Computer Programming: C

By: Ritu Devi

Function

- A function is a self-contained subprogram that is meant to do some specific, well-defined task.
- A C program consists of one or more functions. If a program has only one function then it must be the main() function.

Advantages of using function

- Modularity: A program can be divided into functions, each of which performs some specific task.
- Reusability of repeated code.
- The program becomes easily understandable, modifiable and easy to debug and test.
- Functions can be stored in a library and reusability can be achieved.

Types of Function

C programs have two types of functions:

1. Library functions :

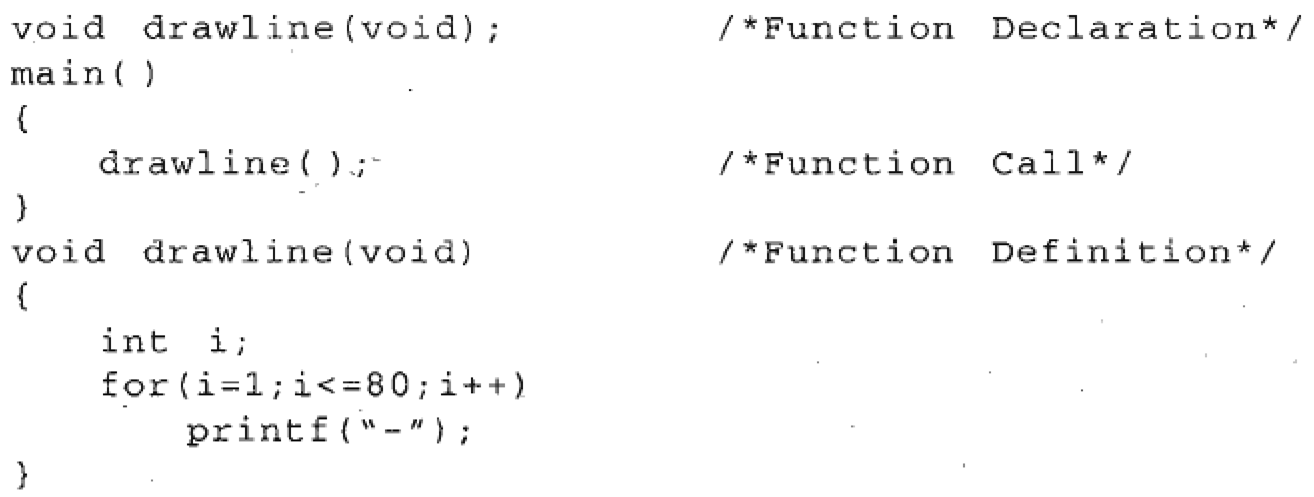
- These functions are present in the C library and they are predefined. For example sqrt( ), printf(), scanf(), printf(), strlen(), strcmp() and so on.
- To use a library function we have to include corresponding header file using the preprocessor directive #include.

2. User-defined functions:

- Users can create their own functions for performing any specific task of the program.

- To create and use these functions, we should know about these three things-

  1. Function definition .
  2. Function declaration
  3. Function call
- For example



```
void  drawline(void);              /*Function  Declaration*/
main( )
{
    drawline( );                   /*Function  Call*/
}
void  drawline(void)               /*Function  Definition*/
{
    int  i;
    for(i=1;i<=80;i++)
        printf("-");
}
```

1. Function Definition

- A function definition consists of two parts -a function header and a function body. The general syntax of a function definition is:

```
return_type   func_name( type1 arg1,type2 arg2,……… )
{
     local variables declarations;
     statement;
     ……….
     return(expression);
}
```

- The return_type is optional and if omitted, it is assumed to be int by default.
- func_name can be any valid C identifier
- The list of arguments declared in parenthesis are known as **formal arguments**, and used to accept values.

2. Function Call

- The function definition describes what a function can do, but to actually use it in the program the function should be called somewhere.
  - func_name(arg1, arg2, arg3 …);
- The arguments arg 1, arg2, ,…are called **actual arguments**.
- Here func_name is known as the **called function** while the function in which this function call is placed is known as the **calling function.**
- The code of a function is executed only when it is called by some other function.
- A function can be called more than once, so the code is executed each time it is called.

3. Function Declaration

- The function declaration is also known as the function prototype and it informs the compiler about the following three things-

  1.Name of the function.

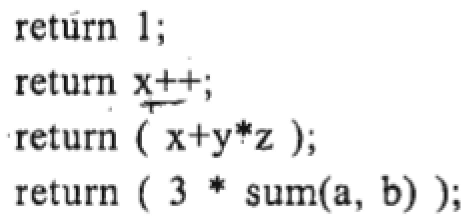
  2. Number and type of arguments received by the function.

  3. Type of value returned by the function.

- The general syntax is:
  - return_type func_name(typel argl, type2 arg2, .. , ...);
  - The names of the arguments are optional.

return statement

- Can appear anywhere inside the body of the function. There are two ways in which it can be used- .
  - return;
  - return (expression);
- Here ***return*** is keyword.
- expression may be any constant, variable, expression or even any other function call(which returns a value).For e.g.,



```
return 1;
return x++;
return ( x+y*z );
return ( 3 * sum(a, b) );
```

```
#include<stdio.h>
void funct(int age,float ht);
main()
{
    int age;
    float ht;
    printf(“Enter age and height: ” );
    scanf(“%d %f”, &age, &ht);
    funct(age,ht);
}
void funct(int age,float ht)
{
    if(age>25)
    {
        printf(“Age should be less than 25\n”);
        return;
    }
    if(ht<5)
    {
        printf(“Height should be more than 5\n”);
        return;
    }
    printf(“Selected\n”);
}
```

Function Arguments

- The calling function sends some values to the called function for communication; these values are called arguments or parameters.
- **Actual arguments:** The arguments which are mentioned in the function call are known as actual arguments
- For Example:
  - Func(x);
  - Func(22,43);
  - Func(a*b,c+d);
  - Func(1,2,sum(x,y));

- **Formal arguments:** The name of the arguments, which are mentioned in the function definition are called formal or dummy arguments since they are used just to hold the values that are sent by the calling function.

- The *order, number* and *type* of actual arguments in the function call should match with the order, number and type of formal arguments in the function definition,

- Program to understand formal and actual arguments.

```
main( )
{
    int  a=6,b=3;
    func(a,b);
    func(15,4);
    func(a+b,a-b);
}
func(int  a,int  b)
{
    printf("a  =  %d      b  =  %d\n",a,b);
}
```

Output:

```
a = 6       b = 3
a = 15      b = 4
a = 9       b = 3
```

- The main() function is a user defined function but the name, number and type of arguments are predefined in the language. The operating system calls the main() function and main() returns a value of integer type to the operating system.

- The definition, declaration and call of main( ) function-
  - Function Declaration - By the C compiler.
  - Function Definition - By the programmer.
  - Function Call - By the operating system.

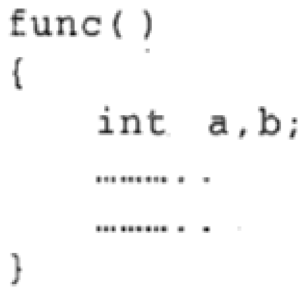
Scope of data in C

- A scope is a region of the program, and the scope of variables refers to the area of the program where the variables can be accessed after its declaration.

Local, Global And Static Variables

Local Variables/ Local scope:

- The variables that are defined within the body of a function or a block, are local to that function block only and are called local variables. For example:

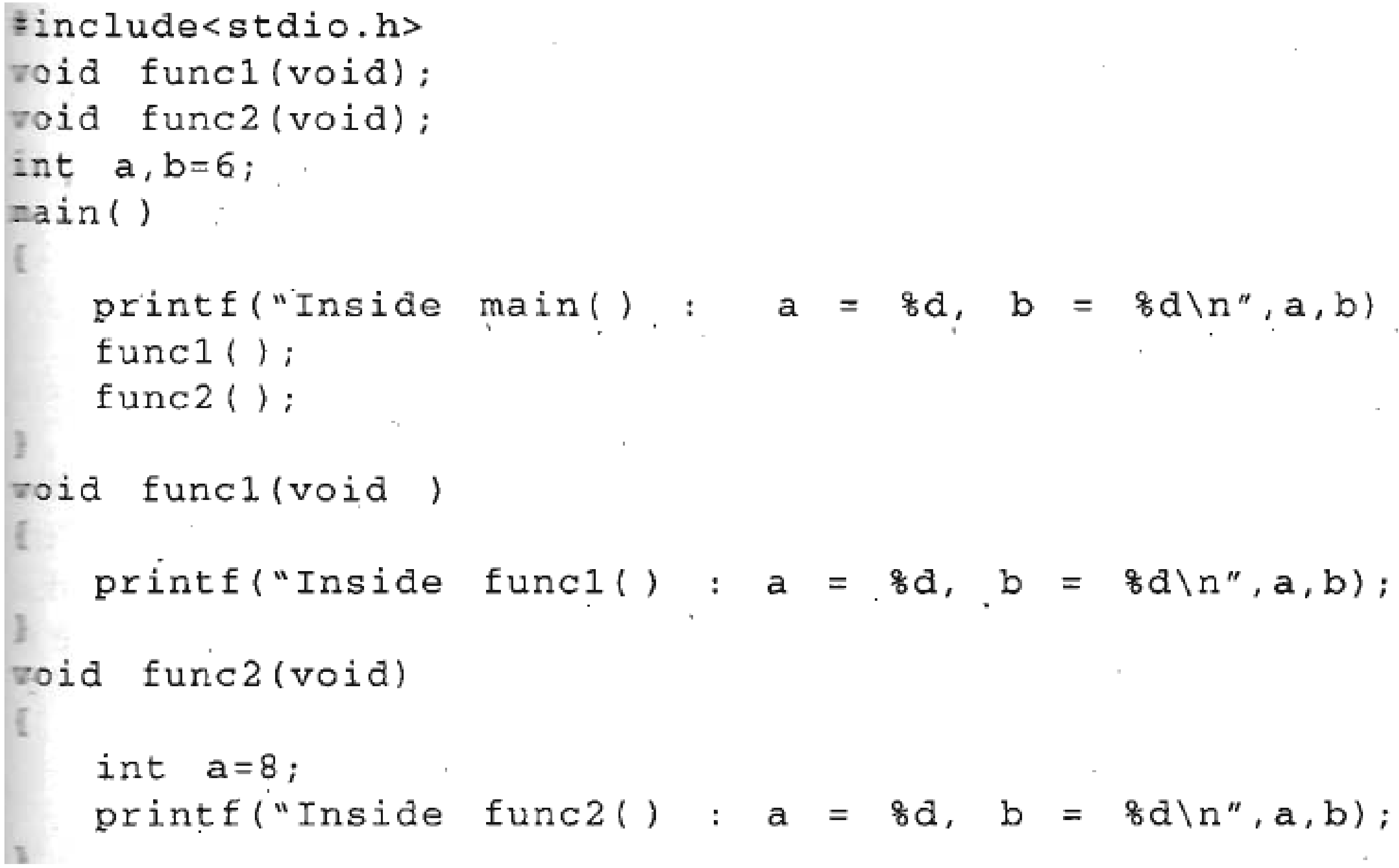


```
func( )
{
    int  a,b;
    ……… . .
    ……… . .
}
```

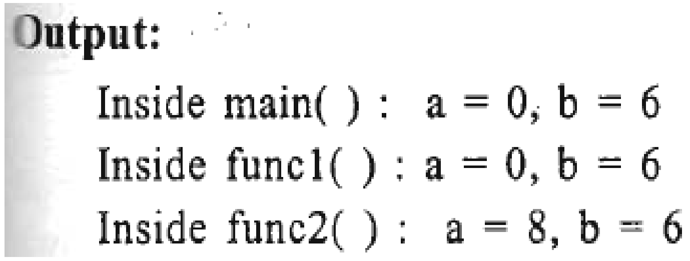
- Local variables can be used only in those functions or blocks, in which they are declared.

Global Variables/Global scope:

- The variables that are defined outside any function are called global variables: All functions in the program can access and modify global variables.
- Global variables are automatically initialized to 0 at the time of declaration.
- If there is a conflict between a local and global variable, the local variable gets the precedence.



```
#include<stdio.h>
void  func1(void);
void  func2(void);
int  a,b=6;
main()
{
    printf(“Inside  main()  :    a  =  %d,  b  =  %d\n”,a,b)
    func1();
    func2();
}
void  func1(void  )
{
    printf(“Inside  func1()  :  a  =  %d,  b  =  %d\n”,a,b);
}
void  func2(void)
{
    int  a=8;
    printf(“Inside  func2()  :  a  =  %d,  b  =  %d\n”,a,b);
}
```



Output:

```
    Inside main( ) :  a = 0, b = 6
    Inside func1( ) : a = 0, b = 6
    Inside func2( ) :  a = 8, b = 6
```

- **Static Variables:** Static variables are declared by writing keyword static in front of the declaration
  - static data_type var_name;
- A static variable is initialized only once and the value of a static variable is retained between function calls. If a static variable is not initialized then it is automatically initialized to 0.

```
#include<stdio.h>
void func(void);
main()
{
    func( );
    func( );
    func( );
}
void func(void)
{
    int a=10;
    static int b=10;
    printf("a = %d    b = %d\n",a,b);
    a++;
    b++;
}
```

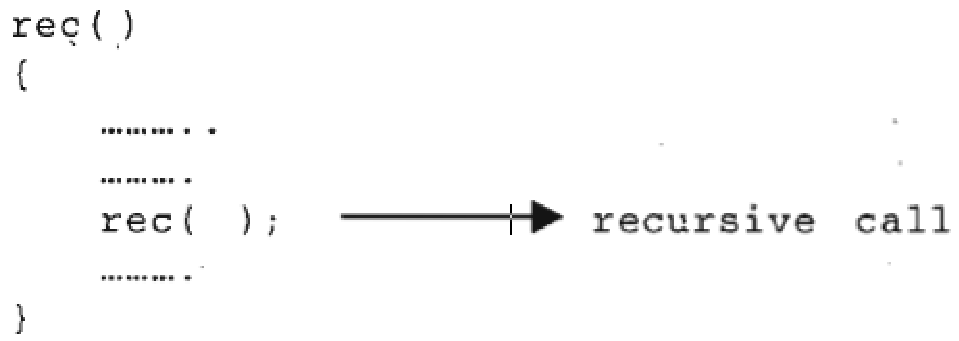
Output:

```
    a = 10   b = 10
    a = 10   b = 11
    a = 10   b = 12
```

Recursion

- A function can call itself. Such a process is called recursion.
- The function that calls itself inside function body again and again is known as recursive function.
- In recursion the calling function and the called function are same.

```
 main()
{
        .......
      rec();
         .......
}
```



```
rec()
{
    ........
    ........
    rec( );
    ........
}
```

- There should be a terminating condition to *stop* recursion, otherwise rec( ) will keep on calling itself infinitely and will never return.

- Using recursive algorithm, certain problems can be solved quite easily.
- There are some problems which can be solved using recursion are:

  (i) Factorial.

  (ii) Power.

  (iii) Fibonacci numbers.

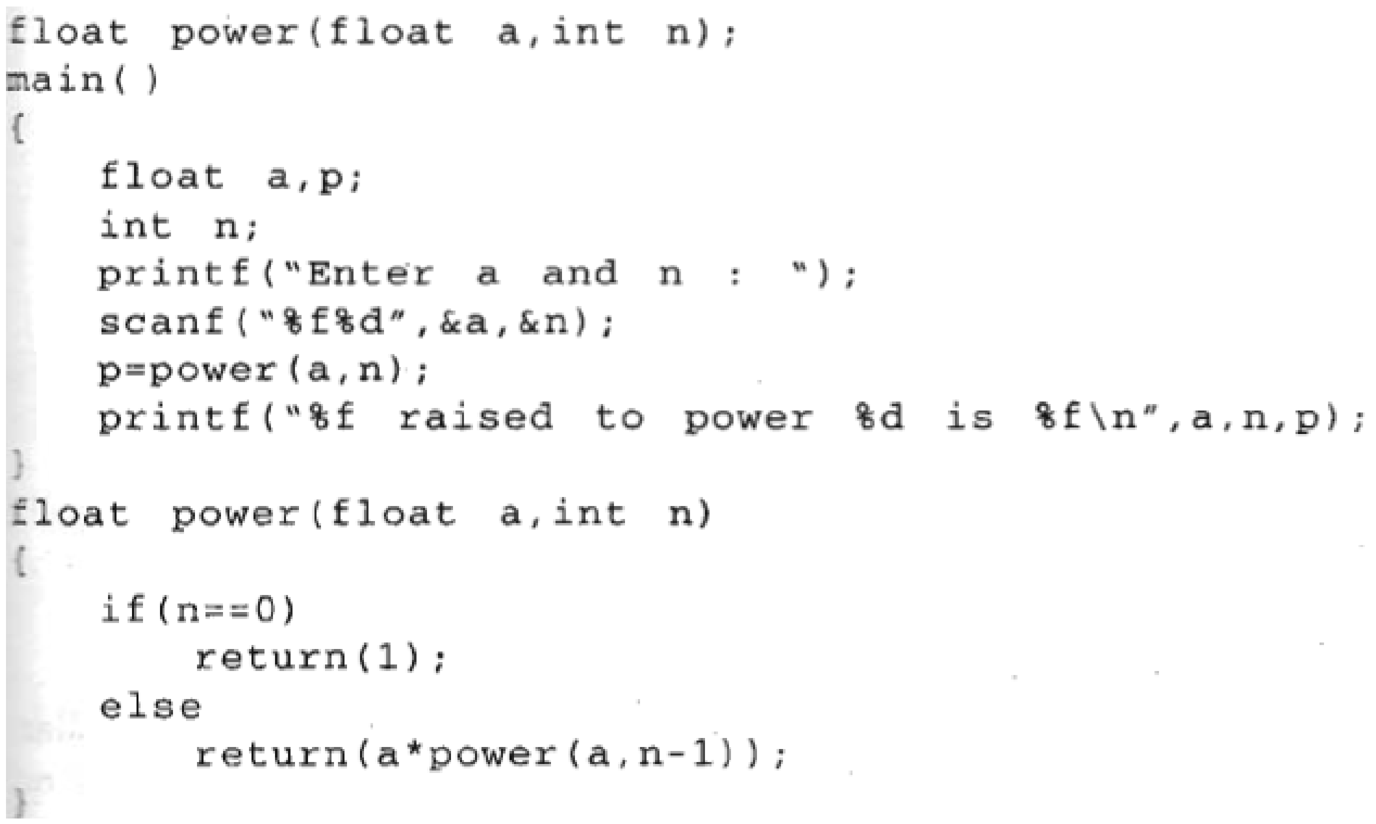
  (iv) Inorder/Preorder/Postorder Tree Traversals.

  (v) DFS of Graph.

* Factorial of a given number using recursion:

```
long  fact(int  n);
main()
{
    int  num;
    printf("Enter  a  number  :  ");
    scanf("%d",&num);
    printf("Factorial  of  %d  is  %ld\n",num,fact(num));
}
long  fact(int  n)
{
    if  (n==0)
        return(1);
    else
        return(n*fact(n-1));
}
```

- Program to raise a floating point number to a positive integer using recursion.



```c
float  power(float  a,int  n);
main( )
{
    float  a,p;
    int  n;
    printf("Enter  a  and  n  :  ");
    scanf("%f%d",&a,&n);
    p=power(a,n);
    printf("%f  raised  to  power  %d  is  %f\n",a,n,p);
}
float  power(float  a,int  n)
{
    if(n==0)
        return(1);
    else
        return(a*power(a,n-1));
}
```

- Program to generate fibonacci series using recursion

```
main()
{
    int nterms, i;
    printf("Enter number of terms : ");
    scanf("%d", &nterms);
    for(i=0;i<nterms;i++)
        printf("%d   ",fib(i));
    printf("\n");
}
int fib(int n)          /*recursive function that returns nth term of
                        fibonacci series*/
{
    if(n==0||n==1)
        return(1);
    else
        return(fib(n-1)+fib(n-2));
}
```

Advantages and disadvantages of using recursion over iteration.

Advantages:

- provides a clean and simple way to write code.
- Some problems are inherently recursive like tree traversals, DFS search etc. For such problems, it is preferred to write recursive code.

Disadvantages:

- Every recursive program can be written iteratively and vice versa is also true.
- The recursive program has greater space requirements than iterative program as all functions will remain in the stack until the condition remains true.
- It also has greater time requirements because of function calls and returns overhead.

Pointer and Functions

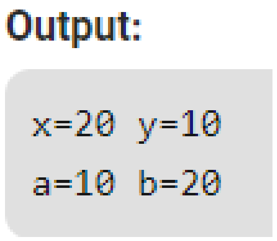
Arguments can be passed in two ways:

- Call by value
- Call by reference

Call by value	**Call by reference**
Only the values of the arguments are passed to the function	Address of the arguments are passed to the function
Any changes made to the formal arguments do not change actual arguments.	Any changes made to the formal arguments change the actual arguments also.

Call by value

```
int main()
{
   int a = 10, b = 20;
   swap(a, b);
   printf("a=%d b=%d\n", a, b);
   return 0;
}
void swap(int x, int y)
{
   int t= x;
   x = y;
     y = t;
   printf("x=%d y=%d\n", x, y);
}
```

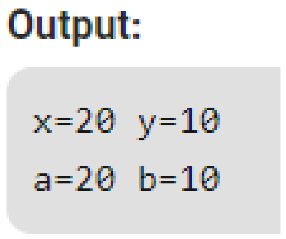


Output:

```
x=20 y=10
a=10 b=20
```

Call by reference

```
int main()
{
    int a = 10, b = 20;
    swap(&a, &b);
    printf("a=%d b=%d\n", a, b);
    return 0;
}
void swap(int *x, int *y)
{
    int t= *x;
        *x =*y;
        *y = t;
    printf("x=%d y=%d\n", *x, *y);
}
```

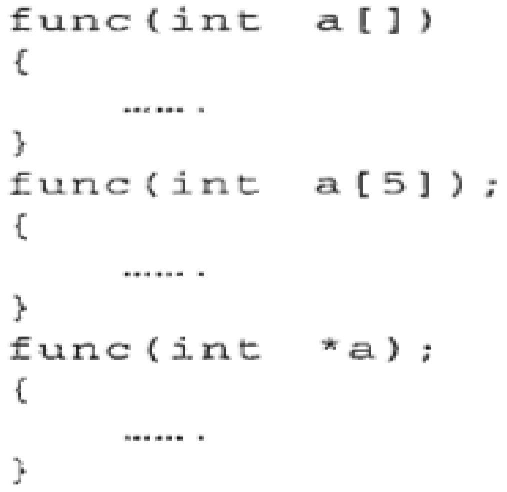


Output:

```
x=20 y=10
a=20 b=10
```

Passing an array to a Function

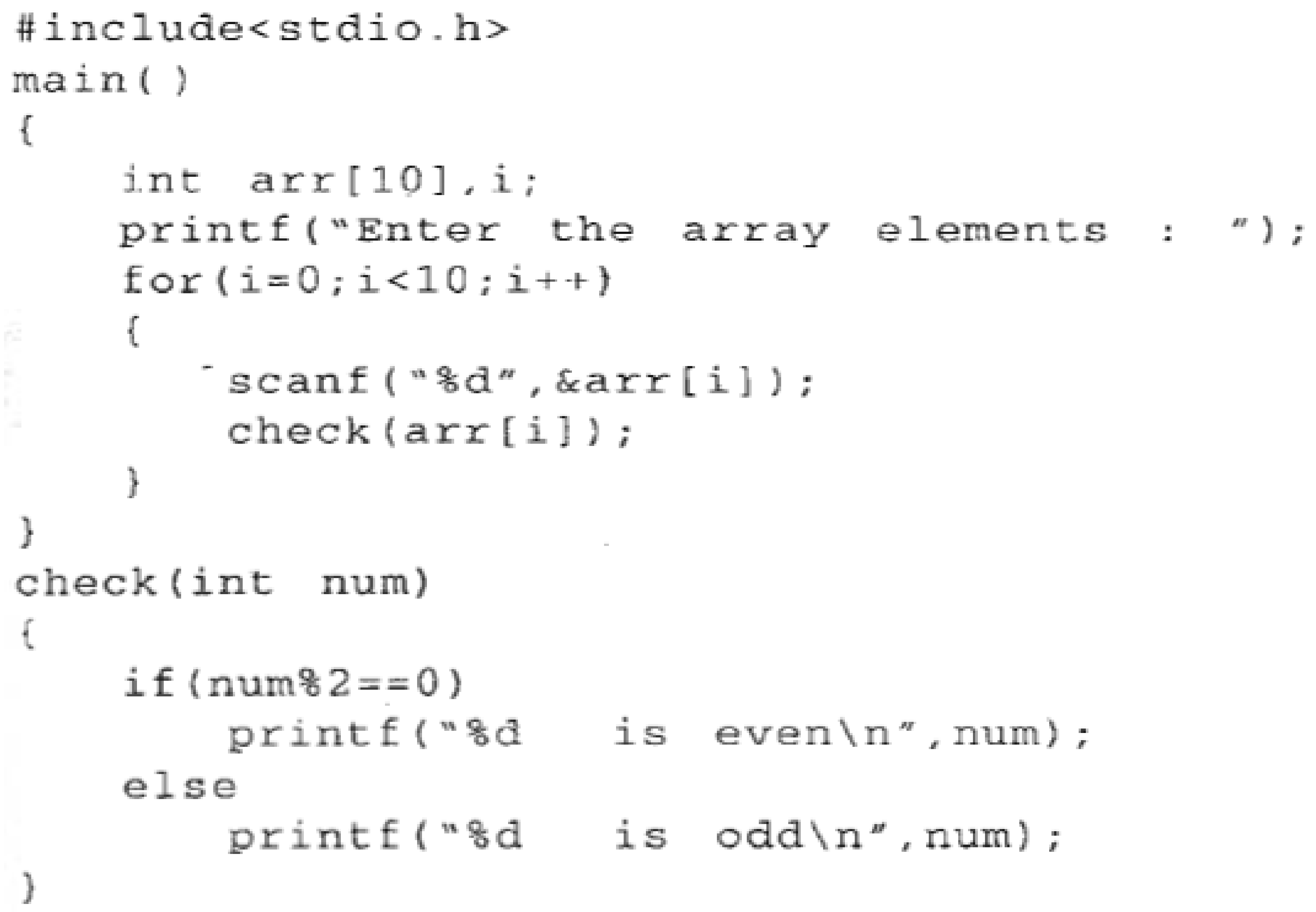
- When an array is passed to a function, the changes made inside the function affect the original array.
- There are three ways of declaring the formal parameter, which has to receive the array.



```
func(int  a[])
{
      ......
}
func(int  a[5]);
{
      ......
}
func(int  *a);
{
      ......
}
```

- In all the three cases the compiler reserves space only for a pointer variable inside the function.

Program to pass array elements to a function



```
#include<stdio.h>
main( )
{
     int  arr[10],i;
     printf(“Enter  the  array  elements  :  ”);
     for(i=0;i<10;i++)
     {
          scanf(“%d”,&arr[i]);
          check(arr[i]);
     }
}
check(int  num)
{
     if(num%2==0)
          printf(“%d   is  even\n”,num);
     else
          printf(“%d   is  odd\n”,num);
}
```

- passing an entire array to a function:

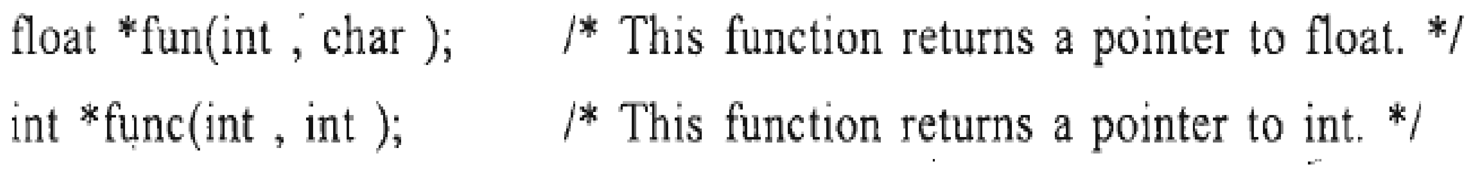
```
main( )
 {
     int num[ ] = { 24, 34, 12, 44, 56, 17 } ;
      dislpay ( &num[0], 6 ) ;
}
display ( int *j, int n )
 {
     int i ;
     for ( i = 0 ; i <= n - 1 ; i++ )
     {
       printf ( "\nelement = %d", *j ) ;
       j++ ;
     }
}
```

- The following two function calls are same:

```
display ( &num[0], 6 ) ;
display ( num, 6 ) ;
```

Function Returning Pointer

- We can have a function that returns a pointer. For e.g., Function declaration can be given as:



```
float *fun(int , char );      /* This function returns a pointer to float. */
int *func(int , int );        /* This function returns a pointer to int. */
```

- While returning a pointer, make sure that the memory address returned by the pointer will exist even after the termination of function.

```
int *fun(int *p,int n);
main()
{
    int arr[10]={1,2,3,4,5,6,7,8,9,10},n,*ptr;
    n=5;
    ptr=fun(arr,n);
    printf("Value of arr = %u, Value of ptr = %u, value of *ptr = %d\n",
    arr,ptr,*ptr);
}
int *fun(int *p,int n)
{
    p=p+n;
    return p;
}
```

Output:

```
    Value of arr = 65104, Value of ptr = 65114, Value of *ptr = 6
```

Returning More Than One Value From A Function

- we can return only one value from a function through return statement. This limitation can be overcome by using call by reference.

```
#include<stdio.h>
main()
{
    int  a,b,sum,diff,prod;
    a=6;
    b=4;
    func(a,b,&sum,&diff,&prod);
    printf("Sum = %d, Difference = %d, Product = %d\n",sum,diff,prod);
}
func(int x,int y,int *ps,int *pd,int *pp)
{
    *ps=x+y;
    *pd=x-y;
    *pp=x*y;
}
```

Output:

```
    Sum = 10, Difference = 2 , Product = 24
```

Command Line Arguments

- The function main() can accept two parameters. The definition of main() when it accepts parameters can be written as-

```
main(int argc, char *argv [ ])
{
                    ....
        ....
}
```

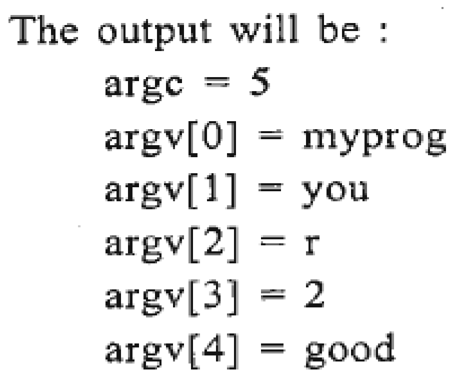
- These parameters are conventionally named argc(argument counter) and argv(argument vector), and are used to access the arguments supplied at the command prompt

- Suppose the program is compiled and executable file is created that is myprog.exe.
- For example myprog consist of the following code.

```
main (int argc, char * argv [ ])
{       int i;
        printf("argc=%d\n",argc) ;
        for(i=0;i<argc;i++)
        printf("argv[%d]=%s\n",i;argv[i]);
}
```

- Now the program is executed at the command prompt, and arguments are supplied to it as

```
  myprog you r 2 good
```



```
The output will be :
      argc = 5
      argv[0] = myprog
      argv[1] = you
      argv[2] = r
      argv[3] = 2
      argv[4] = good
```

- The parameters **argc** represents the number of arguments on command line. Total five arguments are supplied at the command prompt(including program name).
- All the arguments supplied at the command line are stored internally as strings and their addresses stored in the array of pointers named **argv**.