functions in C

The function in C language is also known as procedure or subroutine in other programming languages.

To perform any task, we can create a function. A function can be called many times. It provides modularity and code reusability.

Advantage of functions in C

There are many advantages of functions.

1) Code Reusability

By creating functions in C, you can call it many times. So we don't need to write the same code again and again.

2) Code optimization

It makes the code optimized, we don't need to write much code.

Suppose, you have to check 3 numbers (781, 883 and 531) whether it is a prime number or not. Without using the function, you need to write the prime number logic 3 times. So, there is a repetition of code.

But if you use functions, you need to write the logic only once and you can reuse it several times.

Types of Functions

There are two types of functions in C programming:

1. Library Functions: are the functions which are declared in the C header files such as scanf(), printf(), gets(), puts(), ceil(), floor() etc.

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.

Declaration of a function

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

1. return\_type function\_name(data\_type parameter...){

2. //code to be executed

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 C function that doesn't return any value from the function.

Example without return value:

1. void hello(){

2. printf("hello c");

3. }

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 C function that returns int value from the function.

Example with return value:

1. int get(){

2. return 10;

3. }

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.

1. float get(){

2. return 10.2;

3. }

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

Parameters in C Function

A c function may have 0 or more parameters. You can have any type of parameter in C program such as int, float, char etc. The parameters are also known as formal arguments.

Example of a function that has 0 parameter:

1. void hello(){

2. printf("hello c");

3. }

Example of a function that has 1 parameter:

1. int cube(int n){

2. return n\*n\*n;

3. }

Example of a function that has 2 parameters:

1. int add(int a, int b){

2. return a+b;

3. }

Calling a function in C

If a function returns any value, you need to call the function to get the value returned from the function. The syntax of calling a function in c programming is given below:

1. variable=function\_name(arguments...);

1) variable: The variable is not mandatory. If the function return type is void, you must not provide the variable because void functions don't return any value.

2) function\_name: The function\_name is the name of the function to be called.

3) arguments: You need to provide arguments while calling the C function. It is also known as actual arguments.

Example to call a function:

1. hello();//calls function that doesn't return a value

2. int value=get();//calls function that returns value

3. int value2=add(10,20);//calls parameterized function by passing 2 values

Example of C function with no return statement

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

1. #include <stdio.h>

2. #include <conio.h>

3. //defining function

4. void hello(){

5. printf("hello c programming");

6. }

7. void main(){

8. clrscr();

9.

10. hello();//calling a function

11. hello();

12. hello();

13.

14. getch();

15. }

Output

hello c programming

hello c programming

hello c programming

Example of C function with return statement

Let's see the simple program of function in c language.

1. #include <stdio.h>

2. #include <conio.h>

3. //defining function

4. int cube(int n){

5. return n\*n\*n;

6. }

7. void main(){

8. int result1=0,result2=0;

9. clrscr();

10.

11. result1=cube(2);//calling function

12. result2=cube(3);

13.

14. printf("%d \n",result1);

15. printf("%d \n",result2);

16.

17. getch();

18. }

Output

8

27

Call by value and call by reference in C

There are two ways to pass value or data to function in C language: call by value and call by reference. The original value is not modified in the call by value but it is modified in the call by reference.

Let's understand call by value and call by reference in c language one by one.

Call by value in C

In call by value, the original value is not modified.

In call by value, the value being passed to the function is locally stored by the function parameter in the stack memory location.

If you change the value of the function parameter, it is changed for the current function only.

It will not change the value of the variable inside the caller method such as main().

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

1. #include <stdio.h>

2. #include <conio.h>

3. void change(int num) {

4. printf("Before adding value inside function num=%d \n",num);

5. num=num+100;

6. printf("After adding value inside function num=%d \n", num);

7. }

8.

9. int main() {

10. int x=100;

11. clrscr();

12.

13. printf("Before function call x=%d \n", x);

14. change(x);//passing value in function

15. printf("After function call x=%d \n", x);

16.

17. getch();

18. return 0;

19. }

Output

Before function call x=100

Before adding value inside function num=100

After adding value inside function num=200

After function call x=100

Call by reference in C

In call by reference, the original value is modified because we pass a reference (address).

Here, the address of the value is passed in the function, so actual and formal arguments share the same address space.

Hence, value changed inside the function is reflected inside as well as outside the function.

Note: To understand the call by reference, you must have the basic knowledge of pointers.

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

1. #include <stdio.h>

2. #include <conio.h>

3. void change(int \*num) {

4. printf("Before adding value inside function num=%d \n",\*num);

5. (\*num) += 100;

6. printf("After adding value inside function num=%d \n", \*num);

7. }

8.

9. int main() {

10. int x=100;

11. clrscr();

12.

13. printf("Before function call x=%d \n", x);

14. change(&x);//passing reference in function

15. printf("After function call x=%d \n", x);

16.

17. getch();

18. return 0;

19. }

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

No. Call by value Call by reference

1 A copy of the value is passed to the function An address of value is passed to the function

2 Changes made inside the function is not reflected on other functions Changes made inside the function is reflected outside the function also

3 Actual and formal arguments will be created in different memory location Actual and formal arguments will be created in same memory location

Recursion in C

When the function is called within the same function, it is known as recursion in C. The function which calls the same function is known as a recursive function.

A function that calls itself, and doesn't perform any task after the function call, is known as tail recursion. In tail recursion, we generally call the same function with the return statement. An example of tail recursion is given below.

Let's see a simple example of recursion.

1. recursionfunction(){

2.

3. recursionfunction();//calling self function

4.

5. }

Example of tail recursion in C

Let's see an example to print factorial number using tail recursion in C language.

1. #include<stdio.h>

2. #include<conio.h>

3. int factorial (int n)

4. {

5. if ( n < 0)

6. return -1; /\*Wrong value\*/

7. if (n == 0)

8. return 1; /\*Terminating condition\*/

9. return (n \* factorial (n -1));

10. }

11.

12. void main(){

13. int fact=0;

14. clrscr();

15. fact=factorial(5);

16. printf("\n factorial of 5 is %d",fact);

17.

18. getch();

19. }

Output

factorial of 5 is 120

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

Storage Classes in C

Storage classes are used to define scope and life time of a variable. There are four storage classes in C programming.

o auto

o extern

o static

o register

Storage Classes Storage Place Default Value Scope Life-time

auto RAM Garbage Value Local Within function

extern RAM Zero Global Till the end of main program, May be declared anywhere in the program

static RAM Zero Local Till the end of main program, Retains value between multiple functions call

register Register Garbage Value Local Within function

1) auto

The auto keyword is applied to all local variables automatically. It is the default storage class that is why it is known as automatic variable.

1. #include <stdio.h>

2. void main(){

3. int a=10;

4. auto int b=10;//same like above

5. printf("%d %d",a,b);

6. }

Output:

10 10

2) register

The register variable allocates memory in register than RAM. Its size is same of register size. It has a faster access than other variables.

It is recommended to use the register variable only for quick access such as in counter.

Note: We can't get the address of register variable.

1. register int counter=0;

3) static

The static variable is initialized only once and exists till the end of the program. It retains its value between multiple functions call.

The static variable has the default value 0 which is provided by compiler.

1. #include <stdio.h>

2. void func() {

3. static int i=0;//static variable

4. int j=0;//local variable

5. i++;

6. j++;

7. printf("i= %d and j= %d\n", i, j);

8. }

9. void main() {

10. func();

11. func();

12. func();

13. }

Output:

i= 1 and j= 1

i= 2 and j= 1

i= 3 and j= 1

4) extern

The extern variable is visible to all the programs. It is used if two or more files are sharing the same variable or function.

1. extern int counter=0;

C Array

Array in C language is a collection or group of elements (data). All the elements of c array are homogeneous (similar). It has a contiguous memory location.

C array is beneficial if you have to store similar elements. Suppose you have to store marks of 50 students, one way to do this is allotting 50 variables.

So it will be typical and hard to manage. For example, we can not access the value of these variables with only 1 or 2 lines of code.

Another way to do this is array. By using array, we can access the elements easily. Only few lines of code is required to access the elements of array.

Advantage of C Array

1) Code Optimization: Less code to access the data.

2) Easy to traverse data: By using the for loop, we can retrieve the elements of an array easily.

3) Easy to sort data: To sort the elements of an array, we need a few lines of code only.

4) Random Access: We can access any element randomly using the array.

A disadvantage of C Array

1) Fixed Size: Whatever size, we define at the time of declaration of the array, we can't exceed the limit. So, it doesn't grow the size dynamically like LinkedList which we will learn later.

Declaration of C Array

We can declare an array in the c language in the following way.

1. data\_type array\_name[array\_size];

Now, let us see the example to declare an array.

1. int marks[5];

Here, int is the data\_type, marks is the array\_name and 5 is the array\_size.

Initialization of C Array

A simple way to initialize array is by index. Notice that array index starts from 0 and ends with [SIZE - 1].

1. marks[0]=80;//initialization of array

2. marks[1]=60;

3. marks[2]=70;

4. marks[3]=85;

5. marks[4]=75;

C array example

1. #include <stdio.h>

2. #include <conio.h>

3. void main(){

4. int i=0;

5. int marks[5];//declaration of array

6. clrscr();

7.

8. marks[0]=80;//initialization of array

9. marks[1]=60;

10. marks[2]=70;

11. marks[3]=85;

12. marks[4]=75;

13.

14. //traversal of array

15. for(i=0;i<5;i++){

16. printf("%d \n",marks[i]);

17. }//end of for loop

18.

19. getch();

20. }

Output

80

60

70

85

75

C Array: Declaration with Initialization

We can initialize the c array at the time of declaration. Let's see the code.

1. int marks[5]={20,30,40,50,60};

In such case, there is no requirement to define size. So it can also be written as the following code.

1. int marks[]={20,30,40,50,60};

Let's see the full program to declare and initialize the array in C.

1. #include <stdio.h>

2. #include <conio.h>

3. void main(){

4. int i=0;

5. int marks[5]={20,30,40,50,60};//declaration and initialization of array

6. clrscr();

7.

8. //traversal of array

9. for(i=0;i<5;i++){

10. printf("%d \n",marks[i]);

11. }

12.

13. getch();

14. }

Output

20

30

40

50

60

Two Dimensional Array in C

The two-dimensional array in C language is represented in the form of rows and columns, also known as the matrix. It is also known as an array of arrays or list of arrays.

The two dimensional, three dimensional or other dimensional arrays are also known as multidimensional arrays.

Declaration of two dimensional Array in C

We can declare an array in the c language in the following way.

1. data\_type array\_name[size1][size2];

A simple example to declare two-dimensional array is given below.

1. int twodimen[4][3];

Here, 4 is the row number and 3 is the column number.

Initialization of 2D Array in C

A way to initialize the two dimensional array at the time of declaration is given below.

1. int arr[4][3]={{1,2,3},{2,3,4},{3,4,5},{4,5,6}};

Two dimensional array example in C

1. #include <stdio.h>

2. #include <conio.h>

3. void main(){

4. int i=0,j=0;

5. int arr[4][3]={{1,2,3},{2,3,4},{3,4,5},{4,5,6}};

6. clrscr();

7.

8. //traversing 2D array

9. for(i=0;i<4;i++){

10. for(j=0;j<3;j++){

11. printf("arr[%d] [%d] = %d \n",i,j,arr[i][j]);

12. }//end of j

13. }//end of i

14.

15. getch();

16. }

Output

arr[0][0] = 1

arr[0][1] = 2

arr[0][2] = 3

arr[1][0] = 2

arr[1][1] = 3

arr[1][2] = 4

arr[2][0] = 3

arr[2][1] = 4

arr[2][2] = 5

arr[3][0] = 4

arr[3][1] = 5

arr[3][2] = 6

Passing Array to Function in C

To reuse the array operation, we can create functions that receive an array as an argument. To pass array in function, we need to write the array name only in the function call.

1. functionname(arrayname);//passing array

There are 3 ways to declare a function that receives an array as an argument.

First way:

1. return\_type function(type arrayname[])

Declaring blank subscript notation [] is the widely used technique.

Second way:

1. return\_type function(type arrayname[SIZE])

Optionally, we can define size in subscript notation [].

Third way:

1. return\_type function(type \*arrayname)

You can also use the concept of pointer. In pointer chapter, we will learn about it.

C language passing array to function example

1. #include <stdio.h>

2. #include <conio.h>

3. int minarray(int arr[],int size){

4. int min=arr[0];

5. int i=0;

6. for(i=1;i<size;i++){

7. if(min>arr[i]){

8. min=arr[i];

9. }

10. }//end of for

11. return min;

12. }//end of function

13.

14. void main(){

15. int i=0,min=0;

16. int numbers[]={4,5,7,3,8,9};//declaration of array

17. clrscr();

18.

19. min=minarray(numbers,6);//passing array with size

20. printf("minimum number is %d \n",min);

21.

22. getch();

23. }

Output

minimum number is 3