Department Of Computer Science
B.Sc-I
Semester II
Subject- Programming Skills Using 'C'

Unit-I

Pointers & functions

(A) Pointers

One of the most important and powerful features in C language is pointer.

"A pointer is a variable, it may contain the memory address of the another variable." pointer can have any name that is legal for other variable. It is declared in the same manner like other variables. It is always denoted by * operator. A pointer is a variable whose value is, also an address. Each variable has two attributes; address and value. A variable can take any value specified by its data type.

- **Features of pointers:**

a) Pointers are efficient in handling data and associated with array.

b) Pointers are used for saving memory space.

c) Pointers reduce length and complexity of the program.

d) Pointer helps to make letter use of the available memory.

e) Since the pointer data manipulation is done with address, the execution time is faster.

- **Pointer declaration:**

Syntax : data-type * pointer name ;

Example:

int *a

char *b

float *c

int *a - means 'a' contains the address of variable, which is integer variable.

char *b - means 'b' contains the address of variable, which is character variable.

float *c - means 'c'contain the address of variable which is float variable.

- **Accessing variable through pointers:**

Example:

int *p

x = 15 ;

p = & x

Variable	Value	Address
X	15	4001
P	4001	4005

Program to access through variable pointer:

```
# include < stdio.h>
 main ( )
{
int a = 22, *a ;
float b = 2.25 , *b ;
 a = & a ;
 b = & b ;
printf ( " \ n value of a = % d", * a) ;
printf ( " \ n value of b = % d", * b) ;
}
```

Output:

Value of a = 22

Value of b = 2.25

- **Null pointer:**

A pointer is said to be a null pointer when its right value is 0. A null pointer can never point to a valid data. For checking a pointer, it it is assigned to 0, then it is a null pointer and is not valid.

Example:

```
int *a ;

int *b ;

b = a = 0
```

Hence b and a become null pointers after the integer value of 0 is assigned to them.

- **Initializing pointer variable:**

The process of assigning the address of a variable to a pointer variable is known as initialization. The location of the variable in memory depends on system memory. This can be achieved in 'c'through an ampersand (&) sign. The ampersand is an address operator, which is used to access the address of a variable and assign it to a pointer to initialize it. The programs with uninitialized pointers will produce errorneous results. It is therefore important to initialize pointer variables carefully before they are used in the program. We can use the assignment operator to initialize the pointer variable.

Example:

```
p = & n
```

where 'p' contains the address of variable 'n'.

n **p**

- **Introduction To Arrays:**

In C programming, one of the frequently problem is to handle similar types of data. For example: if the user wants to store marks of 500 students, this can be done by creating 500 variables individually but, this is rather tedious and impracticable. These types of problem can be handled in C programming using arrays.

An array in C Programing can be defined as number of memory locations, each of which can store the same data type and which can be references through the same variable name. It is a collective name given to a group of similar quantities. These similar quantities could be marks of 500 students, number of chairs in university, salaries of 300 employees or ages of 250 students. Thus we

can say array is a sequence of data item of homogeneous values (same type). These values could be all integers, floats or characters etc.

We have two types of arrays:

1. One-dimensional arrays.
2. Multidimensional arrays.

1.One Dimensional Arrays:

A **one-dimensional array** is a structured collection of components (often called *array elements*) that can be accessed individually by specifying the position of a component with a single *index* value. Arrays must be declared before they can be used in the program.

Here is the declaration syntax of one dimensional array:

data_type array_name[array_size];

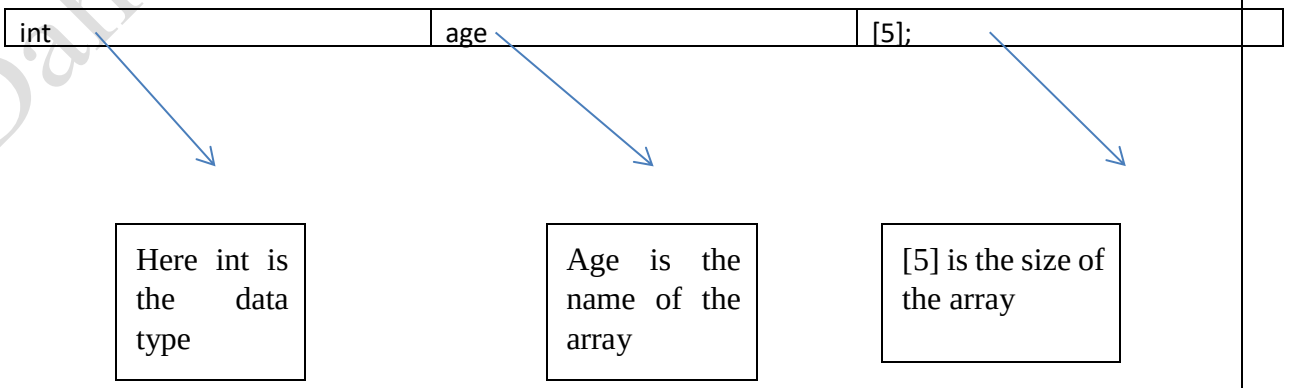
Here "data_type" is the type of the array we want to define

"array_name" is the name given to the array and

"array_size" is the size of the array that we want to assign to the array.

The array size is always mentioned inside the "[]".

For example:
Int age[5];



The following will be the result of the above declarations:

age[0]	age[1]	age[2]	age[3]	age[4]

- **Initializing Arrays**

Initializing of array is very simple in c programming. The initializing values are enclosed within the curly braces in the declaration and placed following an equal sign after the array name. Here is an example which declares and initializes an array of five elements of type int. Array can also be initialized after declaration. Look at the following code, which demonstrate the declaration and initialization of an array.

int age[5]={2,3,4,5,6};

It is not necessary to define the size of arrays during initialization

e.g. : int age[]={2,3,4,5,6};

In this case, the compiler determines the size of array by calculating the number of elements of an array.

age[0]	age[1]	age[2]	age[3]	age[4]
2 ac	3	4	5	6

- **Accessing array elements**

In C programming, arrays can be accessed and treated like variables in C.

For example:

1.scanf("%d",&age[2]);
//statement to insert value in the third element of array age[]

2. printf("%d",age[2]);
//statement to print third element of an array.

Arrays can be accessed and updated using its index.An array of n elements, has indices ranging from 0 to n-1. An element can be updated simply by assigning

A[i] = x;

A great care must be taken in dealing with arrays. Unlike in Java, where array index out of bounds exception is thrown when indices go out of the 0..n-1 range, C arrays may not display any warnings if out of bounds indices are accessed. Instead,compiler may access the elements out of bounds, thus leading to critical run time errors.

Example of array in C programming

```
/* C program to find the sum marks of n students using arrays */
#include <stdio.h>
void main()
{
  int i,n;
  int marks[n];
  int sum=0;
  printf("Enter number of students: ");
  scanf("%d",&n);
 for(i=0;i<n;i++)
 {
   printf("Enter marks of student%d: ",i+1);
  scanf("%d",&marks[i]); //saving the marks in array
   sum+=marks[i];
  }
 printf("Sum of marks = %d",sum);
}
```

Output :

Enter number of students: (input by user) 3

Enter marks of student1: (input by user) 10

Enter marks of student2: (input by user) 29

Enter marks of student3: (input by user) 11

Sum of marks = 50

- **Create the one dimensional array**

To keep things simple we will start by creating an one dimensional character char array of size 6.

```
// 1D char array

char str[6] = "Hello";
```

Three things happens when we create the array.

- S mm6 blocks of memory locations is allocated for the array.
- The characters of the array are stored in that 6 blocks of memory.
- The variable name str points at the first location of the allocated memory locations of the array.

For Example:

```
int i;

char str[6] = "Hello";

 for (i = 0; str[i] != '\0'; i++) {

  printf("%c\n", str[i]);

 }
```

- **Creating pointer variable for the one dimensional array**

We will create a character pointer ptr variable that will hold the address of the character array str.

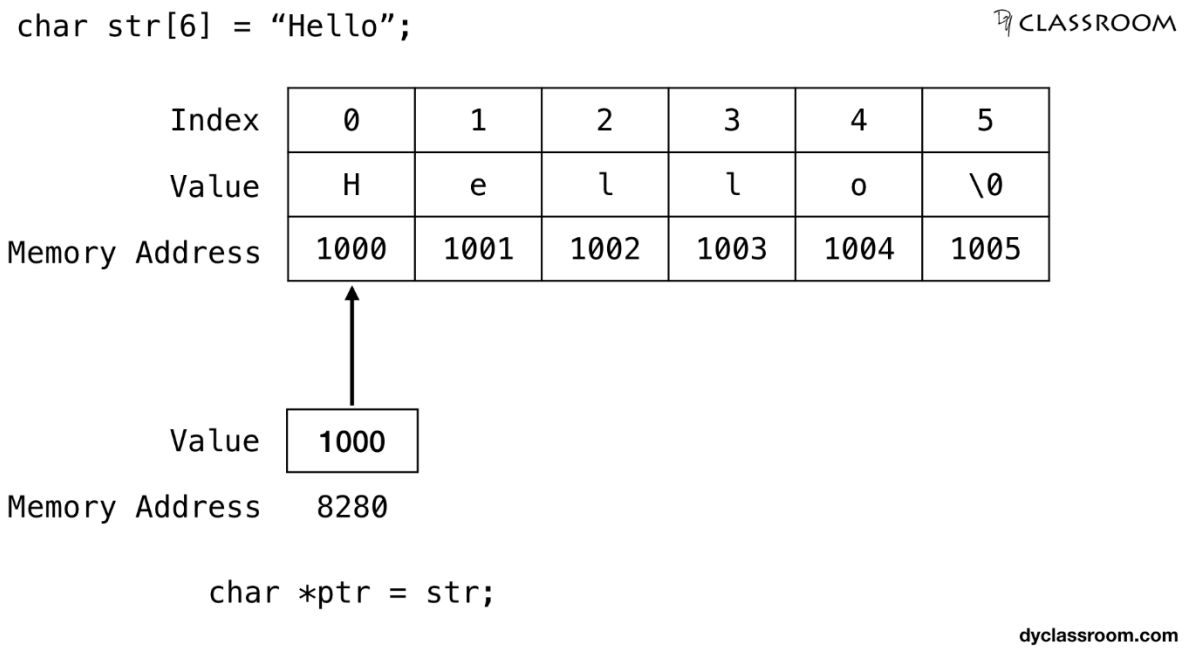
For Example:

```
// array

char str[6] = "Hello";

// pointer

char *ptr = str;
```



Program:

```
#include <stdio.h>
void main()
{
 char str[6] = "Hello";   // character array
 char *ptr = str;            // pointer ptr
                          // pointing at the character array str
  while (*ptr != '\0'
{
   printf("%c\n", *ptr);  )    // print the elements of the array str
   ptr++;                       // make the pointer ptr point at the
  }
 printf("End of code\n");
}
```

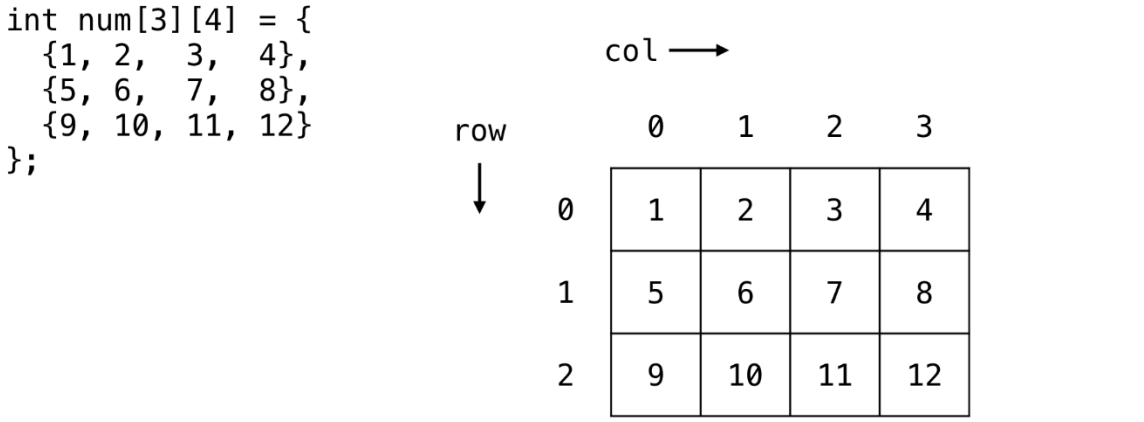
Output:

```
H
e
l
l
o
End of code
```

- ## Creating a two dimensional array

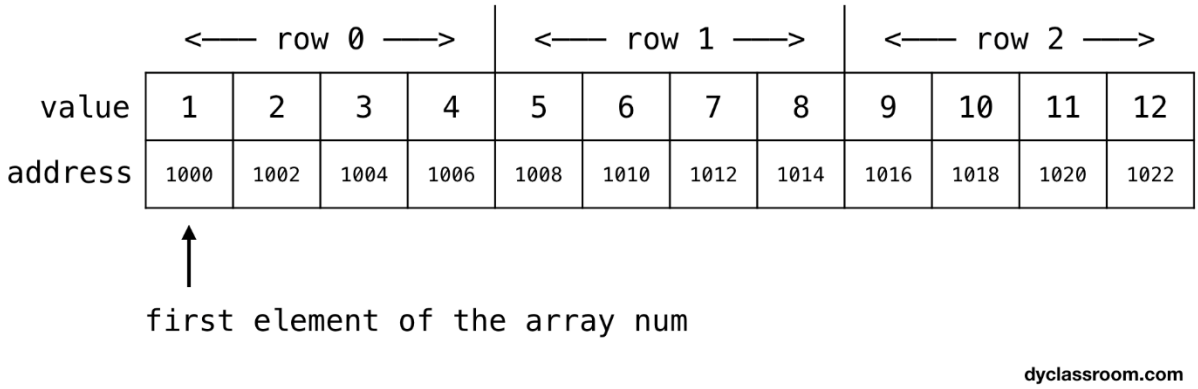
To keep things simple we will create a two dimensional integer array num having 3 rows and 4 columns.

```
int num[3][4] = {
{1, 2,  3,  4},
{5, 6,  7,  8},
{9, 10, 11, 12}
};
```



In the above image we are showing the two dimensional array having 3 rows and 4 columns.
The compiler will allocate the memory for the above two dimensional array **row-wise** meaning the first element of the second row will be placed after the last element of the first row.And if we assume that the first element of the array is at address 1000 and the size of type int is 2 bytes then the elements of the array will get the following allocated memory locations.

```
int num[3][4] = {
  {1, 2,  3,  4},
  {5, 6,  7,  8},
  {9, 10, 11, 12}
};
```



- **Create pointer for the two dimensional array:**

We have created the two dimensional **integer** array num so, our pointer will also be of type int.

We will assign the address of the first element of the array num to the pointer ptr using the **address of** & operator.

int *ptr = &num[0][0];

- **Accessing the elements of the two dimensional array via pointer:**

The two dimensional array num will be saved as a continuous block in the memory. So, if we increment the value of ptr by 1 we will move to the next block in the allocated memory.

In the following code we are printing the content of the num array using for loop and by incrementing the value of ptr.

Program:

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

```
 // 2d array
int num[3][4] = {
  {1, 2,  3,  4},
  {5, 6,  7,  8},
  {9, 10, 11, 12}
};
// pointer ptr pointing at array num
 int *ptr = &num[0][0];
 // other variables
 int
  ROWS = 3,
  COLS = 4,
  TOTAL_CELLS = ROWS * COLS,
  i;
 // print the elements of the array num via pointer ptr
   for (i = 0; i < TOTAL_CELLS; i++)
  {
  printf("%d ", *(ptr + i));
   }
 }
```

Output:

1 2 3 4 5 6 7 8 9 10 11 12

(b) Functions:

1. WHAT IS C FUNCTION?

A large C program is divided into basic building blocks called C function. C functioncontains set of instructions enclosed by "{ }" which performs specific operation in a C program. Actually, Collection of these functions creates a C program.

2. USES OF C FUNCTIONS:

•C functions are used to avoid rewriting same logic/code again and again in a program.
•There is no limit in calling C functions to make use of same functionality wherever required.
•We can call functions any number of times in a program and from any place in a program.
•A large C program can easily be tracked when it is divided into functions.
•The core concept of C functions are, re-usability, dividing a big task into small pieces to achieve the functionality and to improve understandability of very large C programs.

3. C FUNCTION DECLARATION, FUNCTION CALL AND FUNCTION DEFINITION:

There are 3 aspects in each C function. They are,

•Function declaration or prototype – This informs compiler about the function name, function parameters and return value's data type.

•Function call – This calls the actual function

•Function definition – This contains all the statements to be executed.

C functions aspects	syntax
function definition	Return_type function_name (arguments list) { Body of function; }
function call	function_name (arguments list);
function declaration	return_type function_name (argument list);

SIMPLE EXAMPLE PROGRAM FOR C FUNCTION:

As you know, functions should be declared and defined before calling in a C program.
In the below program, function "square" is called from main function.
The value of "m" is passed as argument to the function "square".
This value is multiplied by itself in this function and multiplied value "p" is returned to main function from function "square".

Example:

```
#include<stdio.h>
// function prototype, also called function declaration
float square ( float x );
// main function, program starts from here
 void main( )
{
   float m, n ;
   printf ( "\nEnter some number for finding square \n");
   scanf ( "%f", &m ) ;
   // function call
   n = square ( m ) ;
   printf ( "\nSquare of the given number %f is %f",m,n );
```

```
}
 float square ( float x )   // function definition
{
   float p ;
   p = x * x ;
}
```

OUTPUT:
Enter some number for finding square
2
Square of the given number 2.000000 is 4.000000

4. HOW TO CALL C FUNCTIONS IN A PROGRAM?

There are two ways that a C function can be called from a program. They are,
1.Call by value
2.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.
Different Memory is allocated for both actual and formal parameters. Because, value of actual parameter is copied to formal parameter.
Note:
Actual parameter – This is the argument which is used in function call.
Formal parameter – This is the argument which is used in function definition

EXAMPLE PROGRAM FOR C FUNCTION (USING CALL BY VALUE):

In this program, the values of the variables "m" and "n" are passed to the function "swap".
These values are copied to formal parameters "a" and "b" in swap function and used.
EXAMPLE:

```
#include<stdio.h>
// function prototype, also called function declaration
void swap(int a, int b);
```

```
int main()
{
   int m = 22, n = 44;
   // calling swap function by value
   printf(" values before swap  m = %d \nand n = %d", m, n);
   swap(m, n);
}

void swap(int a, int b)
{
   int tmp;
   tmp = a;
   a = b;
   b = tmp;
   printf(" \nvalues after swap m = %d\n and n = %d", a, b);
}
```

OUTPUT:
values before swap m = 22
and n = 44
values after swap m = 44
and n = 22

2. 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.

EXAMPLE PROGRAM FOR C FUNCTION (USING CALL BY REFERENCE):

In this program, the address of the variables "m" and "n" are passed to the function "swap".
These values are not copied to formal parameters "a" and "b" in swap function.
Because, they are just holding the address of those variables.
This address is used to access and change the values of the variables.

EXAMPLE:

```
#include<stdio.h>
// function prototype, also called function declaration
```

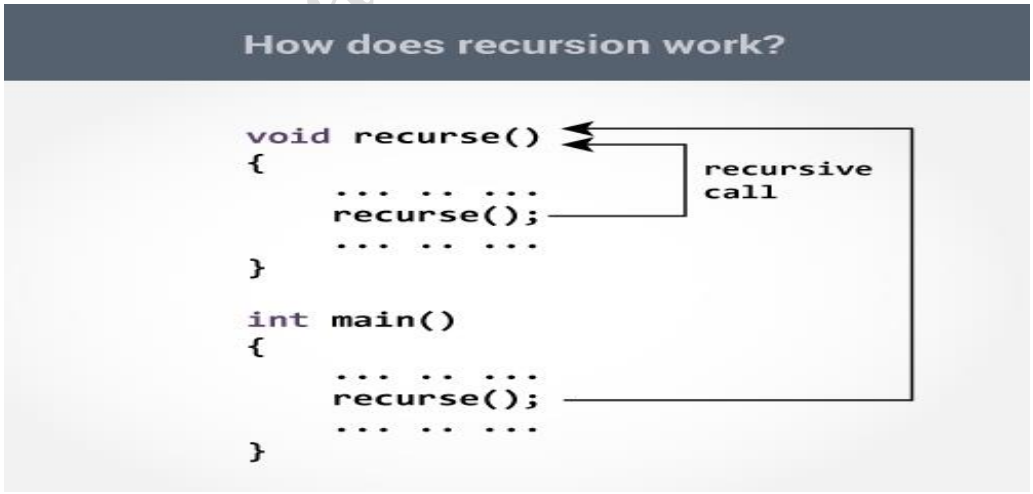
```
void swap(int *a, int *b);
 int main()
{
   int m = 22, n = 44;
   //  calling swap function by reference
   printf("values before swap m = %d \n and n = %d",m,n);
   swap(&m, &n);
}

void swap(int *a, int *b)
{
   int tmp;
   tmp = *a;
   *a = *b;
   *b = tmp;
   printf("\n values after swap a = %d \nand b = %d", *a, *b);
}
```

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

- ## Recursion Function:

A function that calls itself is known as a recursive function. And, this technique is known as recursion.



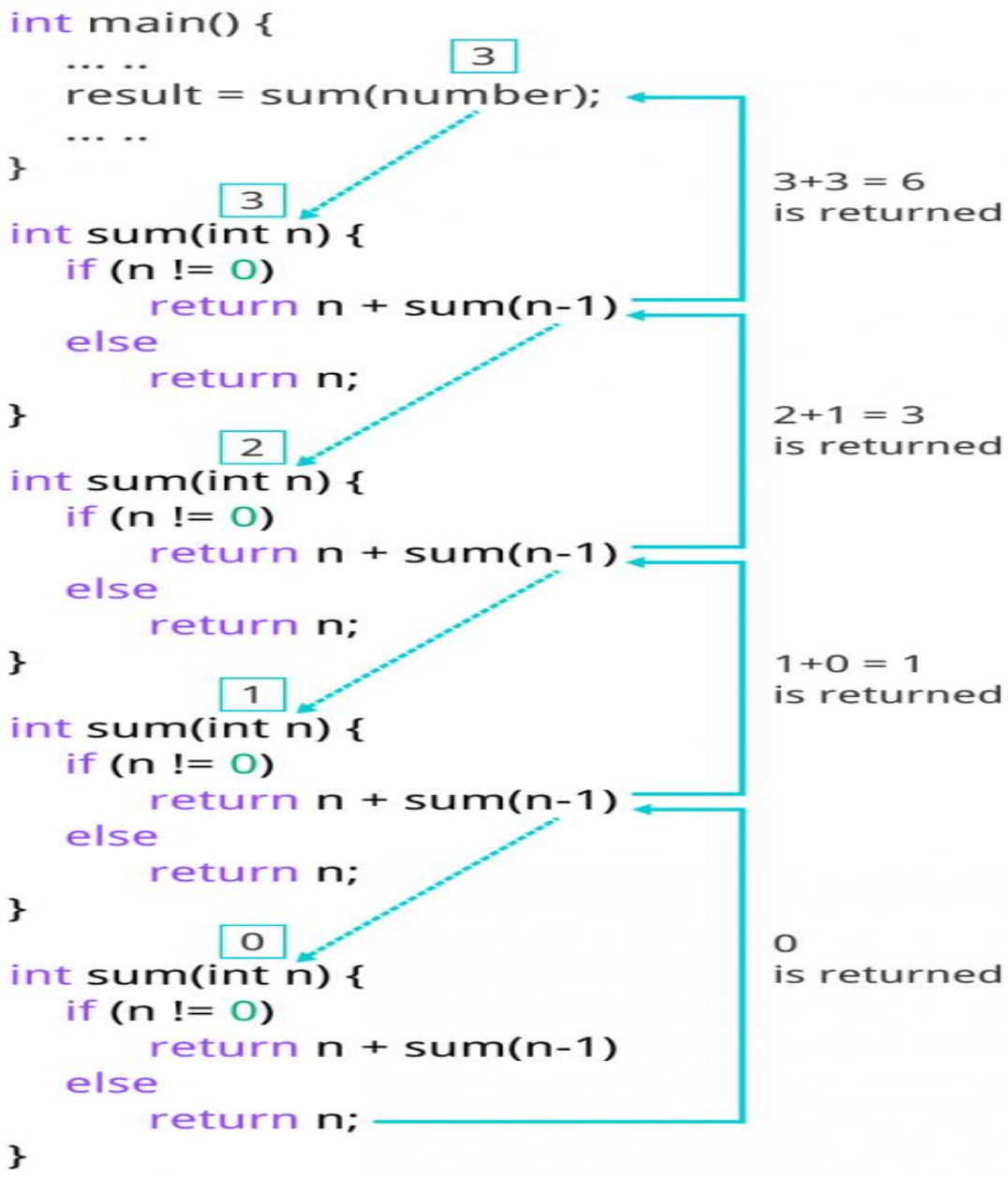
The recursion continues until some condition is met to prevent it.To prevent infinite recursion, if...else statement (or similar approach) can be used where one branch makes the recursive call, and other doesn't.

Example: 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;
}
```

Output:
Enter a positive integer:3
sum = 6

Initially, the sum() is called from the main() function with number passed as an argument.
Suppose, the value of n inside sum() is 3 initially. During the next function call, 2 is passed to the sum() function. This process continues until n is equal to 0.
When n is equal to 0, the if condition fails and the else part is executed returning the sum of integers ultimately to the main() function.
How Its Works?


int main() {
... ..
result = sum(number);
... ..
}
3
int sum(int n) {
if (n != 0)
return n + sum(n-1)
else
return n;
}
3+3 = 6
is returned
2
int sum(int n) {
if (n != 0)
return n + sum(n-1)
else
return n;
}
2+1 = 3
is returned
1
int sum(int n) {
if (n != 0)
return n + sum(n-1)
else
return n;
}
1+0 = 1
is returned
0
int sum(int n) {
if (n != 0)
return n + sum(n-1)
else
return n;
}
0
is returned