Function

Lecture 7

What is Function?

- A number of statements grouped in to a single logical unit is referred to as a function.
- The function main() in the program is executed first.

```
main()
{
}
```

- All other functions are executed from main which calls them directly or indirectly

Function

- There are 2(two) types of functions as:
  - Built-in Functions
  - User Defined Functions

Built in Functions : These functions are also called as 'Library functions'. These functions are provided by system. These functions are stored in library files. e.g.

- scanf()
- printf()

User Defined Functions : The functions which are created by user for program are known as 'User defined functions'.

Function

- Advantage of function
  - Facilities modular programming
  - The use of a function avoids the need for redundant programming of the same instructions
  - It is easy to locate and isolate a faculty function

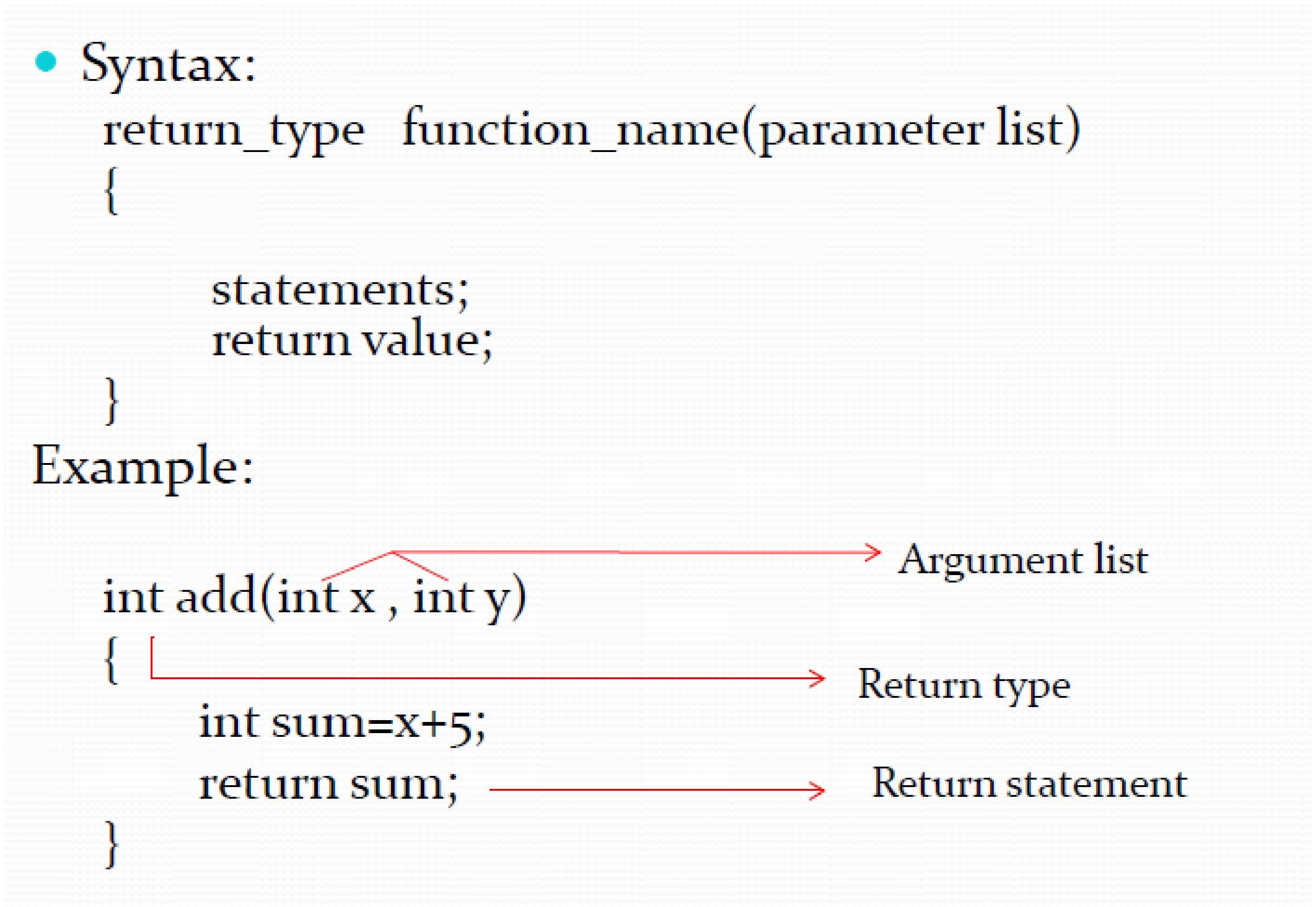
Local variable

- A local variable is a variable that is declared inside a function.
- local variable can only be used in the function where it is declared.

Global variable

- A global variable is a variable that is declared outside all function.
- a global variable can be used in all function

Function



- Syntax:

```
return_type   function_name(parameter list)
{

        statements;
        return value;
}
```

Example:

```
                                          Argument list
int add(int x , int y)
{                                        Return type
       int sum=x+5;
       return sum;                        Return statement
}
```

Function

- Function name is an identifier so function name must satisfy rules for identifiers
- Argument list
  - Contain valid variable names separated with commas
  - Argument variable receive value from calling function
- Return type
  - Specifies type of value that the function returns
- Return value
  - Example : return (expression);

```
int add(int x, int y);          →  Function declaration / function prototype

void main()
{
    int c, a=10, b=20;

    c= add(a, b);               →  Function call

    printf("Sum=%d", c);

}
int  add (int x , int y)        →  Function declarator
{
    int sum;
    sum =x+y;                      Function body           Function definition
    return sum;
}
```

Example: Addition of two number

```
 #include<stdio.h>
 int add(int,int);
 void main()
{
   int num1,num2,res;
   printf("Enter the two numbers");
   scanf("%d%d",&num1, &num2);
   res=add(num1,num2);
printf("The sum is %d ",res);
}
int add(int x,int y)
{
   int s=x+y;
   return(s);
}
```

Program to calculate maximum of three numbers using function

```
#include <stdio.h>
int maximum( int, int, int );
/* function prototype */
 int main()
{
 int a, b, c;
printf( "Enter three integers:" );
```

```
 scanf( "%d%d%d", &a, &b, &c );
 printf( "Maximum is: %d\n",maximum(a,b,c)),
 maximum( a, b, c );
 return 0;
 }
 /* Function maximum definition */
 int maximum( int x, int y, int z)
{
```

```
int max ;

if ( y > x &&y>z )
max = y;
else if ( x > z  )
max=x;
```

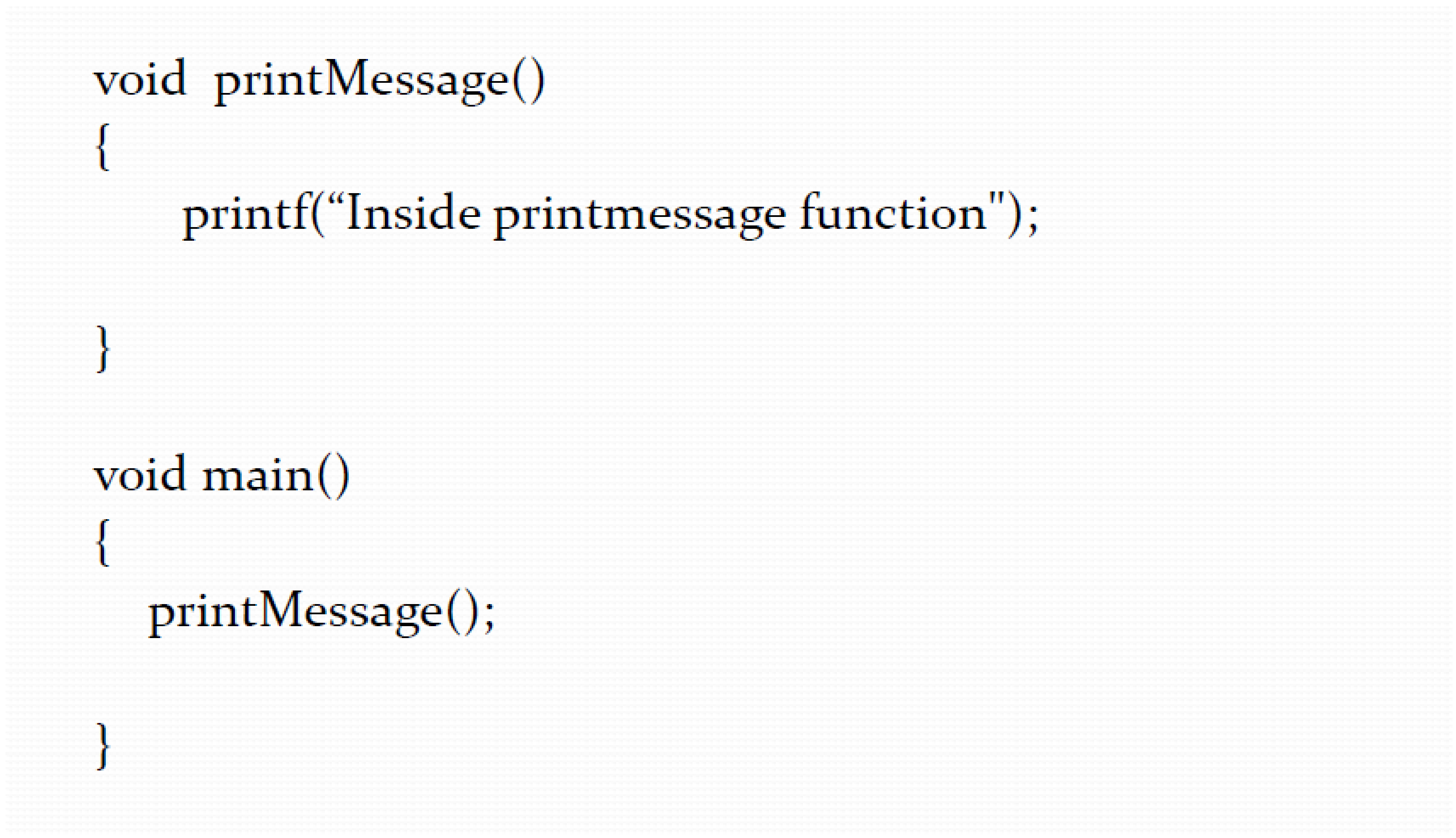
```
 else
 max=z;

 return max;
}
```

Function

- Depending on arguments and return value function can be classified as :-

  - Function with no argument and no return value
  - Function with argument but no return value
  - Function with no argument but return value
  - Function with both arguments and return value

No argument & No return value



```
void  printMessage()
{
      printf(“Inside printmessage function");


}

void main()
{
    printMessage();

}
```

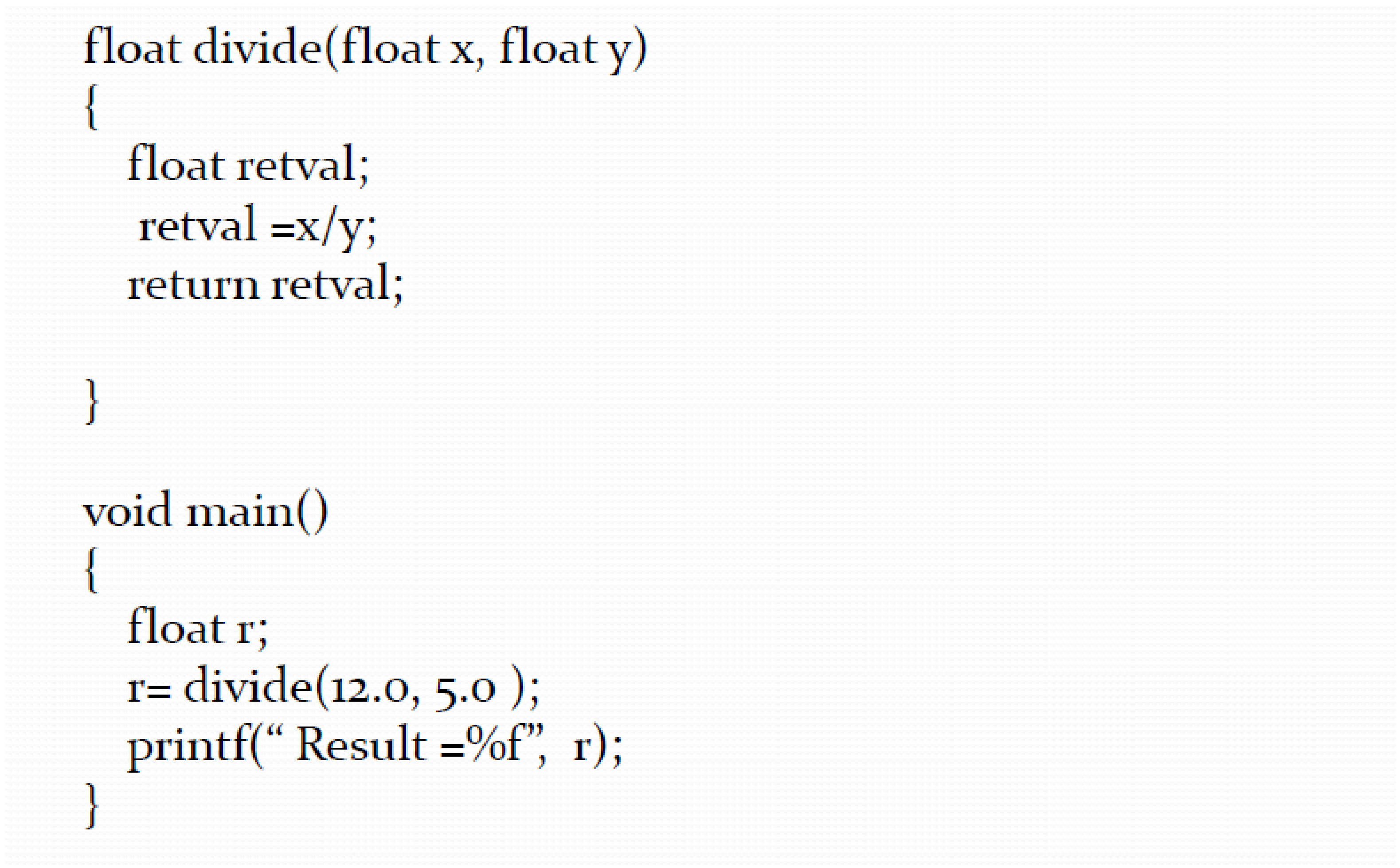
Argument & No return value

```
void  add(int x, int y)
{

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

}

void main()
{
    add(10,20);
    add(25,25);
}
```

No argument but return value

```
int mult()
{
    int x=12, y=5;

    return (x*y);

}

void main()
{
    int a;
    a= mult();
    printf("Multiply =%d", a);
}
```

Argument & Return value



```
float divide(float x, float y)
{
    float retval;
     retval =x/y;
    return retval;

}

void main()
{
    float r;
    r= divide(12.0, 5.0 );
    printf(" Result =%f",  r);
}
```

Nesting function

```
void first_func();
void second_func();
void third_func();
void main()
{
    first_func();
}
void first_func()
{
   printf(“I am in first”);
   second_func();
}
```

```
void second_func()
{
   printf(“I am in second ”);
    third_func();
}
void third_func()
{
   printf(“I am in third”);
}
```

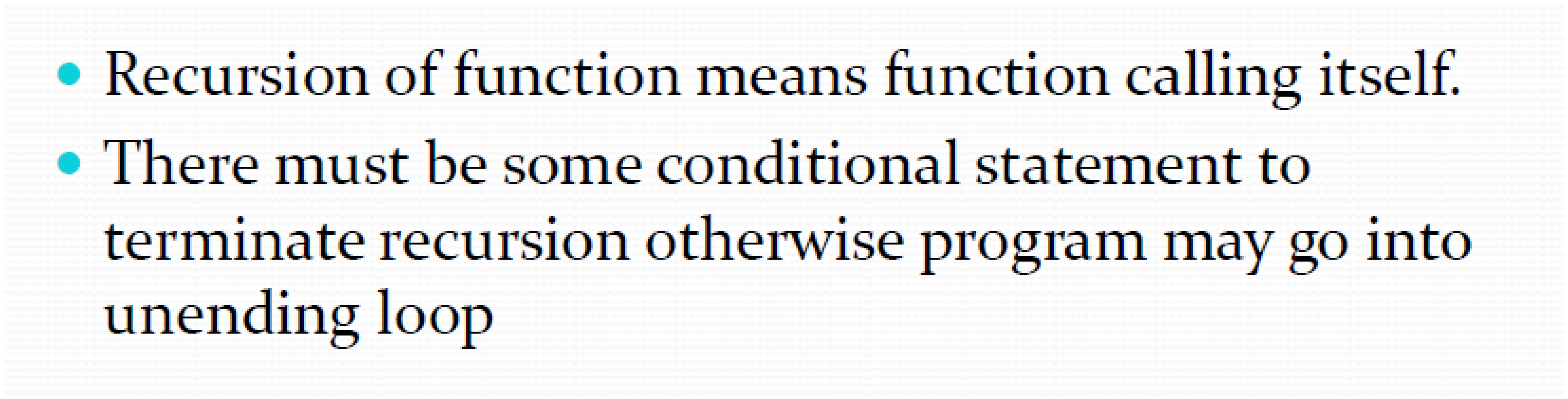
Function: Pass by value

- The value of the corresponding formal argument can be changed within the function, but the value of the actual argument will not change

Function: Pass by value

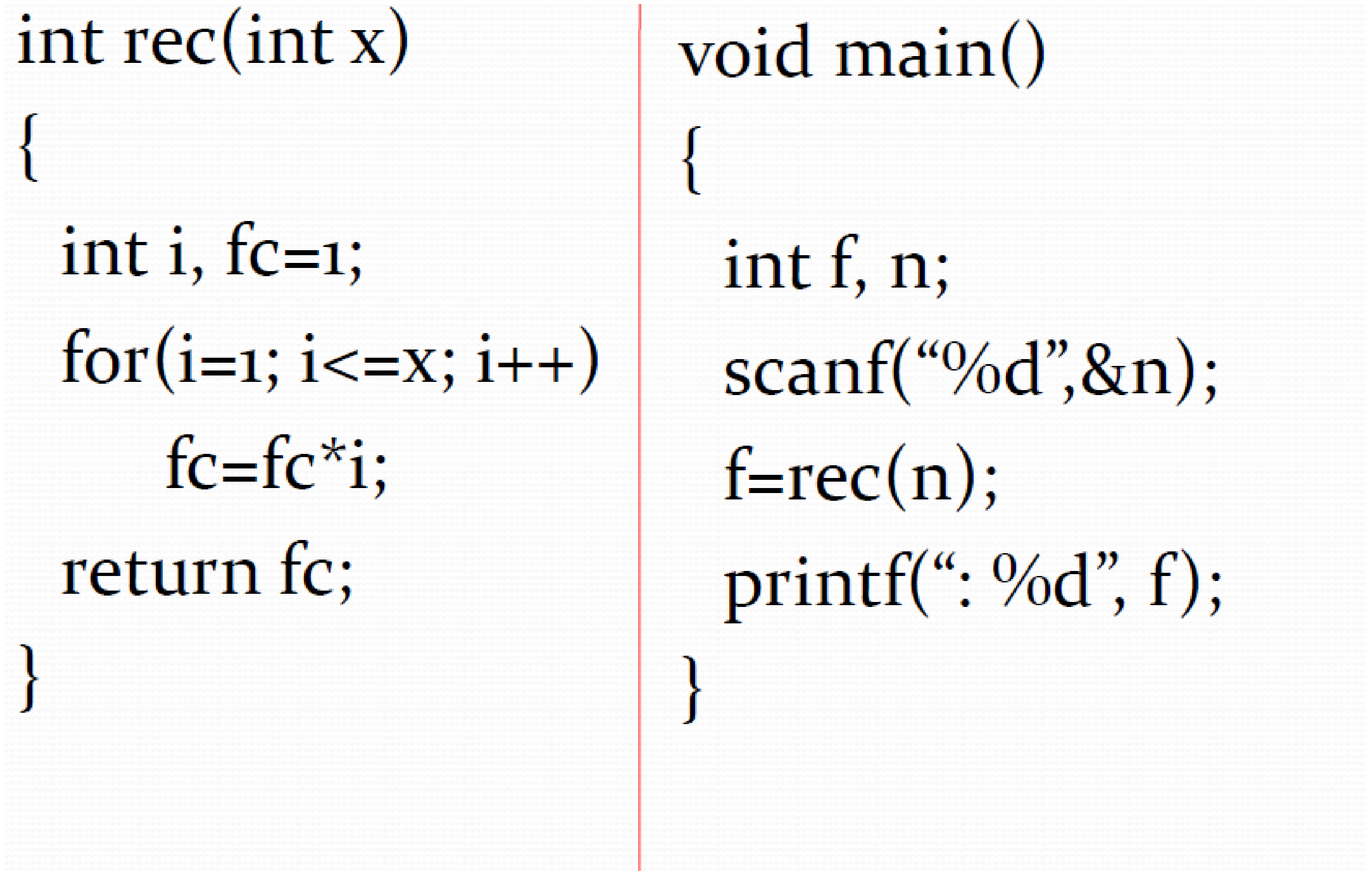
```
void change(int num)
{
    num++;
    printf(“Value in change function :%d”,num);
}

 main()
{

    int  num=10;
    change(num);
    printf(“Value in main function :%d", num);

}
```

Recursion



- Recursion of function means function calling itself.
- There must be some conditional statement to terminate recursion otherwise program may go into unending loop

Factorial (Iteration)



```
int rec(int x)
{
  int i, fc=1;
  for(i=1; i<=x; i++)
        fc=fc*i;
  return fc;
}
```

```
void main()
{
  int f, n;
  scanf(“%d”,&n);
  f=rec(n);
  printf(“: %d”, f);
}
```

Function (recursion)

```
int rec(int x)
{
  int  f;
  if(x==1)
        return (1);
  else
        f=x*rec(x-1);
  return f;
}
```

```
main()
{
  int f, n;
  scanf("%d",&n);
  f=rec(n);
  printf(“%d",f);
}
```