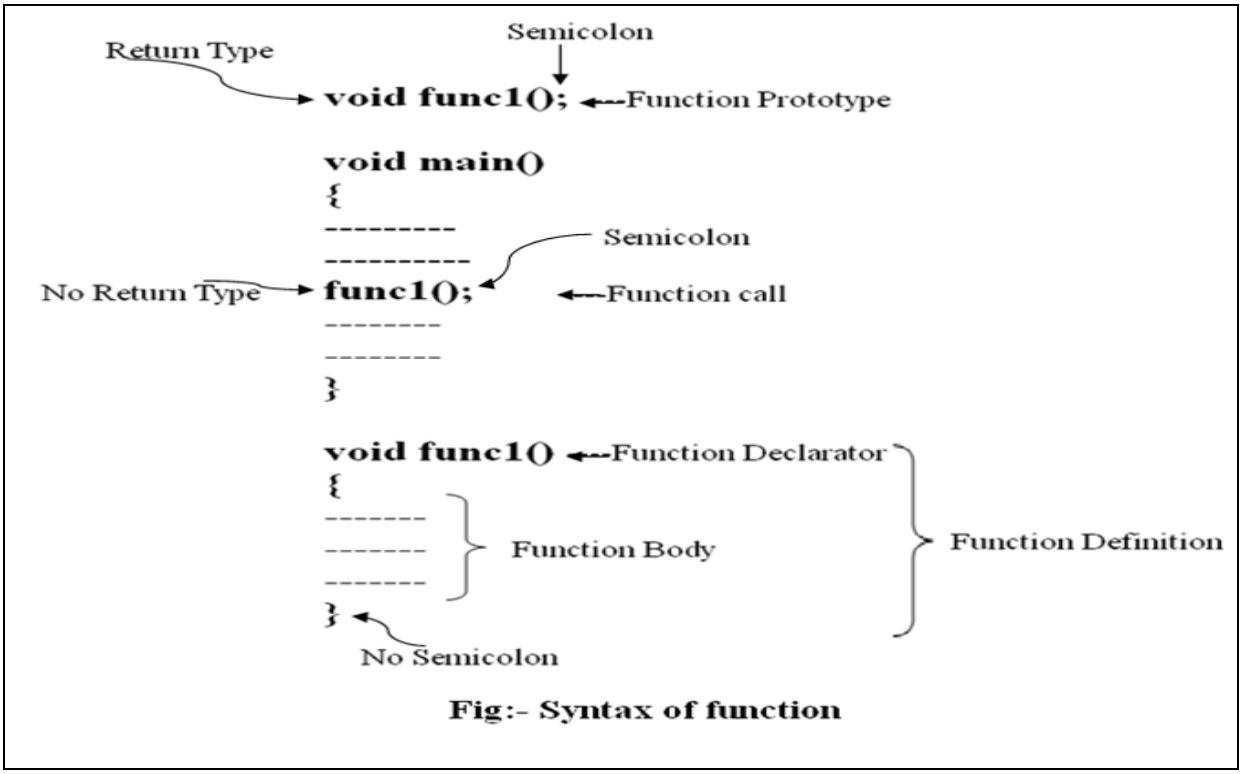


Fig:- Syntax of function

2.

```
#include<stdio.h>
#include<conio.h>
void india();

void main()
{
clrscr();
printf("\nI am in main\n");
india();
printf("\nI am in main again\n");
getch();
}

void india()
{
printf("\nI am in india now\n");
}
```

OUTPUT

```
I am in main

I am in india now

I am in main again
```

3.

```
#include<stdio.h>
#include<conio.h>
void india();

void main()
{
clrscr();
printf("\nI am in main\n");
india();
printf("\nHi I am in main again\n");
india();
printf("\nHello I am in main again\n");
getch();
}

void india()
{
printf("\nI am in india now\n");
}
```

OUTPUT

```
I am in main

I am in india now

Hi I am in main again

I am in india now

Hello I am in main again
```

Types of functions:
A function may belong to any one of the following categories:

1. ***Function with no arguments and no return value.***

Example	Description
void add();	Here void indicates that function does not return value and empty parentheses indicate that function takes no arguments.

2. ***Function with arguments and no return value.***

Example	Description
void add(int, int);	Function does not return value but takes two int arguments.

3. ***Function with no arguments but return value.***

Example	Description
int add();	Function returns integer value and takes no argument.

4. ***Function with arguments and return value.***

Example	Description
int add(int, int);	Function returns integer value and takes two int arguments.

1. Function with no arguments and no return value.

4.

```
#include<stdio.h>
#include<conio.h>
void india();

void main()
{
clrscr();
printf("\nI am in main\n");
india();
printf("\nI am in main again\n");
getch();
}

void india()
{
printf("\nI am in india now\n");
}
```

OUTPUT

```
I am in main

I am in india now

I am in main again
```

2. Function with arguments and no return value.

5.

```
#include<stdio.h>
#include<conio.h>

void square(int);

void main()
{
clrscr();
square(2);
square(12);
square(9);
getch();
}

void square(int l)
{
int k;
k=l*l;
printf("\nSquare is %d\n",k);
}
```

OUTPUT

```
Square is 4

Square is 144

Square is 81
```

6.

```
#include<stdio.h>
#include<conio.h>

void add(int,int);

void main()
{
int a=10,b=20;
clrscr();
add(a,b);
getch();
}

void add(int l,int m)
{
int k;
k=l+m;
printf("\nAddition is %d\n",k);
}
```

OUTPUT

```
Addition is 30
```

3. Function with no arguments but return value.

7.

```
#include<stdio.h>
#include<conio.h>
int add();

void main()
{
int sum;
clrscr();
sum=add();
printf("\nAddition is %d",sum);
getch();
}

int add()
{
int l,m,k;
printf("\nEnter two numbers:\n");
scanf("%d%d",&l,&m);
k=l+m;
return(k);
}
```

OUTPUT

```
Enter two numbers:
10
20

Addition is 30
```

4. Function with arguments and return value.

8.

```
#include<stdio.h>
#include<conio.h>
int add(int,int);

void main()
{
int a=10,b=20,sum;
clrscr();
sum=add(a,b);
printf("\nAddition is %d",sum);
getch();
}

int add(int l, int m)
{
int k;
k=l+m;
return(k);
}
```

OUTPUT

```
Addition is 30
```

9.

```
#include<stdio.h>
#include<conio.h>
int add(int,int);

void main()
{
int a,b,sum;
clrscr();
printf("\nEnter two numbers:\n");
scanf("%d%d",&a,&b);
sum=add(a,b);
printf("\nAddition is %d",sum);
getch();
}

int add(int l, int m)
{
int k;
k=l+m;
return(k);
}
```

OUTPUT

```
Enter two numbers:
10
20

Addition is 30
```

Pointers:-

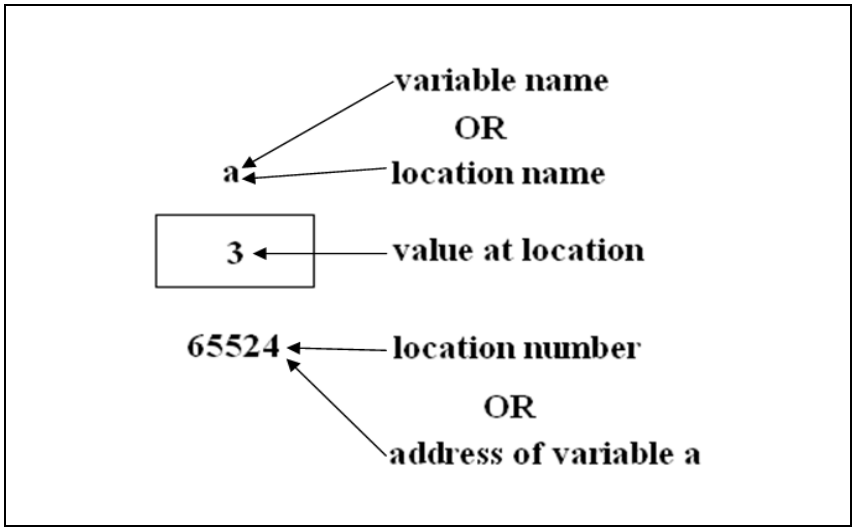
A pointer is a variable that stores address of another variable.

Consider following declaration,

int a=3;

The above statement declares integer variable 'a' and assign 3 to it. Actually this statement tells the compiler to

a) Reserve space in memory to store the integer value.
b) Give 'a' name to that memory location.
c) Store the value 3 at this location.



See above figure, 'a' is nothing but memory location name (or variable name). Value 3 is stored at this location. And 65524 is the address of variable 'a' (or more clearly memory location number).

10. Program show how to print address of variable using address of (&) operator.

```
#include<stdio.h>
#include<conio.h>

void main()
{
int a=3;
clrscr();
printf("\nValue of a=%d",a);
printf("\nAddress of a=%u",&a);
getch();
}
```

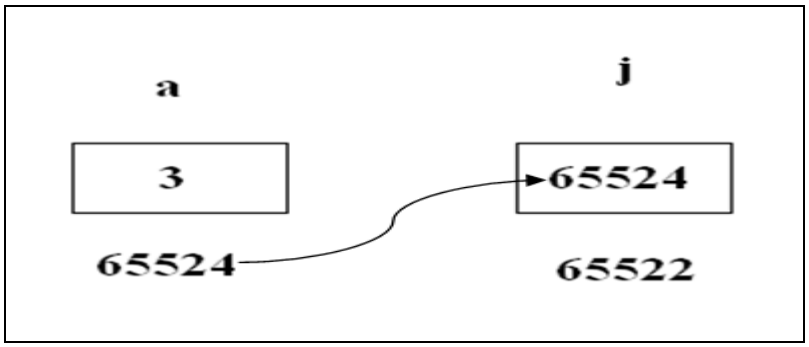
OUTPUT

```
Value of a=3
Address of a=65524
```

& is address of operator that gives address of variable.

Address of variable is always whole number so use %u.

Now see figure below, the variable ‘j’ stores address of another variable ‘a’, so ‘j’ is called as pointer. And we say that ‘j’ is pointing to ‘a’.



Declaration of Pointer:-

Syntax for declaring pointer variable is as follows,

Data_type * pointer-name ;

For example,
int a=3;
int *j;

Initialization of pointer variable:-

We can assign address of variable to the pointer. This is known as pointer initialization. The general form of pointer initialization is as follows,

pointer-name = &(variable-name);

For example,
int a=3;
int *j;
j=&a;

Accessing value of variable through pointer:-

You can access value of variable through pointer using ‘value at address’ operator, which is represented by *.

For example,

int a=3;
int *j;
j=&a;
printf(“\nValue of a=%d”,***j**);

Description,

Here ‘j’ is the pointer because it stores address of variable ‘a’. We can access variable ‘a’ using ‘*j’ that means ‘*j’ gives you value of ‘a’.

11. Program showing how to declare and initialize pointer variable. And how to access variable through pointer variable.

```
#include<stdio.h>
#include<conio.h>

void main()
{
int a=3;
int *j;
clrscr();
j=&a;
printf("\nAddress of a=%u",&a);
printf("\n\nAddress of a=%u",j);

printf("\n\n\nValue of a=%d",a);
printf("\n\nValue of a=%d",*j);

getch();
}
```

OUTPUT

```
Address of a=65524

Address of a=65524


Value of a=3

Value of a=3
```

& is address of operator that gives address of variable.

Address of variable is always whole number so use %u.

In above program we have declared integer variable 'a' and one integer pointer 'j'. Then address of variable 'a' is assigned to pointer 'j'. Now we can access address of variable 'a' using 'j'. And we can access value of variable 'a' using '*j'. So it is clear from above program that we can access variable's address and value using pointer.

12.

```
#include<stdio.h>
#include<conio.h>

void main()
{
int a=3;
int *j;
clrscr();
j=&a;

printf("\n\nValue of a=%d",a);
printf("\n\nValue of a=%d",*j);

*j=*j+1;
printf("\n\nValue of a=%d",a);
printf("\n\nValue of a=%d",*j);
getch();
}
```

OUTPUT

```
Value of a=3

Value of a=3

Value of a=4

Value of a=4
```

Note:-
*j=*j+1;
Increments value of variable 'a' by 1. Because pointer 'j' stores address of 'a'.

In the above program 'j' is a pointer variable holding address of variable 'a'. Therefore when we perform operation on pointer variable 'j' then actually variable 'a' is getting changed.

13. Program showing pointers pointing to char and float variables.

```
#include<stdio.h>
#include<conio.h>
void main()
{
char a='Z';
char *j;
float b=19.88;
float *k;
clrscr();
j=&a;
k=&b;
printf("\nAddress of a=%u",&a);
printf("\n\nAddress of a=%u",j);
printf("\n\nAddress of b=%u",&b);
printf("\n\nAddress of b=%u",k);

printf("\n\n\nValue of a=%c",a);
printf("\n\nValue of a=%c",*j);
printf("\n\nValue of b=%f",b);
printf("\n\nValue of b=%f",*k);
getch();
}
```

OUTPUT

```
Address of a=65525

Address of a=65525

Address of b=65520

Address of b=65520


Value of a=Z

Value of a=Z

Value of b=19.879999

Value of b=19.879999
```

Pointer expressions and Arithmetic

Like other variables pointer variables can also be used in expressions. We can perform addition, subtraction, multiplication and division using pointer.

14. Program showing pointer expressions and arithmetic.

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int a = 10, b = 2;
 int *ptr1 = &a, *ptr2 = &b;
clrscr();
printf(" \n Addition of a and b is    : %d" , *ptr1+*ptr2 );
printf(" \n Subtraction of a and b is : %d" , *ptr1-*ptr2 );
printf(" \n Multiplication of a and b is : %d" , (*ptr1)*(*ptr2) );
printf(" \n Division of a and b is       : %d" , (*ptr1) / (*ptr2) );
getch();
}
```

OUTPUT

```
Addition of a and b is      : 12
Subtraction of a and b is   : 8
Multiplication of a and b is : 20
Division of a and b is      : 5
```

Call By Value and Call By Reference

Call By Value:-

15.

```
#include<stdio.h>
#include<conio.h>

void swap(int,int);

void main()
{
int a=10,b=20;
clrscr();
printf("\nBefore Swapping a=%d   b=%d\n",a,b);
swap(a,b);
printf("\nAfter Swapping a=%d   b=%d\n",a,b);
getch();
}

void swap(int l,int m)
{
l=l+m;
m=l-m;
l=l-m;
}
```

OUTPUT

```
Before Swapping a=10   b=20

After Swapping a=10   b=20
```

Note: - Here we are just passing values of variable 'a' and 'b' to function swap. Therefore operation on variable 'l' and 'm' does not swap 'a' and 'b'.

In above program the particular values of variables a and b are passed to the function 'swap' when this statement **swap(a,b);** in main is executed. Then function 'swap' gives these passed values to arguments l and m. And such call to function is called as 'call by value'.

In short when we call function by passing values or variable then call to function is called as 'call by value'.

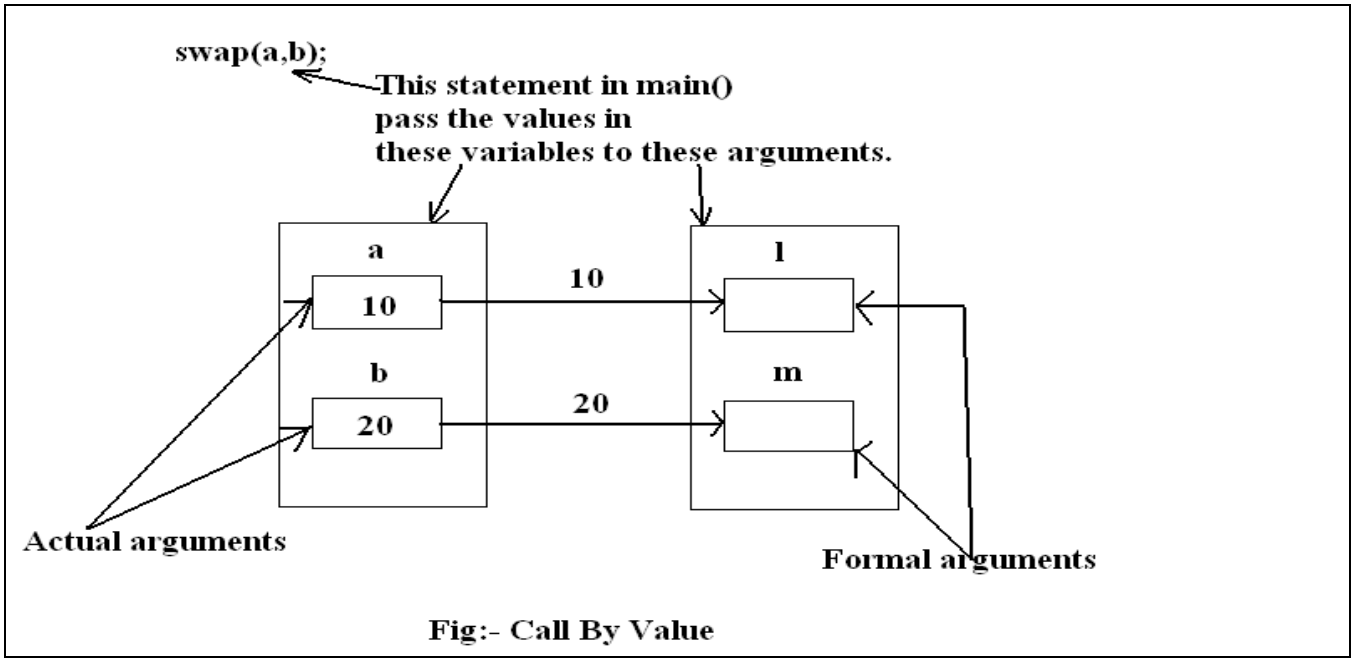


Fig:- Call By Value

Call By Reference:-

16. Program to swap two numbers using call by reference

```
#include<stdio.h>
#include<conio.h>

void swap(int*,int*);

void main()
{
int a=10,b=20;
clrscr();
printf("\nBefore Swapping a=%d   b=%d\n",a,b);
swap(&a,&b);
printf("\nAfter Swapping a=%d   b=%d\n",a,b);
getch();
}

void swap(int *l,int *m)
{
*l=*l+*m;
*m=*l-*m;
*l=*l-*m;
}
```

OUTPUT

```
Before Swapping a=10   b=20

After Swapping a=20   b=10
```

Note: - Here we are passing address of 'a' and 'b' to function swap. Therefore operation on pointers swap 'a' and 'b'. Because in swap function, through pointers, we are actually accessing variable 'a' and 'b'.

In above program the particular address of variables a and b are passed to the function 'swap' when this statement **swap(&a,&b);** in main is executed. Then function 'swap' gives these addresses to pointers l and m. And such call to function is called as 'call by reference'.

In short, when we call function by passing address of variables then call to function is called as 'call by reference'.

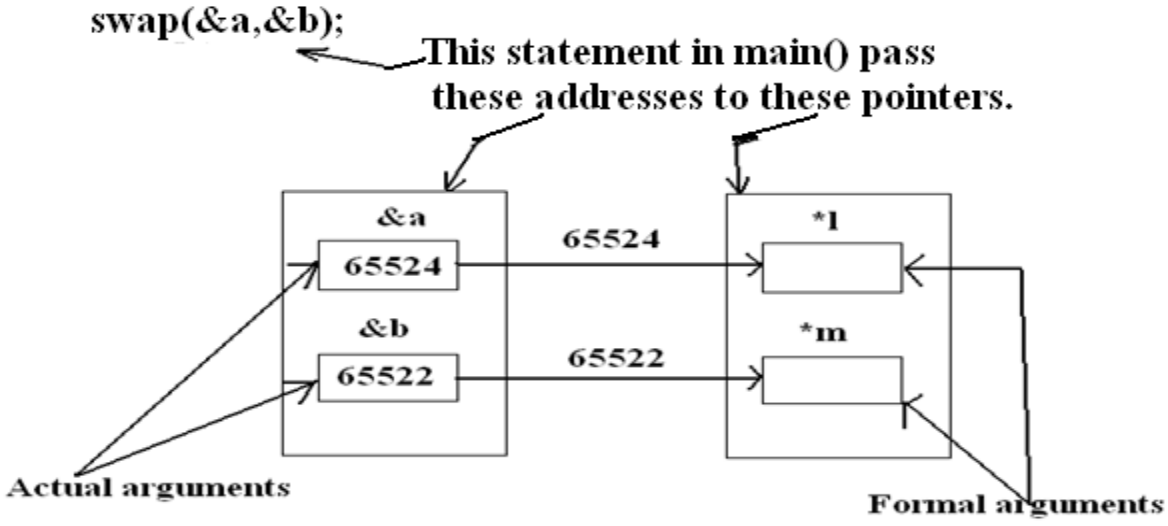


Fig:- Call By Reference

SOME IMPORTANT PROGRAMS ON FUNCTIONS

17. Write a program to calculate the factorial of a number entered through the keyboard, using function?

```
#include<stdio.h>
#include<conio.h>
void factorial(int);
void main()
{
int a;
clrscr();
printf("\nEnter a number:");
scanf("%d",&a);
factorial(a);
getch();
}

void factorial(int n)
{
int i,fact=1;
for(i=1;i<=n;i++)
{
fact=fact*i;
}
printf("\nFactorial of a number is %d",fact);
}
```

OUTPUT

```
Enter a number:5

Factorial of a number is 120
```

18. Any year is entered through the keyboard. Write a function to determine whether the year is a leap year or not?

```
#include<stdio.h>
#include<conio.h>

void leapyear(int);

void main()
{
int year;
clrscr();
printf("\nEnter Year: ");
scanf("%d",&year);
leapyear(year);
getch();
}

void leapyear(int year)
{
if(year%4==0&&year%100!=0||year%400==0)
printf("%d is leap year",year);
else
printf("%d is not a leap year",year);
}
```

OUTPUT

```
Enter Year: 1800
1800 is not a leap year
```

19. Write a function which receives a float and an int from main(), finds the product of these two and returns the product which is printed through main()?

```
#include<stdio.h>
#include<conio.h>
float product(int,float);

void main()
{
int a;
float b,prod;
clrscr();
printf("\nEnter int and float value: ");
scanf("%d%f",&a,&b);
prod=product(a,b);
printf("\nProduct= %f",prod);
getch();
}

float product(int a, float b)
{
float prod;
prod=a*b;
return(prod);
}
```

OUTPUT

```
Enter int and float value: 10 2.5

Product= 25.000000
```

20. Write a function which receives three values from main(), finds the greater one?

```
#include<stdio.h>
#include<conio.h>

void greater(int,int,int);

void main()
{
int a,b,c;
clrscr();
printf("\nEnter three numbers:");
scanf("%d%d%d",&a,&b,&c);
greater(a,b,c);
getch();
}

void greater(int a,int b,int c)
{
if(a>b&&a>c)
printf("\n%d is greater",a);
else if(b>c)
printf("\n%d is greater",b);
else
printf("\n%d is greater",c);
}
```

OUTPUT

```
Enter three numbers:10 20 30

30 is greater
```

Scope and Storage Class of Variables:-

The scope of variable determines which parts of the program can access variable. And storage class of variable determines how long variable stays in existence.

We'll look at variable with four storage classes,

1. Automatic Variables (Local Variables):-

Variables declared within function body are called automatic variables ***or local variables***. Variable declared within function body are automatic by default. A keyword auto can be used to specify an automatic variable.

- ***a. Lifetime:-***
  As the name implies automatic variables are automatically created when a function is called and automatically destroyed when function returns.
- ***b. Scope(visibility):-***
  Automatic variable can only be accessed within the function in which they are declared.
- ***c. Initialization:-***
  An arbitrary (garbage) value is assigned to automatic variable.

21.

#include<stdio.h>
void foo();
void main()
{
int a=10,b; //automatic var. by default
printf("%d %d",a,b);
}
 	
void foo()
{
printf("%d %d",a,b);
}

OR

	Description

#include<stdio.h>
void foo();
void main()
{
auto int a=10,b;
printf("%d %d",a,b);
}
 	
void foo()
{
printf("%d %d",a,b);
}
	Here variables a & b are declared within main() function so they are accessible only within main(). They are not accessible within foo() function.
	
	The statement int a=10, b; declares two variables and assign 10 to a and garbage value is given to b.

2. External Variables (or Global Variables):-

External variable are declared outside of any function.

- ***a. Lifetime:-***
  They are created when program begins and destroyed when program ends.
- ***b. Scope(visibility):-***
  They are accessible throughout the program. (More clearly in the file).
  Therefore all functions can access them.
- ***c. Initialization:-***
  When they are not explicitly initialized then compiler initializes it to zero, when they are created.

22.

```
#include<stdio.h>
#include<conio.h>
int a;  //global variable //initialized to zero

void show();

void main()
{
clrscr();
printf("\nA=%d",a);
a=99;
printf("\nA=%d",a);
show();
getch();
}

void show()
{
printf("\nA=%d",a);
a=55;
printf("\nA=%d",a);
}
```

OUTPUT

```
A=0
A=99
A=99
A=55
```

Descripton

Here variable 'a' is declared outside of any function, so 'a' is global variable. Therefore all functions can access 'a'.

3. Static variables:-

Static variables are accessible only inside the function in which they are declared but it remains in existence until program ends. They are initialized only once when they are declared. When they are not explicitly initialized then compiler initializes it to zero.

23.

```
#include<stdio.h>
#include<conio.h>
void show();

void main()
{
clrscr();
show();
show();
show();
getch();
}

void show()
{
static int a;
a=a+10;
printf("\nA=%d",a);
}
```

OUTPUT

```
A=10
A=20
A=30
```

4. Register Variables:-

If we declared variables with keyword ***register***, then the values are stored in the machine register instead of storing them in the memory.

24.

```
#include<stdio.h>
#include<conio.h>

void main()
{
register int a;
a=5;
clrscr();
printf("\nA=%d",a);
getch();
}
```

OUTPUT

```
A=5
```