C Programming Functions

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

Suppose, a program related to graphics needs to create a circle and color it depending upon the radius and color from the user. You can create two functions to solve this problem:

- create a circle function
- color function

Dividing complex problem into small components makes program easy to understand and use.

Types of function

Depending on whether a function is defined by the user or already included in C compilers, there are two types of functions in C programming

There are two types of function in C programming:

- Standard library functions
- User defined functions

Standard library functions

The standard library functions are built-in functions in C programming to handle tasks such as mathematical computations, I/O processing, string handling etc.

These functions are defined in the header file. When you include the header file, these functions are available for use. 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 `"stdio.h"` header file.

There are other numerous library functions defined under `"stdio.h"`, such as `scanf()`, `fprintf()`, `getchar()` etc. Once you include `"stdio.h"` in your program, all these functions are available for use.

C Library Functions Under Different Header File

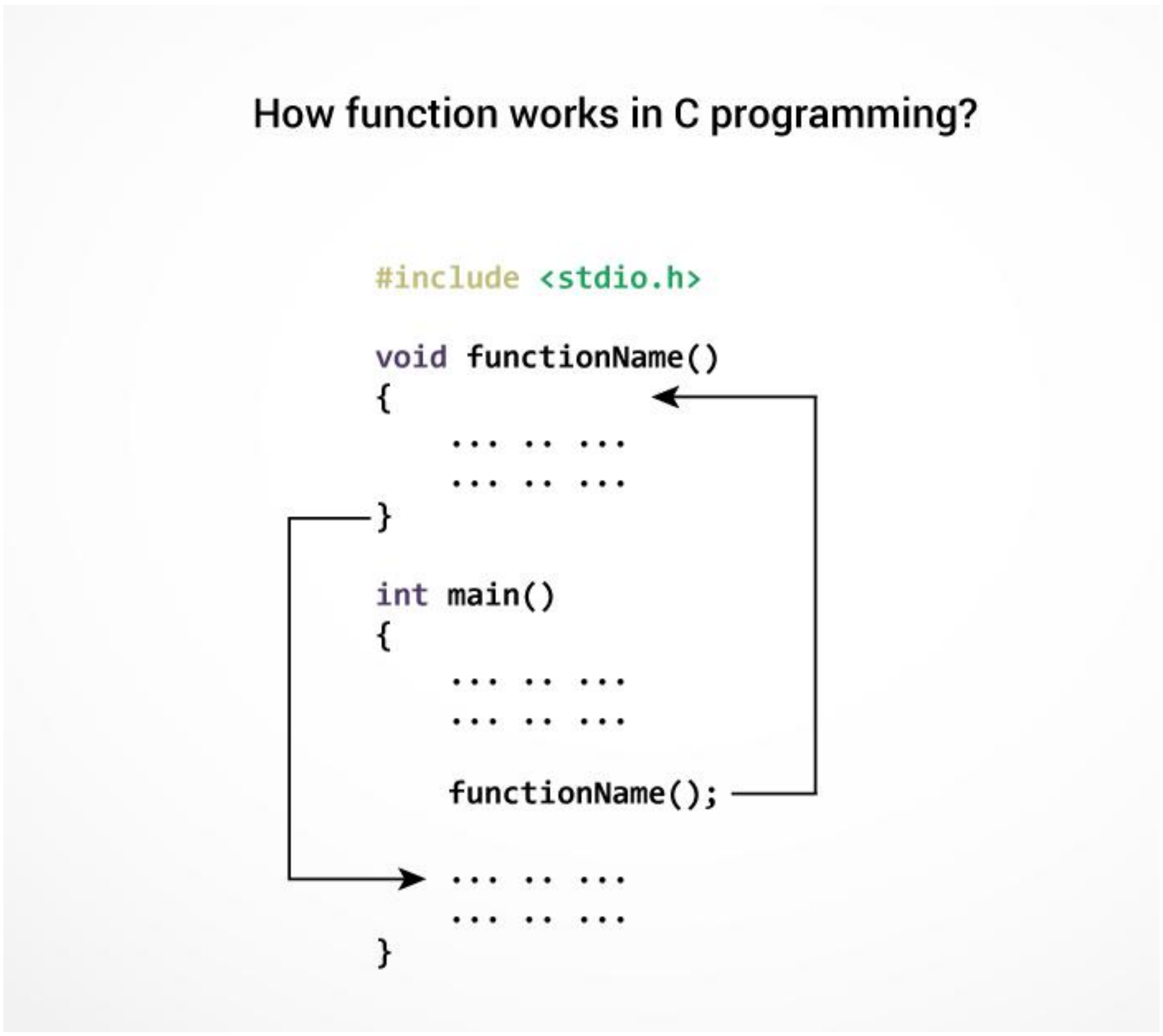
C Header Files	
<assert.h>	Program assertion functions
<ctype.h>	Character type functions
<locale.h>	Localization functions
<math.h>	Mathematics functions
<setjmp.h>	Jump functions
<signal.h>	Signal handling functions
<stdarg.h>	Variable arguments handling functions
<stdio.h>	Standard Input/Output functions
<stdlib.h>	Standard Utility functions
<string.h>	String handling functions
<time.h>	Date time functions

User-defined function

As mentioned earlier, C allow programmers to define functions. Such functions created by the user are called user-defined functions.

You can create as many user-defined functions as you want.

How user-defined function works?



```
#include <stdio.h>

void functionName()
{
    ... .. ...
    ... .. ...
}

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

    functionName();

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

**Remember, function name is an identifier 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.

C User-defined functions ----ud.c

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

Function definition

Function definition contains the block of code to perform a specific task i.e. in this case, adding two numbers and returning it.

Syntax of function definition

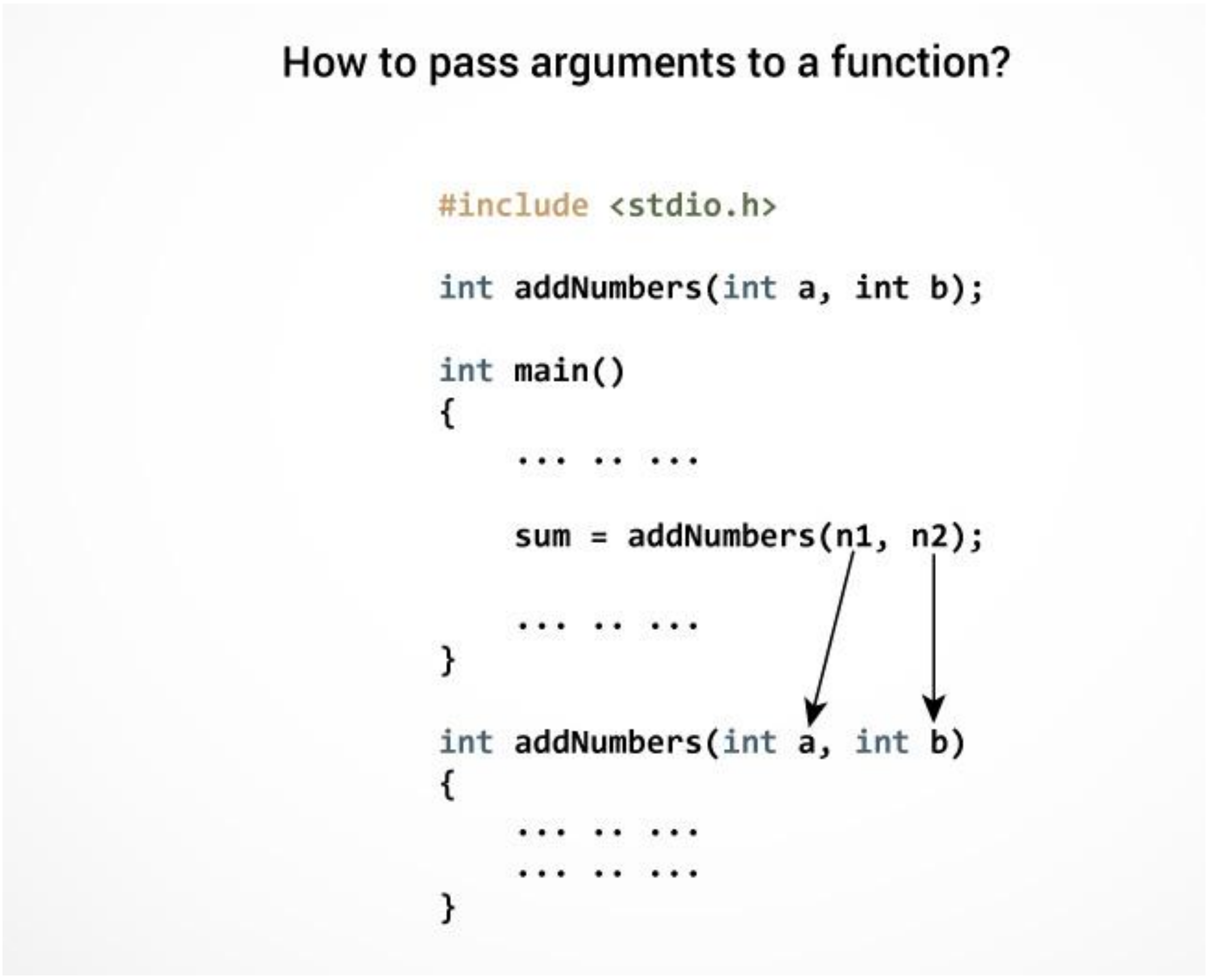
```
returnType functionName(type1 argument1, type2 argument2, ...)

{

    //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 function call.

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



```
#include <stdio.h>

int addNumbers(int a, int b);

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

    sum = addNumbers(n1, n2);

    ... .. ...
}

int addNumbers(int a, int b)
{
    ... .. ...
    ... .. ...
}
```

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

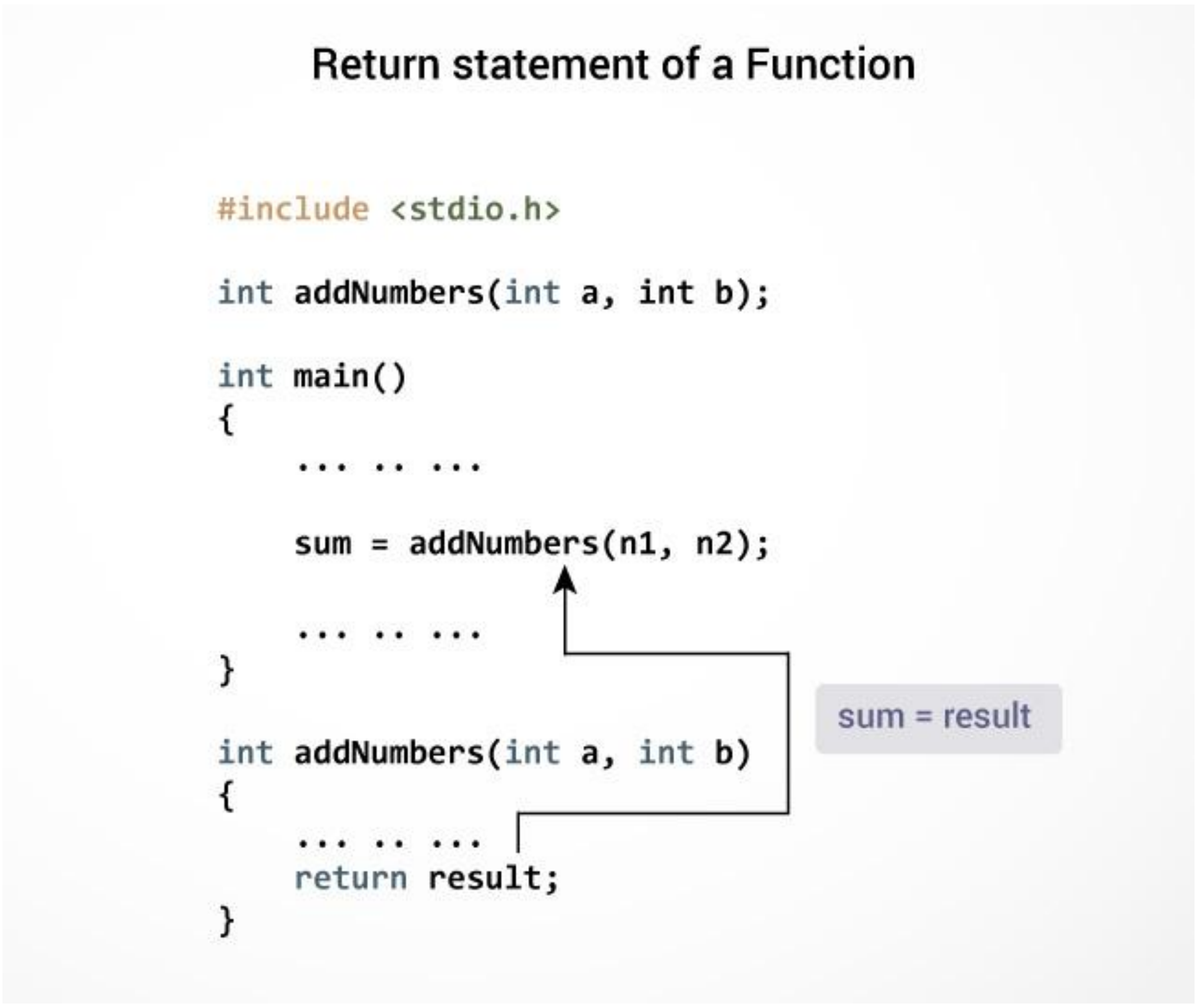
If `n1` is of char type, `a` also should be of char type. If `n2` is of float type, variable `b` also should be of float type.

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 return statement.

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



```
#include <stdio.h>

int addNumbers(int a, int b);

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

    sum = addNumbers(n1, n2);

    ... .. ...
}

int addNumbers(int a, int b)
{
    ... .. ...
    return result;
}
```

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 function prototype and function definition must match.

Types of User-defined Functions in C Programming

No arguments passed and no return Value NANR

No arguments passed but a return value NAYR

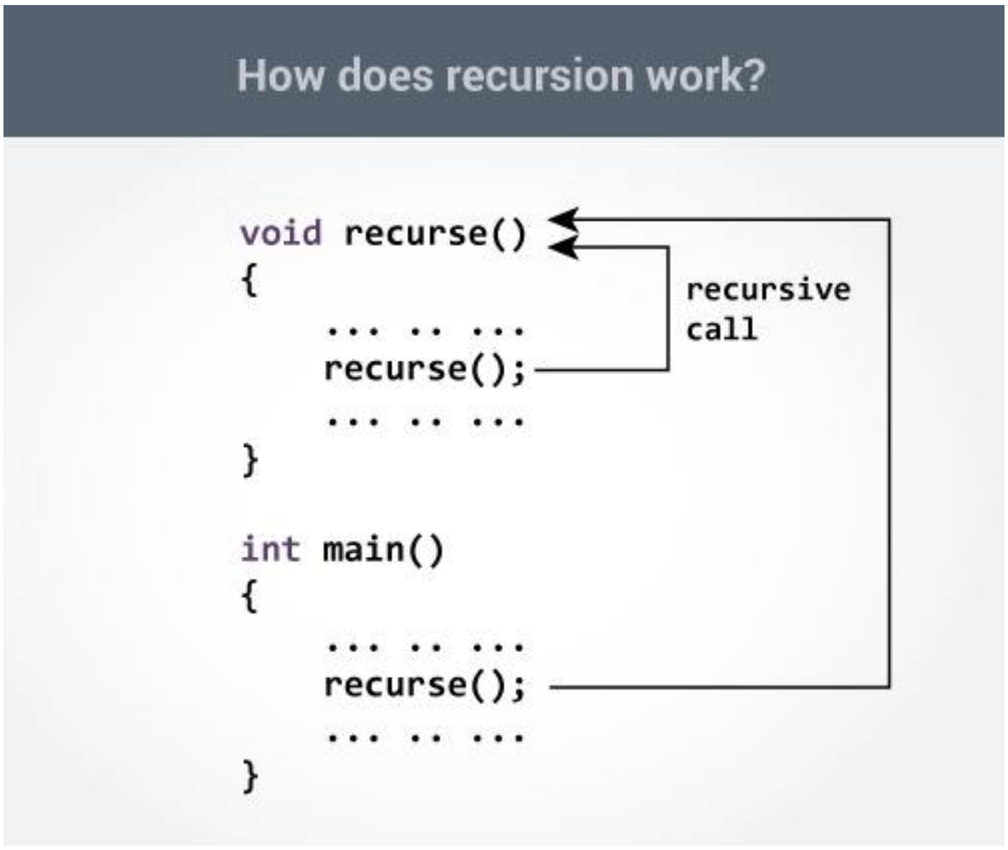
Argument passed but no return value YANR

Argument passed and a return value YAYR

Which approach is better?

C Recursion

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



```
void recurse()
{
    ... .. ...
    recurse();
    ... .. ...
}

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

Sum of Natural Numbers Using Recursion ----rec.c

Scope and Lifetime of a variable

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 the function are automatic or local variables.

The local variables exist only inside the function in which it is declared. When the function exits, the local variables are destroyed.

```
int main() {

    int n; // n is a local variable to main() function

    ... .. ...

}



void func() {

   int n1; // n1 is local to func() function

}
```

In the above code, `n1` is destroyed when `func()` exits. Likewise, `n` gets destroyed when `main()` exits.

Global Variable global.c

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

The extern Storage Class

he **extern** storage class is used to give a reference of a global variable that is visible to ALL the program files. When you use 'extern', the variable cannot be initialized however, it points the variable name at a storage location that has been previously defined.

When you have multiple files and you define a global variable or function, which will also be used in other files, then *extern* will be used in another file to provide the reference of defined variable or function. Just for understanding, *extern* is used to declare a global variable or function in another file.

The extern modifier is most commonly used when there are two or more files sharing the same global variables or functions

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 stsic.c

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

```
static int i;
```

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