MODULE 3 :FUNCTIONS

Definition

- A function is a collection of statements that perform a specific task.
- These functions are very useful to read write and debug complex programs.

Types of Functions

- These can be broadly classified into two types
  - Built-in functions
  - User defined functions

Why are functions needed

- Dividing the program into separate modules allows each function to be written and tested separately.
- Understanding, coding and testing multiple separate functions is easier.
- A big program can be broken into smaller function, then divide the workload by writing different functions.
- Functions can make code faster by coding logic once instead of repeating several times.

Using Functions

- A function *f* that uses another function *g* is known as the *calling function* and *g* is known as the *called function*.
- The inputs that a function takes are known as *arguments/ parameters*.
- When a called function returns some result back to the calling function, it is said to *return* that result.
- The calling function may or may not pass *parameters* to the called function. If the called function accepts arguments, the calling function will pass parameters, else it will not do so.
- *Function declaration* is a declaration statement that identifies a function with its name, a list of arguments that it accepts, and the type of data it returns.
- *Function definition* consists of a function header that identifies the function, followed by the body of the function containing the executable code for that function.

- **User defined functions** :-

The user defined function is defined by the user according to its requirements.

- Parts of user defined function.
  - Function Declaration or Function prototype
  - Function call
  - Function Definition

1. Function Declaration or Function prototype :-

- It will inform the compiler about the return type, function name and number of arguments along with the data types.
- syntax:

return_data_type function_name(data_type variable1, data_type variable2,…);

- **return_data_type** :- is the data type of the value that is returned or sent from the function.
- **function_name** :-valid name for the function.
- **data_type variable1, data_type variable2,.. is the list of variables of specified data types.**

Function Declaration or Function prototype :-

- Example:-

int large (int x, int y);

or

int large (int, int);

- is a function declaration with function_name “large” with return _type “integer” and has two arguments “x” and “y” of integer type.
- NOTE:-
  - if we define a function before main ( ) function the there is no need of function declaration
  - if we define the function after main ( ) function then it is mandatory to declare the function because it will inform the compiler.

2. Function definition

- The declared function must define the same to perform the specific task.
- A function definition comprises 2 parts: function header & function body.
- Syntax-

return_data_type function_name(data_type variable1, data_type variable2,…)

```
{
        statements;
        return variable;
}
```

3. Function call The function call statement invokes the function.

- When a function is invoked, compiler jumps to that function to execute the statements that are a part of that function.
- Syntax:

function_name(variable1, variable2,..);

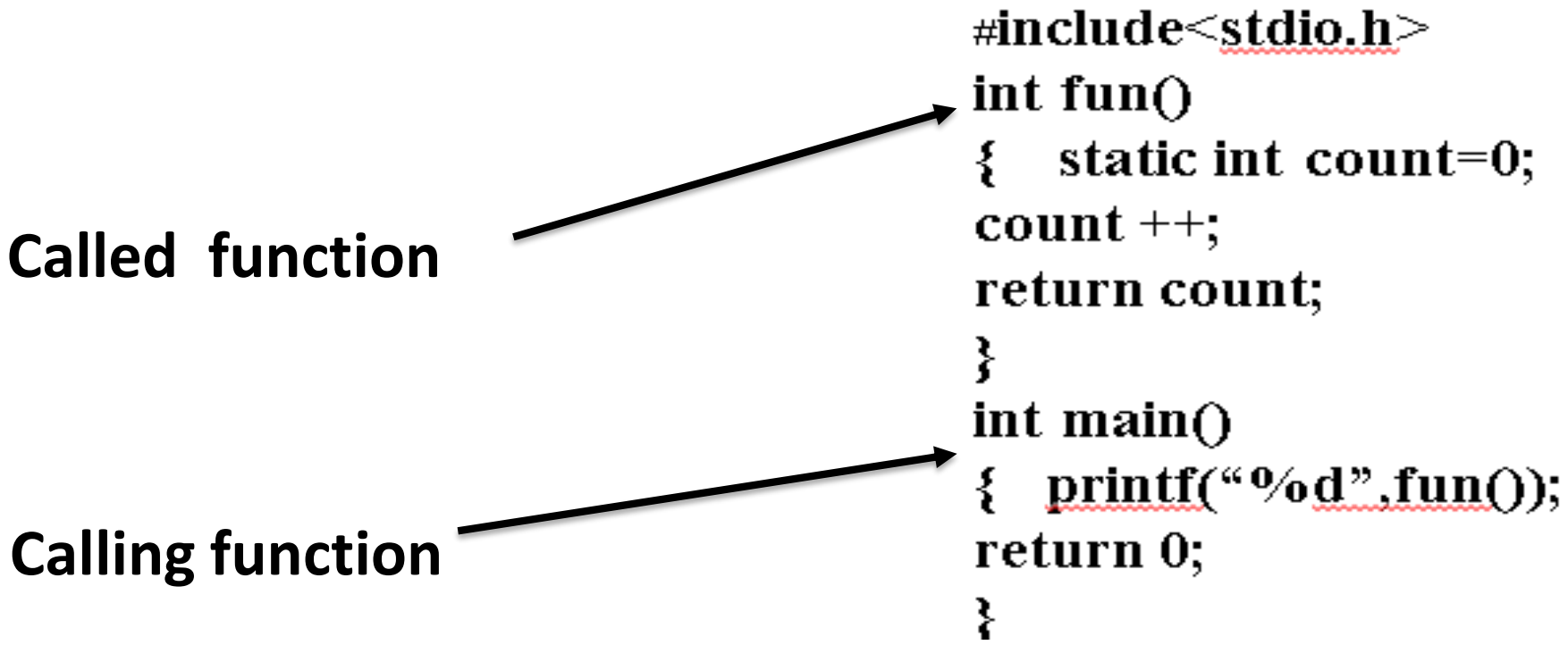
Ex- Write a C program to add 2 integers using functions.

```
#include<stdio.h>
int sum(int a,int b); //function declaration
int main()
{
int num1,num2,total=0;
printf("Enter 2 numbers\n");
scanf("%d%d",&num1,&num2);
total=sum(num1,num2); //function call
printf("Sum = %d\n",total);
return 0;
}

//function definition
int sum(int a,int b)
{
int result;
result=a+b;
return result;
}
```

Calling function and called function :-

- The **function main( )** that calls another function is called calling function.
- **The function** being called by the calling function is known as called function.



return Statement

- A return statement **ends the execution of a function, and returns control to the calling function**.

- Syntax. **return <expression>;**

Example: return 10; return a; return a+b;

- The **value** will be **passed back** to the function where it was called.
- Function that has **void** as its return statement **cannot return** any value to the calling function,

Parameter passing mechanism

There are two methods by which parameters or arguments can be passed to the function

- **Call by value**
- **Call by reference**

Call by value

- In call by value method, the values of actual parameters are copied to formal parameters.
- These 2 different parameters store values in different memory locations.
- One is the original copy and the other is the function copy.
- Any changes made inside functions are not reflected in the actual parameters of the caller.

C program to demonstrate call by value

```
#include<stdio.h>
void add(int n);
int main()
{
    int num=2;
    printf(“Value of num before calling function=%d”,num);
    add(num);
     printf(“Value of num after calling function=%d”,num);
     return 0;
}
void add(int n)
{
        n=n+10;
        printf(“Value of num in called function=%d”,n);
}
```

Output-

Value of num before calling function=2

Value of num in called function=12

Value of num after calling function=2

Call by reference

- In call by reference, the address of the actual parameters is passed to the function as the formal parameters.
- Both the actual and formal parameters refer to the same locations.
- Any changes made to formal parameters are actually reflected to actual parameters.

C program to swap two numbers using call by reference

```
#include<stdio.h>
void swap(int *n1,int *n2);
void main()
{
        int a,b;
        printf("Enter two numbers\n");
        scanf("%d%d",&a,&b);
        printf("Before Swapping\n a=%d\t b=%d\n",a,b);
        swap(&a, &b);
        printf("After Swapping\n a=%d\t b=%d\n",a,b);
}
void swap(int *n1, int *n2)
{
        int temp;
        temp=*n1;
        *n1=*n2;
        *n2=temp;
}
```

Advantages and Disadvantage of Call by value and Reference

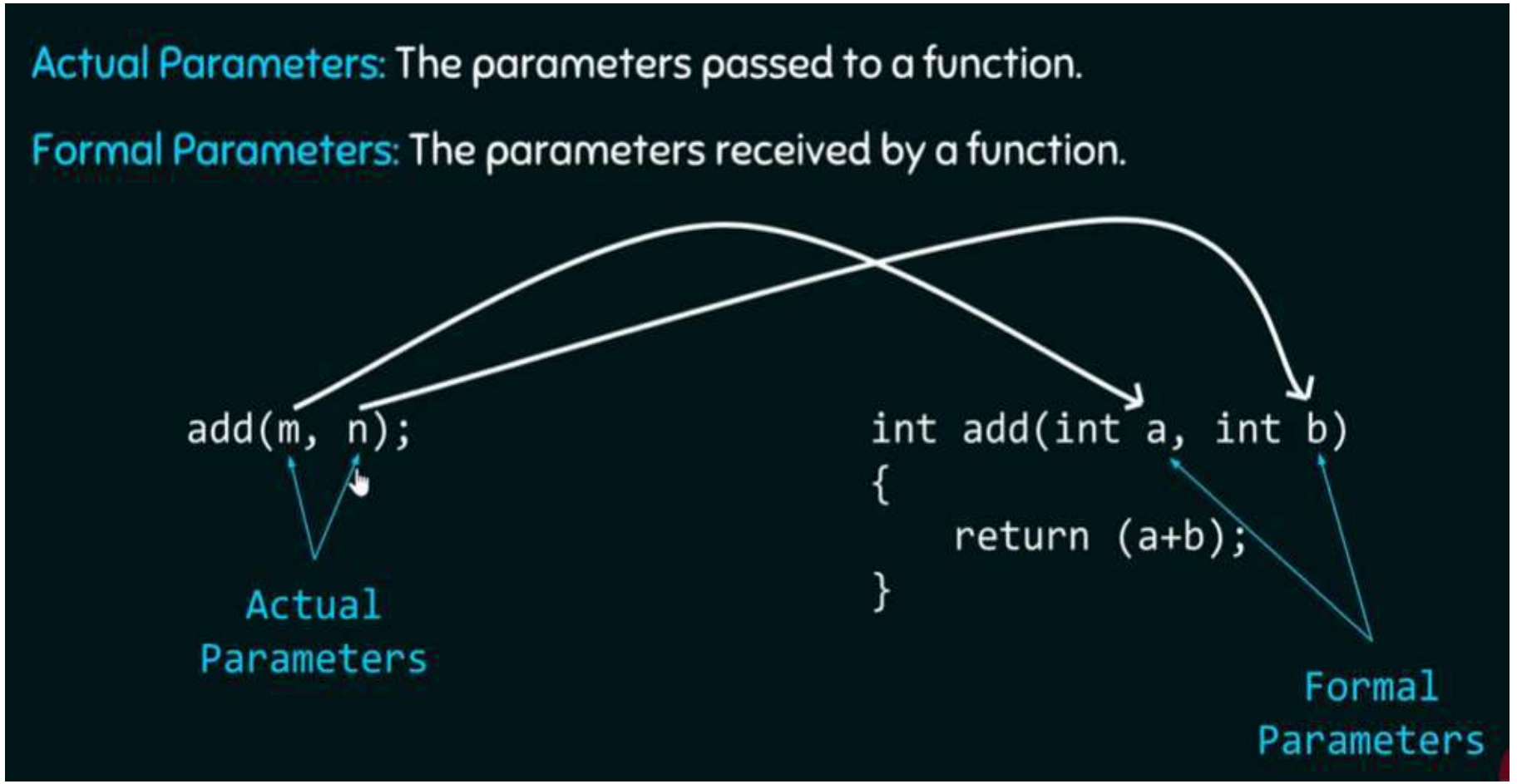
Call by reference

- Advantage: Is more efficient than copying
- Disadvantages:
  - Leads to aliasing: when there are two or more different names for the same storage location
  - Side effects not visible from code itself

Call by value-result

- Has all advantages and disadvantages of call-by-value and call-by-result together.

Actual arguments and Formal arguments :-



Scope of variables

The scope of a variable is the block of code in the entire program where the variable is declared, used, and can be modified.

1. **Block Scope:** A **Block** in C is a set of statements written within the right and left braces. A variable declared within a Block has a Block Scope. Scope of such a variable ends where the block ends.

```
#include<stdio.h>
 int main()  { // Block
 { //Variables within the block
 int a = 8, b = 10;
printf ("The values are: %d, %d\n", a, b);
}
 return  0;
}
```

2.Function scope

- Function scope indicates that a variable is active and visible from beginning to end of a function.
- In C only goto label has function scope.

Scope of variables

3. **Program scope**

Global variables declared outside the function bodies have a **program scope**. The availability of global variables stays for the entire program after its declaration

```c
#include <stdio.h>
int a = 8;//Declare Global Variables
float b = 7.5;
int test(){
  b = b + a;
  return b;
}
int main(){
  //Access a
  printf ("The value of a is: %d\n", a);
  //Access b
  printf ("The value of b is: %f\n", b);
  return 0;
}
```

Output

```
The value of a is: 8
The value of b is: 7.500000
```

Scope of variables

4. File scope

- These variables are usually declared outside of all of the functions and blocks, at the top of the program and can be accessed from any portion of the program.
- The *global static variable* is accessible by all the functions in the same source file as the variable. This variable has a File Scope.

```c
#include <stdio.h>
static int a = 20;
int func() {
  a = a + 20;
  printf ("The value of a is: %d\n", a);
}
int main() {
  func();
  a = a + 5;
  printf ("The value of a is: %d\n", a);
  return 0;
  }
```

```
The value of a is: 40
The value of a is: 45
```

- **Passing Arrays to functions** :- Array elements or an entire array can be passed to a function such a mechanism is called as passing array to the function.

- Write a C program to read and print an array of n numbers by passing array to function.

```
#include<stdio.h>
void read_array(int a[], int n);
void display_array(int a[], int n);
int main()
{
int a[10],n;
printf(“Enter size of array\n”);
scanf(“%d”,&n);
read_array(a,n);
display_array(a,n);
return 0;
}
```

```
void read_array(int a[10], int n)
{
int i;
printf(“Enter elements of array\n”);
for(i=0;i<n;i++)
{
        scanf(“%d”,&a[i]);
}
}

void display_array(int a[10],int n)
{
int i;
printf(“Elements of array are\n”);
for(i=0;i<n;i++)
{
        printf(“%d”,a[i]);
}
}
```

Program to demonstrate passing array to the functions

```
#include<stdio.h>
int largest(int a[20],int n);
void main()
{
   int a[20],n,i,max;
   printf("Enter the value of n\n");
   scanf("%d",&n);
   printf ("Enter %d values\n",n);
   for(i=0;i<n;i++)
    scanf("%d",&a[i]);
   max=largest(a,n);
   printf("Largest element in array=%d\n",max);  }
}
```

```
int largest(int a[20],int n)
{
   int max,i;
   max=a[0];
   for(i=1;i<n;i++)
   {
      if(a[i]>max)
         max=a[i];
   }
   return max;
```

Recursion

- Recursion is the **process in which function calls itself repeatedly until a particular condition is met.**

1.) C program to find factorial of the number using recursion

```
#include<stdio.h>
int factorial(int n);
void main()
{
   int n,res;
   printf("Enter the value of n\n");
    scanf("%d",&n);
    res=factorial(n);
   printf("The factorial of the number=%d\n",res);
}
int factorial(int n)
{
    if(n==0)
    return 1;
   else
     return n*factorial(n-1);
}
```

2.) C program to generate fibonacci series upto N numbers using recursion

```
#include<stdio.h>
int fibonacci(int n);
void main()
{
   int num,i;
   printf("Enter the number\n");
   scanf("%d",&num);
   printf("\nFibonacci Series\n");
   for(i=0;i<num;i++)
   {
      printf("%d\t",fibonacci(i));
   }
}
```

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

3.) Write a C program to calculate exp(x,y) using recursive functions.

```
#include<stdio.h>
int exp(int x,int y);
void main()
{
int num1,num2,res;
printf("Enter 2 numbers\n");
scanf("%d%d",&num1,&num2);
res=exp(num1,num2);
printf("Result= %d\n", res);
}
```

```
int exp(int x,int y)
{
        if(y==0)
          return 1;
       else
        return (x*exp(x,y-1));
}
```

4.) Write a C program to calculate GCD using recursive functions.

```
#include<stdio.h>
int GCD(int x,int y);
void main()
{
int num1,num2,res;
printf("Enter 2 numbers\n");
scanf("%d%d",&num1,&num2);
res=GCD(num1,num2);
printf("GCD= %d\n", res);
}
```

```
int GCD(int x,int y)
 {
     int rem;
     rem=x%y;
     if(rem==0)
         return y;
      else
        return (GCD(y,rem));
  }
```

MODULE-3

ARRAYS AND FUNCTIONS

PART 1 -Arrays

Arrays

- An array is a collection of elements of the same data type.
- The elements of array are stored in consecutive memory locations and are referenced by an index (subscript)

- Consider a situation, where we need to store 5 integer numbers.

Arrays

```
 #include<stdio.h>
int main()
{
int number1=10;
int number2=20;
int number3=30;
int number4=40;
int number5=50;
 printf( "number1: %d \n", number1);
printf( "number2: %d \n", number2);
printf( "number3: %d \n", number3);
printf( "number4: %d \n", number4);
printf( "number5: %d ", number5);
Return 0;
}
```

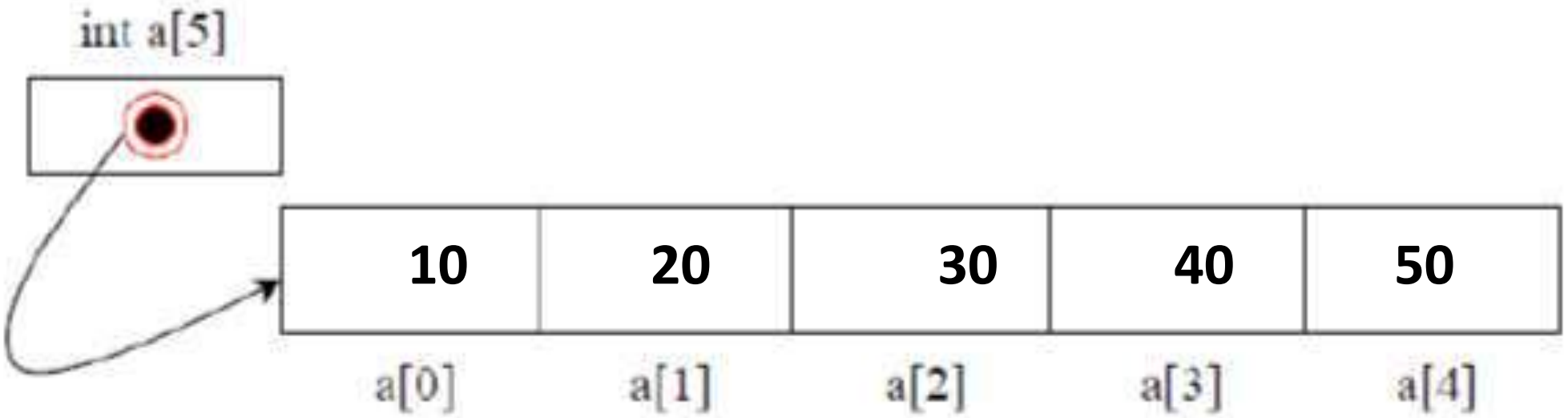
```
int number1=10, number2=20, number3=30, number4=40,
number5=50;
```

```
number1: 10
number2: 20
number3: 30
number4: 40
number5: 50
```

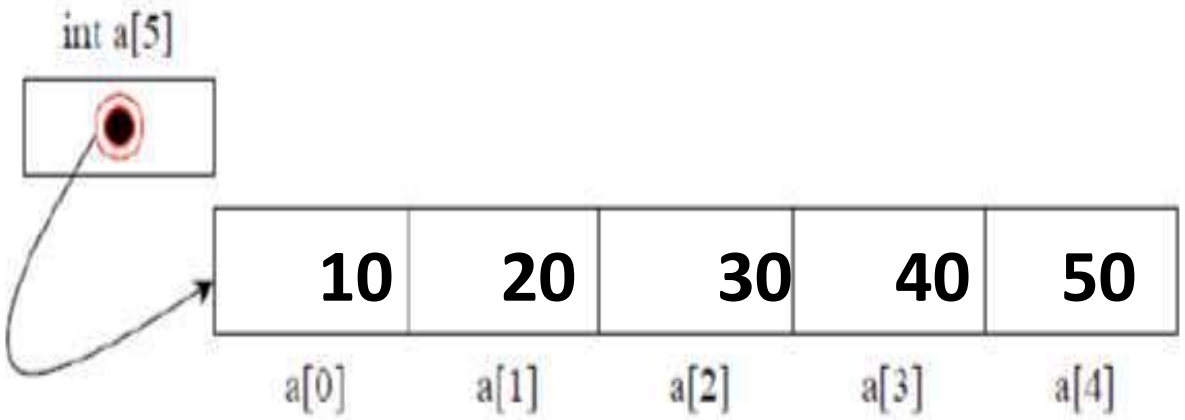
Arrays

- To handle such situation, C language provides a concept called the **ARRAY.**

```
int a[5]={10,20,30,40,50};
```



Arrays



- a[0] holds the first element in the array
- a[1] holds second element in the array
- a[2] holds third element in the array
- a[3] holds fourth element in the array
- a[4] holds fifth element in the array

Types of Arrays

- Single dimensional array or One dimensional array
- Two dimensional array
- Multi dimensional array

Single Dimensional Array Declaration:-

- An Array which has only one subscript is known as Single dimensional array or One dimensional array.
- The individual array elements are processed by using a common array name with different index values that start with **Zero** and ends with **array_size-1**.

Syntax:

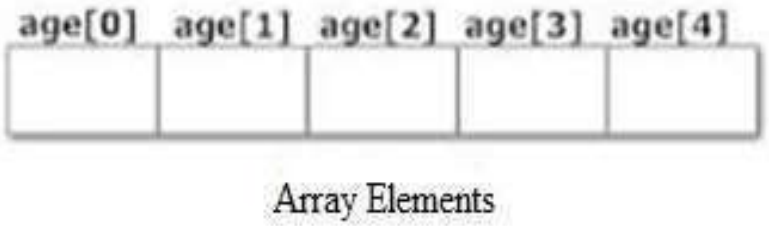
data_type array_name[size];

data_type: can be int, float or char

array_name: is name of the array

size : an integer constant indicating the maximum number of data elements to be stored.

- For ex: int age[5];



- **Total memory=array size * size of datatype**

- Total memory =5*sizeof (int)

= 5*4

=20 bytes.

- Single Dimensional Array :-

- **Rules for declaring Single Dimensional Array :-**

1. An array variable must be declared before being used in a program
2. The declaration must have a data type(int, float, char), variable name and subscript.
3. The subscript represents the size of the array. If the size is declared as 10, programmers can store 10 elements.
4. An array index always starts from **0**.
5. **Example:** if an array variable is declared as **a[10]**, then it ranges from **0 to 9**.
6. Each array element is stored in a separate location.

Initialization of one dimensional arrays/Storing values in arrays

There are 4 ways to initialize / store values in an array.

- Initialization at the time of declaration(static initialization)
- Initialization of array elements one by one
- Partial initialization of array
- Initialization during program execution. (run time initialization)

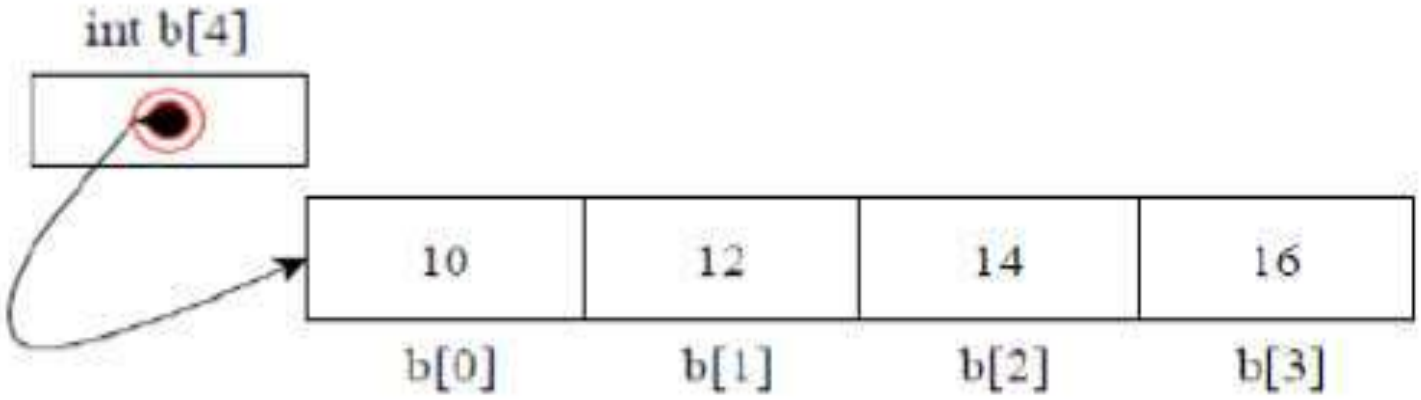
1. Static initialization:- We can initialize the array in the same way as the ordinary values when they are declared.

- The general syntax of initialization of array is

  data_type array_name[size]= {List of values};

- Example:
- int b[4]={10,12,14,16};
- Here each value will be stored in respective index values of the array.

- **Static initialization:-**



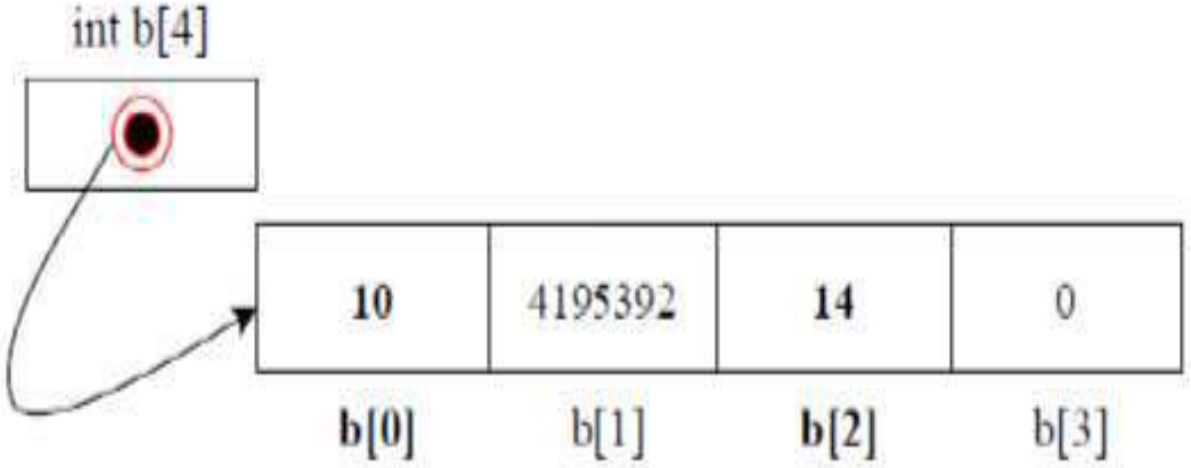
- In location b[0] the value 10 is stored and in location b[1] the value 12 is stored and in location b[2] the value 14 is stored, in location b[3] the value 16 is stored.

2. Initialization of array elements one by one:- Here the user has the liberty to select the locations and store values and the array. This type of initialization is not used much practically.

- Example :int b[4];

b[0]= 10;

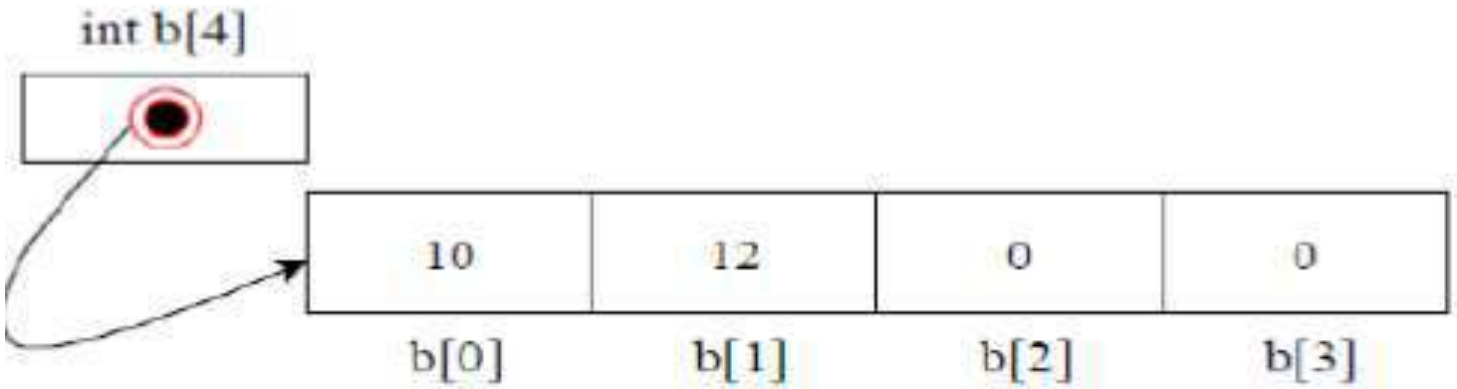
b[2]=14;



Only the array locations specified by the user will contain the values which the user wants the other locations of array will either be 0 or some garbage value.

3. Partial initialization of array :- If the number of values initialized in the array is less than the size of the array then it is called partial initialization. The remaining locations in the array will be initialized to zero or NULL('\0') value automatically.

```
int b[4]={10,12};
```



Here the remaining locations in the array will be initialized to zero.

4. Run Time array Initialization:- If the values are not known by the programmer in advance then the user makes use of run time initialization.

- It helps the programmer to read unknown values from the end users of the program from keyboard by using input function **scanf().**
- Here looping construct is used to read the input values from the keyboard and store them sequentially in the array.

- **Demonstrating run time initialization (C Program to read and print 5 elements of array)**

```
#include<stdio.h>
void main()
{
int b[5],i;
printf(“Enter 5 elements\n”);
for(i=0;i<5;i++)
{
        scanf(“%d”,&b[i]);
}
for(i=0;i<5;i++)
{
        printf(“%d”,b[i]);
}

}
```

Write a c program to add two one dimensional array

```
include<stdio.h>
void main()
{
        int a[20 ],b[20],c[20],n, i;
        printf("Enter the number of elements\n");
        scanf("%d",&n);
        printf("Enter the elements of Array A\n");
        for(i=0;i<n;i++)
                scanf("%d",&a[i]);
        printf("Enter the elements of Array B\n");
                for(i=0;i<n;i++)
        scanf("%d",&b[i]);
        printf("Array Addition\n");
        for(i=0;i<n;i++)
                c[i]=a[i]+b[i];
        printf("The resultant array is \n");
        for(i=0;i<n;i++)
                printf("%d\n",c[i]);
}
```

```
Enter the number of elements
5
Enter the elements of Array A
12
11
14
5
6
Enter the elements of Array B
5
1
2
5
8
Array Addition
The resultant array is
17
12
16
10
14
```

Write C program to find largest and smallest number in an array of n elements

```
include<stdio.h>
void main()
{
        int a[10 ],n,i,max,min
        printf("Enter the number of elements\n");
        scanf("%d",&n);
        printf("Enter the values \n");
        for(i=0;i<n;i++)
                scanf("%d",&a[i]);
        max=min=a[0];
        for(i=0;i<n;i++)
        {
                if(a[i]>max)
                        max=a[i];
                if(a[i]<min)
                        min=a[i];
        }
        printf("Largest number=%d\n",max);
        printf("Smallest number=%d\n",min);
}
```

```
Enter the number of elements
3
Enter the values
5
8
5
Largest number=8
Smallest number=5
```

Write C program to read n integer elements in an array and print the same in reverse order

```
#include<stdio.h>
void main()
{
        int a[20 ],n,i;
        printf("Enter the array size");
        scanf("%d",&n);
        Printf("Enter the array elements\n");
        for(i=0;i<n;i++)
                scanf("%d",&a[i]);
        printf("The elements entered in the array are\n");
        for(i=0;i<n;i++)
                printf("%d\t",a[i]);
        printf("The elements of the array in reverse order are\n");
        for(i=n-1;i>=0;i--)
                printf("%d\t",a[i]);
}
```

```
Enter the array size3
Enter the array elements
3
4
5
The elements entered in the array are
3       4       5
The elements of the array in reverse order are
5       4       3
```

Sorting Techniques:-

- The Process of arranging the elements in ascending or descending order is called sorting.

1) Bubble sort:

This sorting algorithm is a comparison based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order.

Bubble sort with **n elements** requires **n - 1** passes.

```
#include<stdio.h>
 void main()
{
int a[50],n,i,j,temp;
printf("Enter the number of elements\n");
scanf("%d",&n);
printf("Enter array elements\n");
for(i=0;i<n;i++)
{
        scanf("%d",&a[i]);
}
printf("The entered elements are\n");
for(i=0;i<n;i++)
{
        printf("%d\t",a[i]);
}
for(i=1;i<n;i++)
{
        for(j=0;j<n-i;j++)
        {
                if(a[j]>a[j+1])
                {
                        temp=sa[j];
                        a[j]=a[j+1];
                        a[j+1]=temp;
} } }
```

```
printf("The sorted elements are\n");
for(i=0;i<n;i++)
{
        printf("%d\t",a[i]);
}
}
```

Output-

Enter the number of elements : 5

Enter the array elements: 10 2 3 5 4

The entered elements are : 10 2 3 5

The sorted elements are: 2 3 4 5 10

Refer your notes

- Write a C program to read and display n integer numbers using an array.
- Write a C program to find the smallest number in an array and also print its position.
- Write a C program to find the largest number in an array and also print its position.
- Write a C program to insert an element at a given location in an array.
- Write a C program to delete a number from a given location in an array.
- Write a C program to implement Bubble sort.

Searching Techniques:

The process of finding a particular element in the large amount of data is called searching.

1.) Linear search:- A Linear search is also called as sequential Search. In this technique we search for a given specific element called as key element in the large list of data in sequential order. If the key element is present in the list of data then the search is successful otherwise search is unsuccessful.

Benefits:

- Simple approach
- Works well for small arrays
- Used to search when the elements are not sorted

Disadvantages:

- Less efficient if the array is large
- If the elements are already sorted, linear search is not efficient.

Linear Search

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
int a[50],i,n,key;
printf("Enter the number of elements\n");
scanf("%d",&n);
printf("Enter the elements in ascending order\n");
for(i=0; i<n; i++)
{
        scanf("%d",&a[i]);
}
printf("Enter the element to be searched\n");
scanf("%d",&key);
for(i=0;i<n;i++)
{
        if(key==a[i])
        {
                printf("Successful Search &
element is found at position = %d\n",i+1);
                exit(0);
        }
        printf("Unsuccessful Search\n");
 }
```

2. Binary Search: It is fast search algorithm which works on the principle of divide and conquer. For this algorithm to work properly the data collection should be in the sorted form .

1. Divides the array into three sections:

– middle element

– elements on one side of the middle element

– elements on the other side of the middle element

2. If the middle element is the correct value, done. Otherwise, go to step 1. using only the half of the array that may contain the correct value.

3. Continue steps 1. and 2. until either the value is found or there are no more elements to examine

Binary Search

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
int low, high, n, mid, i, a[50], key ;
printf("Enter the number of elements\n");
scanf("%d",&n);
printf("Enter the elements in ascending order\n");
for(i=0; i<n; i++)
{
        scanf("%d",&a[i]);
}
printf("Enter the element to be searched\n");
scanf("%d",&key);
low=0;
high=n-1;
while(low<=high)
{
        mid=(low+high)/2;
        if(key == a[mid])
        {
                printf("Successful Search &
element is found at position = %d\n",mid+1);
                exit(0);
        }
        if(key>a[mid])
        {
                low=mid+1;
        }
        else
        {
                high=mid-1;
        }
}
printf("Unsuccessful Search\n");
 }
```

Two Dimensional Array

- To store data in the form of matrices or tables, 2D arrays are used.
- An array which has two subscripts is known as two dimensional arrays.
- The first subscript represents number of rows and the second subscript represents number of columns.

Declaring two dimensional array-

Syntax:

data_type array_name[size1][size2];

where,

data_type: is the type of data to be stored

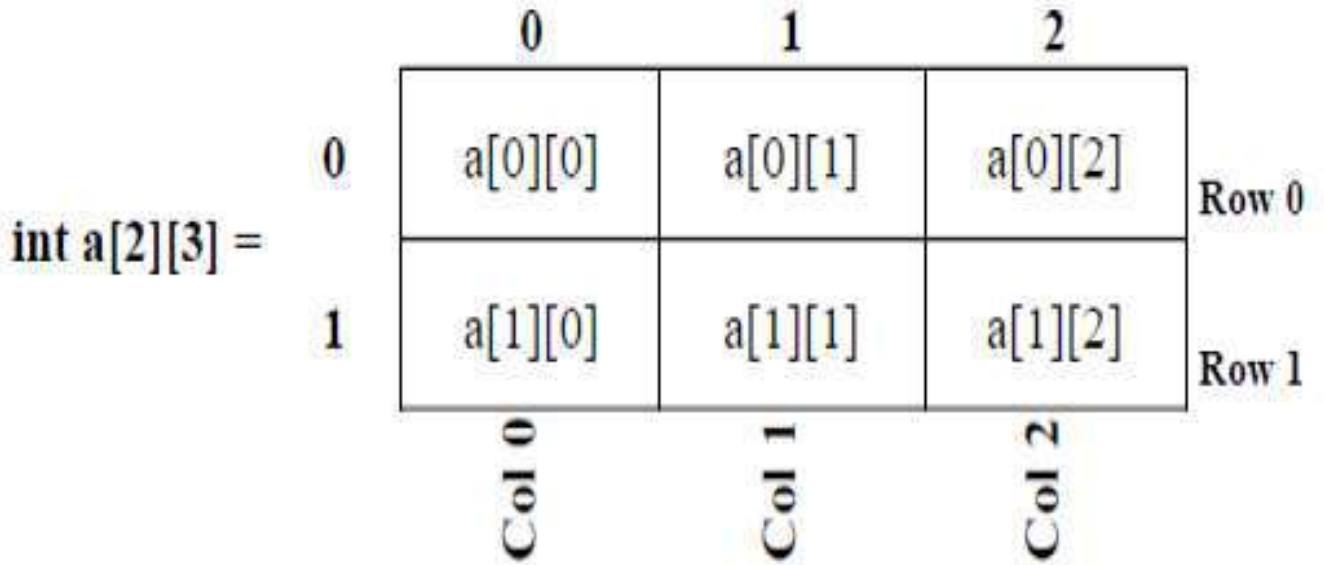
array_name: is name of the array

[size1]: indicates number of rows in the array

[size2]: indicates the number of columns in the array.

Example:

int a[2][3];



Initialization of Two Dimensional Arrays

- Similar to one-dimensional array, the elements of a two-dimensional array can be initialized Either one at a time or all at once.

- The syntax is:

data_type array_name[size1][size2] = {value1, value2, , value n}

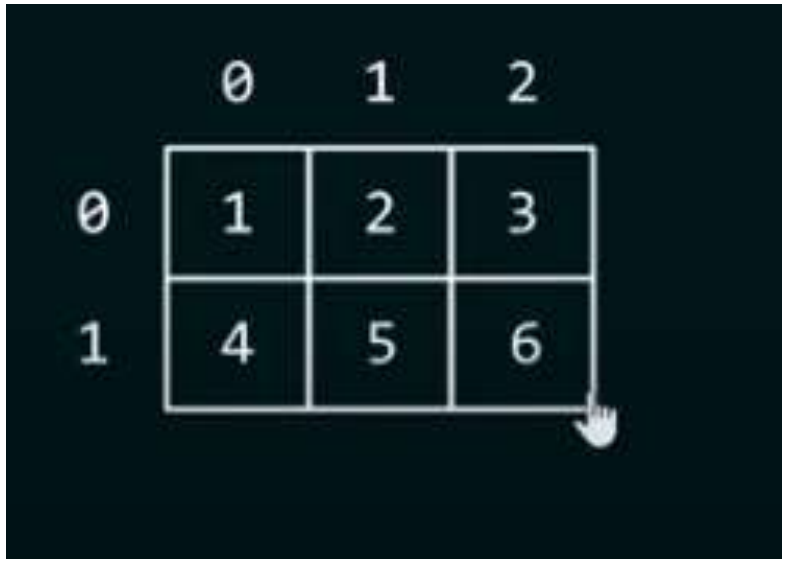
Initialization of two dimensional array :

1)Initializing row by row:- A two dimensional array can be initialized by specifying bracketed values for each row.

Example:

Method 1:

```
int a[2][3] = {1, 2, 3, 4, 5, 6};
```

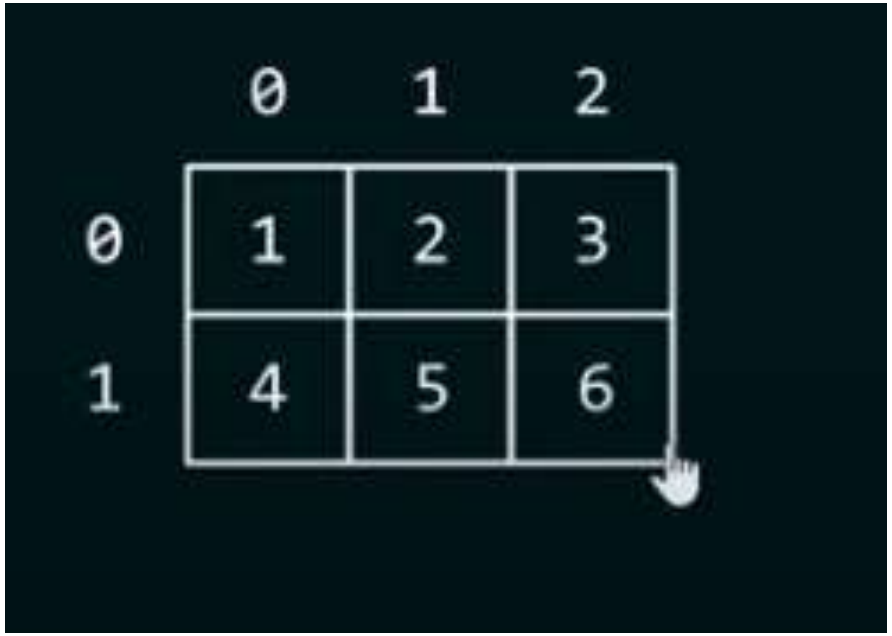


2.) Elements in first row are initialized first and then elements of second are initialized.

Method 2:

```
int a[2][3] = {{1, 2, 3}, {4, 5, 6}};
```

Row 1 **Row 2**




```
a[0][0] = 1;
a[0][1] = 2;
a[0][2] = 3;
a[1][0] = 4;
a[1][1] = 5;
a[1][2] = 6;
```

3.)**Partial array initialization**

- If some values are missing in initializer, then it is automatically set to 0.
- For ex:

```
int matrix[2][3]= {1, 23,44 };
```

Here, first row will be initialized with values, but second row will be initialized to 0.

Operations on 2D arrays

- Sum of 2 matrices
- Difference of 2 matrices
- Transpose of a matrix
- Product of 2 matrices

Write a c program to add two matrices

```c
#include<stdio.h>
int main()
{
int a[20 ][20],b[20][20],c[20][20],m,n, i,j;
printf("enter the rows and column of matrix\n");
scanf("%d%d",&m,&n);
printf("Enter the elements of Matrix A\n");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
scanf("%d",&a[i][j]);
}
}
printf("Enter the elements of Matrix B\n");
 for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
scanf("%d",&b[i][j]);
}
}
printf("Matrix Addition\n");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
c[i][j]=a[i][j]+b[i][j];
}
}
printf("The resultant matrix is\n");
 for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
printf("%d\t",c[i][j]);
}
printf("\n");
}
return 0;
}
```

Write a c program to add two matrices

```
enter the rows and column of matrix
3
3
Enter the elements of Matrix A
1 2 1
2 3 4
1 2 3
Enter the elements of Matrix B
4 3 2
1 2 3
1 1 1
Matrix Addition
The resultant matrix is
5       5       3
3       5       7
2       3       4
```

Write a C Program to find Transpose of m X n matrix

```
include<stdio.h>
void main()
{
        int a[20 ][20],b[20][20], m,n, i,j;
        printf("enter the rows and column of matrix\n");
        scanf("%d%d",&m,&n);
        printf("Enter the elements of Matrix\n");
        for(i=0;i<m;i++)
        {
                for(j=0;j<n;j++)
                {
                        scanf("%d",&a[i][j]);
                }
        }
        printf("          elements of Matrix are\n");
        for(i=0;i<m;i++)
        {
                for(j=0;j<n;j++)
                {
                        printf("%d\t",a[i][j]);
                }
                printf("\n");
        }
```

```
printf("Matrix Transpose\n");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
b[j][i]=a[i][j];
}
}
printf("The Transpose of the matrix is \n");
 for(i=0;i<n;i++)
{
for(j=0;j<m;j++)
{
printf("%d\t",b[i][j]);
}
printf("\n");
}
return 0;
}
```

Write a C Program to find Transpose of m X n matrix

```
enter the rows and column of matrix
2 2
Enter the elements of Matrix
1 2 3 4
Display the Matrix
1       2
3       4
Matrix Transpose
The Transpose of the matrix is
1       3
2       4
```

Homework

- **Write a C Program to read display a 3X3 matrix.**
- **Write a C Program to print 12 34**
  56 32
- **Write a C Program to find difference of 2 matrices.**
- **Write a C Program to find Transpose of 3 X 3 matrix**

Multi-Dimensional Arrays

Multidimensional arrays in an array of arrays.

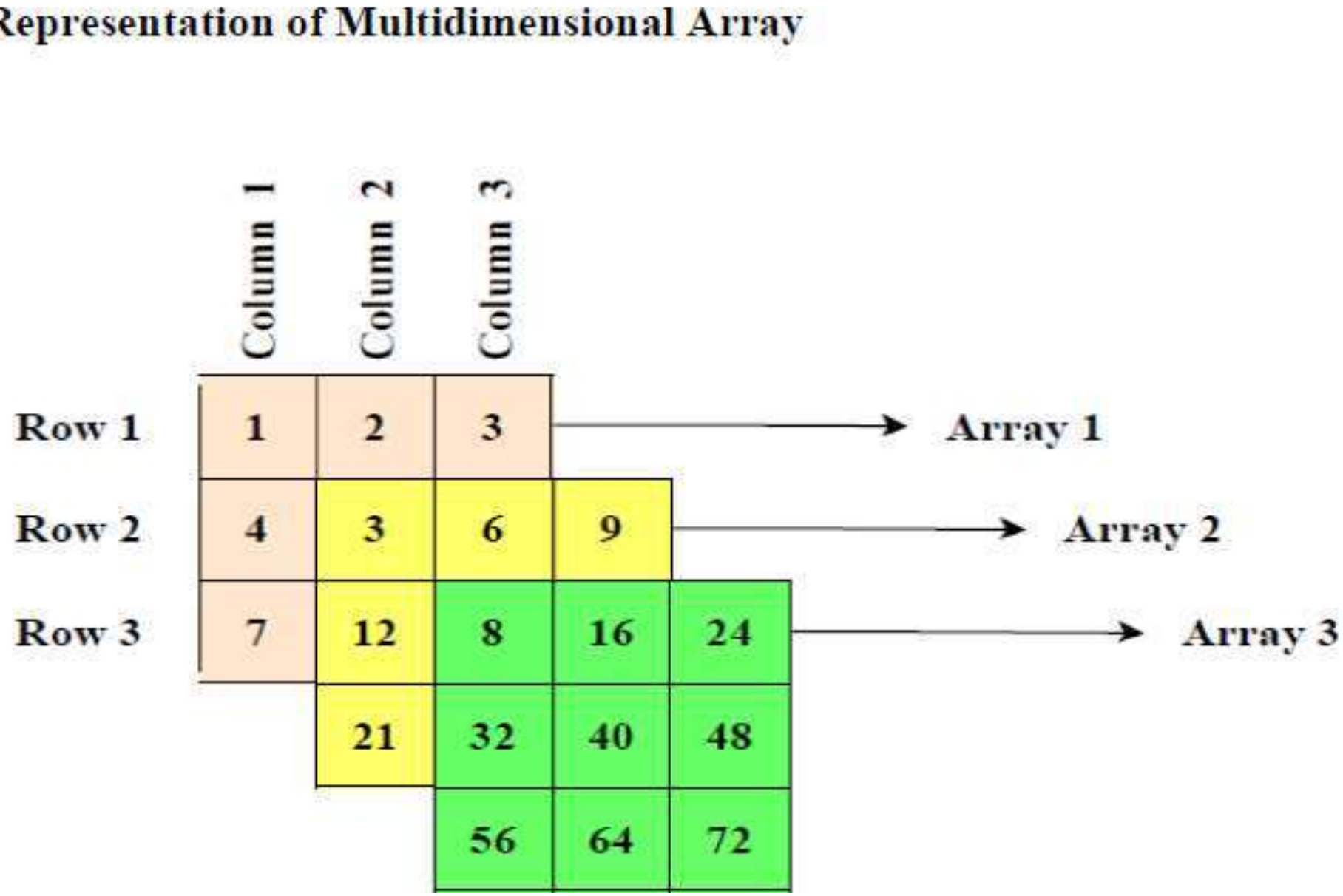
The total number of elements that can be stored in a multidimensional array can be calculated by multiplying the size of all the dimensions.

The general form of multidimensional array is

data_type array_name[s1][s2][s3]….[sn];

Where s1 is the size of ith dimension.

Multi-Dimensional Array



Applications of arrays

- To implement mathematical vectors, matrices, other kinds of rectangular tables.
- Many databases include 1D arrays whose elements are records.
- To sort elements in ascending/descending order.
- To implement data structures like strings, queues, stacks.

FUNCTIONS

Definition

- A function is a collection of statements that perform a specific task.
- These functions are very useful to read write and debug complex programs.

Types of Functions

- These can be broadly classified into two types
  - Built-in functions
  - User defined functions

Why are functions needed

- Dividing the program into separate modules allows each function to be written and tested separately.
- Understanding, coding and testing multiple separate functions is easier.
- A big program can be broken into smaller function, then divide the workload by writing different functions.
- Functions can make code faster by coding logic once instead of repeating several times.

Using Functions

- A function *f* that uses another function *g* is known as the *calling function* and *g* is known as the *called function*.
- The inputs that a function takes are known as *arguments/ parameters*.
- When a called function returns some result back to the calling function, it is said to *return* that result.
- The calling function may or may not pass *parameters* to the called function. If the called function accepts arguments, the calling function will pass parameters, else it will not do so.
- *Function declaration* is a declaration statement that identifies a function with its name, a list of arguments that it accepts, and the type of data it returns.
- *Function definition* consists of a function header that identifies the function, followed by the body of the function containing the executable code for that function.

- **User defined functions** :-

The user defined function is defined by the user according to its requirements.

- Parts of user defined function.
  - Function Declaration or Function prototype
  - Function call
  - Function Definition

1. Function Declaration or Function prototype :-

- It will inform the compiler about the return type, function name and number of arguments along with the data types.
- syntax:

return_data_type function_name(data_type variable1, data_type variable2,…);

- **return_data_type** :- is the data type of the value that is returned or sent from the function.
- **function_name** :-valid name for the function.
- **data_type variable1, data_type variable2,.. is the list of variables of specified data types.**

Function Declaration or Function prototype :-

- Example:-

int large (int x, int y);

or

int large (int, int);

- is a function declaration with function_name “large” with return _type “integer” and has two arguments “x” and “y” of integer type.
- NOTE:-
  - if we define a function before main ( ) function the there is no need of function declaration
  - if we define the function after main ( ) function then it is mandatory to declare the function because it will inform the compiler.

2. Function definition

- The declared function must define the same to perform the specific task.
- A function definition comprises 2 parts: function header & function body.
- Syntax-

```
return_data_type function_name(data_type variable1, data_type variable2,…)
        {
                statements;
                return variable;
        }
```

3. Function call The function call statement invokes the function.

- When a function is invoked, compiler jumps to that function to execute the statements that are a part of that function.
- Syntax:

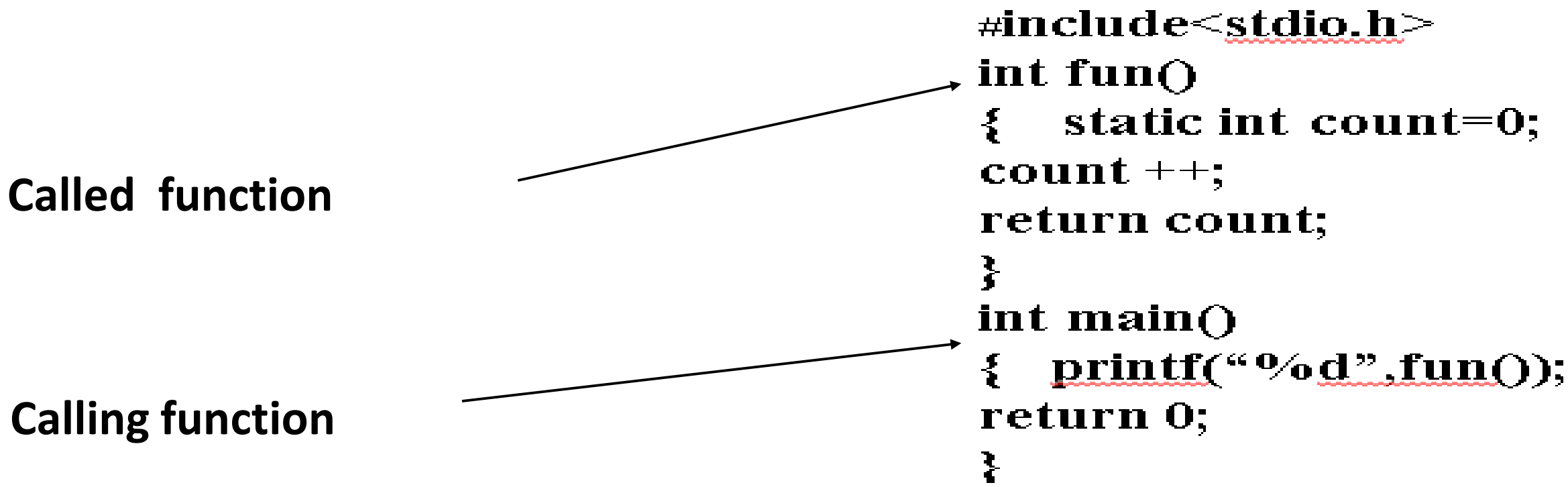
function_name(variable1, variable2,..);

Ex- Write a C program to add 2 integers using functions.

```
#include<stdio.h>
int sum(int a,int b); //function declaration
int main()
{
int num1,num2,total=0;
printf("Enter 2 numbers\n");
scanf("%d%d",&num1,&num2);
total=sum(num1,num2); //function call
printf("Sum = %d\n",total);
return 0;
}
//function definition
int sum(int a,int b)
{
int result;
result=a+b;
return result;
}
```

Calling function and called function :-

- The **function main( )** that calls another function is called calling function.
- **The function** being called by the calling function is known as called function.



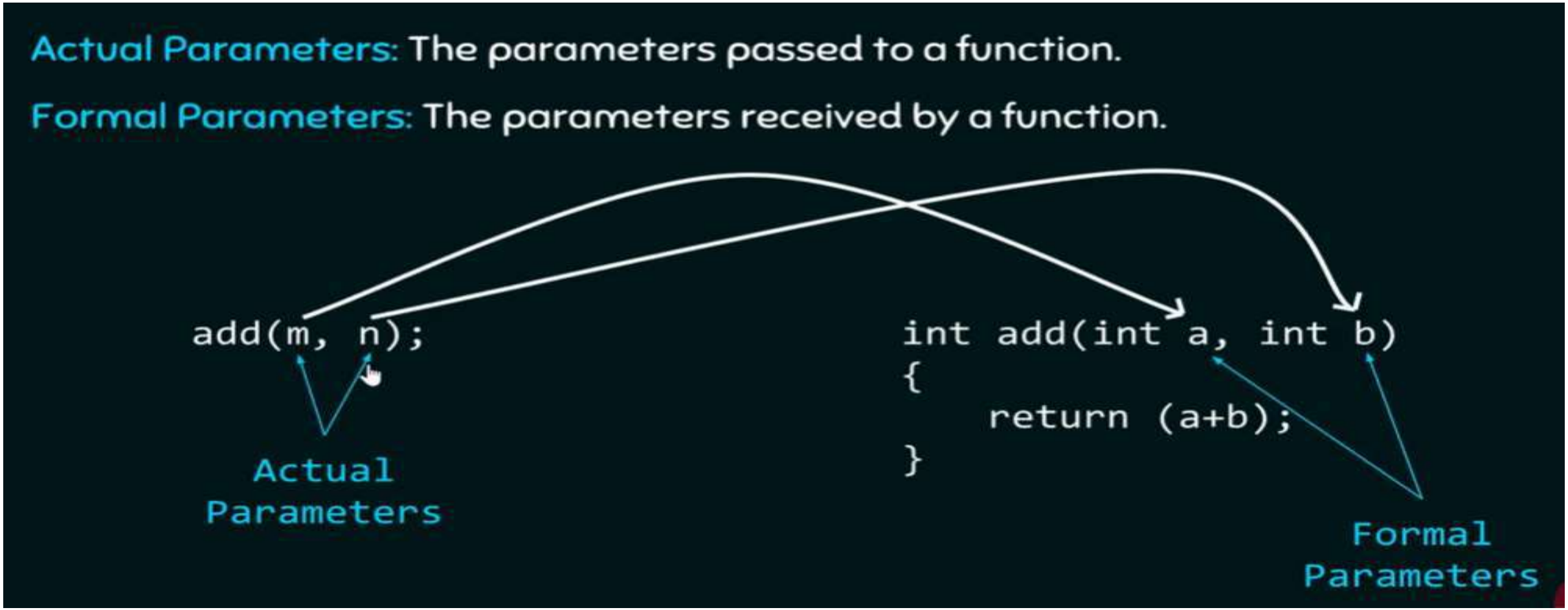
return Statement

- A return statement **ends the execution of a function, and returns control to the calling function**.
- Syntax. **return <expression>;**

Example: return 10; return a; return a+b;

- The **value** will be **passed back** to the function where it was called.
- Function that has **void** as its return statement **cannot return** any value to the calling function,

Actual arguments and Formal arguments :-



Parameter passing mechanism

There are two methods by which parameters or arguments can be passed to the function

- **Call by value**
- **Call by reference**

1. Call by value

- In call by value method, the values of actual parameters are copied to formal parameters.
- These 2 different parameters store values in different memory locations.
- One is the original copy and the other is the function copy.
- Any changes made inside functions are not reflected in the actual parameters of the caller.

C program to demonstrate call by value

```
#include<stdio.h>
void add(int n);
int main()
{
        int num=2;
        printf(“Value of num before calling function=%d”,num);
        add(num);
        printf(“Value of num after calling function=%d”,num);
        return 0;
}
void add(int n)
{
        n=n+10;
        printf(“Value of num in called function=%d”,n);
}
```

Output-

Value of num before calling function=2

Value of num in called function=12

Value of num after calling function=2

2. Call by reference

- In call by reference, the address of the actual parameters is passed to the function as the formal parameters.
- Both the actual and formal parameters refer to the same locations.
- Any changes made to formal parameters are actually reflected to actual parameters.

C program to swap two numbers using call by reference

```
#include<stdio.h>
void swap(int *n1,int *n2);
void main()
{
        int a,b;
        printf("Enter two numbers\n");
        scanf("%d%d",&a,&b);
        printf("Before Swapping\n a=%d\t b=%d\n",a,b);
        swap(&a, &b);
        printf("After Swapping\n a=%d\t b=%d\n",a,b);
}
void swap(int *n1, int *n2)
{
         int temp;
       temp=*n1;
         *n1=*n2;
         *n2=temp;
}
```

Advantages and Disadvantage of Call by value and Reference

Call by reference

- Advantage: Is more efficient than copying
- Disadvantages:
  - Leads to aliasing: when there are two or more different names for the same storage location
  - Side effects not visible from code itself

Call by value-result

- Has all advantages and disadvantages of call-by-value and call-by-result together.

Scope of variables

The scope of a variable is the block of code in the entire program where the variable is declared, used, and can be modified.

1. **Block Scope:** A **Block** in C is a set of statements written within the right and left braces. A variable declared within a Block has a Block Scope. Scope of such a variable ends where the block ends.

```
#include<stdio.h>
 int main()  { // Block
 { //Variables within the block
 int a = 8, b = 10;
printf ("The values are: %d, %d\n", a, b);
}
 return  0;
}
```

2.Function scope

- Function scope indicates that a variable is active and visible from beginning to end of a function.
- In C only goto label has function scope.

Scope of variables

3. **Program scope**

Global variables declared outside the function bodies have a **program scope**. The availability of global variables stays for the entire program after its declaration

```c
#include <stdio.h>
int a = 8;//Declare Global Variables
float b = 7.5;
int test(){
  b = b + a;
  return b;
}
int main(){
  //Access a
  printf ("The value of a is: %d\n", a);
  //Access b
  printf ("The value of b is: %f\n", b);
  return 0;
}
```

Output

```
The value of a is: 8
The value of b is: 7.500000
```

Scope of variables

4. File scope

- These variables are usually declared outside of all of the functions and blocks, at the top of the program and can be accessed from any portion of the program.
- The *global static variable* is accessible by all the functions in the same source file as the variable. This variable has a File Scope.

```c
#include <stdio.h>
static int a = 20;
int func() {
  a = a + 20;
  printf ("The value of a is: %d\n", a);
}
int main() {
  func();
  a = a + 5;
  printf ("The value of a is: %d\n", a);
  return 0;
  }
```

```
The value of a is: 40
The value of a is: 45
```

- **Passing Arrays to functions** :- Array elements or an entire array can be passed to a function such a mechanism is called as passing array to the function.

C program to read and print an array of n numbers by passing array to function.

```
#include<stdio.h>
void read_array(int a[], int n);
void display_array(int a[], int n);
int main()
{
int a[10],n;
printf(“Enter size of array\n”);
scanf(“%d”,&n);
read_array(a,n);
display_array(a,n);
return 0;
}
```

```
void read_array(int a[10], int n)
{
int i;
printf(“Enter elements of array\n”);
for(i=0;i<n;i++)
{
        scanf(“%d”,&a[i]);
}
}
void display_array(int a[10],int n)
{
int i;
printf(“Elements of array are\n”);
for(i=0;i<n;i++)
{
        printf(“%d”,a[i]);
} }
```

C Program to find largest array element by passing array to the functions

```
#include<stdio.h>
int largest(int a[20],int n);
void main()
{
   int a[20],n,i,max;
   printf("Enter the value of n\n");
   scanf("%d",&n);
   printf ("Enter %d values\n",n);
   for(i=0;i<n;i++)
    scanf("%d",&a[i]);
   max=largest(a,n);
   printf("Largest element in array=%d\n",max);
}
```

```
int largest(int a[20],int n)
{
    int max,i;
    max=a[0];
    for(i=1;i<n;i++)
    {
        if(a[i]>max)
            max=a[i];
    }
    return max;
}
```

Recursion

- Recursion is the **process in which function calls itself repeatedly until a particular condition is met.**

1.) Write a C program to find factorial of the number using recursion

```
#include<stdio.h>
int factorial(int n);
void main()
{
   int n,res;
   printf("Enter the value of n\n");
    scanf("%d",&n);
    res=factorial(n);
   printf("The factorial of the number=%d\n",res);
}
int factorial(int n)
{
    if(n==0)
    return 1;
   else
     return n*factorial(n-1);
}
```

2.) Write a C program to generate fibonacci series upto N numbers using recursion

```c
#include<stdio.h>
int fibonacci(int n);
void main()
{
    int num,i;
    printf("Enter the number\n");
    scanf("%d",&num);
    printf("\nFibonacci Series\n");
    for(i=0;i<num;i++)
    {
        printf("%d\t",fibonacci(i));
    }
}
```

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

3.) Write a C program to calculate exp(x,y) using recursive functions.

```
#include<stdio.h>
int exp(int x,int y);
void main()
{
int num1,num2,res;
printf("Enter 2 numbers\n");
scanf("%d%d",&num1,&num2);
res=exp(num1,num2);
printf("Result= %d\n", res);
}
```

```
int exp(int x,int y)
{
        if(y==0)
          return 1;
       else
         return (x*exp(x,y-1));
}
```

4.) Write a C program to calculate GCD of 2 numbers using recursive functions.

```
#include<stdio.h>
int GCD(int x,int y);
void main()
{
int num1,num2,res;
printf("Enter 2 numbers\n");
scanf("%d%d",&num1,&num2);
res=GCD(num1,num2);
printf("GCD= %d\n", res);
}
```

```
int GCD(int x,int y)
 {
    int rem;
    rem=x%y;
    if(rem==0)
       return y;
     else
       return (GCD(y,rem));
  }
```