# Chapter 5

# Function

Er. Pratikshya Shrestha

# Contents

6.1 Function

6.2 Structure of Function

6.3 User-defined and Built-in function

6.4 Components of a function

6.5 Classification of function

6.6 Ways of passing arguments to Function

6.7 Scope of variable

6.8 Storage classes in C

6.9 Returning multiple values by Function

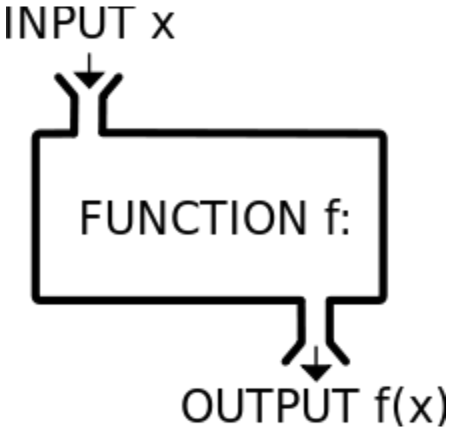
6.10 Passing arrays to function

6.11 Passing structures to function

6.12 pre-processor directives

# 6.1 Function

- A function is a self-contained block of statements that performs a particular specified task in a program.
- When the functions are called through the *main()* or other calling procedure, the functions perform their task.



# Advantages of function

- Manageability
- Code Reusability
- Logical clarity
- Minimize workload

# 6.2 User-defined and Library Function

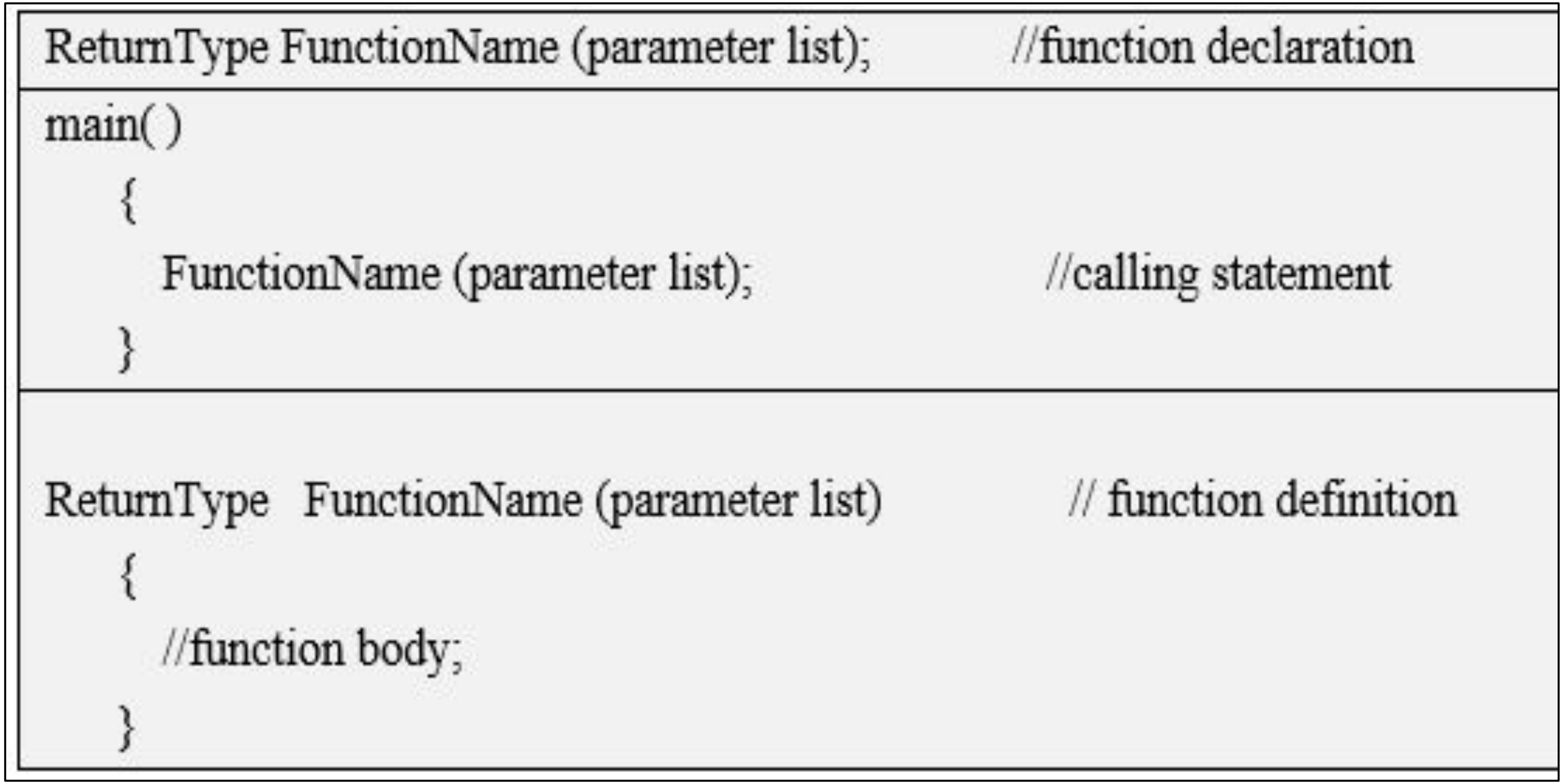
**User-defined function:**

- Defined by user
- User put the name, return-type and arguments themselves.
- Eg:
  - add()
  - main()

**Library function:**

- Built-in functions
- Already available functions in C-library.
- Eg:
  - printf( )
  - scanf()
  - sqrt()
  - pow()

# 6.3 Structure of a function

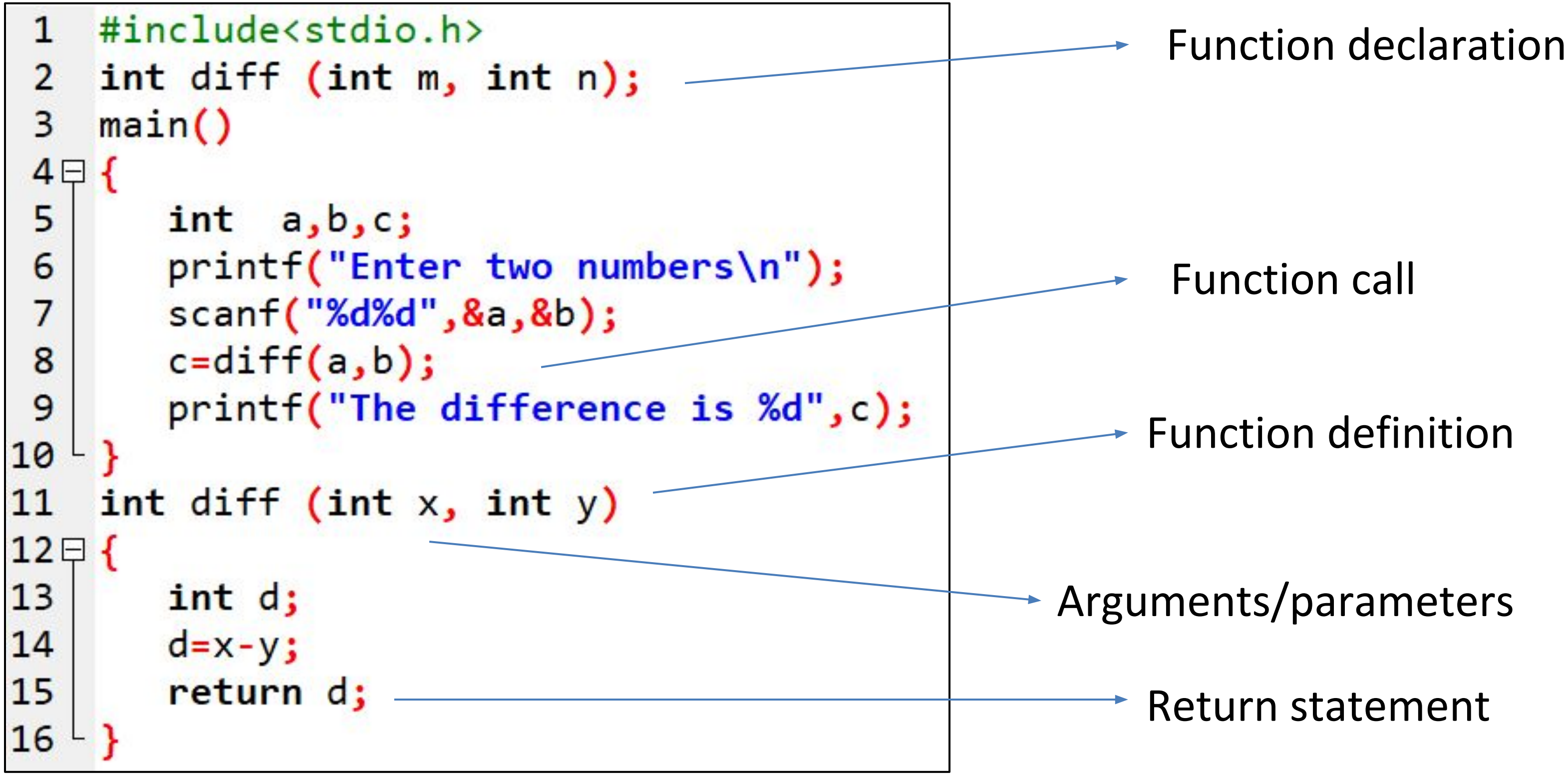


```
ReturnType FunctionName (parameter list);          //function declaration
main( )
    {
       FunctionName (parameter list);                //calling statement
    }

ReturnType   FunctionName (parameter list)            // function definition
    {
       //function body;
    }
```

# 6.4 Components of a function

1. Function declaration
2. Function definition
3. Function call
4. Function arguments/parameters
5. Return statement

# 6.4 Components of a function (Contd.)



```
#include<stdio.h>
int diff (int m, int n);
main()
{
    int  a,b,c;
    printf("Enter two numbers\n");
    scanf("%d%d",&a,&b);
    c=diff(a,b);
    printf("The difference is %d",c);
}
int diff (int x, int y)
{
    int d;
    d=x-y;
    return d;
}
```

Function declaration

Function call

Function definition

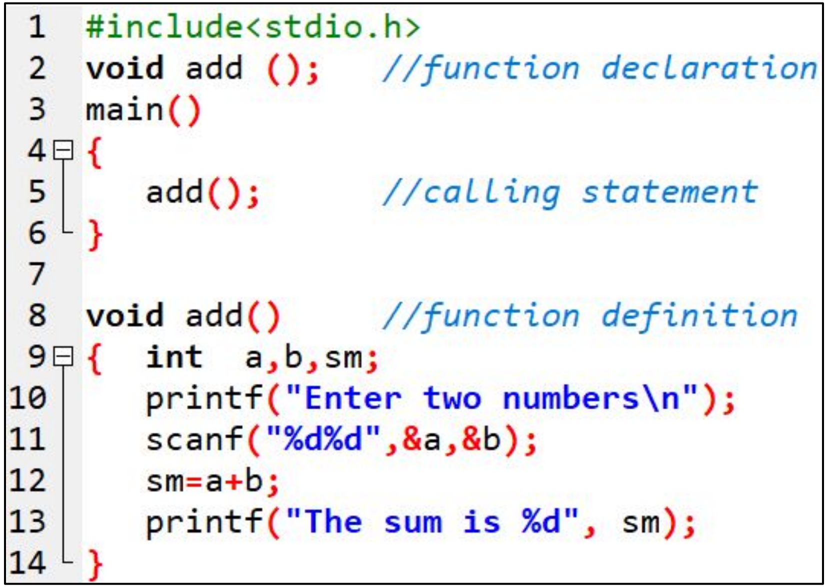
Arguments/parameters

Return statement

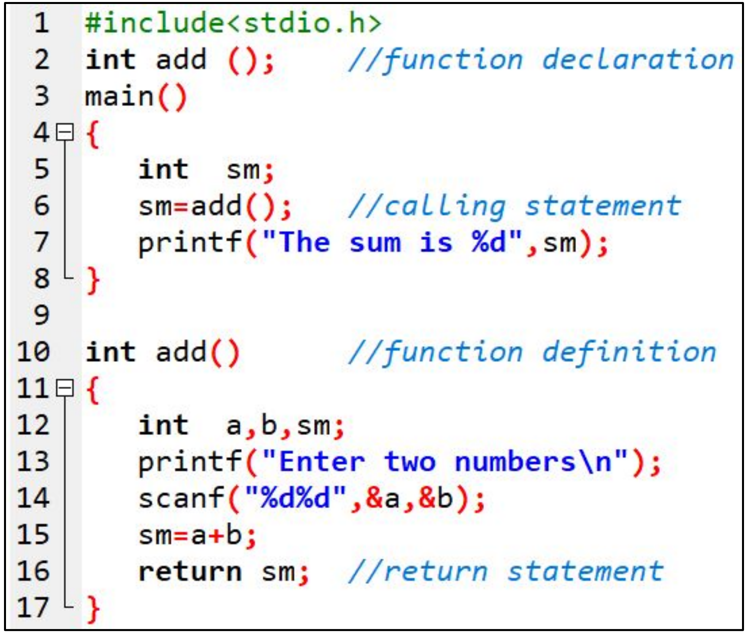
# 6.5 Classification of Function

1. Function with ***no arguments*** and ***no return value***
2. Function with ***no arguments*** but ***return value***
3. Function with ***arguments*** but ***no return value***
4. Function with ***both arguments*** and ***return value***

# 6.5 Classification of Function (Contd.)



```c
#include<stdio.h>
void add ();    //function declaration
main()
{
    add();          //calling statement
}

void add()          //function definition
{   int   a,b,sm;
    printf("Enter two numbers\n");
    scanf("%d%d",&a,&b);
    sm=a+b;
    printf("The sum is %d", sm);
}
```

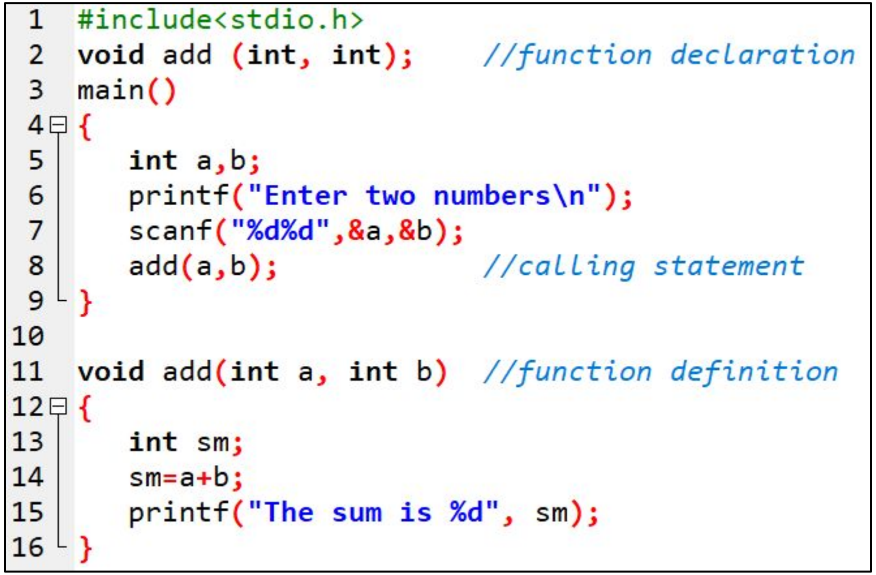


```c
#include<stdio.h>
int add ();     //function declaration
main()
{
    int  sm;
    sm=add();   //calling statement
    printf("The sum is %d",sm);
}

int add()       //function definition
{
    int  a,b,sm;
    printf("Enter two numbers\n");
    scanf("%d%d",&a,&b);
    sm=a+b;
    return sm;  //return statement
}
```

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

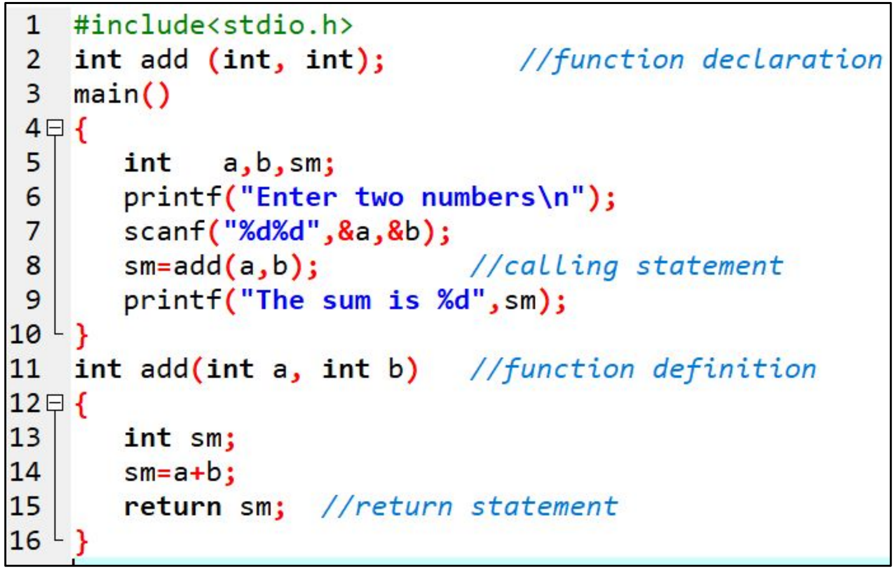
2. Function with ***no arguments*** but ***return value***

# 6.5 Classification of Function (Contd.)



```c
#include<stdio.h>
void add (int, int);     //function declaration
main()
{
    int a,b;
    printf("Enter two numbers\n");
    scanf("%d%d",&a,&b);
    add(a,b);                 //calling statement
}

void add(int a, int b)  //function definition
{
    int sm;
    sm=a+b;
    printf("The sum is %d", sm);
}
```

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



```c
#include<stdio.h>
int add (int, int);          //function declaration
main()
{
    int   a,b,sm;
    printf("Enter two numbers\n");
    scanf("%d%d",&a,&b);
    sm=add(a,b);          //calling statement
    printf("The sum is %d",sm);
}
int add(int a, int b)   //function definition
{
    int sm;
    sm=a+b;
    return sm;  //return statement
}
```

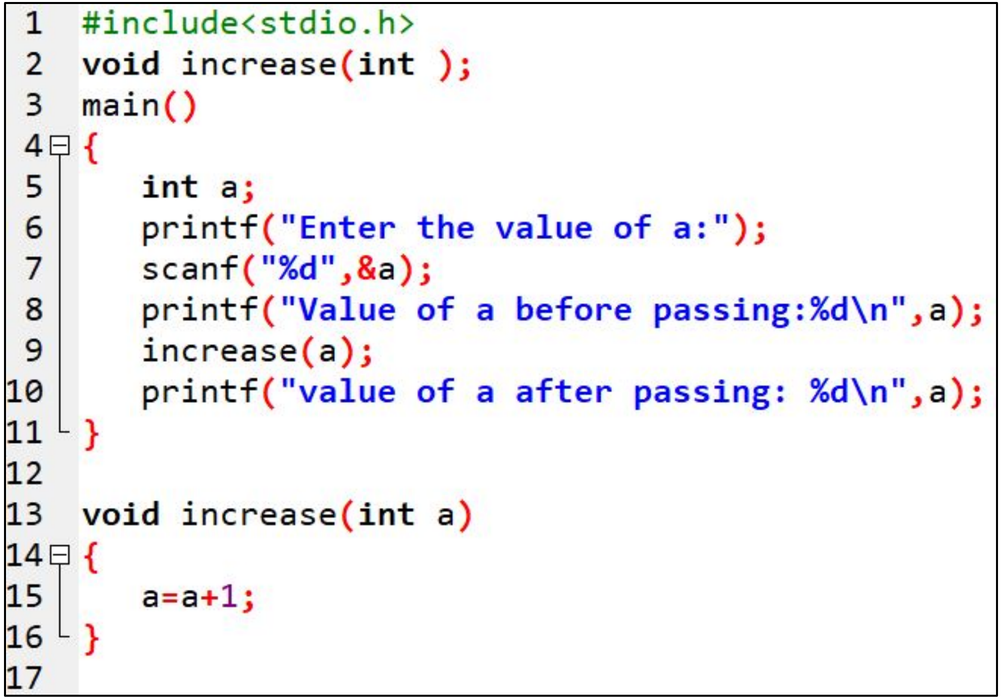
4. ***both arguments*** and ***return value***

# 6.6 Ways of passing arguments to the function

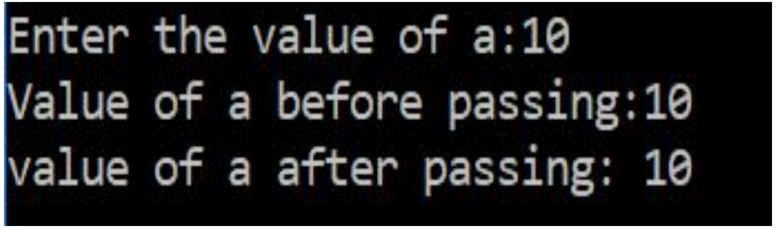
- Passing by Value
- Passing by reference

# Passing by Value

- Only the value is passed
- Any change made in the called function does not make change in the calling function.
- One-way transfer of information
- Advantage: original value remains unchanged
- Disadvantage: only one return value is possible at a single time



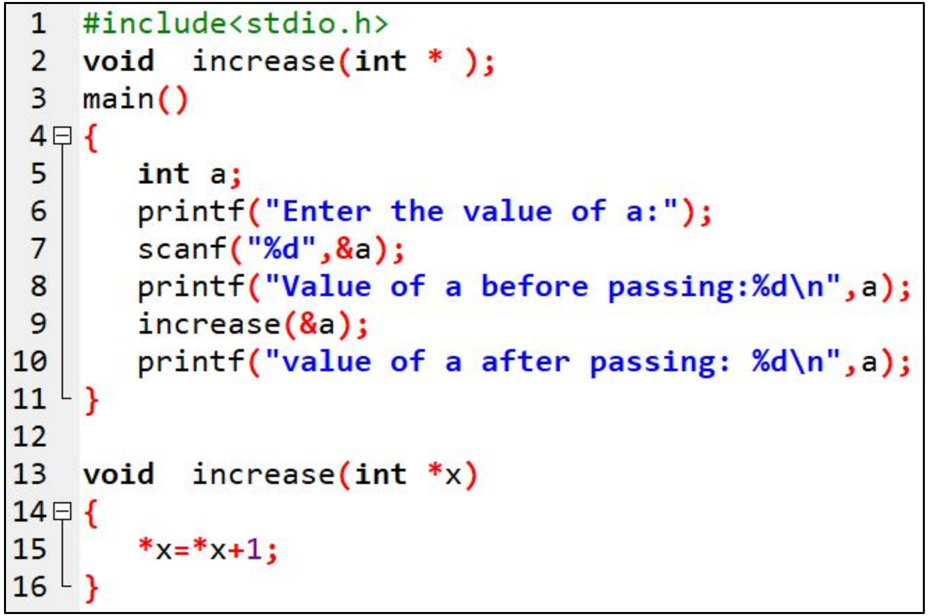
```c
#include<stdio.h>
void increase(int );
main()
{
    int a;
    printf("Enter the value of a:");
    scanf("%d",&a);
    printf("Value of a before passing:%d\n",a);
    increase(a);
    printf("value of a after passing: %d\n",a);
}

void increase(int a)
{
    a=a+1;
}
```



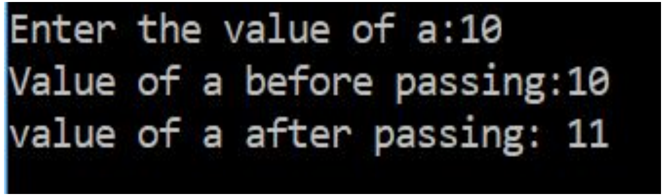
```
Enter the value of a:10
Value of a before passing:10
value of a after passing: 10
```

# Passing by Reference

- The address itself is passed
- Any change made in the called function makes change in the calling function.
- Two-way transfer of information
- Advantage: multiple value return possible
- Disadvantage: Alteration of original value



```
#include<stdio.h>
void  increase(int * );
main()
{
    int a;
    printf("Enter the value of a:");
    scanf("%d",&a);
    printf("Value of a before passing:%d\n",a);
    increase(&a);
    printf("value of a after passing: %d\n",a);
}

void  increase(int *x)
{
    *x=*x+1;
}
```

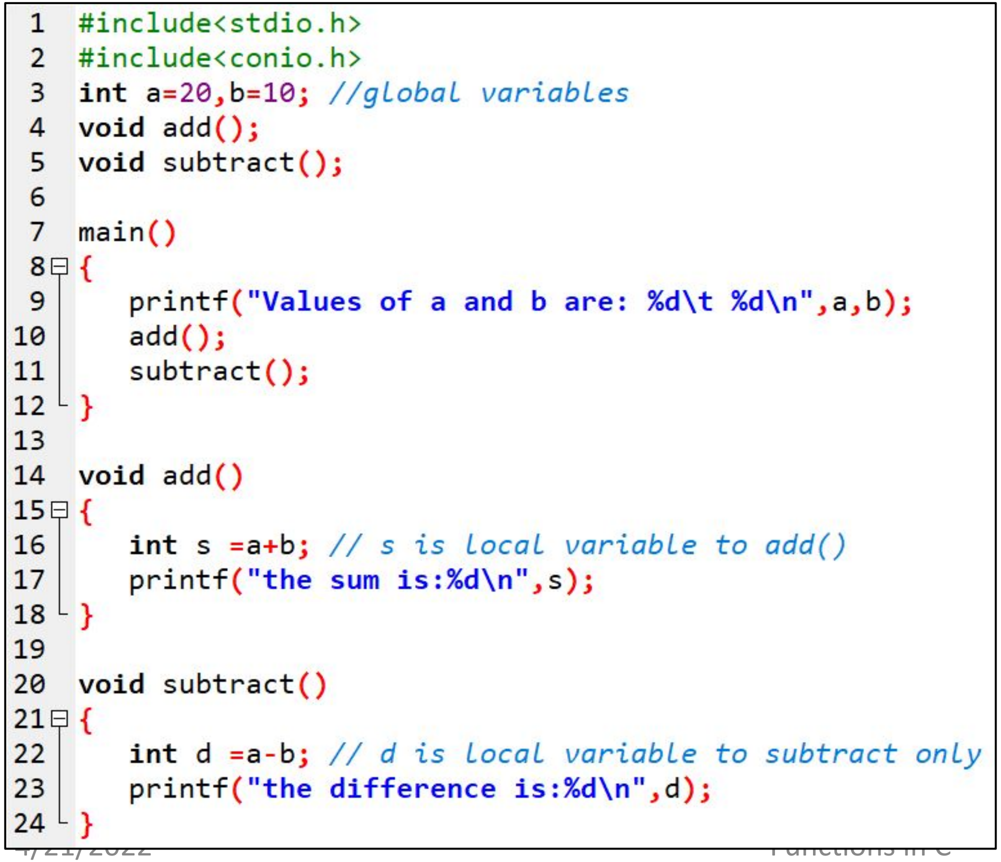


```
Enter the value of a:10
Value of a before passing:10
value of a after passing: 11
```

# 6.7 Scope of variable

- Scope of variables refers to the region of the program where the defined variables can have its existence, and beyond that region the variable cannot be accessed.
- Mainly, there are two scopes of variable in C.
  - Local Scope
    - Variable scope confined within  certain block
  - Global Scope
    - Variable scope is wide throughout the program

# Global and Local scope



```c
#include<stdio.h>
#include<conio.h>
int a=20,b=10; //global variables
void add();
void subtract();

main()
{
    printf("Values of a and b are: %d\t %d\n",a,b);
    add();
    subtract();
}

void add()
{
    int s =a+b; // s is local variable to add()
    printf("the sum is:%d\n",s);
}

void subtract()
{
    int d =a-b; // d is local variable to subtract only
    printf("the difference is:%d\n",d);
}
```

- Here the variables ***a*** and ***b*** are global variables since they can be used by all the functions ***main(),add()*** and ***subtract().***
- But ***s*** is local variable and has its local scope to function ***add()*** only. Likewise, ***d*** has local scope within ***subtract()*** only.

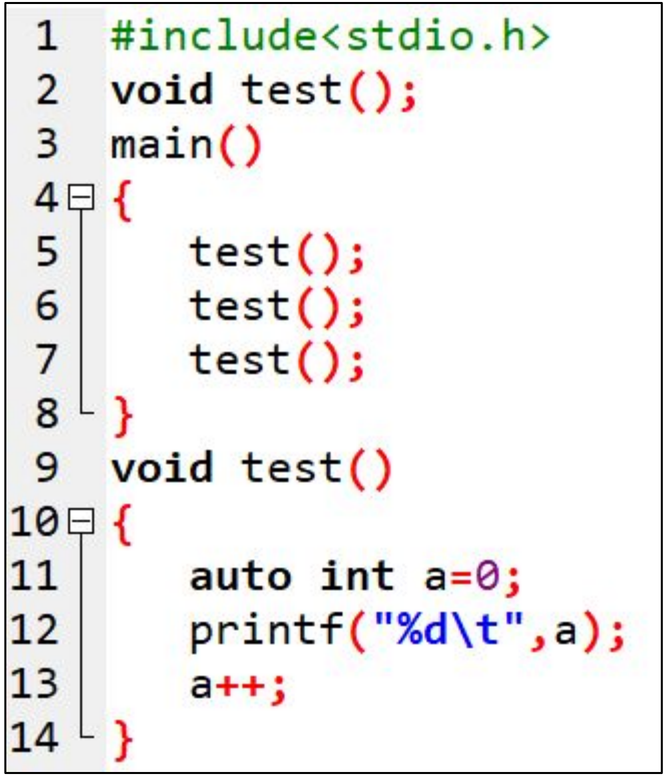
# 6.8 Storage classes in C

- A variable storage class provides the below information:
  - Storage location of variable.
  - Default initial value of variable if not specified.
  - Scope of variable (i.e. Visibility level).
  - Lifetime of variable (how long they exist).

# 6.8 Storage classes in C

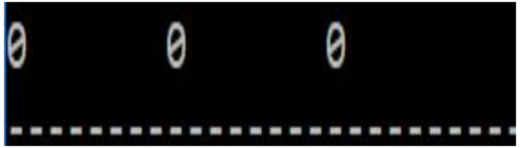
| S.No. | Class | Storage | Default initial value | Scope | Lifetime |
|---|---|---|---|---|---|
| 1. | Automatic | CPU memory | Garbage value | Local | Within the function |
| 2. | Static | CPU memory | Zero | Local | Value persist until the end of program |
| 3. | External | CPU memory | Zero | Global | Till the end of main() program |
| 4. | Register | Register memory | Garbage value | Local | Within the function |

# i) Automatic Storage class



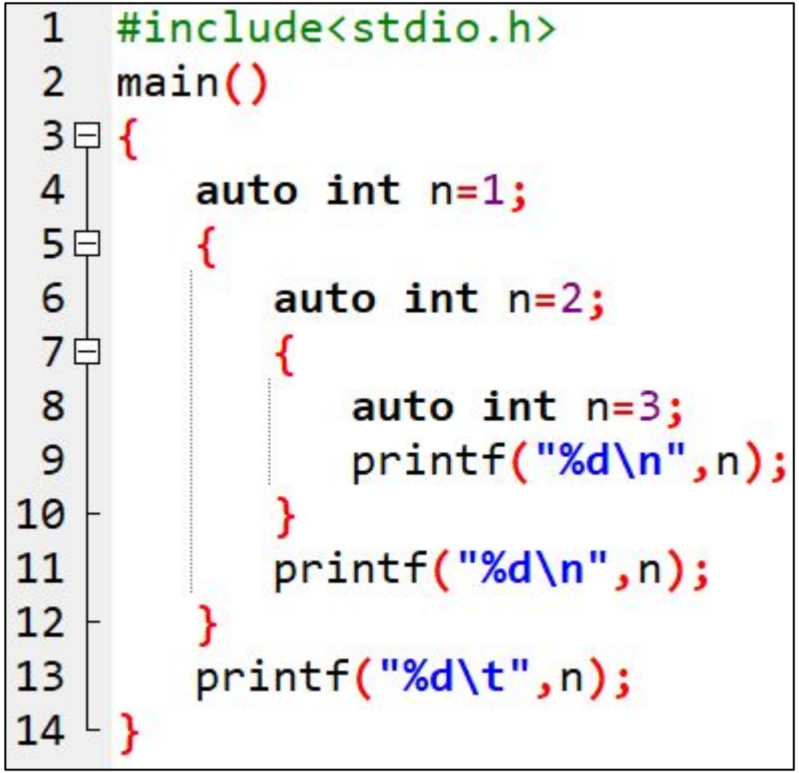
```c
#include<stdio.h>
void test();
main()
{
    test();
    test();
    test();
}
void test()
{
    auto int a=0;
    printf("%d\t",a);
    a++;
}
```

- Such variables are local to the block in which they are defined and
- get destroyed on exit from that block.



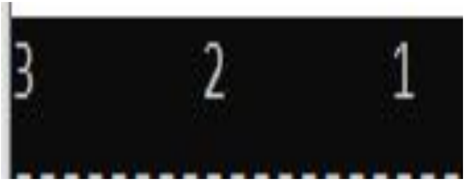
```
0       0       0
```

# i) Automatic Storage class (Contd.)



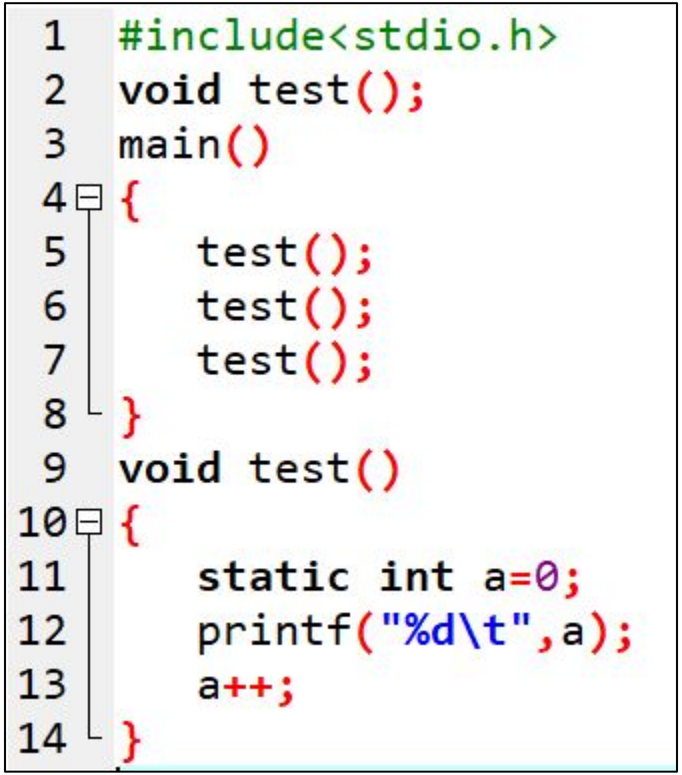
```
#include<stdio.h>
main()
{
    auto int n=1;
    {
        auto int n=2;
        {
            auto int n=3;
            printf("%d\n",n);
        }
        printf("%d\n",n);
    }
    printf("%d\t",n);
}
```

- Keyword used: *auto*
- Default storage class



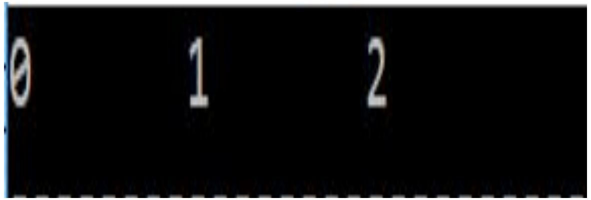
```
3       2       1
```

# ii) Static Storage class



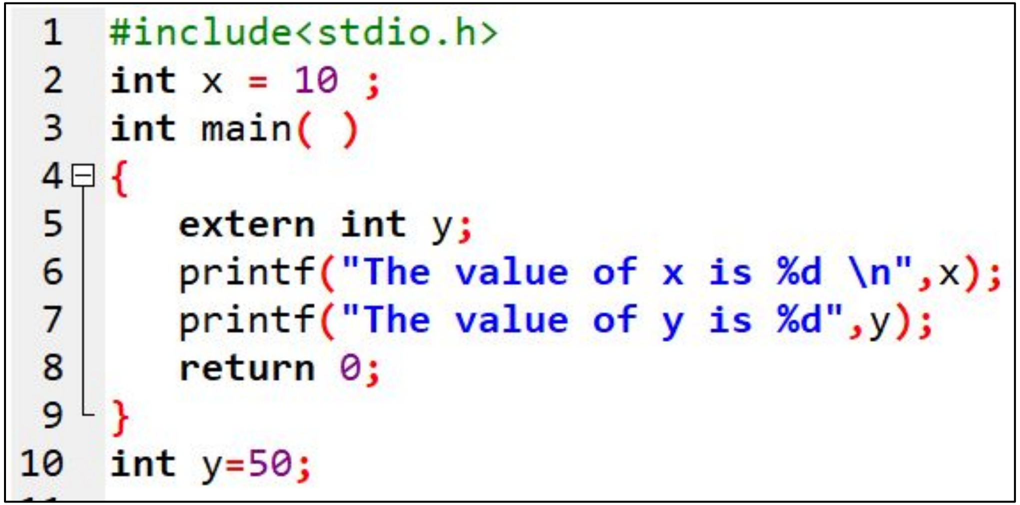
```c
#include<stdio.h>
void test();
main()
{
    test();
    test();
    test();
}
void test()
{
    static int a=0;
    printf("%d\t",a);
    a++;
}
```

- Keyword used: *static*
- Share the same memory
- Accumulate the results

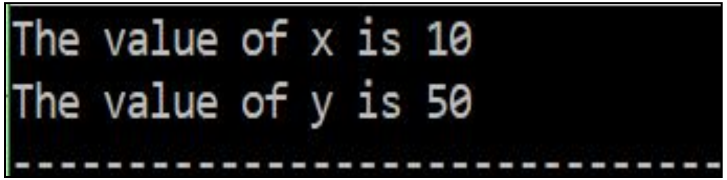


```
0       1       2
```

# iii) External Storage class



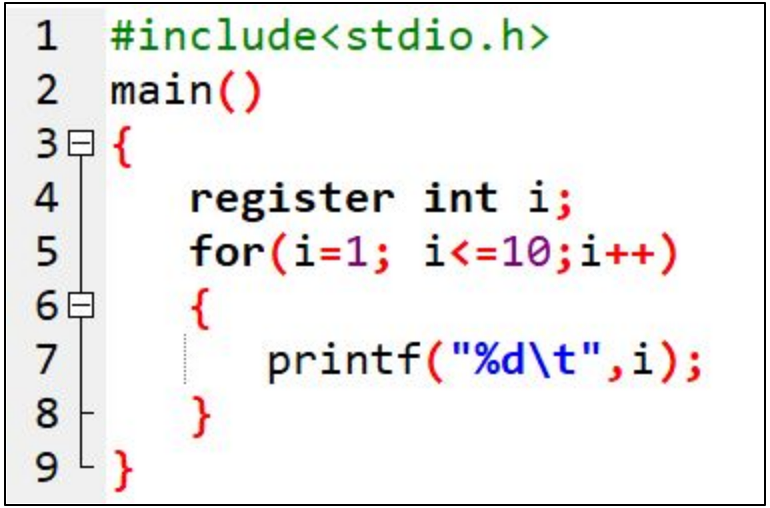
```c
#include<stdio.h>
int x = 10 ;
int main( )
{
    extern int y;
    printf("The value of x is %d \n",x);
    printf("The value of y is %d",y);
    return 0;
}
int y=50;
```



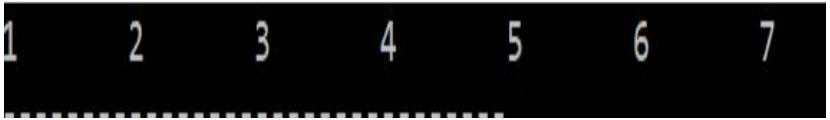
```
The value of x is 10
The value of y is 50
--------------------------------
```

- Keyword used: *extern*
- Active and alive through out the entire program.
- Also called global variables

# iv) Register Storage class



```c
#include<stdio.h>
main()
{
    register int i;
    for(i=1; i<=10;i++)
    {
        printf("%d\t",i);
    }
}
```

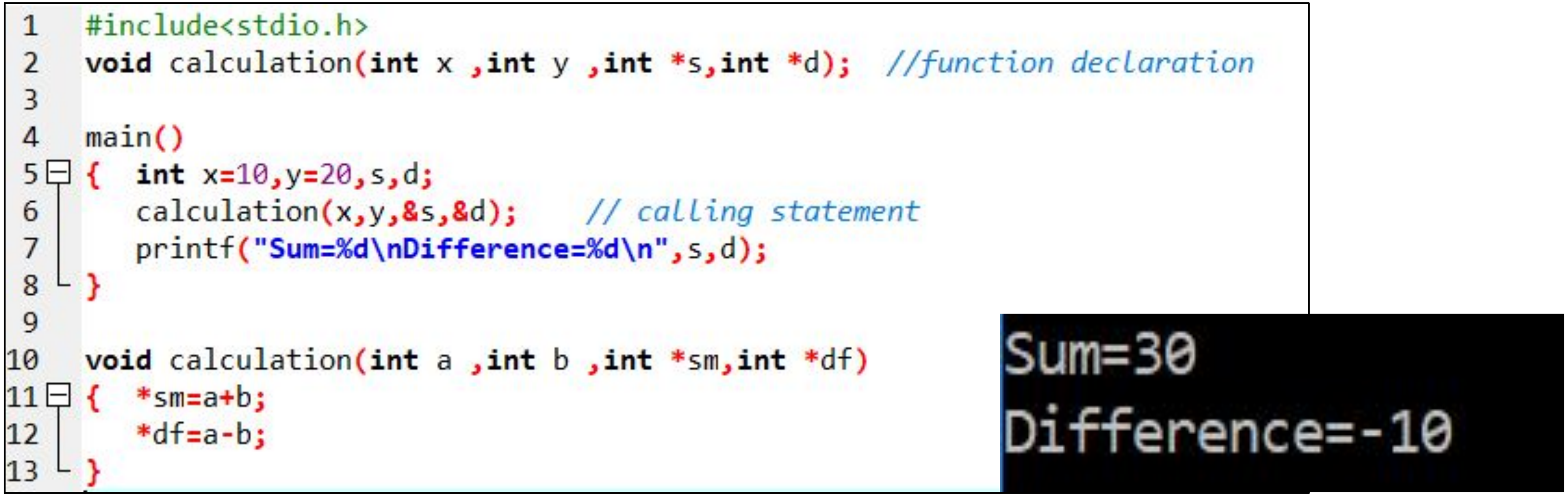


```
1       2       3       4       5       6       7
```

- Keyword used: register
- Variables operate at CPU registers.
- Faster accessing

# 6.9 Returning multiple values by function

- We can return multiple values to the calling function by the use of address operator (&) and indirect operator (*).



```c
#include<stdio.h>
void calculation(int x ,int y ,int *s,int *d);  //function declaration

main()
{   int x=10,y=20,s,d;
    calculation(x,y,&s,&d);    // calling statement
    printf("Sum=%d\nDifference=%d\n",s,d);
}

void calculation(int a ,int b ,int *sm,int *df)
{   *sm=a+b;
    *df=a-b;
}
```

```
Sum=30
Difference=-10
```

# 6.10 Passing arrays to function

Rules for passing array to function:

- The function must be called by passing only the name of array.
- In the function definition, the formal parameter must be an array_type, the size of the array need not to be specified.
- The function prototype must show that the argument is an array.

# 6.10 Passing arrays to function (Contd.)

- Syntax for function prototyping:

  **Return_type Function_name ( datatype arrayname[]);**
- Eg:

  **float Average (float Age[]);**

  Where Average is function name and Age is array name.

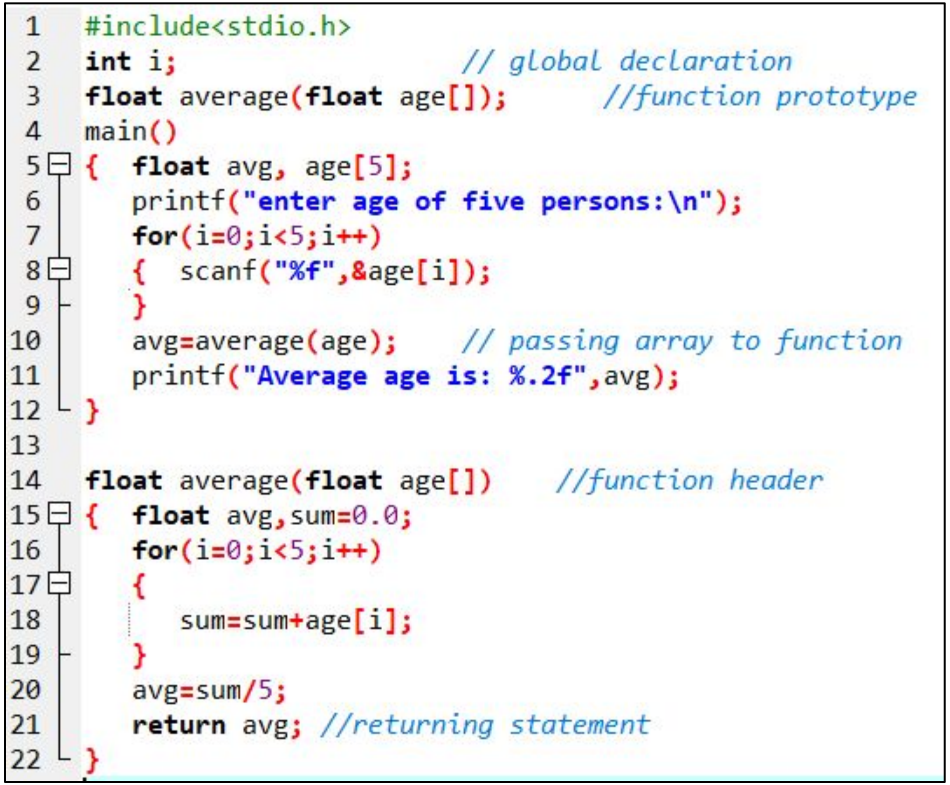
- Syntax for calling or passing array:

  **Function_name (arrayname)**
- Eg:

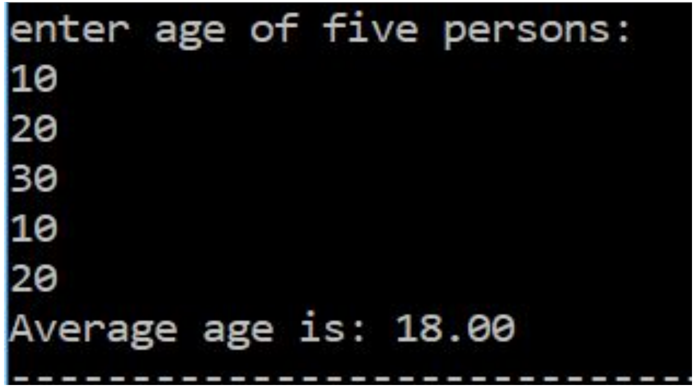
  **Average (Age);**

  Where Average is function name and Age is array name.

# 6.10 Passing arrays to function (Sample program)

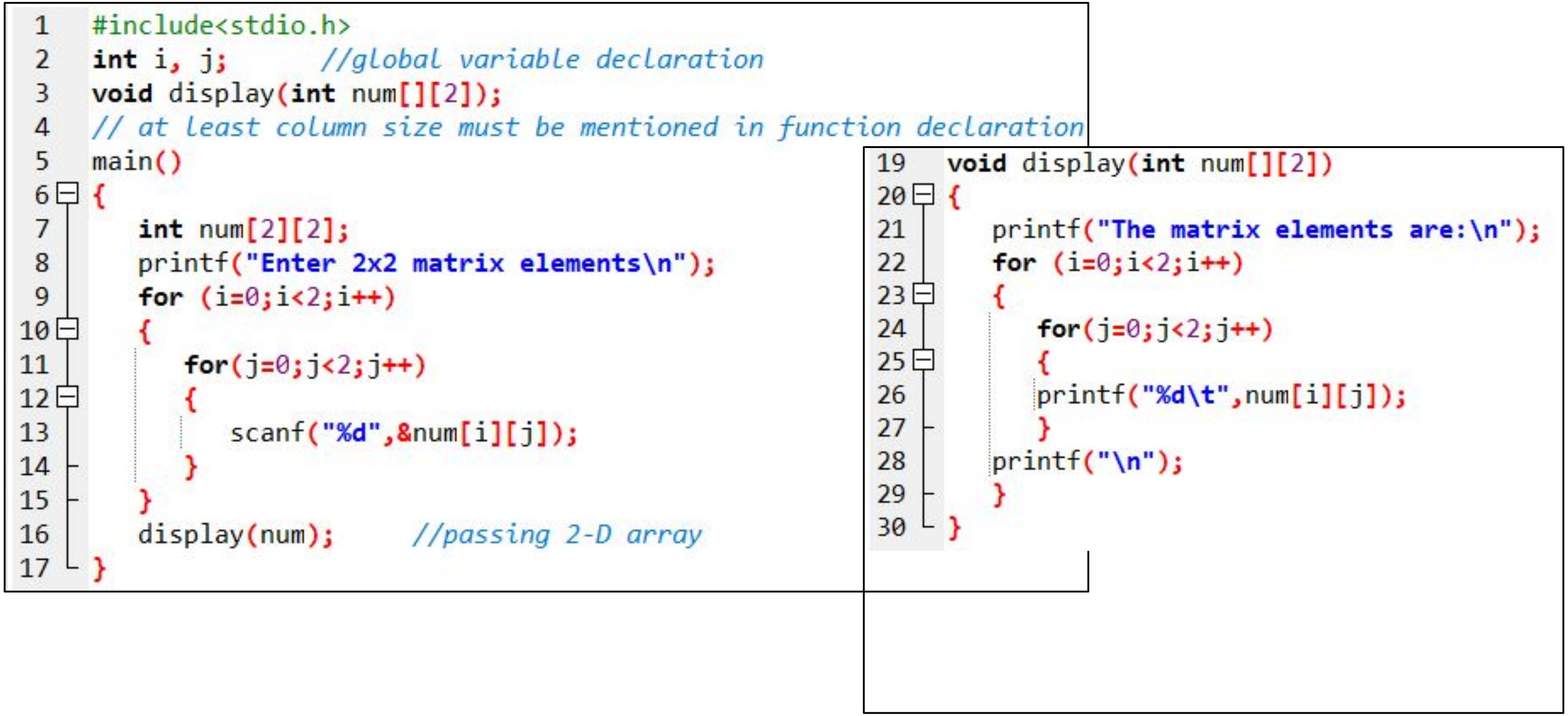


```c
#include<stdio.h>
int i;                      // global declaration
float average(float age[]);      //function prototype
main()
{   float avg, age[5];
    printf("enter age of five persons:\n");
    for(i=0;i<5;i++)
    {   scanf("%f",&age[i]);
    }
    avg=average(age);     // passing array to function
    printf("Average age is: %.2f",avg);
}

float average(float age[])    //function header
{   float avg,sum=0.0;
    for(i=0;i<5;i++)
    {
        sum=sum+age[i];
    }
    avg=sum/5;
    return avg; //returning statement
}
```



```
enter age of five persons:
10
20
30
10
20
Average age is: 18.00
-----------------------------
```

# 6.10 Passing arrays to function (Sample program)



```c
#include<stdio.h>
int i, j;       //global variable declaration
void display(int num[][2]);
// at least column size must be mentioned in function declaration
main()
{
    int num[2][2];
    printf("Enter 2x2 matrix elements\n");
    for (i=0;i<2;i++)
    {
        for(j=0;j<2;j++)
        {
            scanf("%d",&num[i][j]);
        }
    }
    display(num);     //passing 2-D array
}
```

```c
void display(int num[][2])
{
    printf("The matrix elements are:\n");
    for (i=0;i<2;i++)
    {
        for(j=0;j<2;j++)
        {
        printf("%d\t",num[i][j]);
        }
    printf("\n");
    }
}
```

# Passing arrays to function (Sample program)

- This signifies that the function takes a two-dimensional array as an argument. We can also pass arrays with more than 2 dimensions as a function argument.
- When passing two-dimensional arrays, it is not mandatory to specify the number of rows in the array. However, the number of columns should always be specified.
- For example,

```
void displayNumbers(int num[][2]) {
  // code
 }
```

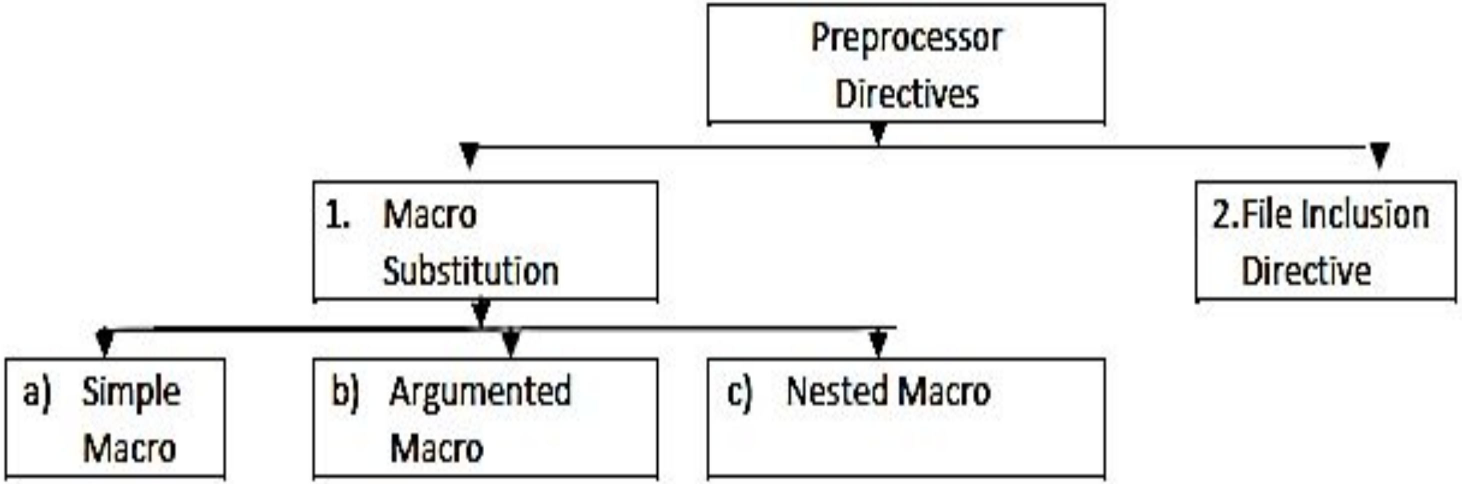
# 6.11 Passing structures to Function

Will be taught in next chapter

# 6.12 Pre-processor directives

- lines that are included in the code of programs preceded by a hash(#).
- can be placed anywhere in the program.
- But often before the *main()* function.

# 6.12 Pre-processor directives (Types)



Preprocessor Directives

1. Macro Substitution

2.File Inclusion Directive

a) Simple Macro

b) Argumented Macro

c) Nested Macro

# i) Macro-Substitution Directive

- uses macros which replaces the defined identifier in the program.
- Such directives accomplish the task under the direction of ***#define*** statement.
- classified into three categories namely:
  - Simple macro
  - Argumented macro
  - Nested macro

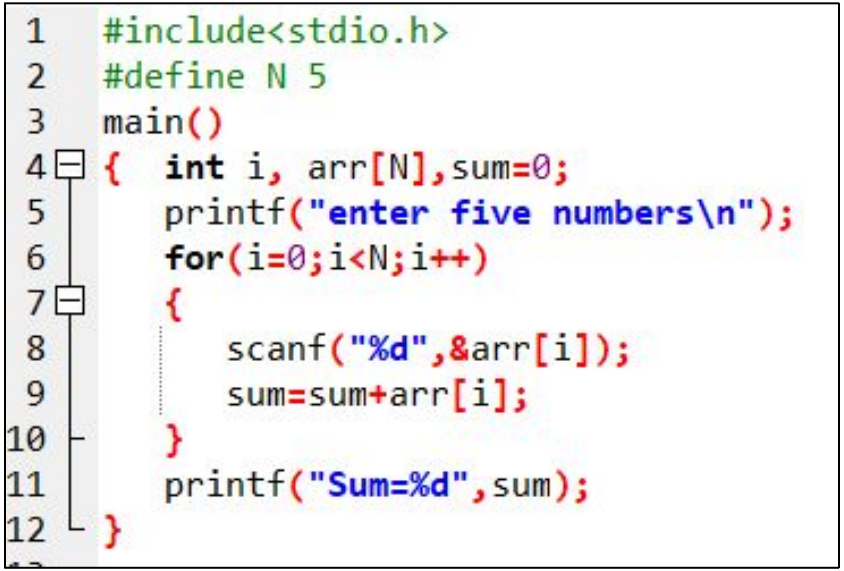
# a) Simple macro

- They are generally used for declaring constants in C programs.
- General form is:

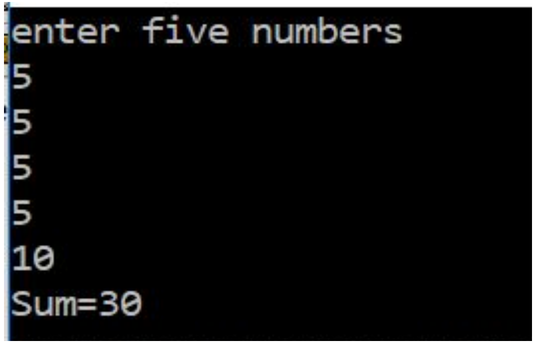
  **#define identifier value**

- Eg:

  **# define N 5**



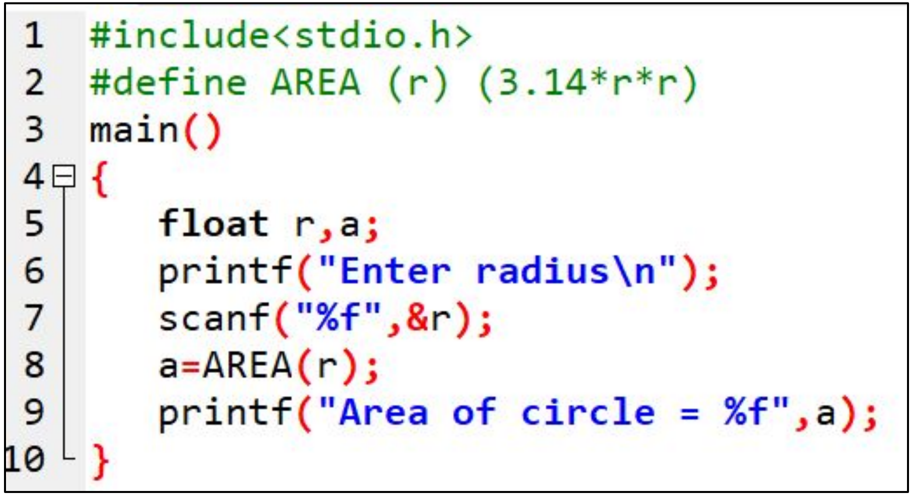
```c
#include<stdio.h>
#define N 5
main()
{   int i, arr[N],sum=0;
    printf("enter five numbers\n");
    for(i=0;i<N;i++)
    {
        scanf("%d",&arr[i]);
        sum=sum+arr[i];
    }
    printf("Sum=%d",sum);
}
```



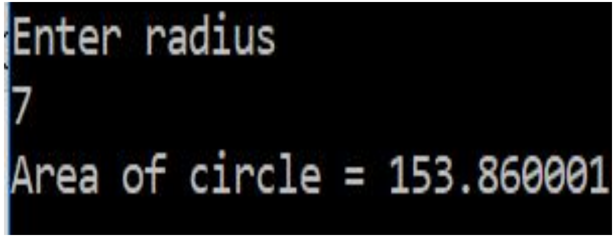
```
enter five numbers
5
5
5
5
10
Sum=30
```

# b) Argumented macro

- This argumented macro substitution permits us to define macro in more complex and uniform useful forms.
- The general form is:
- #define identifier (a1,a2,.....)(expression)
- Eg:
- #define AREA(r)  (3.14*r*r)

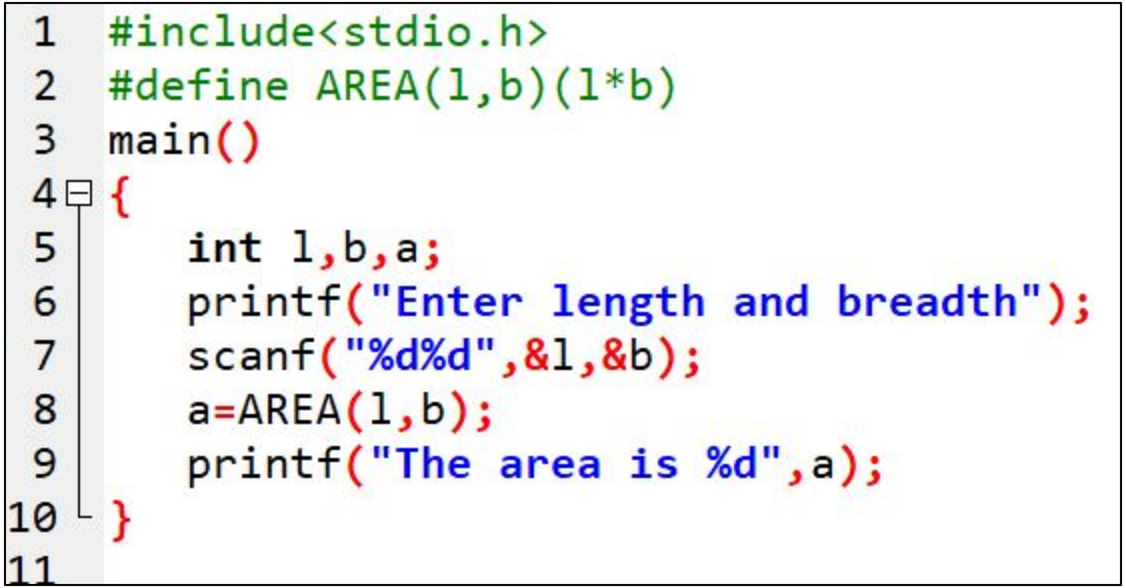


```
#include<stdio.h>
#define AREA (r) (3.14*r*r)
main()
{
    float r,a;
    printf("Enter radius\n");
    scanf("%f",&r);
    a=AREA(r);
    printf("Area of circle = %f",a);
}
```

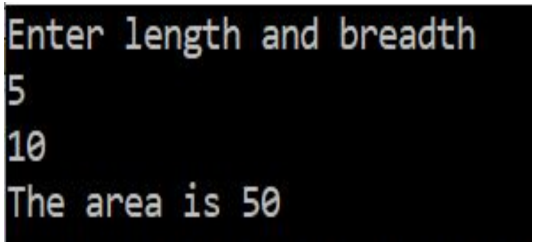


```
Enter radius
7
Area of circle = 153.860001
```

# Sample program

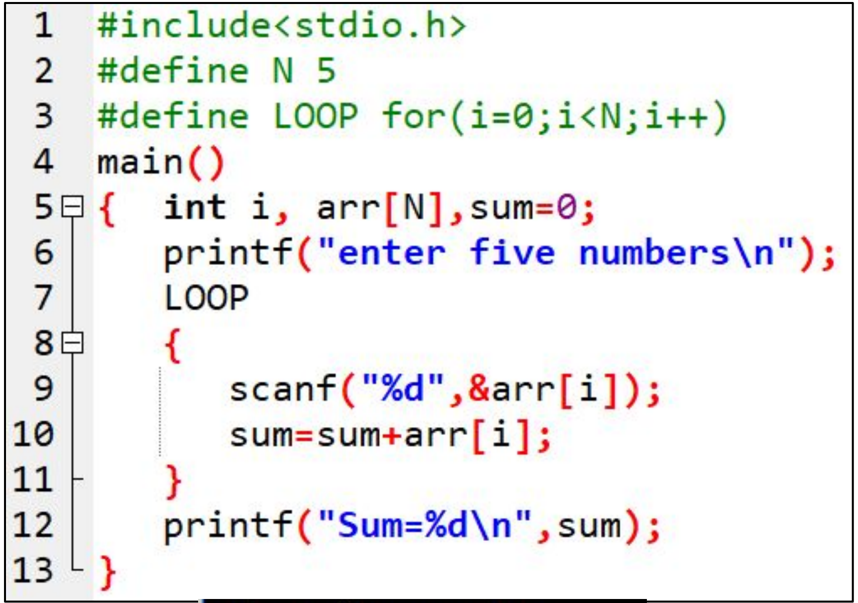


```c
#include<stdio.h>
#define AREA(l,b)(l*b)
main()
{
    int l,b,a;
    printf("Enter length and breadth");
    scanf("%d%d",&l,&b);
    a=AREA(l,b);
    printf("The area is %d",a);
}
```

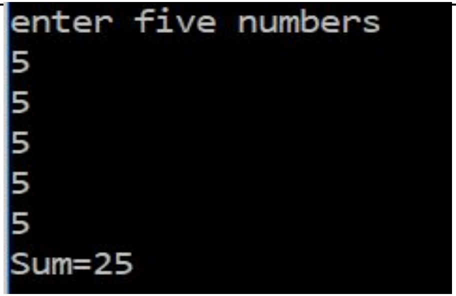


```
Enter length and breadth
5
10
The area is 50
```

# c) Nested Macro



```c
#include<stdio.h>
#define N 5
#define LOOP for(i=0;i<N;i++)
main()
{   int i, arr[N],sum=0;
    printf("enter five numbers\n");
    LOOP
    {
        scanf("%d",&arr[i]);
        sum=sum+arr[i];
    }
    printf("Sum=%d\n",sum);
}
```



```
enter five numbers
5
5
5
5
5
Sum=25
```

# ii) File Inclusive

- are used to include user defined header file inside C programs.
- They insert all the contents of the file into the program.
- They begin with #include and the header file is mentioned inside double quotes (“ “).

# ii) File Inclusive (Contd.)

There are two ways of including header file and they are:

***<u>Using angle bracket</u>***

- The general form is:

  #include <filename>

- Eg:

  #include <stdio.h>

***<u>Using double quotes</u>***

- The general form is:

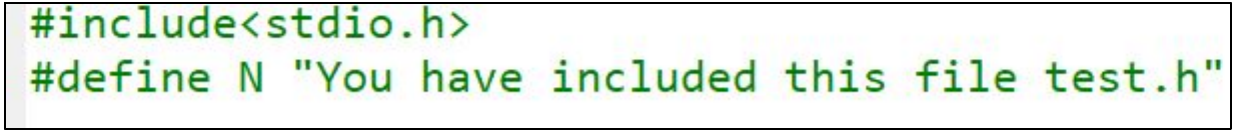
  #include “filename”

- Eg:

  #include “myfile.c”

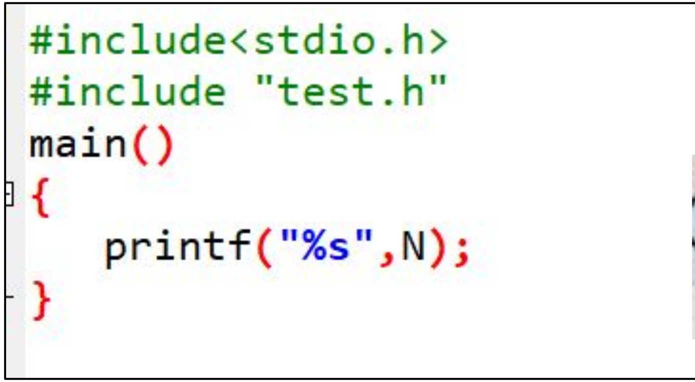
# Sample program

create a file ***test.h***

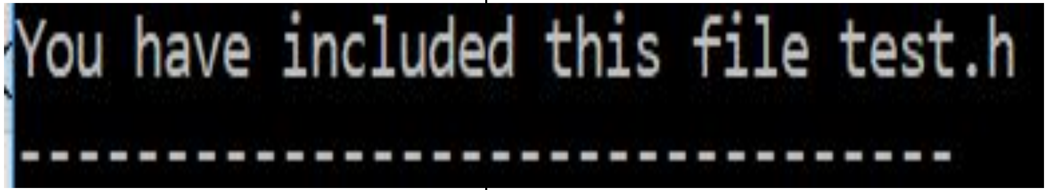


```
#include<stdio.h>
#define N "You have included this file test.h"
```

Now let us write another program which uses a file inclusion directive.



```
#include<stdio.h>
#include "test.h"
main()
{
    printf("%s",N);
}
```



```
You have included this file test.h
--------------------------------
```

# 6.13 Recursive Function

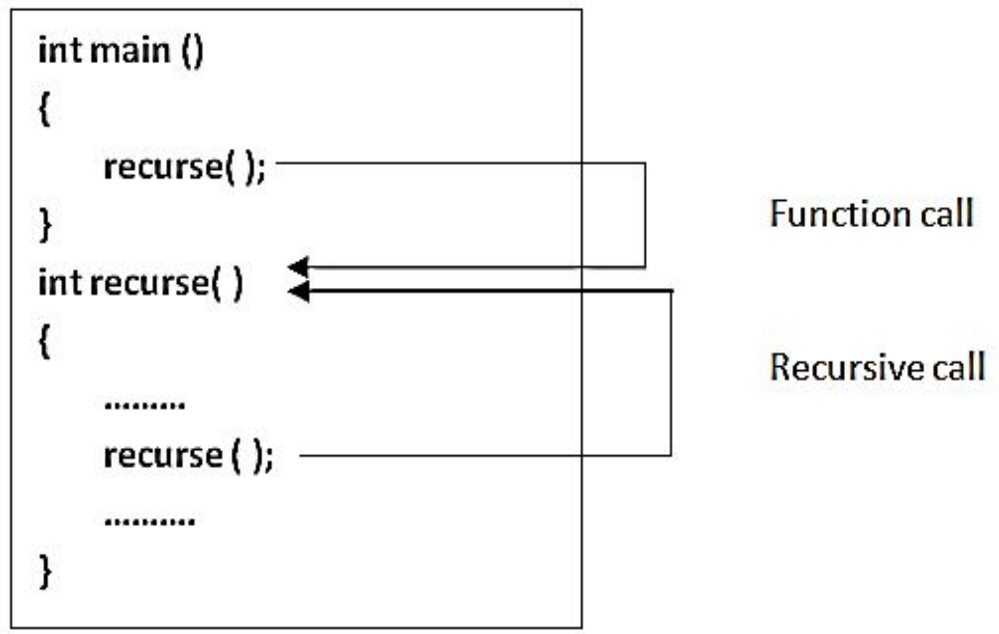
- function that calls to itself.
- to solve a smaller version of its tasks until a final call which does not require a self call.
- Thus, a function that calls itself is known as a recursive function and this technique is called recursion.
- Many iterative or repetitive problems can be written in recursive form.

# 6.13 Recursive Function (Contd.)

**Base Criteria:**

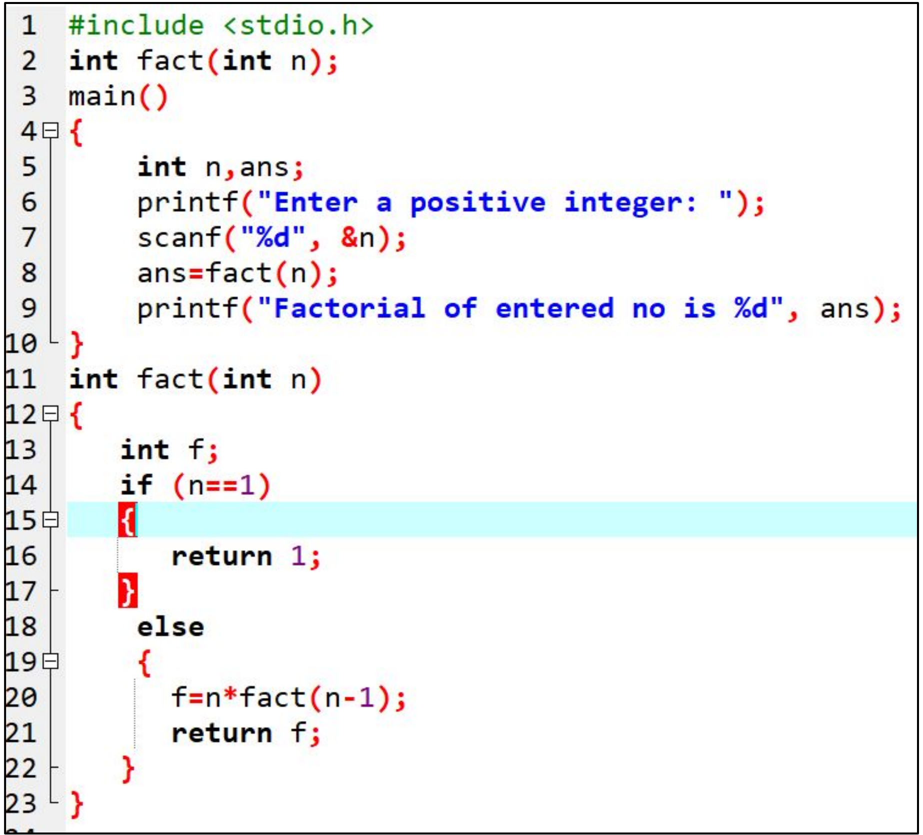
- Each time a function calls itself and it must be closer to a solution.
- There must be a decision criterion for stopping the process which is also called a base criterion.

# Basic Structure of programming using recursion

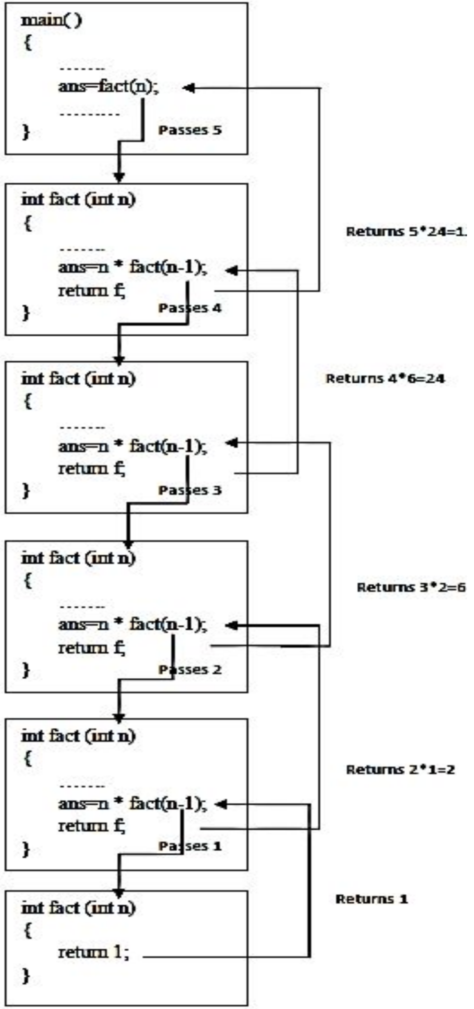


```
int main ()
{
      recurse( );
}
int recurse( )
{
      .........
      recurse ( );
      ..........
}
```

# WAP using recursion to enter a number and find its factorial.



```c
#include <stdio.h>
int fact(int n);
main()
{
    int n,ans;
    printf("Enter a positive integer: ");
    scanf("%d", &n);
    ans=fact(n);
    printf("Factorial of entered no is %d", ans);
}
int fact(int n)
{
   int f;
   if (n==1)
   {
       return 1;
   }
    else
    {
       f=n*fact(n-1);
       return f;
    }
}
```



```
main( )
{
      -------
      ans=fact(n);
      ---------
}
```

```
int fact (int n)
{
      ------
      ans=n * fact(n-1);
      return f;
}
```

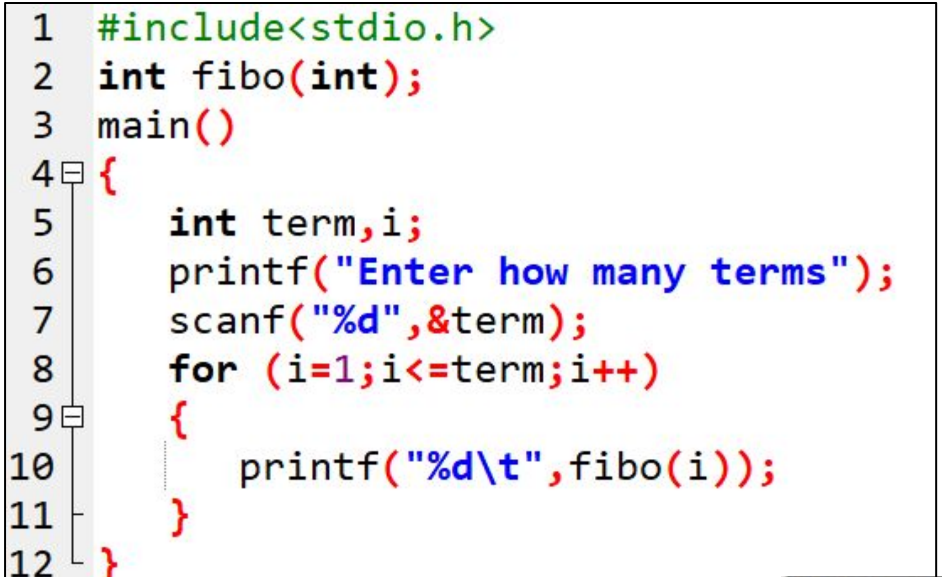
```
int fact (int n)
{
      ------
      ans=n * fact(n-1);
      return f;
}
```

```
int fact (int n)
{
      ------
      ans=n * fact(n-1);
      return f;
}
```

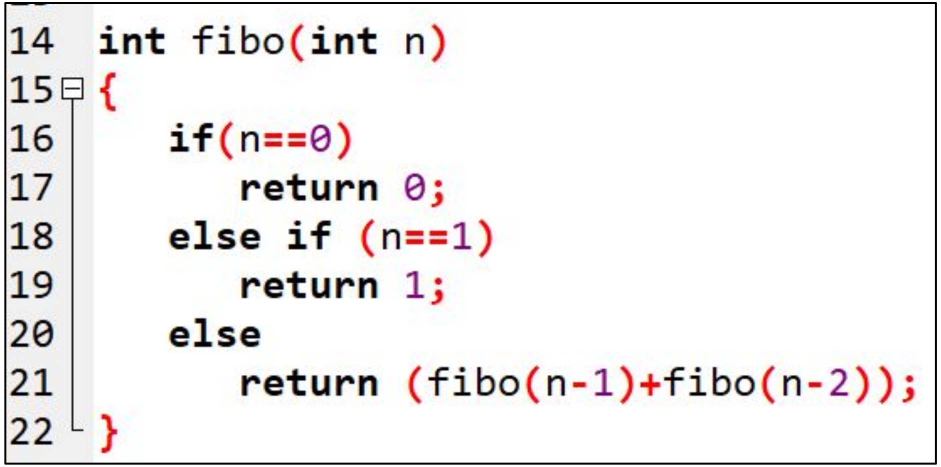
```
int fact (int n)
{
      ------
      ans=n * fact(n-1);
      return f;
}
```

```
int fact (int n)
{
      return 1;
}
```

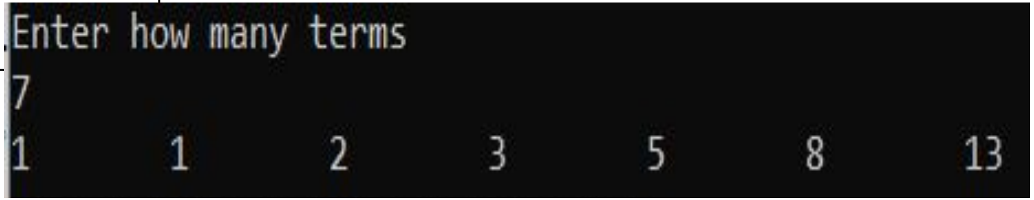
# WAP using recursion to print Fibonacci series up to n<sup>th</sup> terms



```c
#include<stdio.h>
int fibo(int);
main()
{
    int term,i;
    printf("Enter how many terms");
    scanf("%d",&term);
    for (i=1;i<=term;i++)
    {
        printf("%d\t",fibo(i));
    }
}

```

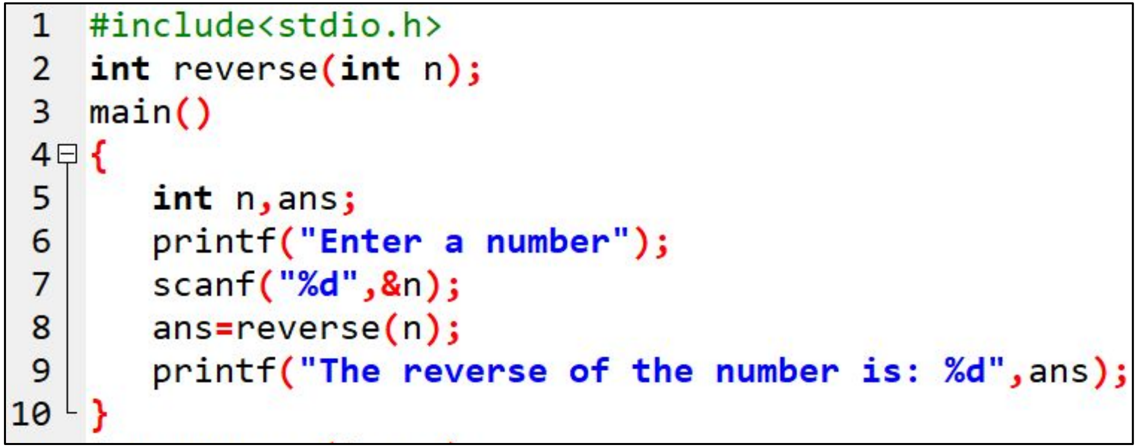


```c
int fibo(int n)
{
    if(n==0)
        return 0;
    else if (n==1)
        return 1;
    else
        return (fibo(n-1)+fibo(n-2));
}
```

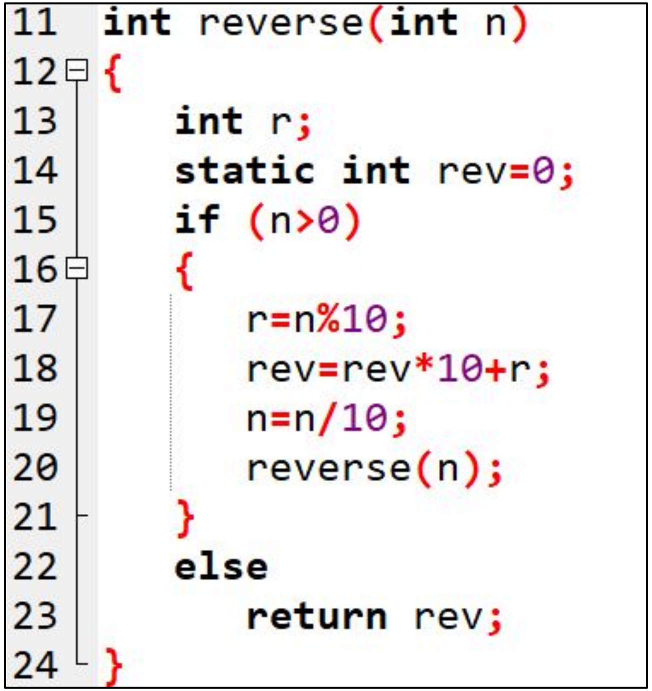


```
Enter how many terms
7
1       1       2       3       5       8       13
```

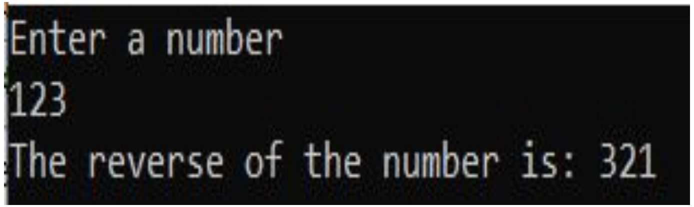
# WAP to input any positive integer and find its reverse using recursion.



```c
#include<stdio.h>
int reverse(int n);
main()
{
    int n,ans;
    printf("Enter a number");
    scanf("%d",&n);
    ans=reverse(n);
    printf("The reverse of the number is: %d",ans);
}
```

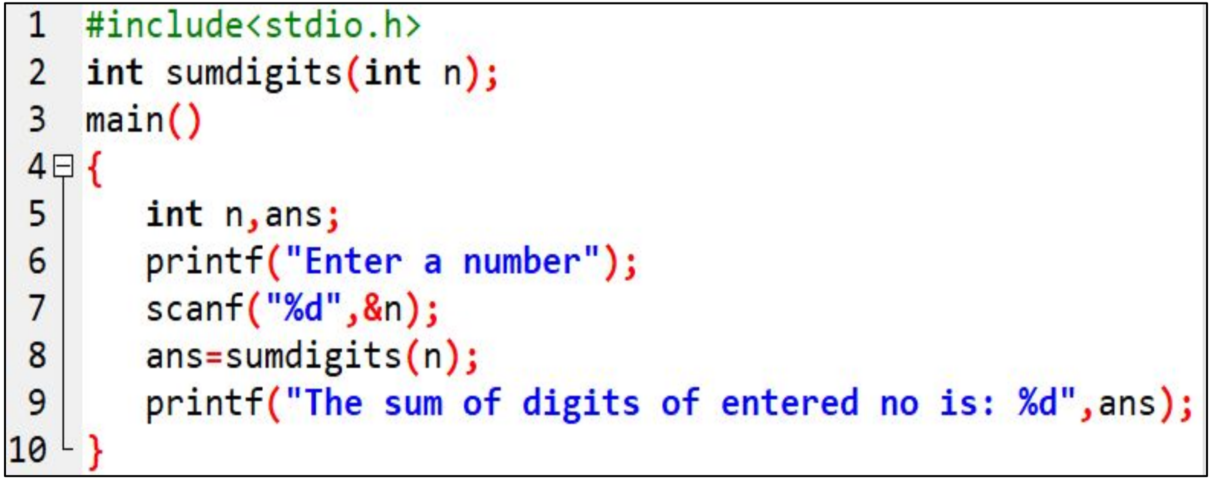


```c
int reverse(int n)
{
    int r;
    static int rev=0;
    if (n>0)
    {
        r=n%10;
        rev=rev*10+r;
        n=n/10;
        reverse(n);
    }
    else
        return rev;
}
```

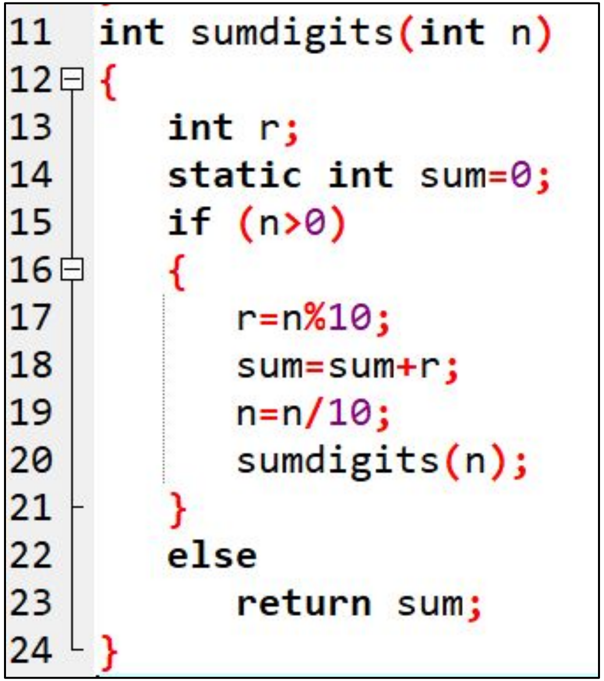


```
Enter a number
123
The reverse of the number is: 321
```

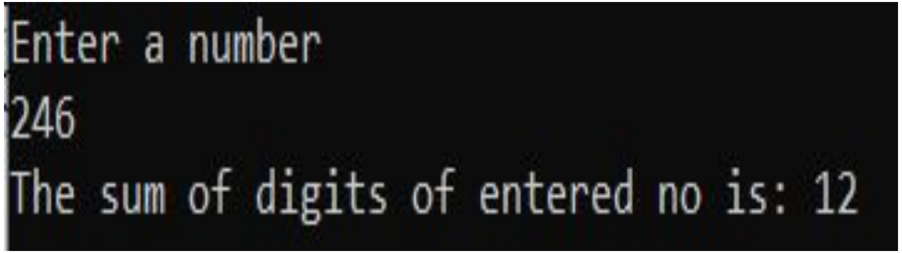
# WAP to input any positive integer and find the sum of digits in it.



```c
#include<stdio.h>
int sumdigits(int n);
main()
{
    int n,ans;
    printf("Enter a number");
    scanf("%d",&n);
    ans=sumdigits(n);
    printf("The sum of digits of entered no is: %d",ans);
}
```

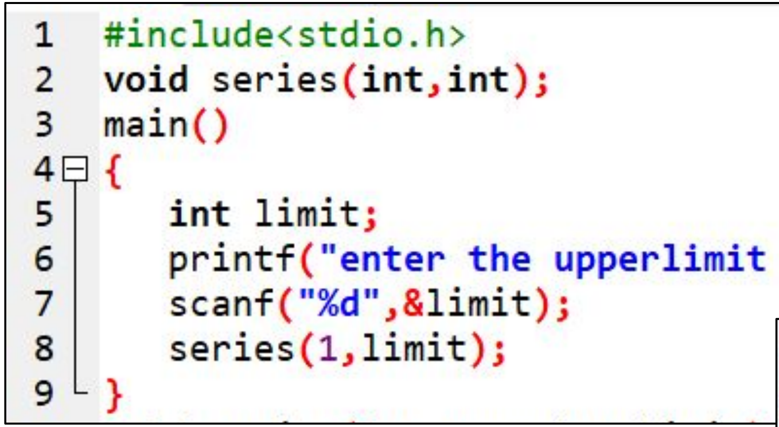


```c
int sumdigits(int n)
{
    int r;
    static int sum=0;
    if (n>0)
    {
        r=n%10;
        sum=sum+r;
        n=n/10;
        sumdigits(n);
    }
    else
        return sum;
}
```

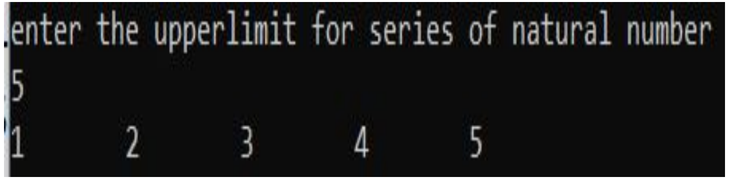


```
Enter a number
246
The sum of digits of entered no is: 12
```

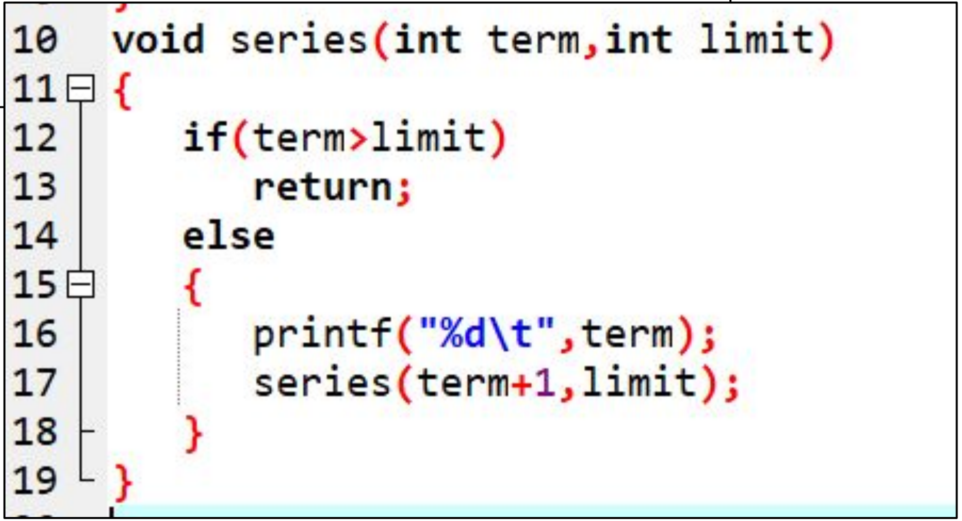
# WAP to find the sum of n natural numbers using recursion.



```c
#include<stdio.h>
void series(int,int);
main()
{
    int limit;
    printf("enter the upperlimit for series of natural number\n");
    scanf("%d",&limit);
    series(1,limit);
}
```



```
enter the upperlimit for series of natural number
5
1       2       3       4       5
```



```c
void series(int term,int limit)
{
    if(term>limit)
        return;
    else
    {
        printf("%d\t",term);
        series(term+1,limit);
    }
}
```

# End of Chapter