BYJU'S EXAM PREP

Vision 2023

A Course for GATE & PSUs

Computer Science Engineering

Programming and Data structures

CHAPTER 2

Recursion

CHAPTER 2

PROGRAMMING & DATA STRUCTURES

RECURSION

Functions:

In c, we can divide a large program into the basic building blocks known as function. The function contains the set of programming statements enclosed by {}. A function can be called multiple times to provide reusability and modularity to the C program. In other words, we can say that the collection of functions creates a program.

Advantage of functions in C

There are the following advantages of C functions.

- By using functions, we can avoid rewriting the same logic/code again and again in a program.
- We can call C functions any number of times in a program and from any place in a program.
- We can track a large C program easily when it is divided into multiple functions.
- Reusability is the main achievement of C functions.
- However, Function calling is always overhead in a C program.

1. FUNCTION ASPECTS

There are three aspects of a C function.

1.1 Function declaration A function must be declared globally in a c program to tell the compiler about the function name, function parameters, and return type.

1.2 Function call Function can be called from anywhere in the program. The parameter list must not differ in function calling and function declaration. We must pass the same number of functions as it is declared in the function declaration.

1.3 Function definition It contains the actual statements which are to be executed. It is the most important aspect to which the control comes when the function is called. Here, we must notice that only one value can be returned from the function.

The syntax of creating function in c language is given below:

```
return_type function_name(data_type parameter...)
{
//code to be executed
}
```

SN	C function aspects	Syntax
1	Function declaration	return_typefunction_name (argument list);
2	Function call	function_name (argument_list)
3	Function definition	return_typefunction_name (argument list) {function body;}

Examples

Given below is the source code for a function called **max()**. This function takes two parameters num1 and num2 and returns the maximum value between the two –

```
/* function returning the max between two numbers */
int max(int num1, int num2) {

   /* local variable declaration */
   int result;

   if (num1 > num2)
      result = num1;
   else
      result = num2;

   return result;
}
```

Function Declarations

A function **declaration** tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately.

A function declaration has the following parts –

```
return_type function_name( parameter list );
```

For the above defined function max(), the function declaration is as follows –

```
int max(int num1, int num2);
```

Parameter names are not important in function declaration only their type is required, so the following is also a valid declaration –

```
int max(int, int);
```

Function declaration is required when you define a function in one source file and you call that function in another file. In such a case, you should declare the function at the top of the file calling the function.

Calling a Function

While creating a C function, you give a definition of what the function has to do. To use a function, you will have to call that function to perform the defined task.

When a program calls a function, the program control is transferred to the called function. A called function performs a defined task and when its return statement is executed or when its function-ending closing brace is reached, it returns the program control back to the main program.

To call a function, you simply need to pass the required parameters along with the function name, and if the function returns a value, then you can store the returned value. For example –

```
#include <stdio.h>
/* function declaration */
int max(int num1, int num2);
int main () {
   /* local variable definition */
   int a = 100;
   int b = 200;
   int ret;
   /* calling a function to get max value */
   ret = max(a, b);
   printf( "Max value is : %d\n", ret );
   return 0;
}

/* function returning the max between two numbers */
int max(int num1, int num2) {

   /* local variable declaration */
   int result;
   if (num1 > num2)
      result = num1;
   else
      result = num2;
   return result;
}
```

We have kept max() along with main() and compiled the source code. While running the final executable, it would produce the following result –

Max value is : 200

Function Arguments

If a function is to use arguments, it must declare variables that accept the values of the arguments. These variables are called the **formal parameters** of the function.

Formal parameters behave like other local variables inside the function and are created upon entry into the function and destroyed upon exit.

While calling a function, there are two ways in which arguments can be passed to a function−

Sr.No.	Call Type & Description
1	**Call by value** This method copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.
2	**Call by reference** This method copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the argument.

By default, C uses **call by value** to pass arguments. In general, it means the code within a function cannot alter the arguments used to call the function.

2. TYPES OF FUNCTIONS

There are two types of functions in C programming:

1. **Library Functions**:

   The standard library functions are built-in functions in C programming.

   These functions are defined in header files. For example,

   - The printf() is a standard library function to send formatted output to the screen (display output on the screen). This function is defined in the stdio.h header file. Hence, to use the printf()function, we need to include the stdio.h header file using #include <stdio.h>.
   - The sqrt() function calculates the square root of a number. The function is defined in the math.h header file.

2. **User-defined functions**: are the functions which are created by the C programmer, so that he/she can use it many times. It reduces the complexity of a big program and optimizes the code.

How user-defined function works

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

int main()
{
   ... .. ...
   ... .. ...

   functionName();

   ... .. ...
   ... .. ...
}
```

The execution of a C program begins from the main() function.

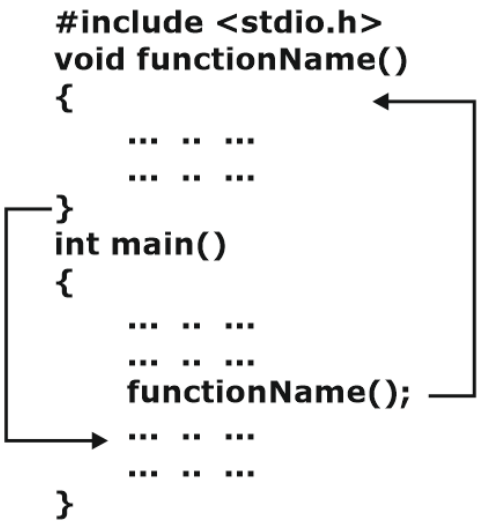
When the compiler encounters functionName();, control of the program jumps to

void functionName()

And, the compiler starts executing the codes inside functionName().

The control of the program jumps back to the main() function once code inside the function definition is executed.

How function works in C programming?



Note, function names are identifiers and should be unique.

Advantages of user-defined function

1. The program will be easier to understand, maintain and debug.
2. Reusable codes that can be used in other programs
3. A large program can be divided into smaller modules. Hence, a large project can be divided among many programmers.

3. RETURN VALUE

A C function may or may not return a value from the function. If you don't have to return any value from the function, use void for the return type.

Let's see a simple example of a C function that doesn't return any value from the function.

Example without return value:

```
void hello(){
printf("hello c");
}
```

If you want to return any value from the function, you need to use any data type such as int, long, char, etc. The return type depends on the value to be returned from the function.

Let's see a simple example of a C function that returns int value from the function.

Example with return value:

```
int get(){
return 10;
}
```

In the above example, we have to return 10 as a value, so the return type is int. If you want to return floating-point value (e.g., 10.2, 3.1, 54.5, etc), you need to use float as the return type of the method.

```
float get(){
return 10.2;
}
```

Now, you need to call the function, to get the value of the function.

1. **Function with no argument and no return value :** When a function has no arguments, it does not receive any data from the calling function. Similarly when it does not return a value, the calling function does not receive any data from the called function.

   Syntax :

   Function declaration : void function();

   Function call : function();

   Function definition :

```
void function()
{
  statements;
}
```

```
// C code for  function with no
//  arguments and no return value

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

Output:

The total amount is 5000.000000

2. **Function with arguments but no return value :** When a function has arguments, it receive any data from the calling function but it returns no values.

Syntax :

Function declaration : void function (int);

Function call : function(x);

Function definition:

```
void function( int x )
{
  statements;
}
```

```
// C code for function
// with argument but no return value
#include <stdio.h>
void function(int, int[], char[]);
int main()
{
```

```
    int a = 20;
    int ar[5] = { 10, 20, 30, 40, 50 };
    char str[30] = "gradeup";
    function(a, &ar[0], &str[0]);
    return 0;
}

void function(int a, int* ar, char* str)
{
    int i;
    printf("value of a is %d\n\n", a);
    for (i = 0; i < 5; i++) {
        printf("value of ar[%d] is %d\n", i, ar[i]);
    }
    printf("\nvalue of str is %s\n", str);
}
```

Output:

value of a is 20

value of ar[0] is 10

value of ar[1] is 20

value of ar[2] is 30

value of ar[3] is 40

value of ar[4] is 50

The given string is : gradeup

3. **Function with no arguments but returns a value :** There could be occasions where we may need to design functions that may not take any arguments but returns a value to the calling function. An example for this is the getchar function, it has no parameters but it returns an integer type data that represents a character. Syntax :

Function declaration : int function();

Function call : function();

Function definition :

```
int function()
{
    statements;
     return x;
}
```

```
// C code for function with no arguments
// but have return value
#include <math.h>
#include <stdio.h>

int sum();
int main()
{
    int num;
    num = sum();
    printf("\nSum of two given values = %d", num);
    return 0;
}

int sum()
{
    int a = 50, b = 80, sum;
    sum = sqrt(a) + sqrt(b);
    return sum;
}
```

Output:

Sum of two given values = 16

Function declaration : int function (int);

Function call : function(x);

Function definition:

```
int function( int x )
{
  statements;
  return x;
}
```

```
// C code for function with arguments
// and with return value

#include <stdio.h>
#include <string.h>
int function(int, int[]);
int main()
```

```
{
    int i, a = 20;
    int arr[5] = { 10, 20, 30, 40, 50 };
    a = function(a, &arr[0]);
    printf("value of a is %d\n", a);
    for (i = 0; i < 5; i++) {
        printf("value of arr[%d] is %d\n", i, arr[i]);
    }
    return 0;
}

int function(int a, int* arr)
{
    int i;
    a = a + 20;
    arr[0] = arr[0] + 50;
    arr[1] = arr[1] + 50;
    arr[2] = arr[2] + 50;
    arr[3] = arr[3] + 50;
    arr[4] = arr[4] + 50;
    return a;
}
```

Output:

value of a is 40

value of arr[0] is 60

value of arr[1] is 70

value of arr[2] is 80

value of arr[3] is 90

value of arr[4] is 100

4. DIFFERENT ASPECTS OF FUNCTION CALLING

A function may or may not accept any argument. It may or may not return any value. Based on these facts, There are four different aspects of function calls.

a) function without arguments and without return value

b) function without arguments and with return value

c) function with arguments and without return value

d) function with arguments and with return value

5. TYPES OF FUNCTION CALL

5.1. Call by value in C

- In the call by value method, the value of the actual parameters is copied into the formal parameters. In other words, we can say that the value of the variable is used in the function call in the call by value method.
- In the call by value method, we can not modify the value of the actual parameter by the formal parameter.
- In call by value, different memory is allocated for actual and formal parameters since the value of the actual parameter is copied into the formal parameter.
- The actual parameter is the argument which is used in the function call whereas the formal parameter is the argument which is used in the function definition.

Let's try to understand the concept of call by value in c language by the example given below:

```
void change(int num) {
printf("Before adding value inside function num=%d \n",num);
num=num+100;
printf("After adding value inside function num=%d \n", num);
}
int main() {
   int x=100;
printf("Before function call x=%d \n", x);
   change(x);//passing value in function
printf("After function call x=%d \n", x);
return 0;
}
```

Output-

```
Before function call x=100
Before adding value inside function num=100
After adding value inside function num=200
After function call x=10
```

Examples

1.

```
#include <stdio.h>
void swap(int , int); //prototype of the function
int main()
{
   int a = 10;
   int b = 20;
```

```
    printf("Before swapping the values in main a = %d, b = %d\n",a,b); // printing the
       value of a and b in main
    swap(a,b);
    printf("After swapping values in main a = %d, b = %d\n",a,b); // The value of actual
        parameters do not change by changing the formal parameters in call by value, a
       = 10, b = 20
}
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); // Formal paramet
       ers, a = 20, b = 10
}
```

Output

Before swapping the values in main a = 10, b = 20

After swapping values in function a = 20, b = 10

After swapping values in main a = 10, b = 20

2.

```
#include <stdio.h>
int increment(int var)
{
   var = var+1;
   return var;
}

int main()
{
  int num1=20;
  int num2 = increment(num1);
  printf("num1 value is: %d", num1);
  printf("\nnum2 value is: %d", num2);

  return 0;
}
```

Output:

num1 value is: 20

num2 value is: 21

Explanation

We passed the variable num1 while calling the method, but since we are calling the function using call by value method, only the value of num1 is copied to the formal parameter var. Thus change made to the var doesn't reflect in the num1.

5.2. Call by reference in C

In call by reference, the address of the variable is passed into the function call as the actual parameter.

The value of the actual parameters can be modified by changing the formal parameters since the address of the actual parameters is passed.

In call by reference, the memory allocation is similar for both formal parameters and actual parameters. All the operations in the function are performed on the value stored at the address of the actual parameters, and the modified value gets stored at the same address.

Consider the following example for the call by reference.

```
1. #include<stdio.h>
void change(int *num) {
printf("Before adding value inside function num=%d \n",*num);
   (*num) += 100;
printf("After adding value inside function num=%d \n", *num);
}
int main() {
   int x=100;
printf("Before function call x=%d \n", x);
   change(&x);//passing reference in function
printf("After function call x=%d \n", x);
return 0;
}
```

Output

```
Before function call x=100
Before adding value inside function num=100
After adding value inside function num=200
After function call x=200
```

Difference between call by value and call by reference in c

2.
```
#include <stdio.h>
void swap(int *, int *); //prototype of the function
int main()
{
   int a = 10;
   int b = 20;
   printf("Before swapping the values in main a = %d, b = %d\n",a,b); // printing the v
      alue of a and b in main
   swap(&a,&b);
   printf("After swapping values in main a = %d, b = %d\n",a,b); // The values of actu
      al parameters do change in call by reference, a = 10, b = 20
    }
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); // Formal para
      meters, a = 20, b = 10
}
```

Output

```
Before swapping the values in main a = 10, b = 20
After swapping values in function a = 20, b = 10
After swapping values in main a = 20, b = 10
```

3.
```
#include <stdio.h>
void increment(int  *var)
{
   /* Although we are performing the increment on variable
    * var, however the var is a pointer that holds the address
    * of variable num, which means the increment is actually done
    * on the address where the value of num is stored.
    */
   *var = *var+1;
}
int main()
{
    int num=20;
```

```
    /* This way of calling the function is known as call by
     * reference. Instead of passing the variable num, we are
     * passing the address of variable num
     */
    increment(&num);
    printf("Value of num is: %d", num);
  return 0;
}
```

Output:

Value of num is: 21

4. include <stdio.h>

```
void disp( int *num)
{
   printf("%d ", *num);
}

int main()
{
    int arr[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 0};
    for (int i=0; i<10; i++)
    {
       /* Passing addresses of array elements*/
       disp (&arr[i]);
    }

    return 0;
}
```

Output:

1 2 3 4 5 6 7 8 9 0

Difference between call by value and call by reference

Parameters	Call by value	Call by reference
Definition	While calling a function, when you pass values by copying variables, it is known as "Call By Values."	While calling a function, in programming language instead of copying the values of variables, the address of the variables is used, it is known as "Call By References.
Arguments	In this method, a copy of the variable is passed.	In this method, a variable itself is passed.
Effect	Changes made in a copy of a variable never modify the value of the variable outside the function.	Change in the variable also affects the value of the variable outside the function.
Alteration of value	Does not allow you to make any changes in the actual variables.	Allows you to make changes in the values of variables by using function calls.
Passing of variable	Values of variables are passed using a straightforward method.	Pointer variables are required to store the address of variables.
Value modification	Original value not modified.	The original value is modified.
Memory Location	Actual and formal arguments will be created in different memory location	Actual and formal arguments will be created in the same memory location
Safety	Actual arguments remain safe as they cannot be modified accidentally.	Actual arguments are not Safe. They can be accidentally modified, so you need to handle arguments operations carefully.
Default	Default in many programming languages like C++.PHP. Visual Basic NET, and C#.	It is supported by most programming languages like JAVA, but not by default.

Advantages of using Call by value method

Pros/benefits of a call by value method:

- The method doesn't change the original variable, so it is preserving data.
- Whenever a function is called it, never affect the actual contents of the actual arguments.
- Value of actual arguments passed to the formal arguments, so any changes made in the formal argument does not affect the real cases.

Advantages of using Call by reference method

Pros of using call by reference method:

- The function can change the value of the argument, which is quite useful.
- It does not create duplicate data for holding only one value which helps you to save memory space.
- In this method, there is no copy of the argument made. Therefore it is processed very fast.
- Helps you to avoid changes done by mistake
- A person reading the code never knows that the value can be modified in the function.

Disadvantages of using Call by value method

Here, are major cons/drawbacks of a call by value method:

- Changes to actual parameters can also modify corresponding argument variables
- In this method, arguments must be variables.
- You can't directly change a variable in a function body.
- Sometime argument can be complex expressions
- There are two copies created for the same variable which is not memory efficient.

Disadvantages of using Call by reference method

Here, are major cons of using call by reference method:

- Strong non-null guarantee. A function taking in a reference needs to make sure that the input is non-null. Therefore, null checks need not be made.
- Passing by reference makes the function not pure theoretically.
- A lifetime guarantee is a big issue with references. This is specifically dangerous when working with lambdas and multi-threaded programs.

6. RECURSION IN C

Recursion is the process which comes into existence when a function calls a copy of itself to work on a smaller problem. Any function which calls itself is called a recursive function, and such function calls are called recursive calls. Recursion involves several numbers of recursive calls. However, it is important to impose a termination condition of recursion. Recursion code is shorter than iterative code however it is difficult to understand.

Recursion cannot be applied to all the problems, but it is more useful for the tasks that can be defined in terms of similar subtasks. For Example, recursion may be applied to sorting, searching, and traversal problems.

In the following example, recursion is used to calculate the factorial of a number.

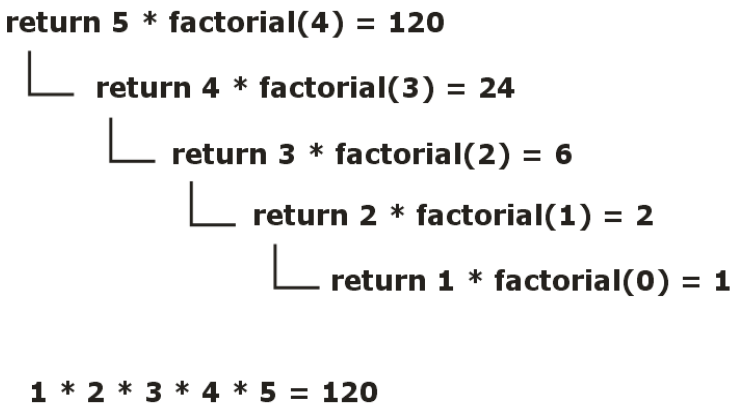
```
int fact(int n)
{
   if (n==0)
   {
      return 0;
   }
   else if ( n == 1)
   {
      return 1;
   }
   else
   {
      return n*fact(n-1);
   }
}
```

Output

// let n=5

factorial = 120

We can understand the above program of the recursive method call by the figure given below:



6.1. Types of Recursion

(i) Tail recursion

(ii) Non Tail recursion

(iii) Indirect recursion

(iv) nested recursion

(i) Tail recursion: -

In the programs the very last statement is a recursion call. And there is no other statement after that, then it is called Tail recursion.

WAP of print array elements using recursion.

```
Int a[0…..4] = {10, 20, 30, 40, 50}
               i        j
Print_array (a, i, j)
{
if (i==j)
  print (a[i]);
Else
  print (a[i]);
print_array (a, i+1, j);            recursion(Tail recursion)
  }
```

O/P:- 10, 20, 30, 40, 50

Non-recursion program:-

```
For (i =0;  i<j; i+t)
Print (a[i]);
```

→ In the tail recursion unnecessarily wasting the stuck space.

→ In tail recursion it is easy to write an equivalent non-recursion programme.

(ii) Non- tail recursion:-

In the non tail recursion recursion call can be anywhere other than the last statement.

```
A(int n)
{ if (n ≤ 1) returns;
Else
  A(n-2);
  print (n);
  A(n-1);
  print (n-2);
}
```

What is the O/P of A(5) = ?

O/P :- 3 2 0 1 5 2 0 4 3 2 0 1 2 3

Note:-

It is difficult to write an equivalent non-recursive program.

(iii) Indirect- Recursion: -

Two or more functions calling each other are called Indirect- Recursion.

```
Eg:-  A(int n)
{
If (n ≤ 1) return;
Else
 {
   B(n-2);
   print(n);
   B(n-1);
   print(n-2);
  }
}
```

```
B(int-n)
 {
    If (n ≤ 1) return;
    Else
     {
 print(n-3);
  A(n-1);
  A(n-2);
        }
    }
```

What is the o/p of A(5) = ?

O/P:- 0, 2, 0, 5, 1, 3, -1, 1, 2, 0

(iv) Nested Recursion:-

A recursion function which is looping itself as a parameter to a recursion call is called the nested recursion.

$$A(m,n)=\begin{cases}\eta+1, \text{ if } m=0\\ A(m-1,1), \text{if } n=0\\ A(m-1,A(m,n-1)), \text{otherwise}\end{cases}$$

6.2. Recursive Function

A recursive function performs the tasks by dividing it into the subtasks. There is a termination condition defined in the function which is satisfied by some specific subtask. After this, the recursion stops and the final result is returned from the function.

The case at which the function doesn't recur is called the base case whereas the instances where the function keeps calling itself to perform a subtask, is called the recursive case. All the recursive functions can be written using this format.

Pseudocode for writing any recursive function is given below.

```
if (test_for_base)
{
   return some_value;
}
else if (test_for_another_base)
{
   return some_another_value;
}
```

```
else
{
   // Statements;
   recursive call;
}
```

Example of recursion in C

Let's see an example to find the nth term of the Fibonacci series.

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

Output

Enter the value of n?12

144

Example: Which of the following is correct output for the program code given below?

```
Main ( )
{
Void fun ( );
fun ( );
fun ( );
}
void fun ( );
{
static int i = 1;
auto int j = 5;
printf ("%d", (i++));
printf ("%d", (j++));
}
```

A.1 5 2 6 3 7

B.2 6 3 7 4 8

C.1 5 2 5

D.1 5 2 5 3 5

Answer: C

Solution :

An object whose storage class is auto, is reinitialized at every function call whereas an object whose storage class static persists its value between different function calles.

When the function fun ( ) is called for the first time, value of i and j are printed and sequentially incremented. During the second function call, i returns its incremented value whereas j is initialized, hence i will print 2 and j will print 5 again.

****