**C Functions**

In this tutorial, you will be introduced to functions (both user-defined and standard library functions) in C programming. Also, you will learn why functions are used in programming.

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

Suppose, you need to create a program to create a circle and color it. You can create two functions to solve this problem:

* create a circle function
* create a color function

Dividing a complex problem into smaller chunks makes our program easy to understand and reuse.

**Types of function**

There are two types of function in C programming:

* [Standard library functions](https://www.programiz.com/c-programming/library-function)
* [User-defined functions](https://www.programiz.com/c-programming/c-user-defined-functions)

**Standard 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.

## Library Functions in Different Header Files

| C Header Files | Description |
| --- | --- |
| <assert.h> | Program assertion functions |
| [<ctype.h>](https://www.programiz.com/c-programming/library-function/ctype.h) | Character type functions |
| <locale.h> | Localization functions |
| [<math.h>](https://www.programiz.com/c-programming/library-function/math.h) | Mathematics functions |
| <setjmp.h> | Jump functions |
| <signal.h> | Signal handling functions |
| <stdarg.h> | Variable arguments handling functions |
| <stdio.h> | Standard Input/Output functions |
| <stdlib.h> | Standard Utility functions |
| [<string.h>](https://www.programiz.com/c-programming/library-function/string.h) | String handling functions |
| <time.h> | Date time functions |

### User-defined function

You can also create functions as per your need. Such functions created by the user are known as user-defined functions.

## How user-defined function works?

#include <stdio.h>

void functionName()

{

... .. ...

... .. ...

}

int main()

{

... .. ...

... .. ...

functionName();

... .. ...

... .. ...

}

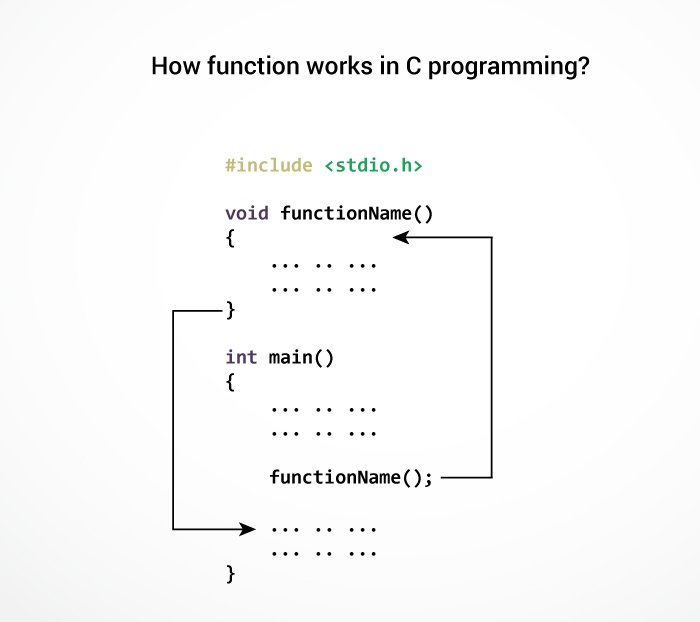
The execution of a C program begins from the main() function.

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.

Working of C Function

Note, function names are identifiers and should be unique.

This is just an overview of user-defined functions. Visit these pages to learn more on:

* [User-defined Function in C programming](https://www.programiz.com/c-programming/c-user-defined-functions)
* [Types of user-defined Functions](https://www.programiz.com/c-programming/types-user-defined-functions)

**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.

# C User-defined functions

In this tutorial, you will learn to create user-defined functions in C programming with the help of an example.

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

C allows you to define functions according to your need. These functions are known as user-defined functions. For example:

Suppose, you need to create a circle and color it depending upon the radius and color. You can create two functions to solve this problem:

* createCircle() function
* color() function

## Example: User-defined function

Here is an example to add two integers. To perform this task, we have created an user-defined addNumbers().

#include <stdio.h>

int addNumbers(int a, int b); // function prototype

int main()

{

int n1,n2,sum;

printf("Enters two numbers: ");

scanf("%d %d",&n1,&n2);

sum = addNumbers(n1, n2); // function call

printf("sum = %d",sum);

return 0;

}

int addNumbers(int a, int b) // function definition

{

int result;

result = a+b;

return result; // return statement

}

## Function prototype

A function prototype is simply the declaration of a function that specifies function's name, parameters and return type. It doesn't contain function body.

A function prototype gives information to the compiler that the function may later be used in the program.

### Syntax of function prototype

returnType functionName(type1 argument1, type2 argument2, ...);

In the above example, int addNumbers(int a, int b); is the function prototype which provides the following information to the compiler:

1. name of the function is addNumbers()
2. return type of the function is int
3. two arguments of type int are passed to the function

The function prototype is not needed if the user-defined function is defined before the main() function.

## Calling a function

Control of the program is transferred to the user-defined function by calling it.

### Syntax of function call

functionName(argument1, argument2, ...);

In the above example, the function call is made using addNumbers(n1, n2); statement inside the main() function.

## Function definition

Function definition contains the block of code to perform a specific task. In our example, adding two numbers and returning it.

#### Syntax of function definition

returnType functionName(type1 argument1, type2 argument2, ...)

{

//body of the function

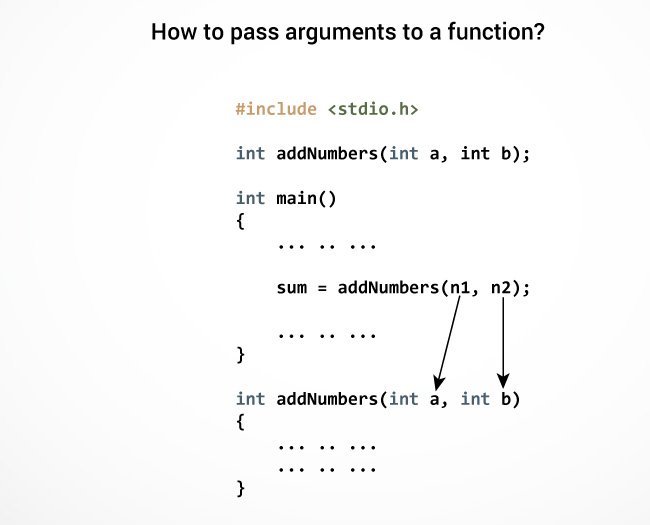
}

When a function is called, the control of the program is transferred to the function definition. And, the compiler starts executing the codes inside the body of a function.

## Passing arguments to a function

In programming, argument refers to the variable passed to the function. In the above example, two variables n1 and n2 are passed during the function call.

The parameters a and b accepts the passed arguments in the function definition. These arguments are called formal parameters of the function.

Passing Argument to Function

The type of arguments passed to a function and the formal parameters must match, otherwise, the compiler will throw an error.

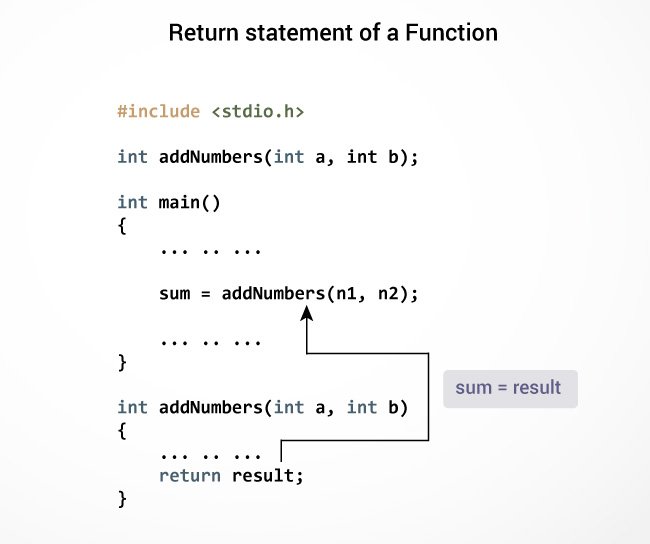
If n1 is of char type, a also should be of char type. If n2 is of float type, variable b also should be of float type.

A function can also be called without passing an argument.

## Return Statement

The return statement terminates the execution of a function and returns a value to the calling function. The program control is transferred to the calling function after the return statement.

In the above example, the value of the result variable is returned to the main function. The sum variable in the main() function is assigned this value.

Return Statement of Function

### Syntax of return statement

return (expression);

For example,

return a;

return (a+b);

The type of value returned from the function and the return type specified in the function prototype and function definition must match.

# Types of User-defined Functions in C Programming

In this tutorial, you will learn about different approaches you can take to solve the same problem using functions.

These 4 programs below check whether the integer entered by the user is a prime number or not.

The output of all these programs below is the same, and we have created a user-defined function in each example. However, the approach we have taken in each example is different.

## Example 1: No arguments passed and no return value

#include <stdio.h>

void checkPrimeNumber();

int main()

{

checkPrimeNumber(); // argument is not passed

return 0;

}

// return type is void meaning doesn't return any value

void checkPrimeNumber()

{

int n, i, flag = 0;

printf("Enter a positive integer: ");

scanf("%d",&n);

for(i=2; i <= n/2; ++i)

{

if(n%i == 0)

{

flag = 1;

}

}

if (flag == 1)

printf("%d is not a prime number.", n);

else

printf("%d is a prime number.", n);

}

The checkPrimeNumber() function takes input from the user, checks whether it is a prime number or not and displays it on the screen.

The empty parentheses in checkPrimeNumber(); statement inside the main() function indicates that no argument is passed to the function.

The return type of the function is void. Hence, no value is returned from the function.

## Example 2: No arguments passed but a return value

#include <stdio.h>

int getInteger();

int main()

{

int n, i, flag = 0;

// no argument is passed

n = getInteger();

for(i=2; i<=n/2; ++i)

{

if(n%i==0){

flag = 1;

break;

}

}

if (flag == 1)

printf("%d is not a prime number.", n);

else

printf("%d is a prime number.", n);

return 0;

}

// returns integer entered by the user

int getInteger()

{

int n;

printf("Enter a positive integer: ");

scanf("%d",&n);

return n;

}

The empty parentheses in the n = getInteger(); statement indicates that no argument is passed to the function. And, the value returned from the function is assigned to n.

## Example 3: Argument passed but no return value

#include <stdio.h>

void checkPrimeAndDisplay(int n);

int main()

{

int n;

printf("Enter a positive integer: ");

scanf("%d",&n);

// n is passed to the function

checkPrimeAndDisplay(n);

return 0;

}

// return type is void meaning doesn't return any value

void checkPrimeAndDisplay(int n)

{

int i, flag = 0;

for(i=2; i <= n/2; ++i)

{

if(n%i == 0){

flag = 1;

break;

}

}

if(flag == 1)

printf("%d is not a prime number.",n);

else

printf("%d is a prime number.", n);

}

The integer value entered by the user is passed to the checkPrimeAndDisplay() function.

Here, the checkPrimeAndDisplay() function checks whether the argument passed is a prime number or not and displays the appropriate message.

## Example 4: Argument passed and a return value

#include <stdio.h>

int checkPrimeNumber(int n);

int main()

{

int n, flag;

printf("Enter a positive integer: ");

scanf("%d",&n);

// n is passed to the checkPrimeNumber() function

// the returned value is assigned to the flag variable

flag = checkPrimeNumber(n);

if(flag == 1)

printf("%d is not a prime number",n);

else

printf("%d is a prime number",n);

return 0;

}

// int is returned from the function

int checkPrimeNumber(int n)

{

int i;

for(i=2; i <= n/2; ++i)

{

if(n%i == 0)

return 1;

}

return 0;

}

The input from the user is passed to the checkPrimeNumber() function.

The checkPrimeNumber() function checks whether the passed argument is prime or not.

If the passed argument is a prime number, the function returns 0. If the passed argument is a non-prime number, the function returns 1. The return value is assigned to the flag variable.

Depending on whether flag is 0 or 1, an appropriate message is printed from the main() function.

## Which approach is better?

Well, it depends on the problem you are trying to solve. In this case, passing argument and returning a value from the function (example 4) is better.

A function should perform a specific task. The checkPrimeNumber() function doesn't take input from the user nor it displays the appropriate message. It only checks whether a number is prime or not.

# C Recursion

In this tutorial, you will learn to write recursive functions in C programming with the help of an example.

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

### How recursion works?

void recurse()

{

... .. ...

recurse();

... .. ...

}

int main()

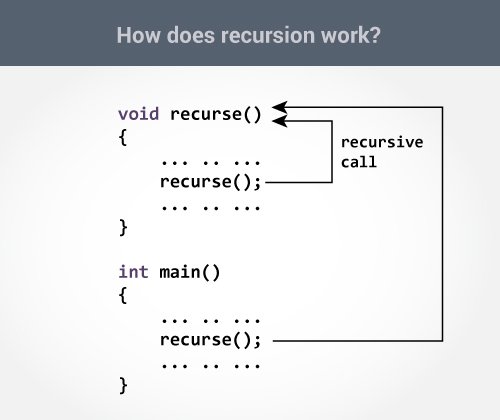
{

... .. ...

recurse();

... .. ...

}

Working of Recursion

The recursion continues until some condition is met to prevent it.

To prevent infinite recursion, [if...else statement](https://www.programiz.com/c-programming/c-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**

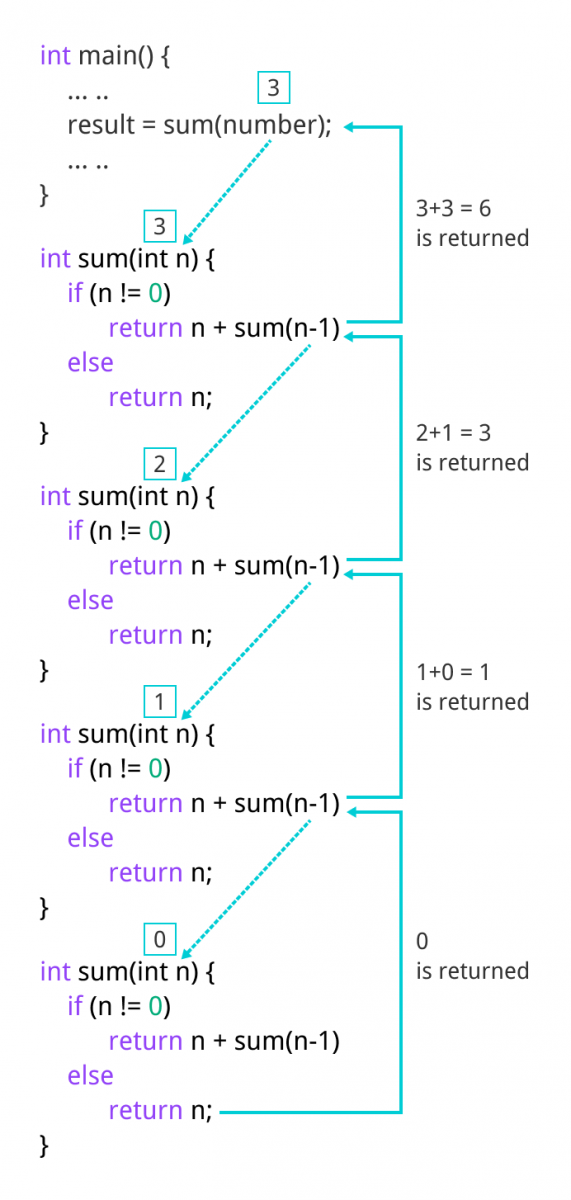
Enter a positive integer:3

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.

Sum of Natural Numbers

### Advantages and Disadvantages of Recursion

Recursion makes program elegant. However, if performance is vital, use loops instead as recursion is usually much slower.

That being said, recursion is an important concept. It is frequently used in [data structure and algorithms](https://www.programiz.com/dsa). For example, it is common to use recursion in problems such as tree traversal.

**C Storage Class**

In this tutorial, you will learn about scope and lifetime of local and global variables. Also, you will learn about static and register variables.

Every variable in C programming has two properties: type and storage class.

Type refers to the data type of a variable. And, storage class determines the scope, visibility and lifetime of a variable.

There are 4 types of storage class:

1. automatic
2. external
3. static
4. register

**Local Variable**

The variables declared inside a block are automatic or local variables. The local variables exist only inside the block in which it is declared.

Let's take an example.

#include <stdio.h>

int main(void) {

for (int i = 0; i < 5; ++i) {

printf("C programming");

}

// Error: i is not declared at this point

printf("%d", i);

return 0;

}

When you run the above program, you will get an error undeclared identifier i. It's because i is declared inside the for loop block. Outside of the block, it's undeclared.

Let's take another example.

int main() {

int n1; // n1 is a local variable to main()

}

void func() {

int n2; // n2 is a local variable to func()

}

In the above example, n1 is local to main() and n2 is local to func().

This means you cannot access the n1 variable inside func() as it only exists inside main(). Similarly, you cannot access the n2 variable inside main() as it only exists inside func().

**Global Variable**

Variables that are declared outside of all functions are known as external or global variables. They are accessible from any function inside the program.

### Example 1: Global Variable

#include <stdio.h>

void display();

int n = 5; // global variable

int main()

{

++n;

display();

return 0;

}

void display()

{

++n;

printf("n = %d", n);

}

**Output**

n = 7

Suppose, a global variable is declared in file1. If you try to use that variable in a different file file2, the compiler will complain. To solve this problem, keyword extern is used in file2 to indicate that the external variable is declared in another file.

## Register Variable

The register keyword is used to declare register variables. Register variables were supposed to be faster than local variables.

However, modern compilers are very good at code optimization, and there is a rare chance that using register variables will make your program faster.

Unless you are working on embedded systems where you know how to optimize code for the given application, there is no use of register variables.

## Static Variable

A static variable is declared by using the static keyword. For example;

static int i;

The value of a static variable persists until the end of the program.

### Example 2: Static Variable

#include <stdio.h>

void display();

int main()

{

display();

display();

}

void display()

{

static int c = 1;

c += 5;

printf("%d ",c);

}

**Output**

6 11

During the first function call, the value of c is initialized to 1. Its value is increased by 5. Now, the value of c is 6, which is printed on the screen.

During the second function call, c is not initialized to 1 again. It's because c is a static variable. The value c is increased by 5. Now, its value will be 11, which is printed on the screen.