Functions in C

Functions in C
Functions in C
Functions in C

Course Code: CSE-121

Course Title: Structural Programming Language

Khandaker Jannatul Ritu, Lecturer, BAIUST

Outlines

C Functions

A **function in C** is a set of statements that when **called** perform some **specific task**. It is the basic building block of a C program that provides modularity and code reusability. The programming statements of a function are enclosed within **{ } braces**, having certain meanings and performing certain operations. They are also called **subroutines or procedures** in other languages.
In this article, we will learn about functions, function definition. declaration, arguments and parameters, return values, and many more.

Syntax of Functions in C
The syntax of function can be divided into 3 aspects:
1.Function Declaration
2.Function Definition
3.Function Calls

```c
#include <stdio.h>

int sum(int a, int b)
{
  return a + b;
}

int main()
{
  int add = sum(10, 30);
  printf("Sum is: %d", add);
}
```

Why Function?

Before Using Function

```
#include <stdio.h>
int main(){
   int length, width;
   scanf("%d %d", &length, &width);
   printf("Area of Rectangle : %d\n", length * width);

   scanf("%d %d", &length, &width);
   printf("Area of Rectangle : %d\n", length * width);

   scanf("%d %d", &length, &width);
   printf("Area of Rectangle : %d\n", length * width);

   scanf("%d %d", &length, &width);
   printf("Area of Rectangle : %d\n", length * width);
}
```

After Using Function

```
#include <stdio.h>

void rectange(int l, int w){
   printf("Area of Rectangle : %d\n", l*w);
}

int main(){
   rectange(14,5);
   rectange(3,2);
   rectange(1,8);
   rectange(12,9);
}
```

Why Function?

Before Using Function

```
#include <stdio.h>
int main(){
    char student1[20];
    scanf("%s", student1);
    printf("Hi %s !\n", student1);

    char student2[20];
    scanf("%s", student2);
    printf("Hi %s !\n", student2);

    char student3[20];
    scanf("%s", student3);
    printf("Hi %s !\n", student3);
}
```

After Using Function

```
#include <stdio.h>
void student_name(){
    char student[20];
    printf("What is your name ? = ");
    scanf("%s", student);
    printf("Hi %s !\n", student);
}
int main(){
    student_name();
    student_name();
    student_name();
    student_name();
}
```

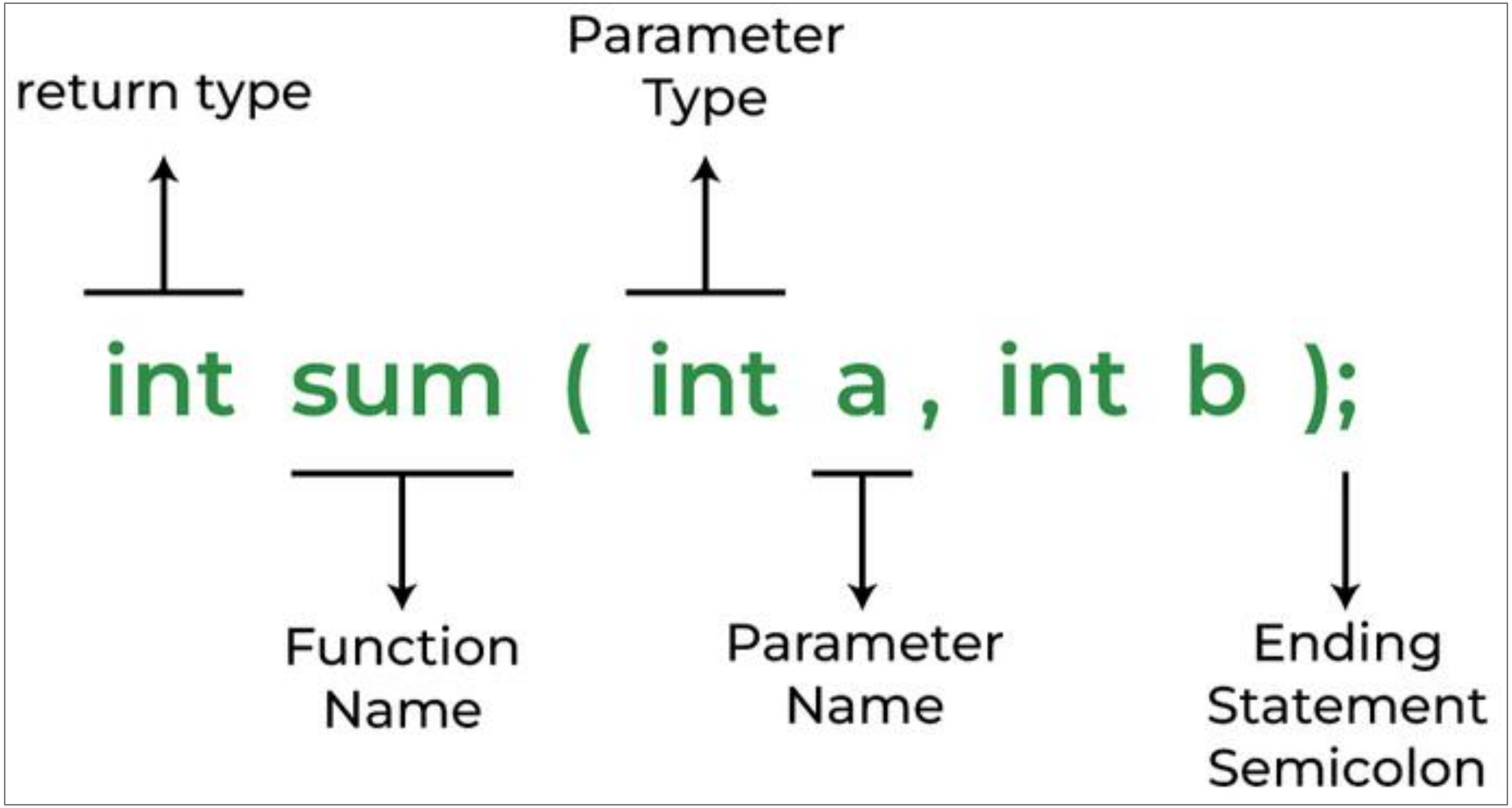
Before Using Function

```
#include <stdio.h>
int main(){
    int n1; scanf("%d", &n1);
    int array1[n1];
    for(int i=0; i<n1; i++){
        scanf("%d", &array1[i]);
    }
    for(int i=0; i<n1; i++){
        printf("%d ", array1[i]);
    }
    printf("\n");
    int n2; scanf("%d", &n2);
    int array2[n2];
    for(int i=0; i<n2; i++){
        scanf("%d", &array2[i]);
    }
    for(int i=0; i<n2; i++){
        printf("%d ", array2[i]);
    }
}
```

After Using Function

```
#include <stdio.h>
void MyArray(){
    printf("Enter array size: ");
    int n; scanf("%d", &n);
    int array[n];
    printf("Enter values: ");
    for(int i=0; i<n; i++){
        scanf("%d", &array[i]);
    }
    printf("Output: ");
    for(int i=0; i<n; i++){
        printf("%d ", array[i]);
    }
    printf("\n\n");
}
int main(){
    MyArray();
    MyArray();
    MyArray();
}
```

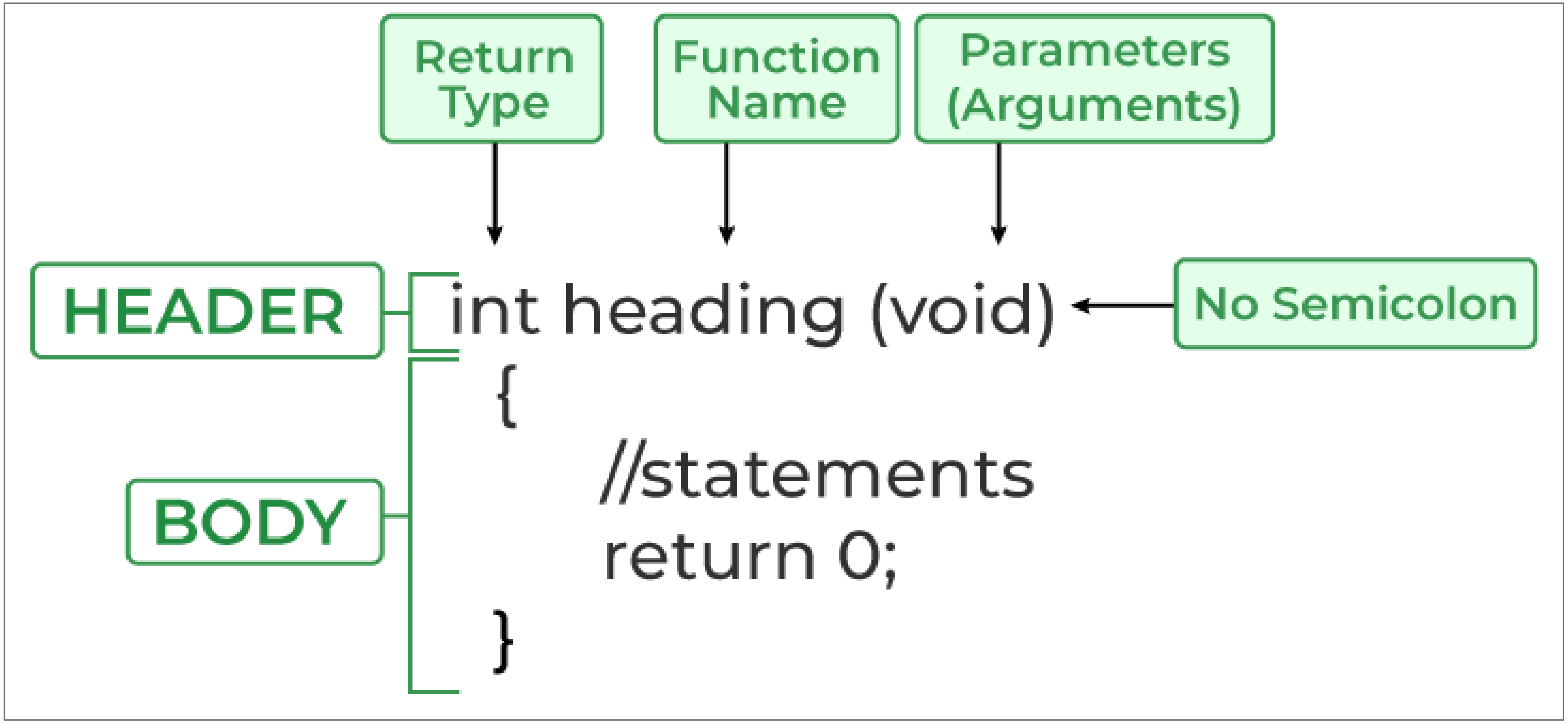
❑ Function Declaration



Functions are the block of code that is executed every time they are called during an execution of a program.

❑ Function definition

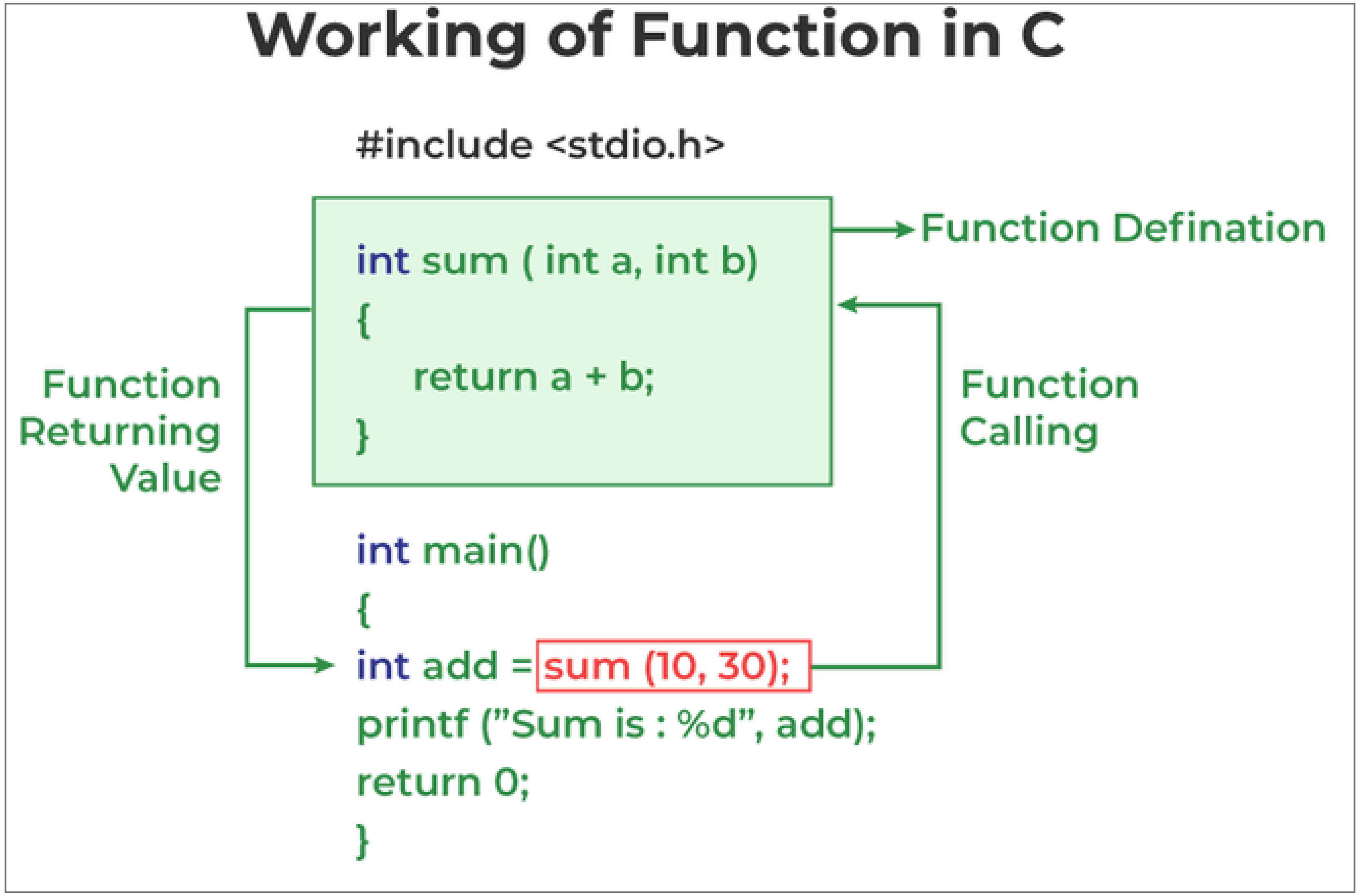
The function definition consists of actual statements which are executed when the function is called (i.e. when the program control comes to the function).



```
int heading (void)
  {
      //statements
      return 0;
  }
```

❑ Function Call

A function call is a statement that instructs the compiler to execute the function. We use the function name and parameters in the function call.



```
#include <stdio.h>

int sum ( int a, int b)
{
    return a + b;
}

int main()
{
int add = sum (10, 30);
printf ("Sum is : %d", add);
return 0;
}
```

❑ Function return type

Function return type tells what type of value is returned after all function is executed. When we don't want to return a value, we can use the void data type.

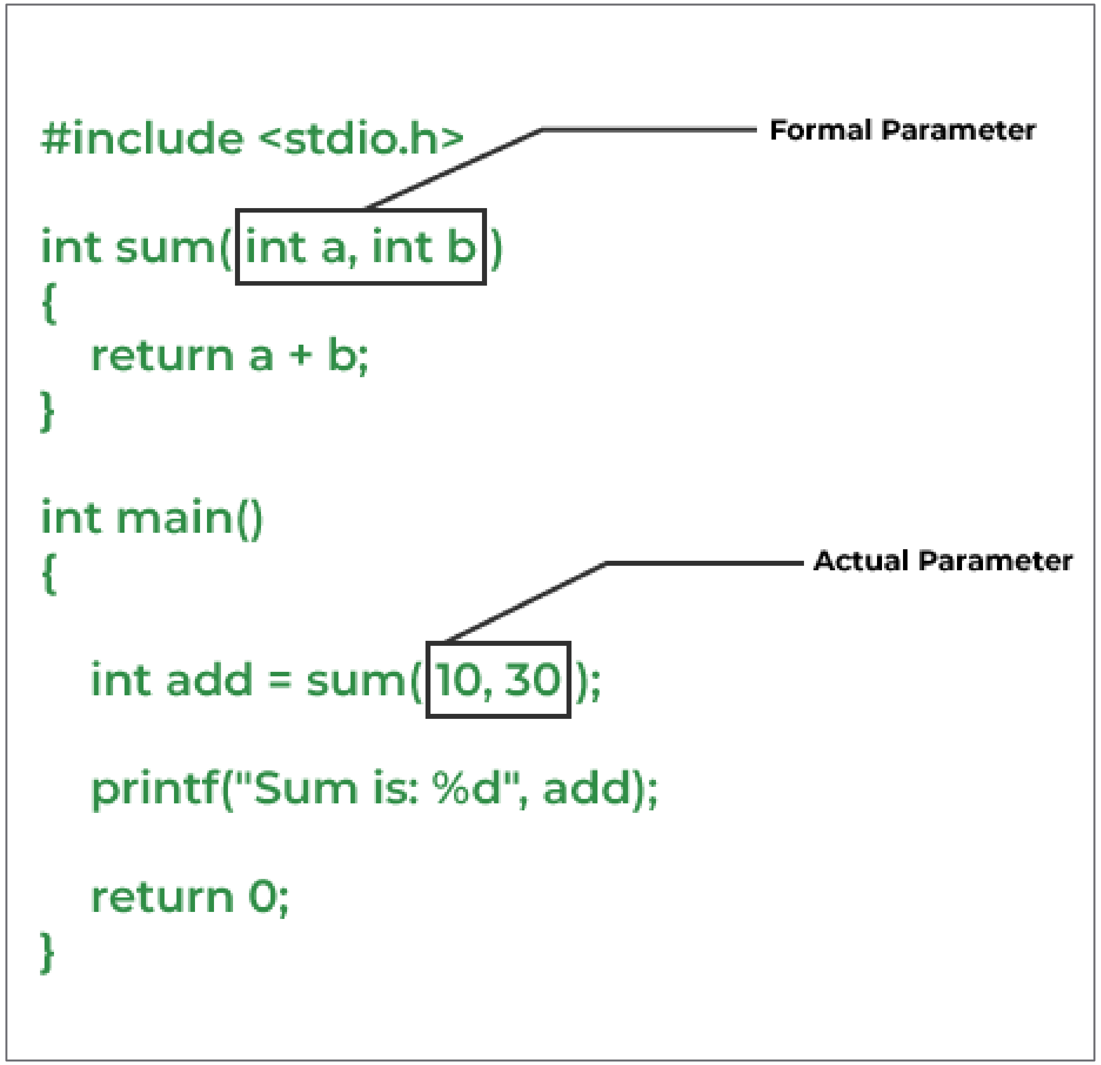
Example:
int func(parameter_1,parameter_2);
The above function will return an integer value after running statements inside the function.

❑ Parameters in Functions:-

<u>Function Arguments </u>(also known as Function Parameters) are the data that is passed to a function.

```
int function_name(int var1, int var2);
```

Formal parameter
Actual parameter/ Function arguments/Function parameter



```
#include <stdio.h>

int sum(int a, int b)
{
   return a + b;
}

int main()
{

   int add = sum(10, 30);

   printf("Sum is: %d", add);

   return 0;
}
```

Q-1. What is the actual and formal parameter?

Formal parameter: The variables declared in the function prototype is known as Formal arguments or parameters.
Actual parameter: The values that are passed in the function are known as actual arguments or parameters.

Q-2. What is the difference between function arguments and parameters?

Function parameters are the values declared in a function declaration. Whereas, function arguments are the values that are passed in the function during the function call.

Example:

```
int func(int x,int y);
func(10,20);
```

Here, int x and int y are parameters while,
10 and 20 are the arguments passed to the function.

Q-3. Can we return multiple values from a C Function?

No, it is generally not possible to return multiple values from a function. But we can either use pointers to static or heap memory locations to return multiple values or we can put data in the structure and then return the structure.

Conditions Of Return Type And Arguments

✓ **Types of Function According to Arguments and Return Value**

Functions can be differentiated into 4 types according to the arguments passed and value returns these are:

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

1. Function with no argument and no return value

```
#include <stdio.h>
void myfunction()
{
    printf("Hello World");
}
int main()
{
    myfunction();
}
```

```
Output:
Hello World
```

2. Function with arguments but no return value

```c
#include <stdio.h>
void myfunction(int x)
{
    printf("The power of %d = %d", x, x*x);
}
int main(){
    myfunction(5);
}
```

Output:

```
The power of 5 = 25
```

3. Function with no arguments but returns a value

```c
#include <stdio.h>
int myfunction()
{
    return 10*10;
}
int main()
{
    int answer = myfunction(); ///method-1
    printf("%d \n", answer);
    printf("%d", myfunction()); ///method-2
}
```

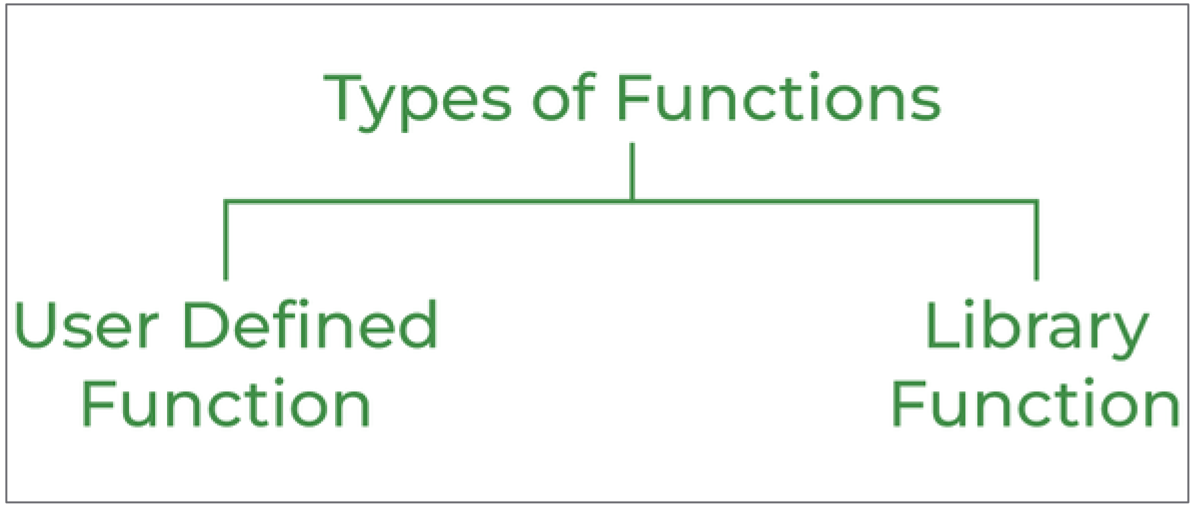
```
Output:
100
100
```

4. Function with arguments and return value

```c
#include <stdio.h>
int myfunction(int x, int y, int z)
{
    int add = x+y+z;
    return add/3;
}
int main()
{
    int answer = myfunction(2,4,6); ///method-1
    printf("%d \n", answer);
    printf("%d", myfunction(10,4,6)); ///method-2
}
```

```
Output:
4
6
```

❑ Types of Functions:



1. Library Function / built-in function

A compiler package already exists that contains these functions, each of which has a specific meaning and is included in the package. Built-in functions have the advantage of being directly usable without being defined, whereas user-defined functions must be declared and defined before being used.

For Example: pow(), sqrt(), strcmp(), strcpy()

✓ Advantages of C library functions

- C Library functions are easy to use and optimized for better performance.
- C library functions save a lot of time i.e, function development time.
- C library functions are convenient as they always work.

Library/Built-in functions:

```
#include <math.h>
#include <stdio.h>
int main()
{
  double Number;
  Number = 49;
  double squareRoot = sqrt(Number);
  printf("The Square root of %.2lf = %.2lf", Number, squareRoot);
}
```

2. User Defined Function

Functions that the programmer creates are known as User-Defined functions or **“tailor-made functions”**. User-defined functions can be improved and modified according to the need of the programmer. Whenever we write a function that is case-specific and is not defined in any header file, we need to declare and define our own functions according to the syntax.

Advantages of User-Defined Functions

- Changeable functions can be modified as per need.
- The Code of these functions is reusable in other programs.
- These functions are easy to understand, debug and maintain.

```c
#include <stdio.h>
int sum(int a, int b)
{
  return a + b;
}
int main()
{
  int a = 30, b = 40;
  int res = sum(a, b);
  printf("Sum is: %d", res);
}
```

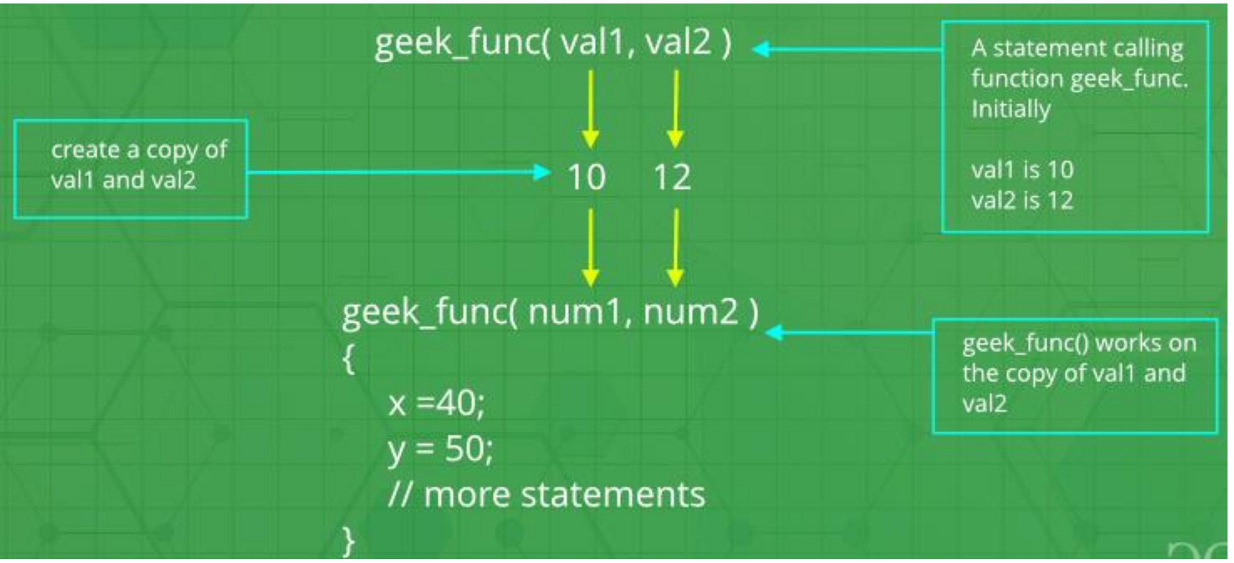
❑ Passing Parameters to Functions

We can pass arguments to the C function in two ways:
1.Pass by Value/ Call by value
2.Pass by Reference/ Call by reference

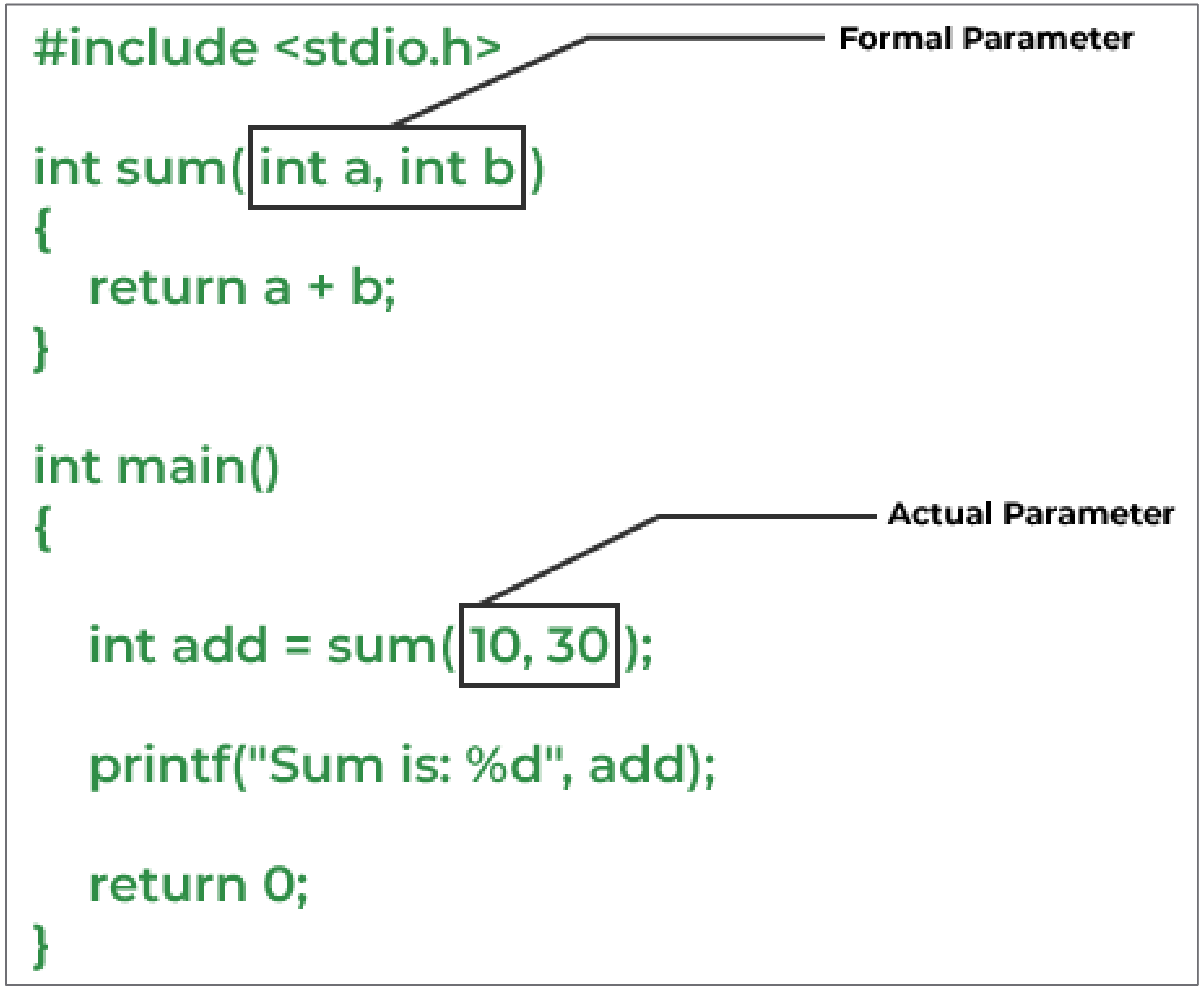
1. Pass By Value

In call by value method of parameter passing, the values of actual parameters are copied to the function's formal parameters.

There are two copies of parameters stored 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.


geek_func(val1, val2)
A statement calling function geek_func. Initially
val1 is 10
val2 is 12
create a copy of val1 and val2
10
12
geek_func() works on the copy of val1 and val2

```
geek_func( num1, num2 )
{
    x =40;
    y = 50;
    // more statements
}
```



```
#include <stdio.h>

int sum(int a, int b)
{
    return a + b;
}

int main()
{

    int add = sum(10, 30);

    printf("Sum is: %d", add);

    return 0;
}
```

// C program to illustrate call by value

```c
#include <stdio.h>
void func(int a, int b)
{
    a += b;
    printf("In func, a = %d b = %d\n", a, b);
}
int main(void)
{
    int x = 5, y = 7;
    func(x, y);
    printf("In main, x = %d y = %d\n", x, y);
}
```

```c
#include <stdio.h>
void swap(int a, int b)
{
    int temp = a;
    a = b;
    b = temp;
}
int main()
{
    int x = 10, y = 20;
    printf("before swap are: %d, %d\n", x, y);
    swap(x, y);
    printf("after swap are: %d, %d", x, y);
}
```

2. Pass by Reference

In call by reference method of parameter passing, the address of the actual parameters is passed to the function as the formal parameters. In C, we use pointers to achieve call-by-reference.

•Both the actual and formal parameters refer to the same locations.

•Any changes made inside the function are actually reflected in the actual parameters of the caller.

```
#include <stdio.h>
void swap(int* a, int* b){
   int temp = *a;
   *a = *b;
   *b = temp;
}
int main(){
   int x = 10, y = 20;
   printf("x and y before swap are: %d, %d\n", x, y);
   swap(&x, &y);
   printf("x and y after swap are: %d, %d", x, y);
}
```

➢ Difference between the Call by Value and Call by Reference in C

Call by Value	Call by Reference
In this method the value of actual parameters are copied to formal parameters.	In this method the address of actual parameters are copied to formal parameters
The changes made to formal parameters does not affect the actual parameters.	The changes made to the formal parameters reflect the changes to the actual parameters.
We can pass constant values as an argument in call by value method.	We cannot pass constant values as an argument in this method.
Call by value is considered safer as original data is preserved	Call by reference is risky as it allows direct modification in original data

❑ Passing Array To A Function

```
#include <stdio.h>
void check(int pupil[], int sz){
    for(int i=0; i<sz; i++){
        printf("%d ", pupil[i]);
    }
}
int main(){
    int N; scanf("%d", &N);
    int student[N];
    for(int i=0; i<N; i++){
        scanf("%d", &student[i]);
    }
    check(student, N);
}
```

✓ Careful, It is not passing array as a parameter to a function!

```
#include <stdio.h>
void MyArray(){
    int n; scanf("%d", &n);
    int array[n];
    for(int i=0; i<n; i++){
        scanf("%d", &array[i]);
    }
    for(int i=0; i<n; i++){
        printf("%d ", array[i]);
    }
}
int main(){
    MyArray();
}
```

❑ Passing Characters To A Function

```
#include <stdio.h>

void myfunction(char* str){
    printf("Output: %s", str);
}

int main()
{
    int str[100];
    scanf("%s", str);
    myfunction(str);
}
```

❑ C program to find the x^y without library functions

```c
#include <stdio.h>
int power(int x, int y){
    int ans = 1;
    for(int i=1; i<=y; i++){
        ans = ans * x;
    }
    return ans;
}
int main(){
    printf("power :%d\n", power(2,4));
    printf("power :%d\n", power(5,2));
    printf("power :%d\n", power(2,5));
}
```

❑ Exercise problems of functions: solve this functions

Ex-1:

```
#include <math.h>
#include <stdio.h>
int sum();
int main()
{
        int num;
        num = sum();
        printf("Sum of two given values = %d", num);
        return 0;
}
int sum()
{
        int a = 50, b = 80, sum;
        sum = sqrt(a) + sqrt(b);
        return sum;
}
```

Ex-2:

```
#include <stdio.h>
void value(void);

void main() {
value();
}
void value(void)
{
        float year = 1, period = 5, amount = 5000, inrate = 0.12;
        float sum;
        sum = amount;
        while (year <= period) {
                sum = sum * (1 + inrate);
                year = year + 1;
        }
        printf(" The total amount is :%f", sum);
}
```

Ex-3:

```
#include <stdio.h>
int sum(int x, int y)
{
        int c;
        c = x + y;
        return c;
}
int main()
{
        int a = 3, b = 2;
        int c = sum(a, b);
        printf("Sum of %d and %d : %d", a, b, c);
}
```

Ex-4:

```
#include <stdio.h>
int sum(int, int);
int sum(int x, int y)
{
        int sum;
        sum = x + y;
        return x + y;
}
int main()
{
        int x = 10, y = 11;
        int result = sum(x, y);
        printf("Sum of %d and %d = %d ", x, y, result);
}
```

Ex-5:

```
#include <stdio.h>
void myfunction(float area)
{
   printf("Area : %d", area);
}
int main()
{
   int a,b,c;
}
```

Ex-6:

```
#include <stdio.h>
void myfunction()
{
   int x;
   printf("Enter a value: ");
   scanf("%d", &x);
   printf("The value is : %d", x);
}
int main()
{
   myfunction();
}
```

Ex-7:

```
#include <stdio.h>
void flag(float sum){
   printf("sum : %f", sum);
}
void myfunction(int x, float y)
{
   float sum = x + y;
   flag(sum);
}
int main(){
   int a;
   float b;
   printf("Enter two value: ");
   scanf("%d %f", &a, &b);
   myfunction(a, b);
}
```

Ex-8

```
#include <stdio.h>
int checkOddEven(int n1)
{
   return (n1 % 2);
}
int main(){
   int n1;
   printf("Input any number : ");
   scanf("%d", &n1);
   if(checkOddEven(n1))
   {
      printf("The entered number is odd.\n\n");
   }
   else
   {
      printf("The entered number is even.\n\n");
   }
}
```

Ex-9:

```
#include <stdio.h>
#include <string.h>
void fun(char* arr)
{
    int i;
    unsigned int n = strlen(arr);
    printf("n = %d\n", n);
    for (i = 0; i < n; i++)
        printf("%c ", arr[i]);
}
int main()
{
    char arr[] = { 'g', 'e', 'e', 'k', 's', 'q', 'u', 'i', 'z' };
}
```

Ex-10:

```
#include <stdio.h>
#include <string.h>
void fun(char* arr)
{
    int i;
    unsigned int n = strlen(arr);
    printf("n = %d\n", n);
    for (i = 0; i < n/2; i++)
        printf("%c ", arr[i]);
}
int main()
{
    char arr[] = "geeksquiz";
    fun(arr);
}
```

Ex-11:

```
#include <stdio.h>
void printString(char* str)
{
    printf("Array of Characters: ");

    int i = 0;
    while (str[i] != '\0') {
        printf("%c \n", str[i]);
        i++;
    }
}
int main()
{
    char arr[] = "String";
    printString(arr);
}
```

❑ Use of Global variable in function

Ex-1:

```
#include <stdio.h>
int value = 100;

void check(){
   printf("check() function : %d\n", value);
}

int main(){
   check();
   printf("main() function: %d\n", value);
   check();
}
```

Ex-2:

```
#include <stdio.h>
void check(){
   int value = 100;
   printf("check() function : %d\n", value);
}
int main(){
   int value = -5;
   check();
   printf("main() function: %d\n", value);
   check();
}
```

Ex-3:

```
#include <stdio.h>
int answer;
void check(){
   printf("inside check() function: %d\n", answer);
}
void check1(){
   printf("inside check1() function: %d\n", answer);
}
int main(){
   scanf("%d", &answer);
   check();
   check1();
}
```

Ex-4:

```
#include <stdio.h>
int student[100];
int sz;
void check(){
   for(int i=0; i<sz; i++){
      printf("%d ", student[i]);
   }
}
int main(){
   scanf("%d", &sz);
   for(int i=0; i<sz; i++){
      scanf("%d", &student[i]);
   }
   check();
}
```

Ex-5:

```
#include <stdio.h>
char str[100];
int sz;
void check(){
   printf("%s", str);
}
int main(){
   scanf("%s", str); //&--> No need in string
   sz = strlen(str);
   check();
}
```

Ex-6:

```
#include <stdio.h>
#define sz 10
int name[sz];
int main(){
  for(int i=0; i<sz; i++)
  {
     printf("%d ", name[i]);
  }
  printf("\n");

  int fame[sz];
  for(int i=0; i<sz; i++)
  {
     printf("%d ", fame[i]);
  }
}
```