5 Functions

State the need of function.

A function is a self contained block of statements which is used to perform a specific task.

Need:

1. It is possible to code any program utilizing only main function but it leads to a number of problems.
2. The program may become too large and complex and as a result the task of debugging, testing and maintaining becomes difficult.
3. If a program is divided into functional parts, then each part may be independently coded and later combined into a single unit.
4. These independently coded programs are called as subprograms that are much easier to understand, debug and test.

5.1User defined functions:

[A] A function is a self contained block of statements which is used to perform a specific task.

Need:

1. It is possible to code any program utilizing only main function but it leads to a number of problems.
2. The program may become too large and complex and as a result the task of debugging, testing and maintaining becomes difficult.
3. If a program is divided into functional parts, then each part may be independently coded and later combined into a single unit.
4. These independently coded programs are called as subprograms that are much easier to understand, debug and test.

[B] Functions those are developed by the user for their Programs are known as User Defined Programs. When a user wants to make his Own Function, then he must have to follow the Following Operations.

C allow programmers to define functions. Such functions created by the user are called user-defined functions.

Depending upon the complexity and requirement of the program, we can create as many user-defined functions as we want

1) Function Declaration or Prototyping

2) Function Defining

3) Calling a Function

User-defined functions can be categorized as:

- Function with no arguments and no return value
- Function with no arguments and a return value
- Function with arguments and no return value
- Function with arguments and a return value.

Advantages of user-defined function

1. The program will be easier to understand, maintain and debug.
2. Reusable codes that can be used in other programs
3. A large program can be divided into smaller modules. Hence, a large project can be divided among many programmers

5.2 A multifunction program,

5.3 The form of ‘C’ function

```
#include <stdio.h>

void functionName()

{

    ... .. ...

    ... .. ...

}

int main()

{

    ... .. ...

    ... .. ...

    functionName();

    ... .. ...

    ... .. ...

}
```

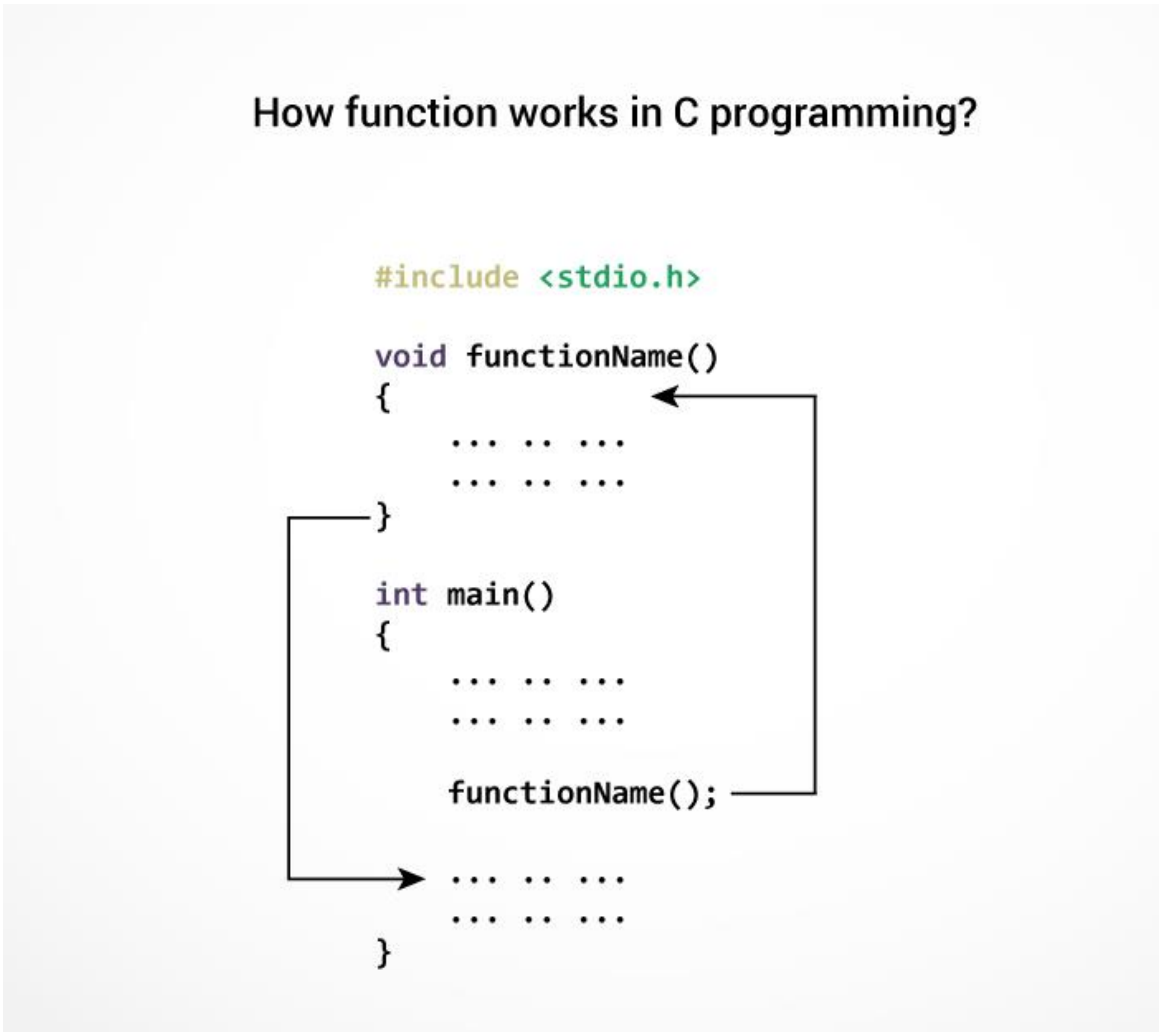
The execution of a C program begins from the main() function.

When the compiler encounters functionName(); inside the main function, control of the program jumps to

void functionName()

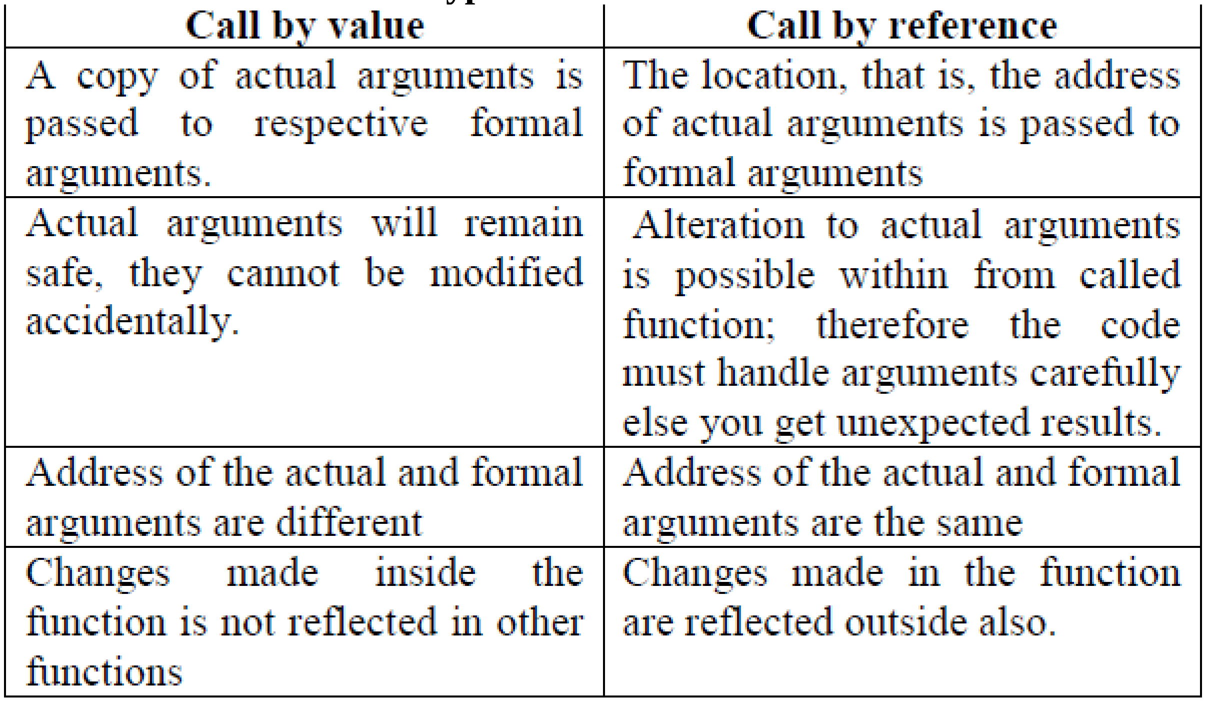
And, the compiler starts executing the codes inside the user-defined function.

The control of the program jumps to statement next to functionName(); once all the codes inside the function definition are executed.



```
#include <stdio.h>

void functionName()
{
    ... .. ...
    ... .. ...
}

int main()
{
    ... .. ...
    ... .. ...

    functionName();

    ... .. ...
    ... .. ...
}
```

5.4 Return value and their types



Call by value	Call by reference
A copy of actual arguments is passed to respective formal arguments.	The location, that is, the address of actual arguments is passed to formal arguments
Actual arguments will remain safe, they cannot be modified accidentally.	Alteration to actual arguments is possible within from called function; therefore the code must handle arguments carefully else you get unexpected results.
Address of the actual and formal arguments are different	Address of the actual and formal arguments are the same
Changes made inside the function is not reflected in other functions	Changes made in the function are reflected outside also.

CALL BY VALUE	**CALL BY REFERENCE**
Passing the value of variable to the calling function	Passing the address of variable to the calling function.
Changes will not be reflected back in main function.	Changes will be reflected back in main function.

Example:
void main()
{
    int x=10,y=20;
    printf("%d%d",x,y);
    swap(x,y);
}
void swap(int a,int b)
{
    int c;
    c=a;
    a=b;
    b=c;
    /*changes here do not affect in values
     of x and y in main function..*/
}
	Example:
	void main()
	{
	    int x=10,y=20;
	    printf("%d%d",x,y);
	    swap(&x,&y);
	}
	void swap(int *a,int *b)
	{
	   int c;
	   c=*a;
	  *a=*b;
	   *b=c ;
	/*changes here do affect in values of x and y
	in main function..*/
	}

5.5 Calling a function

```
#include <stdio.h>
int addNumbers(int a, int b);         // function prototype
int main()
{
    int n1,n2,sum;

    printf("Enters two numbers: ");
    scanf("%d %d",&n1,&n2);
    sum = addNumbers(n1, n2);        // function call
    printf("sum = %d",sum);
    return 0;
}
int addNumbers(int a,int b)          // function definition
{
    int result;
    result = a+b;
    return result;                  // return statement
}
```

[1] Function prototype

A function prototype is simply the declaration of a function that specifies function's name, parameters and return type. It doesn't contain function body.

A function prototype gives information to the compiler that the function may later be used in the program.

Syntax of function prototype

```
returnType functionName(type1 argument1, type2 argument2,...);
```

In the above example, `int addNumbers(int a, int b);` is the function prototype which provides following information to the compiler:

1. name of the function is `addNumbers()`
2. return type of the function is `int`
3. two arguments of type `int` are passed to the function

The function prototype is not needed if the user-defined function is defined before the `main()` function.

[2] Calling a function

Control of the program is transferred to the user-defined function by calling it.

Syntax of function call

```
functionName(argument1, argument2, ...);
```

In the above example, function call is made using `addNumbers(n1,n2);` statement inside the `main()`.

[3] Function definition

Function definition contains the block of code to perform a specific task i.e. in this case, adding two numbers and returning it

Function definition

Function definition contains the block of code to perform a specific task i.e. in this case, adding two numbers and returning it

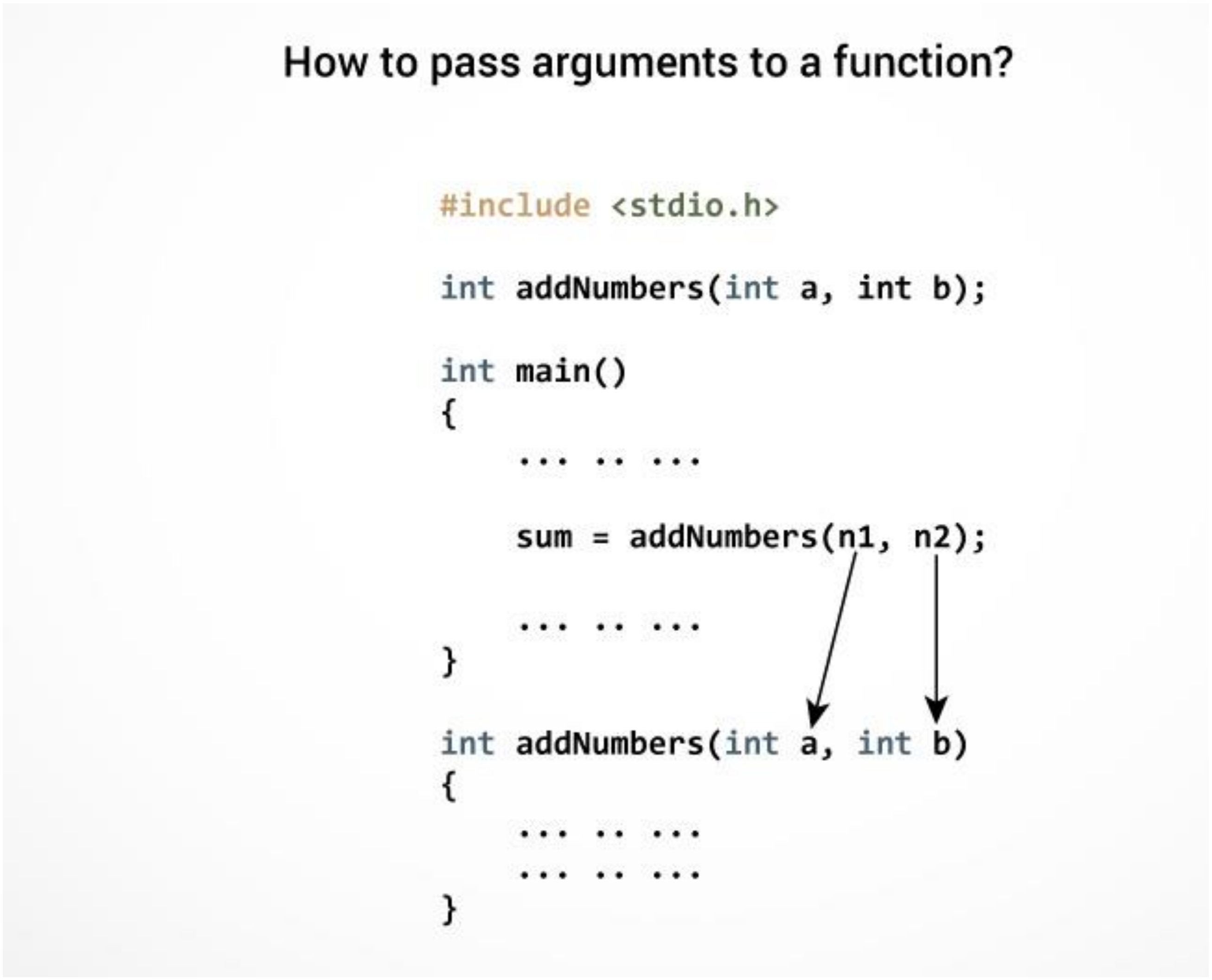
```
returnType functionName(type1 argument1, type2 argument2, ...)

{

   //body of the function

}
```

When a function is called, the control of the program is transferred to the function definition. And, the compiler starts executing the codes inside the body of a function.

Passing arguments to a function

In programming, argument refers to the variable passed to the function. In the above example, two variables n1 and n2 are passed during function call.

The parameters a and b accepts the passed arguments in the function definition. These arguments are called formal parameters of the function.



```
#include <stdio.h>

int addNumbers(int a, int b);

int main()
{
    ... .. ...

    sum = addNumbers(n1, n2);

    ... .. ...
}

int addNumbers(int a, int b)
{
    ... .. ...
    ... .. ...
}
```

The type of arguments passed to a function and the formal parameters must match, otherwise the compiler throws error.

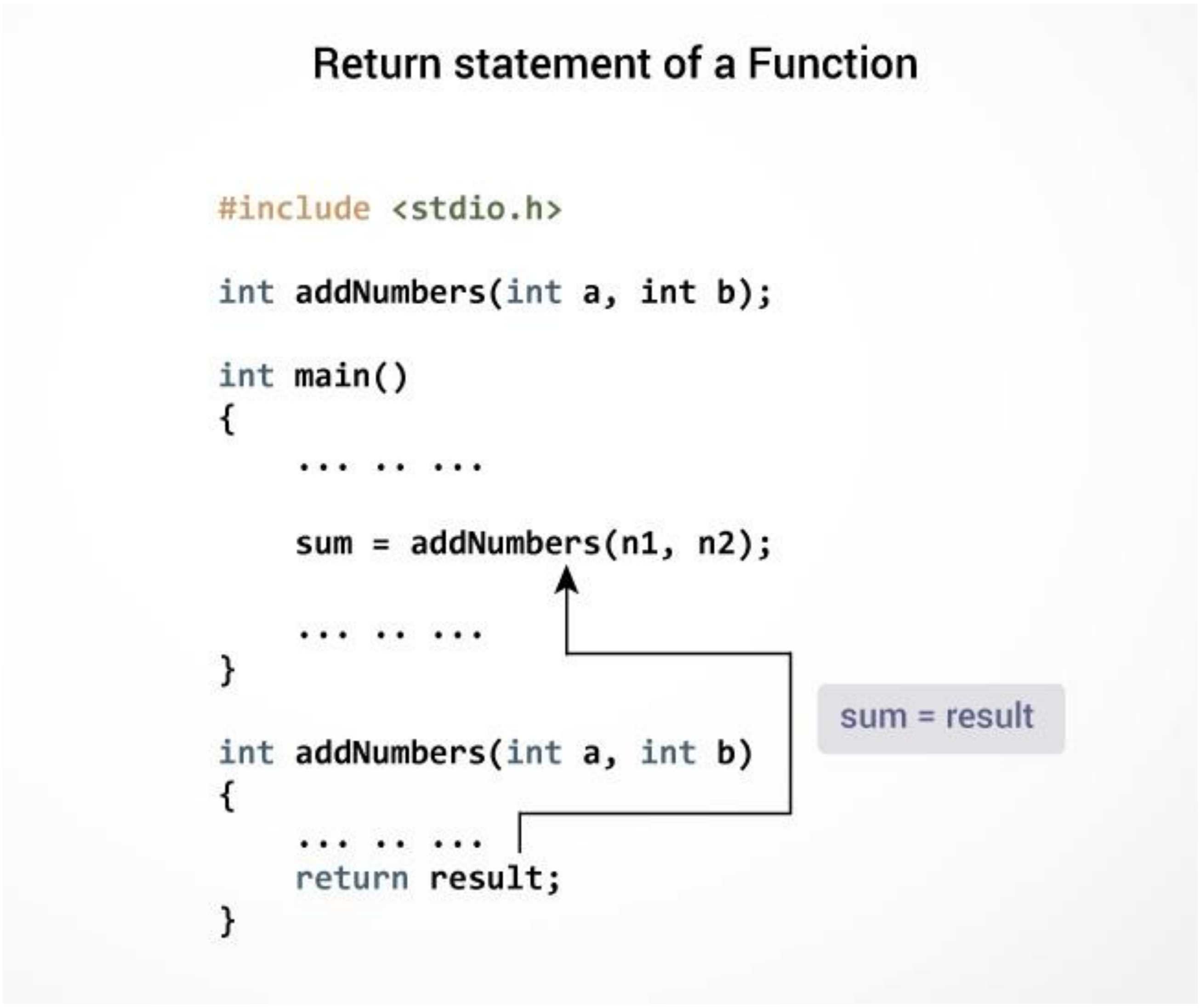
If n1 is of char type, a also should be of char type. If n2 is of float type, variable b also should be of float type.

A function can also be called without passing an argument.

Return Statement

The return statement terminates the execution of a function and returns a value to the calling function. The program control is transferred to the calling function after return statement.

In the above example, the value of variable result is returned to the variable sum in the main() function.



Return statement of a Function

```
#include <stdio.h>

int addNumbers(int a, int b);

int main()
{
    ... .. ...

    sum = addNumbers(n1, n2);

    ... .. ...
}

int addNumbers(int a, int b)
{
    ... .. ...
    return result;
}
```

Syntax of return statement

```
return (expression);
```

For example,

```
return a;

return (a+b);
```

The type of value returned from the function and the return type specified in function prototype and function definition must match.

5.7 No arguments and no return value
1. Example for Function with no arguments and no return values.

```
#include<stdio.h>

void main()

{

void evenodd(void)

clrscr();

evenodd();

getch();

}

void evenodd()

{

int num=25;

if(num%2==0)

printf("%d is even",num);

else

printf("%d is odd",num);

}
```

5.8 Arguments with return value

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

void main()

{

int fact(int);

int no;

clrscr();

printf("\n Enter number:");

scanf("%d",&no);

printf("\nfactorial of number:%d",fact(no));

getch();

}

int fact(int n)

{

int fact=1,i=0;

for(i=1;i<=n;i++)

{

fact=fact*i;

}
return fact;
}
```

5.9 Nesting of functions and recursion

Recursive function is the process in which function calls itself.

Advantages :

• Reduces length of the program
• Reduces unnecessary calling of a function.
• Useful when same solution is to be applied many times.

When a function is called by itself within its own body then that process is called as Recursion. I

process chaining occurs.

Example:

```
#include<stdio.h>

#include<conio.h>

int fact(int);

void main()

{

int a,b;

clrscr();

printf("Enter a number:");

scanf("%d",&a);

b=fact(a);

printf("\nFactorial of a given number is:%d",b);

getch();

}
```

```
int fact(int n)

{

int f;

if (n==1)

return(1);

else

f=n*fact(n-1); /*recursion occurs here*/

return(f);

}
```

In above example recursion is used where function named fact is called by itself to get the factorial of an entered number.

Write a function to print factorial of a number.
(Note: Any Category of function shall be consider)

```
void factorial()
{
int no,fact=1,i;
printf("Enter number:");
scanf("%d",&no);
for(i=1;i<=no;i++)
fact=fact*i;
printf("\n Factorial of %d is %d",no,fact);
}
```

Write a function to display fibonnacci series up to given number using recursion .Usefunction name "Fibbo".

```
#include<stdio.h>
#include<conio.h>
int Fibbo(int n);
void main()
{
```

```
int c,i=0,n;
clrscr();
printf("Enter fibonacci number:");
scanf("%d",&n);
printf("\nFibonacci number series:");
for(c=1;c<=n;c++)
{
printf("%d ",Fibbo(i));
i++;
}
getch();
}
int Fibbo(int n)
{
if(n==0)
return(1);
else if(n==1)
return(1);
else
return(Fibbo(n-1)+Fibbo(n-2));
}
```

State call by value with example.

When a function is called by passing actual parameters in function name then it is called as call by value. In call by value method actual parameters are copied to formal parameters. The called function works on the copy and not on original values of the parameters. Because of this the original data cannot be changed accidently.

Example:

```
#include<stdio.h>

#include<conio.h>

main()

{
```

```
int p,a,b;
printf("Enter two numbers:";
scanf("%d %d",&a,&b);
p=product(a,b);
}
product(int x,int y)
{
int p1=1;
p1=x*y;
return(a,b);
}
```

Write a program to perform addition, subtraction, multiplication and division of two integer numbers using function.

```
#include<stdio.h>
#include<conio.h>
void add(int p,int q);
void subtraction (int p,int q);
void multiplication(int p,int q);
void division(int p,int q);
void main()
{
int m,n,sum,sub,mul,div;
clrscr();
```

```
printf("enter two integer numbers::");
scanf("%d %d",&m,&n);
add(m,n);
subtraction(m,n);
multiplication(m,n);
division(m,n);
getch();
}
void add(int p,int q)
{
int sum;
sum=p+q;
printf("Addition of a numberis %d\n",sum);
}
void subtraction(int p,int q)
{
float sub;
sub=p-q;
printf("\n subtraction of a number %f",sub);
}
void multiplication(int p, int q)
{
int mul;
```

```
mul=p*q;
printf("\nmutiplication of a number=%d\n",mul);
}
void division(int p,int q)
{
float div;
div=p/q;
printf("\n Division of a number=%f\n",div);
}
```

Syntax to declare a function:

```
Return_type function_name(datatype arg1,datatype arg2,…,datatype
arg n)
{
Function body;
}
```

Example:-

```
void square(int no)
{
printf("%d",no*no);
}
```

In the above example variable no is passed as an argument to the function square.this function displays square of no on screen

AND THAT IS END OF FUNCTIONS.