Function

Function

- Number of statement grouped into single logical unit which carries out specific task is referred as function
- main() is a function and this function 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

Function

- Syntax:

```
return_type   function_name(parameter list)
{

        statements;
        return value;
}
```

Example:

```
int add(int x , int y)
{
        int sum=x+5;
        return sum;
}
```

Argument list

Return type

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);

Parts of Function

```
int add(int x, int y);

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

    c= add(a, b);

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

}
int  add (int x , int y)
{
    int sum;
    sum =x+y;
    return sum;
}
```

Function declaration / function prototype

Function call

Function declarator

Function body

Function definition

Function

```
#include <stdio.h>
int  add (int x , int y)
{
    int sum;
    sum =x+y;
    return sum;
}
void main()
{
    int c, a=10, b=20;
    c= add(a,  b);

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

Formal argument

Actual argument

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 but 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 and 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);

}
```

Function with arrays

- It is possible to pass value of array to a function
- Pass by reference : the value of the actual argument will change

```
void read(int arr[], int n)
{
   int i;
   printf("Enter  value :");
   for(i=0; i<n ; i++)
   {
         scanf("%d",& arr[i]);
   }
}
```

```
int findlarge(int arr[], int
n)
{
      int i, large;
      large= arr [0];
      for(i=1; i<n ; i++)
     {
         if(myarray[i]>large)
            large=myarray[i];
      }
         return large;
}
```

```
main()
{
   int arr[4], large;
   read(arr, 4);
   large= findlarge(arr, 4);
   printf("large:%d",large);
}
```

Variable scope

- Variable scope refers to the area of a program in which a variable is visible and its value is available to use

- **Global Scope**
  - It is declare outside any functions
  - It can be used by later blocks of code:

```
int var1;
void func()
{
    printf("%d“,var1);
}
main()
{
    var1=10;
    func();

}
```

Storage class

- C has following storage class
  - Automatic variables
  - External variables
  - Static variables
  - Register variables

Automatic Variables

- Declared inside a function
- Created when function are called
- Destroyed automatically when function exists
- Also know as local or integral variable as they are local to function in which they are declared
- The optional keyword “auto” declares automatic variable explicitly

Automatic Variables

```
 void newfun()
{
    int a=10;
    a++;
    printf("%d",a);
}
main()
{

    newfun();
    printf("%d",a);    // error
}
```

External Variables

- Also know as global variable as these have global scope
- Can be access by any function in the program
- Declared outside function
- Local variable have precedence over global variable with same name in the function when local variables are declared

External Variables

```
int a=20;
void newfunc()
{
   int a=19;
   printf("%d",a);
}
```

Extern Declaration

```
void printmsg()
{
    extern int var1;
    printf("%d",var1)
}
int var1=12;
```

Although variable 'var1' has been defined after both the functions the external declaration of var1 inside the functions informs compiler that var1 is integer type defined somewhere else in the program

Static Variable

- The value persists until the end of the program
- Scope is limited to the function in which they are defined
- Can be either internal or external
- Initialized only once, when the program is compiled

Static Variable

```
void increase()
 {
    static int var2=10;
    var2++;

    printf("%d\n",var2);
}
main()
{
     increase();
     increase();
     increase();
}
```

Register Variable

- Variable is stored in register of machine
- Leads faster execution of program
- Only few variables can be placed in the register
- C will automatically convert register variables into non register variables once the limit is reached
- Declaration

```
register int count;
```

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)

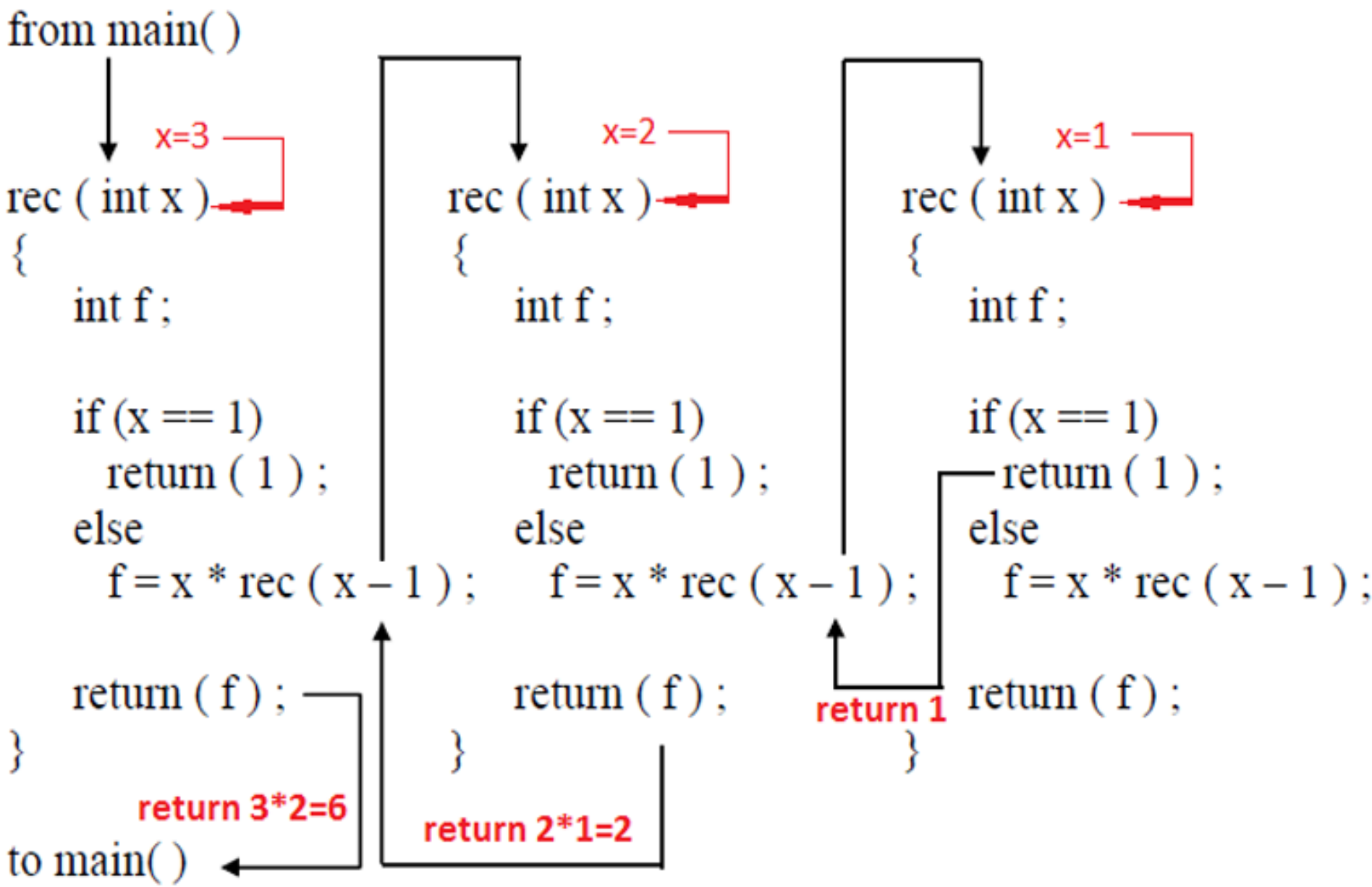
```
int rec(int x)
{
  int i, fc=1;
  for(i=1; i<=x; i++)
      fc=fc*i;
  return fc;
}
```

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

Factorial (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);
}
```


from main( )
x=3
rec (int x)
{
int f ;
if (x == 1)
return (1) ;
else
f = x * rec (x – 1) ;
return (f) ;
}
return 3*2=6
to main( )
x=2
rec (int x)
{
int f ;
if (x == 1)
return (1) ;
else
f = x * rec (x – 1) ;
return (f) ;
}
return 2*1=2
x=1
rec (int x)
{
int f ;
if (x == 1)
return (1) ;
else
f = x * rec (x – 1) ;
return (f) ;
}
return 1