Functions

Topics

- ✓ What Is a C function?
- ✓ Advantages of Function.
- ✓ Working of a C Functions.
- ✓ Types of C function.
- ✓ Function declaration definition, call.
- ✓ Function Invocation
- ✓ Call by value
- ✓ Call by reference
- ✓ Passing Arrays to functions
- ✓ Command
- ✓ Line arguments
- ✓ Recursion
- ✓ Library Functions.

What is a function?

- A function is a block of code that together performs a specific task.

- A large program into the basic building blocks known as function. The function contains the set of programming statements enclosed by " {   } " .

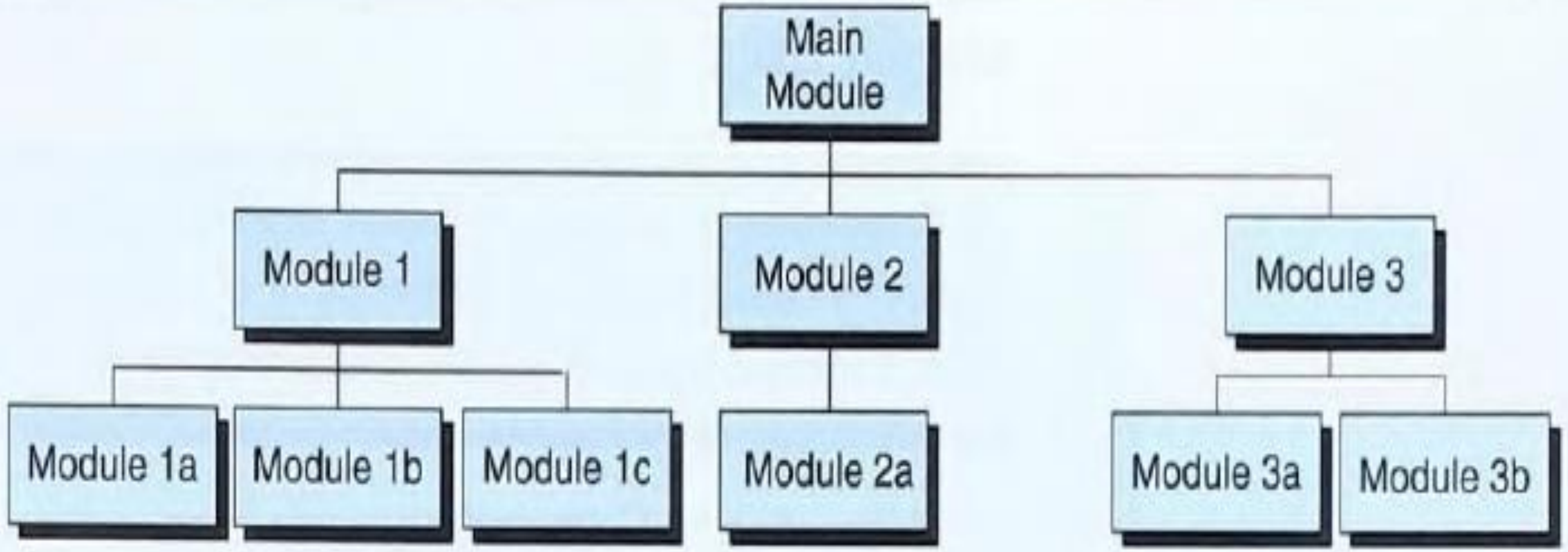
- The execution of the program always starts and ends with main, but it can call other functions to do special tasks.

Why functions?

1.When we write large complex programs, it becomes difficult to keep track of the source code. The job of functions is to divide the large program to many separate modules based on their functionality.

2.Reusability

3.Readability



Structure Chart

Advantages:

1.The function provides modularity.

2.C functions are used to avoid rewriting same logic/code again and again.

3.We can call a function any number of times.

4.A large C program can easily debug when it is divided into functions.

5.A large program can be divided into smaller modules. Hence, a large project can be divided among many programmers.

Working of a Function in Program:

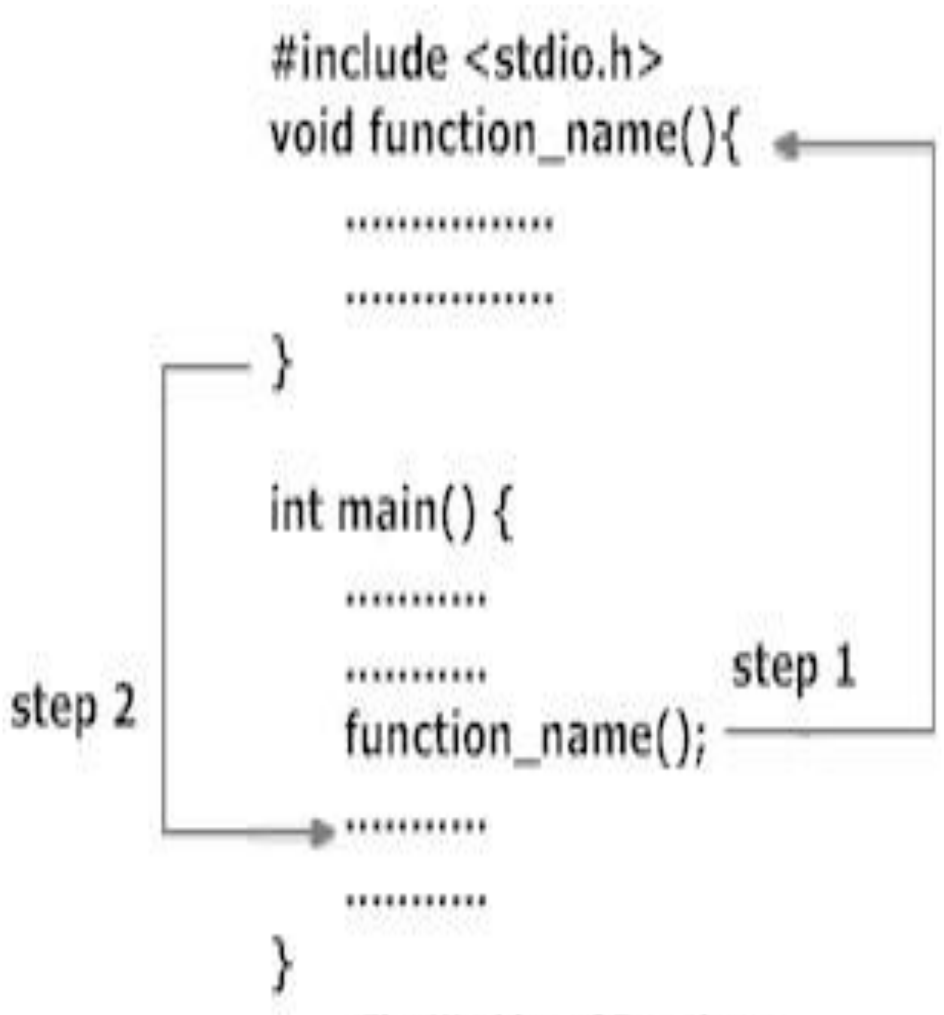
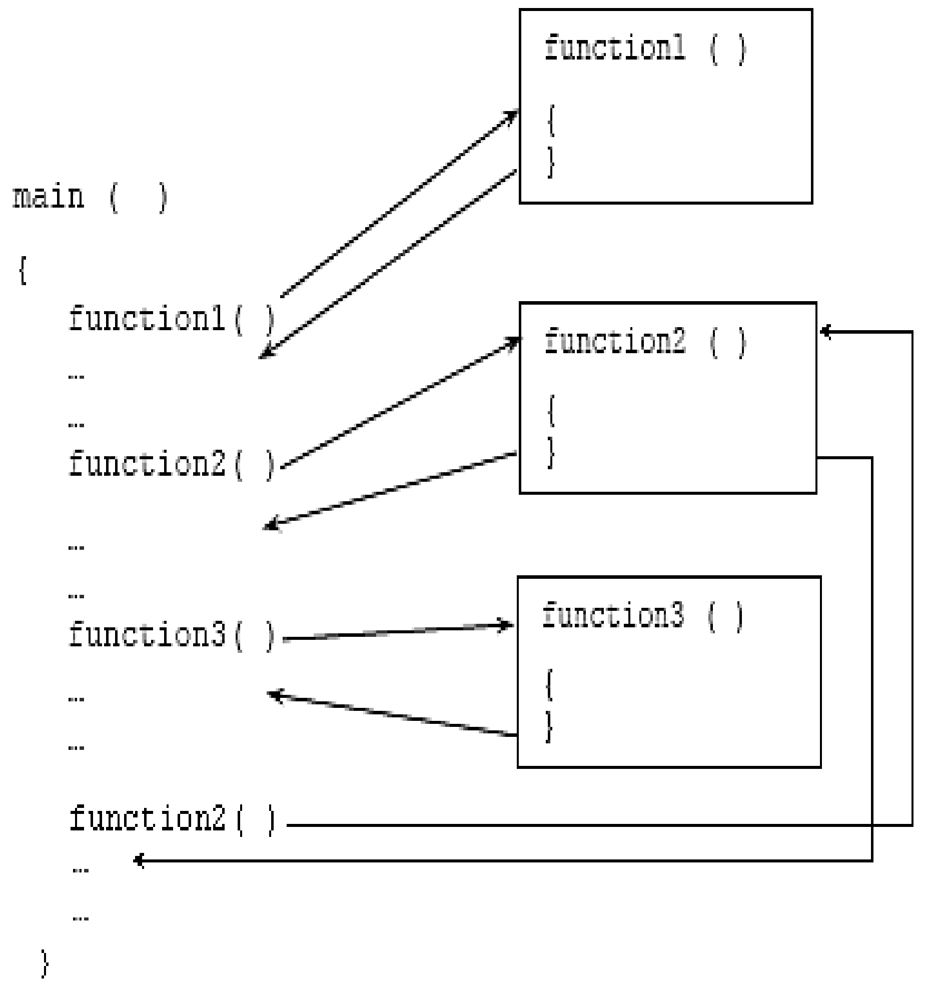

```
#include <stdio.h>
void function_name(){
      ...............
      ...............
}

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

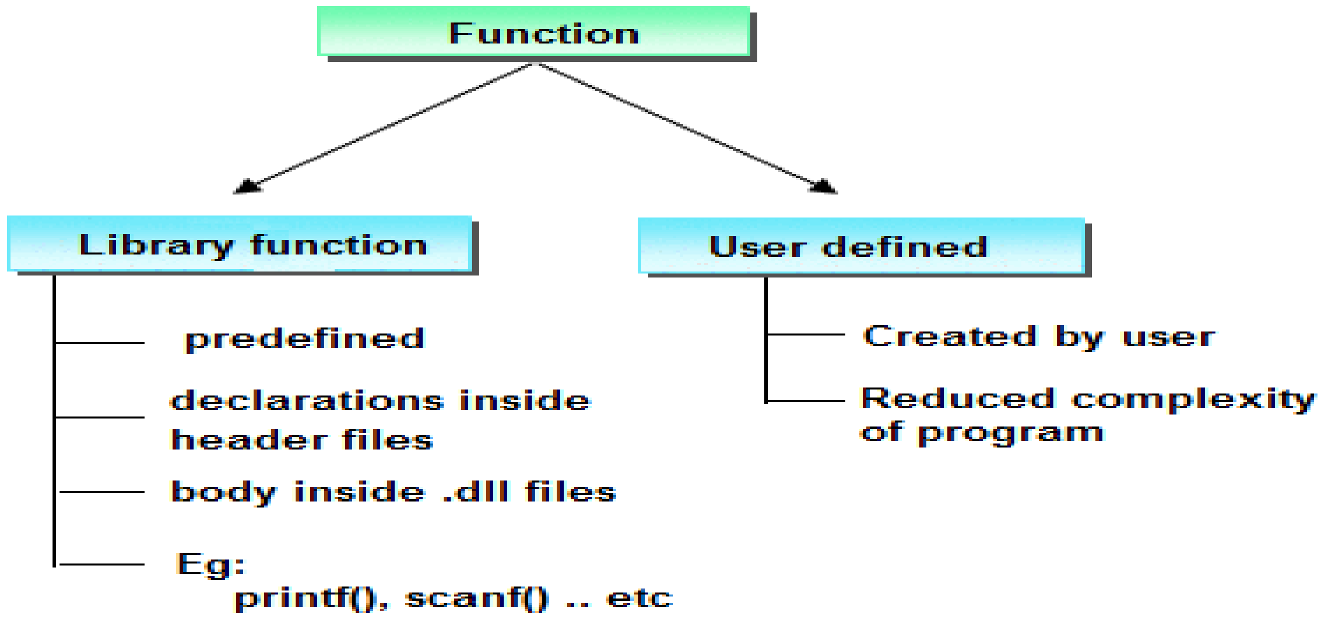
Fig: Working of Functions


```
main (  )
{
   function1( )
   …
   …
   function2( )
   …
   …
   function3( )
   …
   …
   function2( )
   …
   …
 }
```
```
function1 ( )
{
}
```
```
function2 ( )
{
}
```
```
function3 ( )
{
}
```

Types of Functions

‘C’ language supports two types of functions

1) Library function / Built in functions.
2) User defined function.


Function
Library function
User defined
predefined
declarations inside header files
body inside .dll files
Eg: printf(), scanf() .. etc
Created by user
Reduced complexity of program

1)Library functions:

- ✓ Library functions are pre-defined set of functions. Their task is limited.
- ✓ Library functions are not required to be written by us.
- ✓ A user cannot understand the internal working of these functions.
- ✓ Library functions in C language are inbuilt functions which are grouped together and placed in a common place called library.
- ✓ Each library function in C performs specific operation.
- ✓ We can make use of these library functions to get the pre-defined output instead of writing our own code to get those outputs.

- ✓These library functions are created by the persons who designed and created C compilers.

- ✓All C standard library functions are declared in many header files which are saved as file_name.h.

- ✓When we include header files in our C program using “#include<filename.h>” command, all C code of the header files are included in C program. Then, this C program is compiled by compiler and executed.

Some of the Predefined Functions:

1. stdio.h
2. conio.h
3. string.h
4. stdlib.h
5. math.h
6. time.h
7. ctype.h

stdio.h

Function	Description
printf()	This function is used to print the character, string, float, integer, octal and hexadecimal values onto the output screen
scanf()	This function is used to read a character, string, numeric data from keyboard.
getc()	It reads character from file
gets()	It reads line from keyboard
getchar()	It reads character from keyboard
puts()	It writes line to o/p screen
putchar()	It writes a character to screen

C conio.h library functions:

Functions	Description
clrscr()	This function is used to clear the output screen.
getch()	It reads character from keyboard
getche()	It reads character from keyboard and echoes to o/p screen
textcolor()	This function is used to change the text color
textbackground()	This function is used to change text background

string.h

String functions	Description
strcat ( )	Concatenates str2 at the end of str1
strncat ( )	Appends a portion of string to another
strcpy ( )	Copies str2 into str1
strncpy ( )	Copies given number of characters of one string to another
strlen ( )	Gives the length of str1
strcmp ( )	Returns 0 if str1 is same as str2. Returns <0 if strl < str2. Returns >0 if str1 > str2
strcmpi ( )	Same as strcmp() function. But, this function negotiates case. “A” and “a” are treated as same.
strlwr ( )	Converts string to lowercase
strupr ( )	Converts string to uppercase
strrev ( )	Reverses the given string
strset ( )	Sets all character in a string to given character
strnset ( )	It sets the portion of characters in a string to given character

C – stdlib.h library functions

Function	Description
malloc()	This function is used to allocate space in memory during the execution of the program.
calloc()	This function is also like malloc () function. But calloc () initializes the allocated memory to zero. But, malloc() doesn’t
realloc()	This function modifies the allocated memory size by malloc () and calloc () functions to new size
free()	This function frees the allocated memory by malloc (), calloc (), realloc () functions and returns the memory to the system.
abs()	This function returns the absolute value of an integer . The absolute value of a number is always positive. Only integer values are supported in C.
div()	This function performs division operation
abort()	It terminates the C program
exit()	This function terminates the program and does not return any value

C – math.h library functions

Function	Description
floor ( )	This function returns the nearest integer which is less than or equal to the argument passed to this function.
ceil ( )	This function returns nearest integer value which is greater than or equal to the argument passed to this function.
sin ( )	This function is used to calculate sine value.
cos ( )	This function is used to calculate cosine.
exp ( )	This function is used to calculate the exponential "e" to the xth power.
tan ( )	This function is used to calculate tangent.
log ( )	This function is used to calculates natural logarithm.
log10 ( )	This function is used to calculates base 10 logarithm.
sqrt ( )	This function is used to find square root of the argument passed to this function.
pow ( )	This is used to find the power of the given number.

C – time.h library functions

Functions	Description
setdate()	This function used to modify the system date
getdate()	This function is used to get the CPU time
clock()	This function is used to get current system time
time()	This function is used to get current system time as structure
difftime()	This function is used to get the difference between two given times
localtime()	This function shares the tm structure that contains date and time information

C – ctype.h library functions

Functions	Description
isalpha()	checks whether character is alphabetic
isdigit()	checks whether character is digit
isalnum()	Checks whether character is alphanumeric
islower()	Checks whether character is lower case
isupper()	Checks whether character is upper case
tolower()	Checks whether character is alphabetic & converts to lower case
toupper()	Checks whether character is alphabetic & converts to upper case

2. User- Defined Functions

- The function which is developed by the user according to their requirements is called user defined function.
- Depends on the complexity and requirement of the program, we can create as many user – defined functions we want.

User defined function Syntax/ Function prototype:

returntype function_name(list of arguments);

- The function prototype which provides the following information to the compiler:
  1. Return type of the function
  2. Name of the function
  3. List of arguments of respective type are passed to the function

1. function declaration

```
return_type function_name ( argument  list );
```

2. function definition

```
return_type function_name ( arguments list )
{
        Body of function;
}
```

3. function call

```
function_name ( arguments list );
```

Function declaration or prototype:

- In C, all functions must be declared before they are used. This is normally accomplished using a function prototype.

- The function prototype tells the compiler about the function name, type and number of arguments to be passed and return type.

- The compiler will also catch differences between the number of arguments used to call a function and the number of parameters in the function definition.

Syntax:

return_type function_name (type parm_namel, type parm_name2, . . . , type parm_nameN);

- The use of parameter names is optional.

Function Parameters:

Are ***two types:***

- ✓ Actual parameter – This is the argument which is used in function call.
- ✓ Formal parameter – This is the argument which is used in function definition.

Function definition:

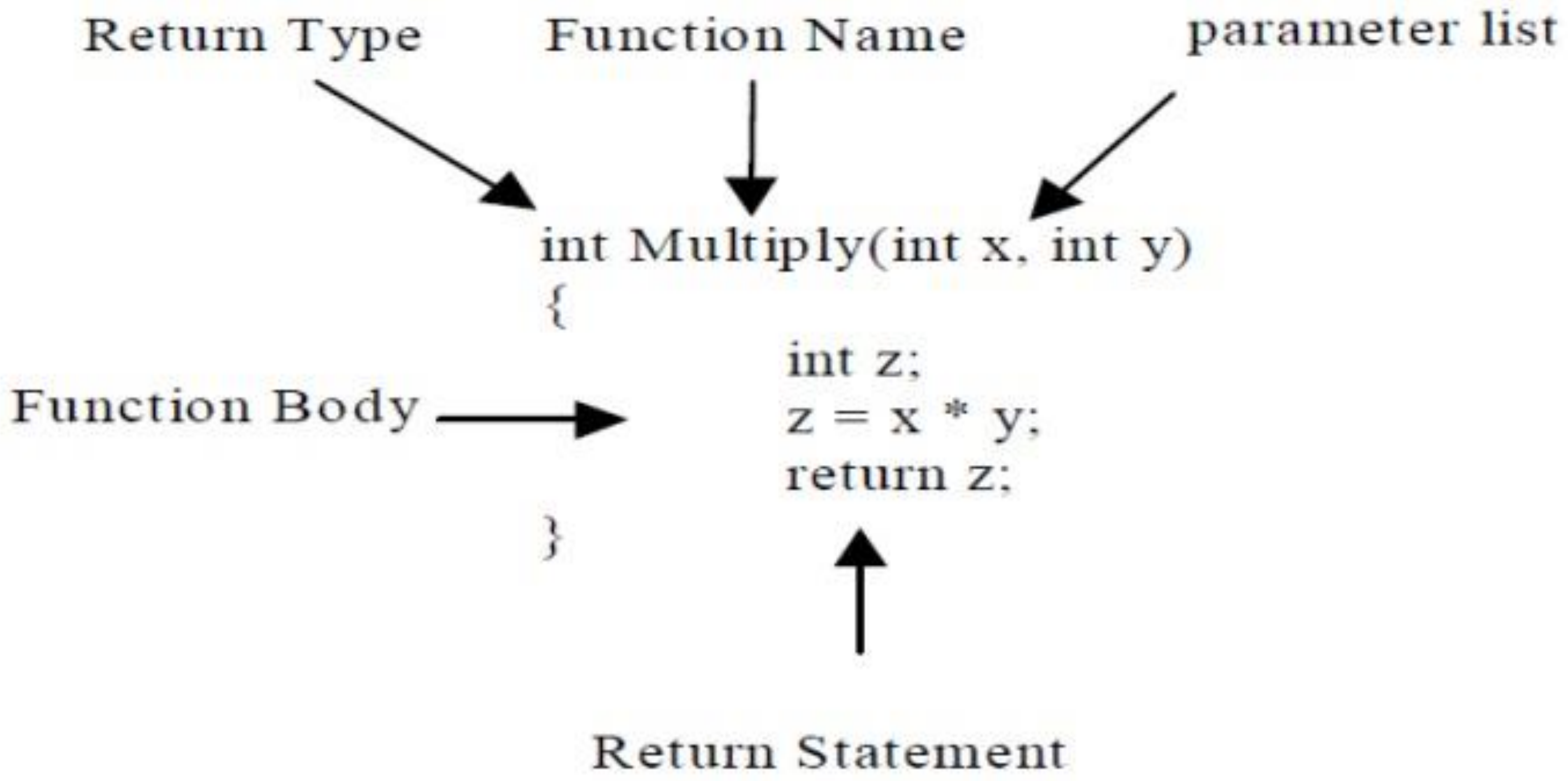
When a function is defined, space is allocated for that function in the memory.

A function definition comprises two parts:

1. Function header
2. Function body.

Syntax is:

```
return-type function-name (type varname1, type varname2, . . . ,
type varnameN)
{
    body of the function;
    return statement;
}
```



```
int Multiply(int x, int y)
{
        int z;
        z = x * y;
        return z;
}
```

Function call statement:

- The function call statement invokes the function, which means the program control passes to that of the function.
- Once the function completes its task, the program control is passed back to the calling environment.

Syntax:

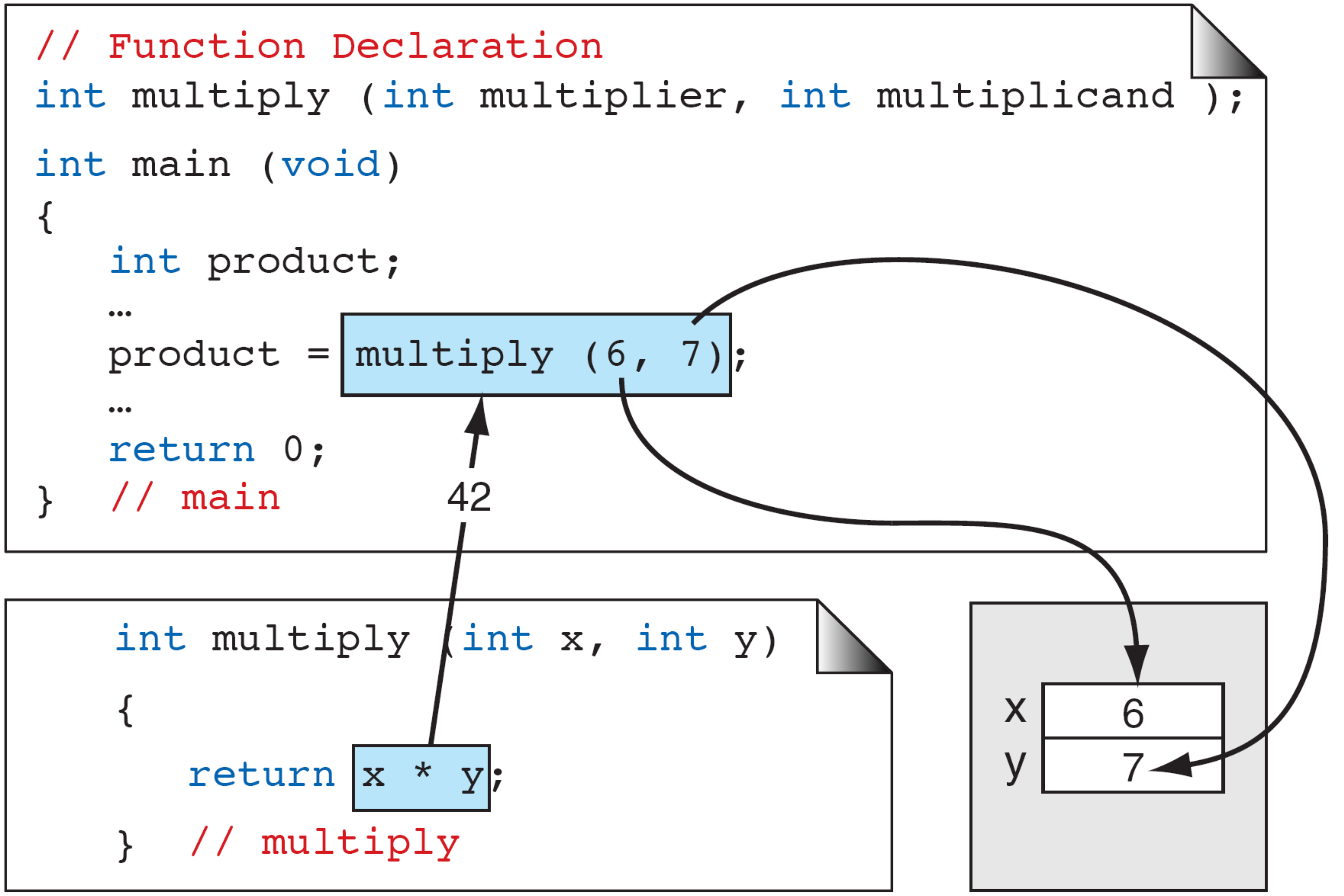
```
function_name ( arguments list );
```

```
#include <stdio.h>
//function declaration
int max(int x, int y);
void main(void)
{
    int a = 10, b = 20,m;
//Function call
    m = max(a, b);
    printf("maximum value is %d", m);

}
```

```
// function definition
int max(int x, int y)
{
    if (x > y)
      return x;
    else
      return y;
}
```

Output:

```
Maximum value is 20
```



```
// Function Declaration
int multiply (int multiplier, int multiplicand );

int main (void)
{
   int product;
   …
   product = multiply (6, 7);
   …
   return 0;
}  // main
```

```
   int multiply (int x, int y)
   {
      return x * y;
   }  // multiply
```

Function Definition

Function invocation

- Function invocation is also called as function call.
- To execute a function we must call it.
- A function can be called or invoked by using function name followed by list of arguments (values) that function definition will receive to perform task.

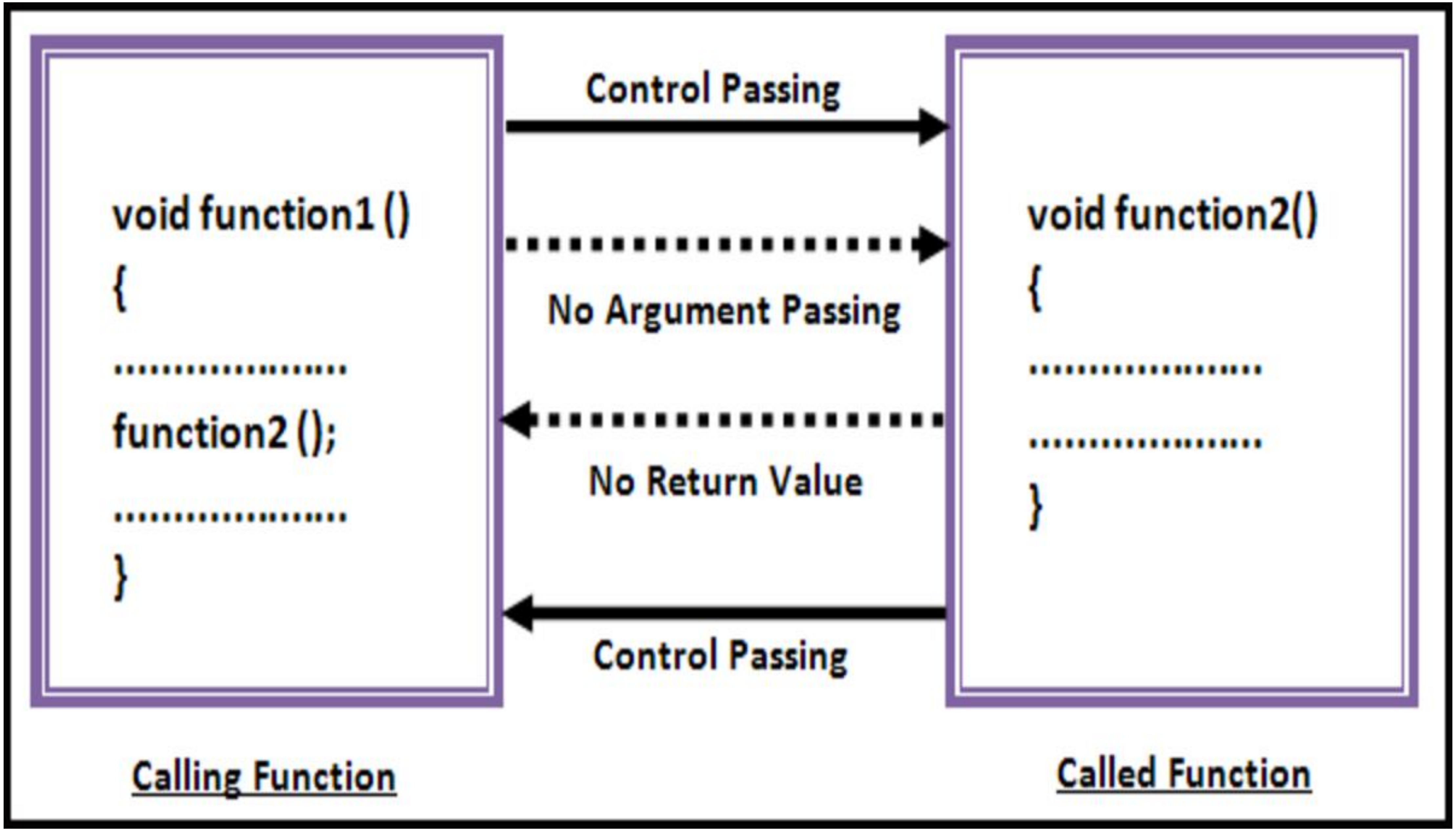
User defined Function can invoke in different types

There are four basic types:

1. Function without arguments and without return value.
2. Functions with arguments and without return value.
3. Functions without arguments and with return value.
4. Functions with argument and with return value.

1. Function without arguments and without return value.

- A C function without any arguments means you cannot pass data (values like int, char etc) to the called function. Similarly, function with no return type does not pass back data to the **calling function**.
- Void functions without parameters does not receive any parameters and also does not return any value .
- This can be used only as a statement because a Void function does not return a value



```
void function1 ()
{
...................
function2 ();
...................
}
```

```
void function2()
{
...................
...................
}
```

Example 1:

```
#include<stdio.h>
void add()
{
        int x,y;
        int result;
        scanf(“%d %d”,&x,&y);
        result = x+y;
        printf ("Sum of %d and %d is %d. \n\n", x, y, result);
}
void main()
{
        add();
}
```

Output:

```
6
7
Sum of 6 and 7 is.
13
```

Example 2

```
#include<stdio.h>
void fact();
void main()
{
      fact();
}
```

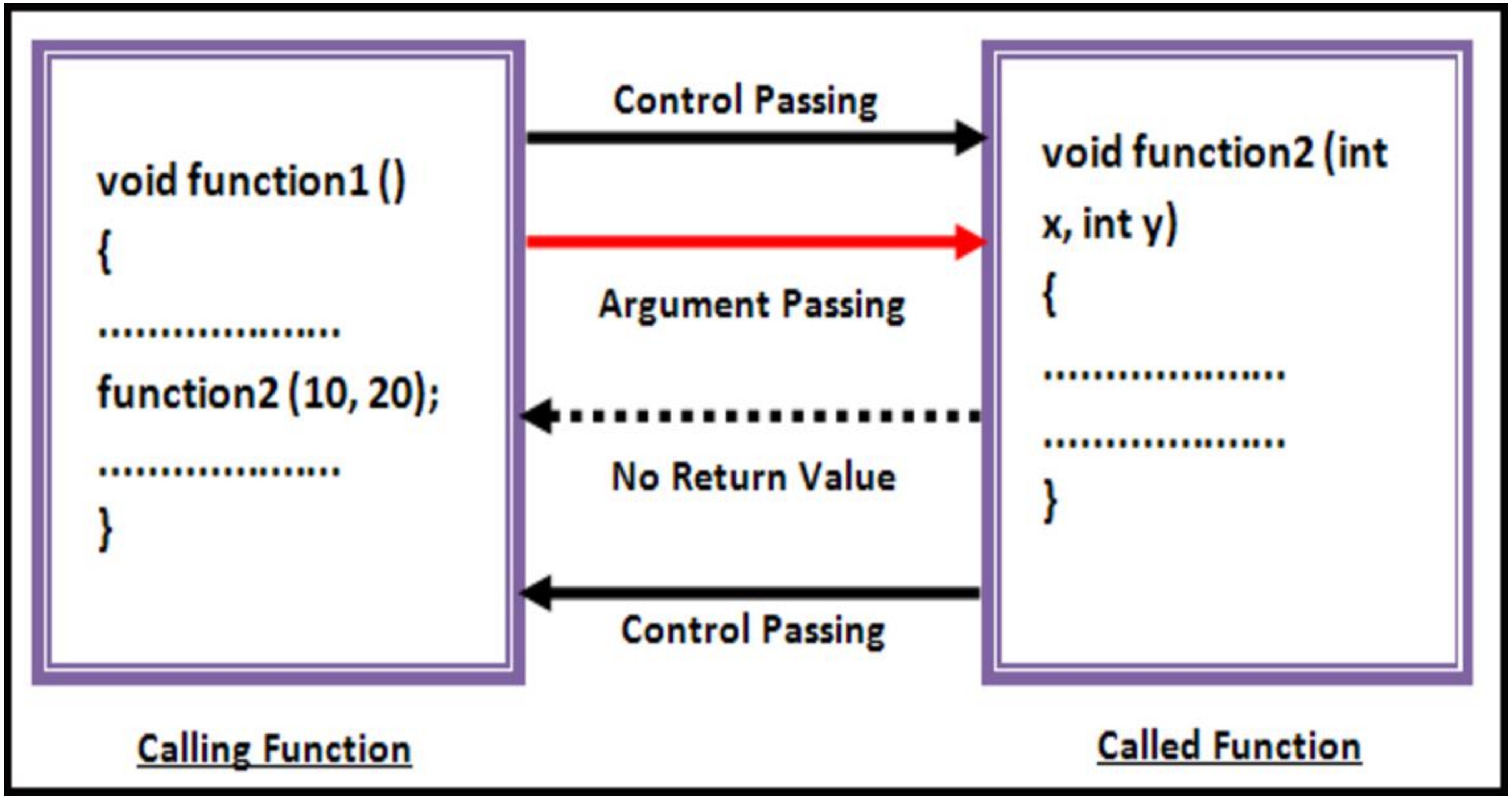
```
void fact()
{
    int i,fact=1,n;
    printf("Enter a Number : ");
    scanf("%d",&n);
    for(i=1; i<=n; i++)
        fact=fact*i;
    printf("\n Factorial of %d is:%d”,n,fact);
}
```

Output:

```
Enter a Number : 6
Factorial of 6 is: 720
```

2. Functions with arguments and without return value

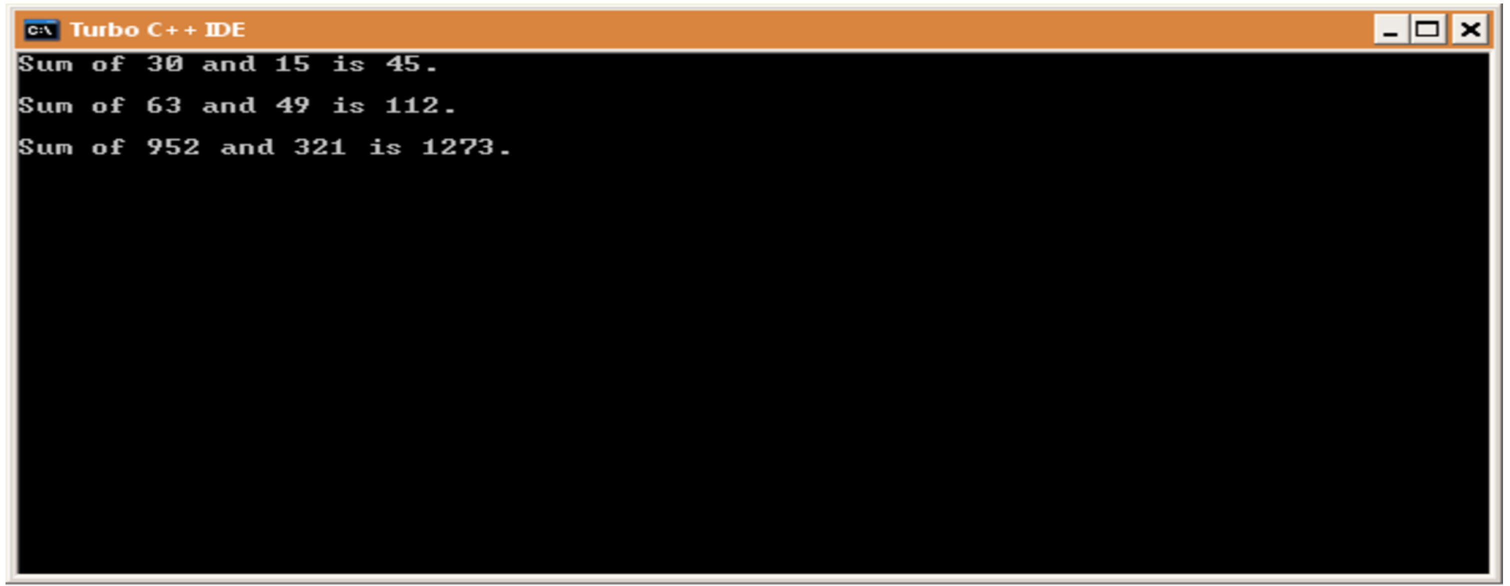
- The function that receives parameter but does not return any value.
- This function takes arguments(parameter list).
- This type of function can accept data from calling function.
- In other words, we can send data to the called function from calling function but we cannot send result data back to the calling function. Rather, it displays the result on the terminal.


void function1 ()
{
....
function2 (10, 20);
....
}
Control Passing
Argument Passing
No Return Value
Control Passing
void function2 (int x, int y)
{
....
....
}
Calling Function
Called Function

Example 1:

```
#include<stdio.h>
void add(int x, int y)
{
        int result;
        result = x+y;
        printf ("Sum of %d and %d is
        %d.\n\n", x, y, result);
}
```

```
void main()
{
    add(30,15);
    add(63,49);
    add(952,321);
}
```



```
Sum of 30 and 15 is 45.
Sum of 63 and 49 is 112.
Sum of 952 and 321 is 1273.
```

Example 2:

```
#include<stdio.h>
void fact(int);
void main()
{
   int num;
   printf("Enter Number: ");
   scanf(“%d”,&num);
   fact(num);
}
```

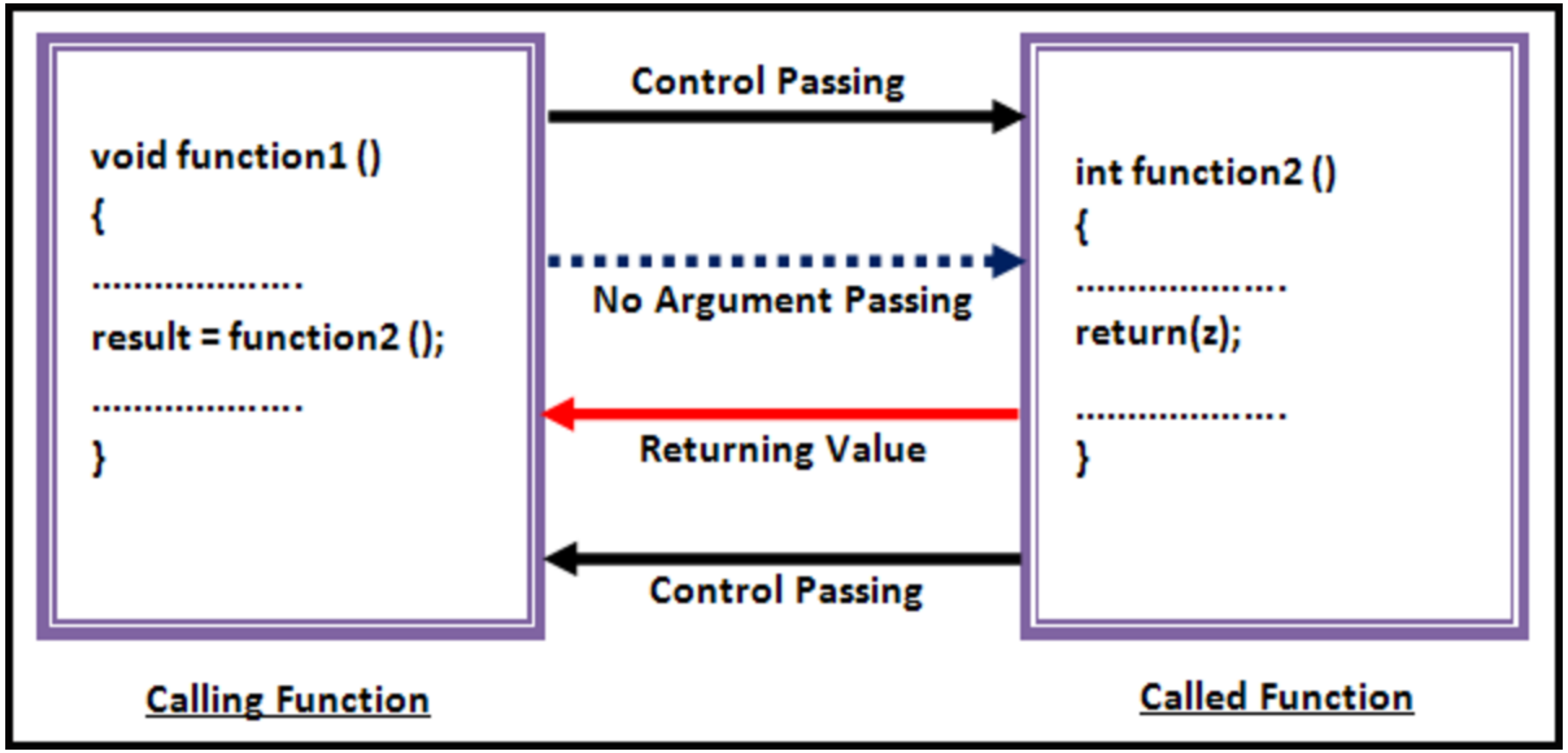
```
void fact(int n)
{
    int i,fact=1;
    for(i=1; i<=n; i++)
          fact=fact*i;
    printf("\nFactorial of %d is:%d",n, fact);
 }
```

Output:

```
Enter Number : 6
Factorial of 6 is: 720
```

3. Functions without arguments and with return value.

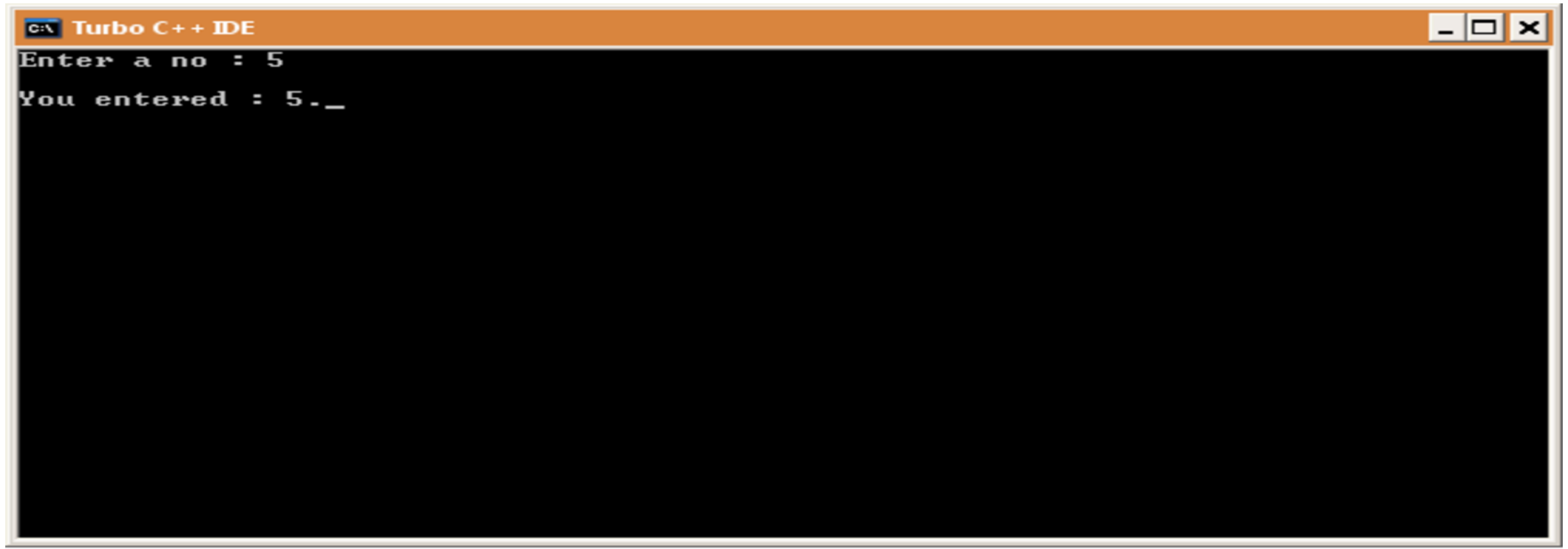
- The function returns a value but does not receive parameters



Example 1:

```
#include<stdio.h>
int input1()
{
int no1;
printf("Enter   a   no   :   ");
scanf("%d",&no1);
return(no1);
}
```

```
void main()
{
    int z;
    z = input1();
    printf("\nYou entered : %d.“,  z);
}
```



```
Enter a no : 5
You entered : 5._
```

Example 1:

```
#include<stdio.h>
int fact(void);
void main()
{
     int f;
     f=fact();
    printf("\nFactorial of %d is: %d ",fact);
}
```

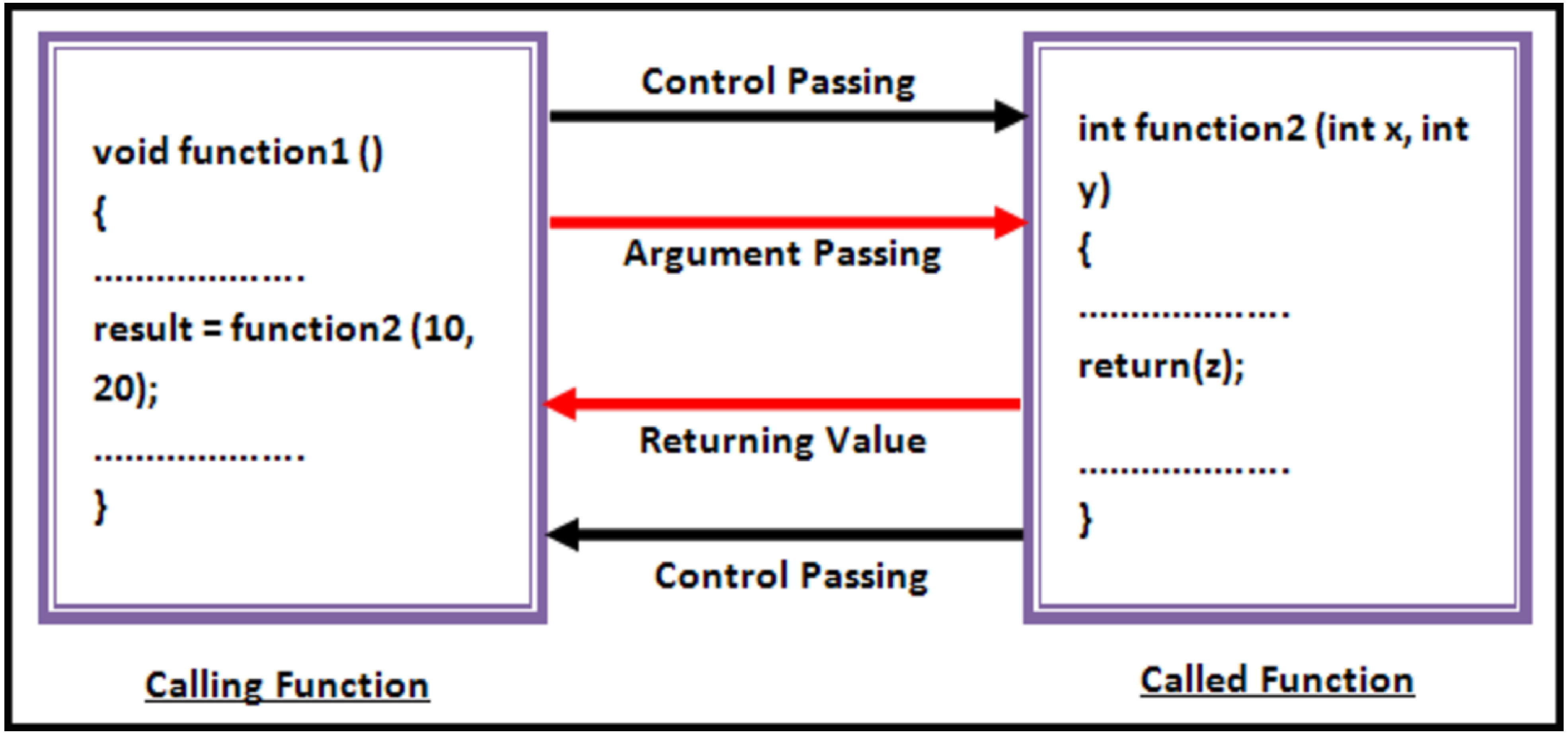
```
int fact()
{
   int i,fact=1,n;
   printf("Enter Number:");
   scanf("%d",&n);
   for(i=1; i<=n; i++)
        fact=fact*i;
     return fact;
}
```

Output:

```
Enter Number : 6
Factorial of 6 is: 720
```

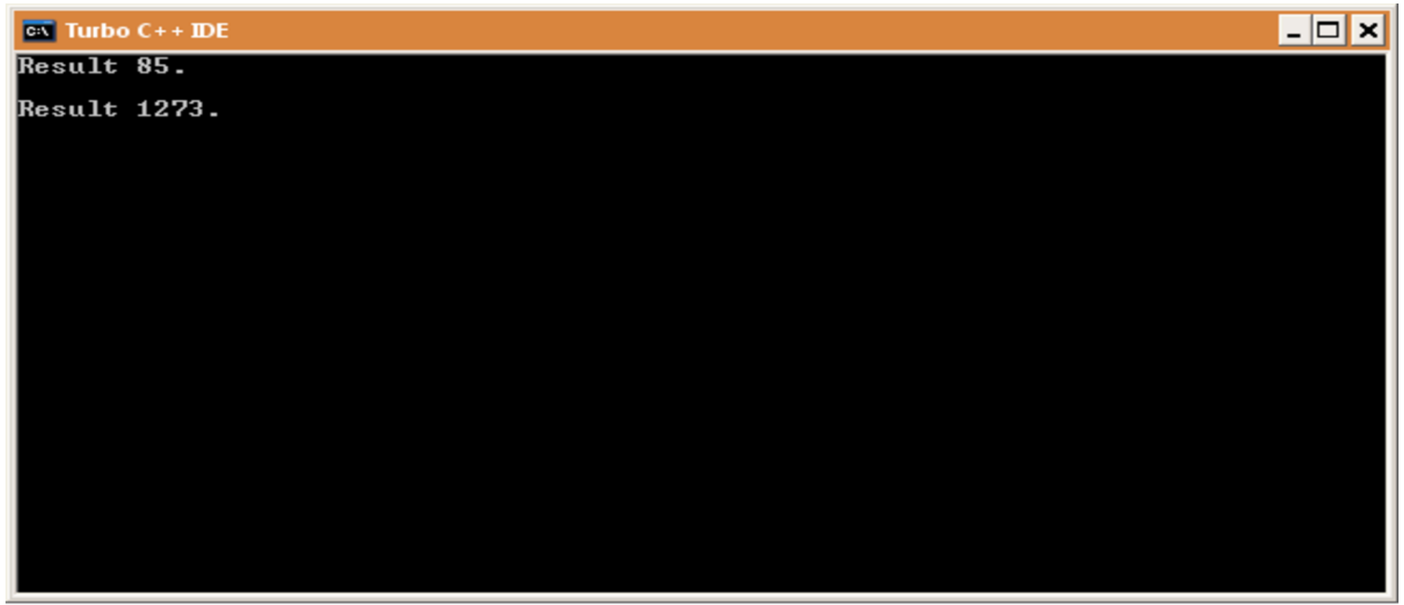
4. Functions with argument and with return value.

- The function receives parameters and return values



```
#include<stdio.h>
int add(int x, int y)
{
        int result;
        result = x+y;
        return(result);
}
```

```
void main()
{
    int z;
    z = add(952,321);
    printf("Result %d.\n\n", add(30,55));
    printf("Result %d.\n\n",z);
}
```



```
Result 85.

Result 1273.
```

Example 2:

```
#include<stdio.h>
int fact(int);
void main()
{
        int i,f,n;
        printf("Enter Number: ");
        scanf("%d",&n);
        f=fact(n);
        printf("\nFactorial of  %d is: %d ",f);
}
```

```
int fact(int n)
{
    int i,f=1;
    for(i=1; i<=n; i++)
          f=f*i;
      return f;
}
```

```
Output:
Enter Number : 6
Factorial of 6 is: 720
```

C Language uses two mechanisms for function calling/ invocation:

- ➢ Pass by values / call by value
- ➢ Pass by reference / call by reference

1. CALL BY VALUE:

- In call by value method, the value of the variable is passed to the function as parameter.
- The value of the actual parameter can not be modified by formal parameter.
- Value of actual parameter is copied to formal parameter.

```
#include<stdio.h>
void swap(int a, int b);
 int main()
{
    int m = 22, n = 44;
    printf(" values before swap in main function m
= %d \n and n =  %d", m, n);
    swap(m, n);
    printf(" values after swap in main function  m =
%d \n and n =   %d", m, n);
}
```

```
void swap(int a, int b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
   printf ("After swapping values in
function a = %d, b = %d\n", a, b);
}
```

Output:

```
values before swap in main function
 m = 22 and n = 44

After swapping values in
Function a=44, b=22

values after swap in main function
 m = 22 and n = 44
```

```
#include <stdio.h>
int sum(int a, int b)
{
        int c=a+b;
        return c;
}
int main( )
{
        int var1 =10;
        int var2 = 20;
        int var3 = sum(var1, var2);
        printf("%d", var3);
        return 0;
}
```

Output:

```
30
```

CALL BY REFERENCE

- In call by reference method, the address of the variable is passed to the function as parameter.
- The value of the actual parameter can be modified by formal parameter.
- Same memory is used for both actual and formal parameters since only address is used by both parameters.

```
#include<stdio.h>
void swap(int *a, int *b);
void main()
{
    int m = 22, n = 44;
    printf("values before swap m = %d \n and n = %d",m,n);
    swap(&m, &n);
     printf("\n values after swap a = %d \nand b = %d", *a, *b);
}

 void swap(int *a, int *b)
 {
     int temp;
     temp = *a;
     *a = *b;
     *b = temp;
  }
```

Output:

```
values before swap m = 22 and n = 44
values after swap a = 44 and b = 22
```

Passing an array to function:

- ➢ Whenever we need to pass a list of elements as argument to any function in C language, it is preferred to do so using an array.
- ➢ But how can we pass an array as argument to a function

- Ex:- #include<stdio.h>
- float findAverage(int marks[]);

Pass Individual Array Elements Method 1:

- **Example 1:**

```
#include <stdio.h>
void display(int age1, int age2)
{
        printf("%d\n", age1);
        printf("%d\n", age2);
 }
 void main()
 {
        int ageArray[] = {2, 8, 4, 12};
        // pass second and third elements to display() ;
        display(ageArray[1], ageArray[2]);
}
```

Output

```
8
4
```

Example 2: Pass Array as an argument to a Function

```
#include <stdio.h>
float calculateSum(float num[],int size);
 void main()
 {
        float result, num[] = {23.4, 55, 22.6, 3, 40.5, 18};
        int n=6;
        result = calculateSum(num,n);
        printf("Result = %f", result);
}
 float calculateSum(float num[],int size)
{
         float sum = 0.0;
         for (int i = 0; i < size; ++i)
         {
                sum += num[i];
         }
         return sum;
}
```

Output:

```
Result :162.50
```

Pass Multidimensional Arrays to a Function

Example 3: Pass two-dimensional arrays

```
#include <stdio.h>
void displayNumbers(int num[2][2]);
void main()
{
        int num[2][2];
        printf("Enter 4 numbers:\n");
        for (int i = 0; i < 2; ++i)
                for (int j = 0; j < 2; ++j)
                        scanf("%d", &num[i][j]);
        displayNumbers(num);
}
```

```
void displayNumbers(int num[2][2])
{
        printf("Displaying:\n");
        for (int i = 0; i < 2; ++i)
                for (int j = 0; j < 2; ++j)
                        printf("%d\n", num[i][j]);
}
```

```
Output:
Enter 4 numbers:
 2 3 4 5
 Displaying: 2 3 4 5
```

Command Line Arguments
&
Recursion

Recursion

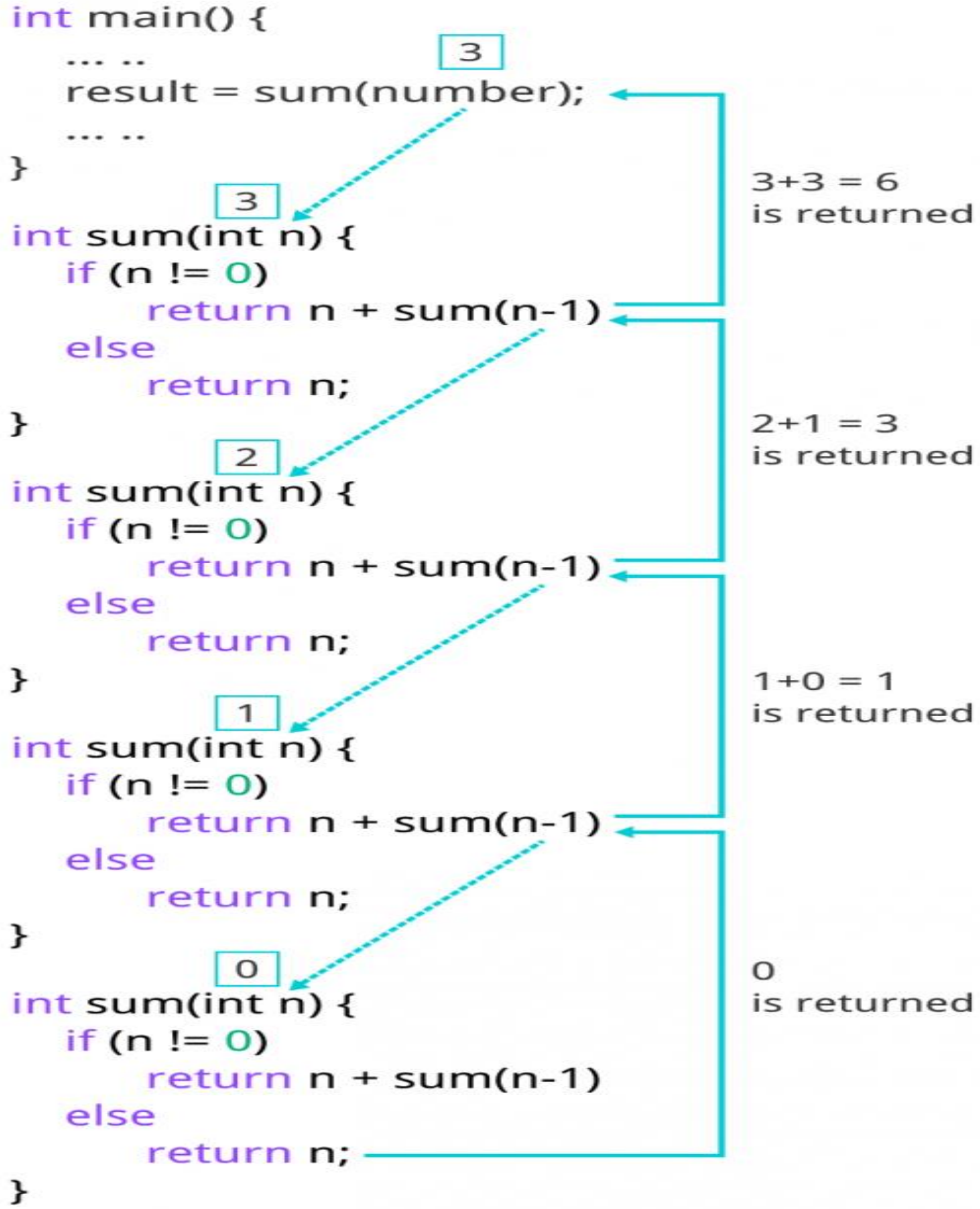
- ➢In C, a function can call itself. In this case, the function is said to be recursive.
- ➢A function that calls itself repeatedly until specified condition is met is known as recursive function and the process of calling function itself is known as recursion.

Syntax for Recursive function:

```
returntype fun_name()
{
   fun_name();     /* function calls itself */
}
int main()
 {
           fun_name();
}
```

Example 1: Sum of Natural numbers using Recursion

```
#include <stdio.h>
int sum(int n);
int main()
{
    int number, result;
    printf("Enter a positive integer: ");
    scanf("%d", &number);
    result = sum(number);
    printf("sum = %d", result);
    return 0;
}
int sum(int n)
{
    if (n != 0) // sum() function calls itself

        return n + sum(n-1);

                else

        return n;

 }
```



```
int main() {
    ... ..
    result = sum(number);
    ... ..
}

int sum(int n) {
    if (n != 0)
        return n + sum(n-1)
    else
        return n;
}

int sum(int n) {
    if (n != 0)
        return n + sum(n-1)
    else
        return n;
}

int sum(int n) {
    if (n != 0)
        return n + sum(n-1)
    else
        return n;
}

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

Example: Factorial of a given number

Iterative Definition

•Factorial(n)= $\left[\begin{array}{ll} 1 & \text{if } n=0 \\ n*(n-1)*(n-2)\ldots3*2*1 & \text{if } n>0 \end{array}\right]$

Recursive Definition

Base Case

Factorial(n) = $\left[\begin{array}{ll} 1 & \text{if } n=0 \\ n*\text{Fcatorial}(n-1) & \text{if } n>0 \end{array}\right]$

General Case

Example

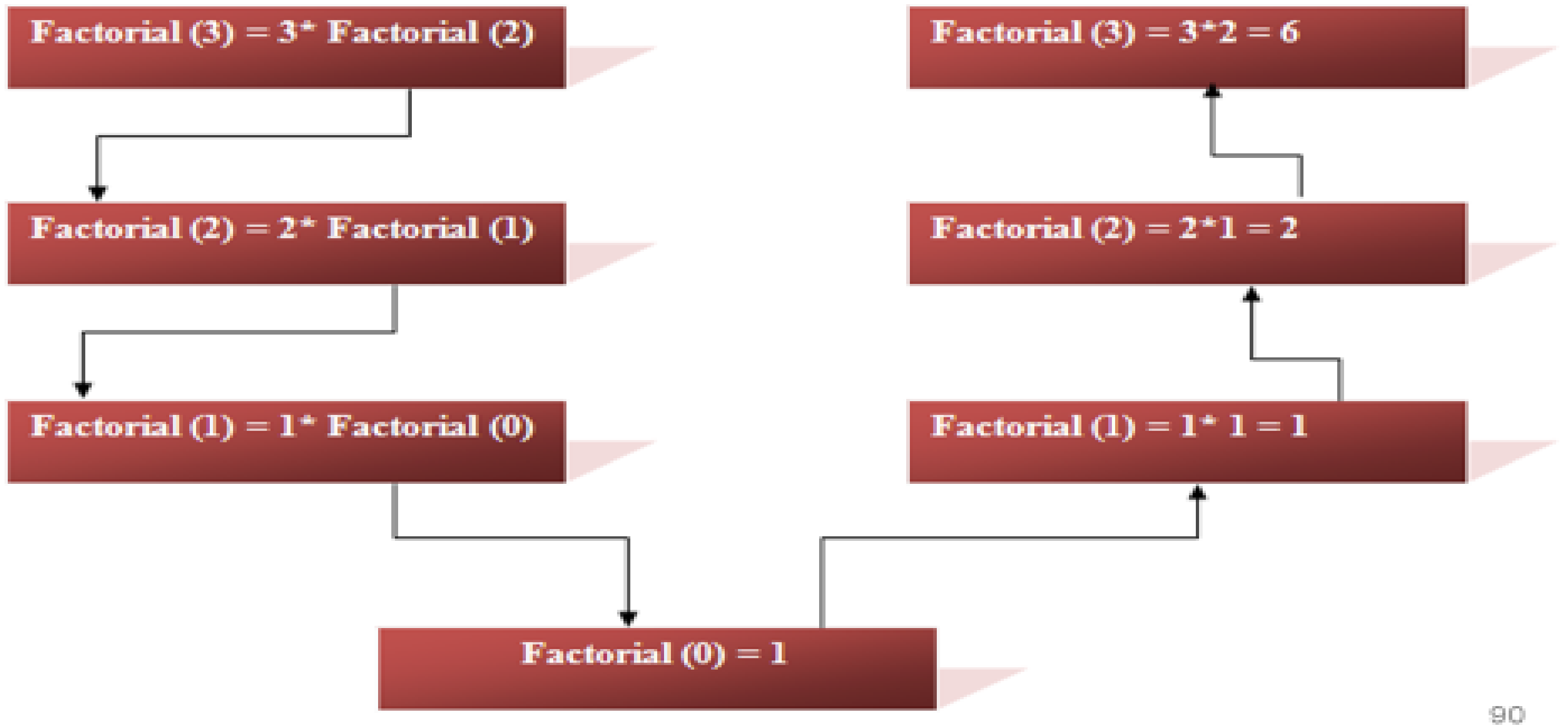
```
void main()
{
        int number;
        long fact;
        printf("Enter a number: ");
        scanf("%d", &number);
        fact = factorial(number);
        printf("Factorial of %d is %ld\n", number, fact);

 }
```

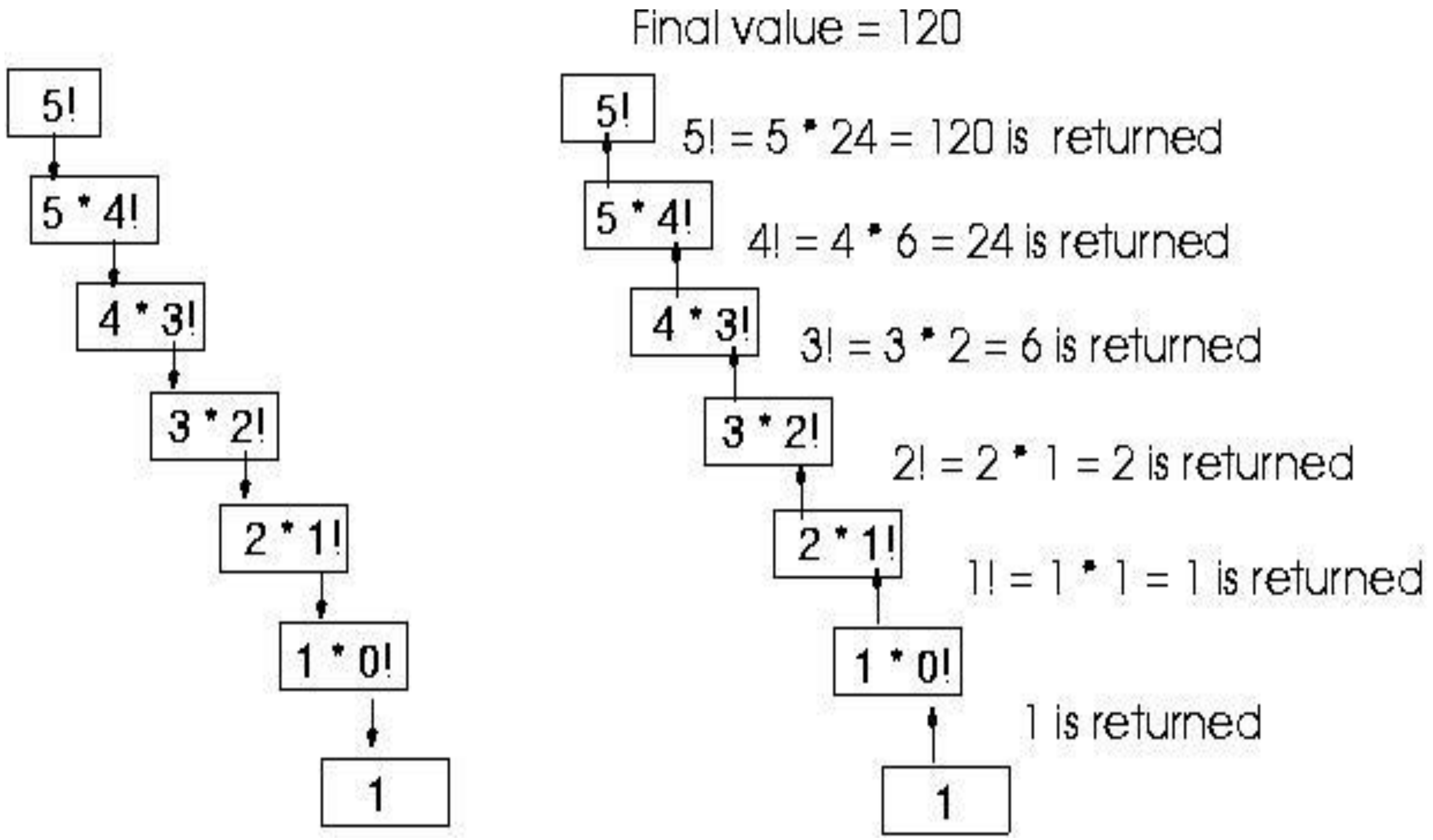
```
long factorial(int n)
{
 if (n == 0)
   return 1;
 else
   return(n * factorial(n-1));
}
```

Output:

```
Enter a number : 5
Factorial of 5 is : 120
```

Recursive representation of Factorial(3) (1 of 2)


Factorial (3) = 3* Factorial (2)
Factorial (2) = 2* Factorial (1)
Factorial (1) = 1* Factorial (0)
Factorial (0) = 1
Factorial (1) = 1* 1 = 1
Factorial (2) = 2*1 = 2
Factorial (3) = 3*2 = 6


5!
5 * 4!
4 * 3!
3 * 2!
2 * 1!
1 * 0!
1
Final value = 120
5!
5! = 5 * 24 = 120 is returned
5 * 4!
4! = 4 * 6 = 24 is returned
4 * 3!
3! = 3 * 2 = 6 is returned
3 * 2!
2! = 2 * 1 = 2 is returned
2 * 1!
1! = 1 * 1 = 1 is returned
1 * 0!
1 is returned
1

Command Line Argument Definition

- Command Line Argument is a parameter supplied to a program when the program is invoked.
- In C it is possible to accept command line arguments.
- Command-line arguments are given after the name of a program in command-line operating systems like DOS or Linux, and are passed in to the program from the operating system.

Command Line Arguments

To use command line arguments in our C program, we must first understand the full declaration of the main function

The prototype for main() looks like:

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

Command Line Arguments Cont…

- argc – no.of parameters
- argv[ ] – parameters list
- each parameter separated by a space
- parameter values are passed to program at the time of execution

Conventional rules

- Arguments are always passed to main()
- There must be two
  - first is an integer
  - second char pointer to an array
- First argument (argv[0]) will always be the name of the calling program.
- argc will always be at least 1

Conventional rules

- The first argument is always argv[0]
- The last argument is always argv[argc-1]
- argv[argc] will always be a null pointer
- Arguments are always passed as character strings. Numbers must be converted from characters to integers, floats, doubles, etc.

```
#include <stdio.h>
void main(int argc, char *argv[] )
{

   printf("Program name is: %s\n", argv[0]);

   if(argc < 2)
      printf("No argument passed through command line.\n");
   else
      printf("First argument is: %s\n", argv[1]);

}
```

```c
#include <stdio.h>
#include<stdlib.h>
void main(int argc, char *argv[])
{
        int a,b,sum;
        if(argc!=3)
        {
                printf("please use \"prg_name value1 value2 \"\n");
                return -1;
        }
        a = atoi(argv[1]);
        b = atoi(argv[2]);
        sum = a+b;
        printf("Sum of %d, %d is: %d\n",a,b,sum);
}
```

Sample Program

```
#include <stdio.h>
void main(int argc, char *argv[])
{
   for (int i=0; i<argc; i++)
         printf( “Argument number #%d --> %s \n ”, i, argv[i]);

}
```

Sample Output:

```
C:\> hello Each word should be a unique
argument
Argument number #0 -->hello
Argument number #1 -->Each
Argument number #2 -->word
Argument number #3 -->should
Argument number #4 -->be
Argument number #5 -->a
Argument number #6 -->unique
Argument number #7 -->argument
```

- Write a program in C to Print Fibonacci Series using recursion.
- Write a program in C to find the sum of digits of a number using recursion.
- Write a program in C to get the largest element of an array using recursion.

Thank you