Programing Methodology in C

Lecture 9 – Functions in C programing Language

By Elubu Joseph

josebulinga@gmail.com

Functions in C

Agenda

1. Introduction to Functions
    i. Category of functions
    ii. Types of Functions
    iii. Types of Function calls

Functions in C

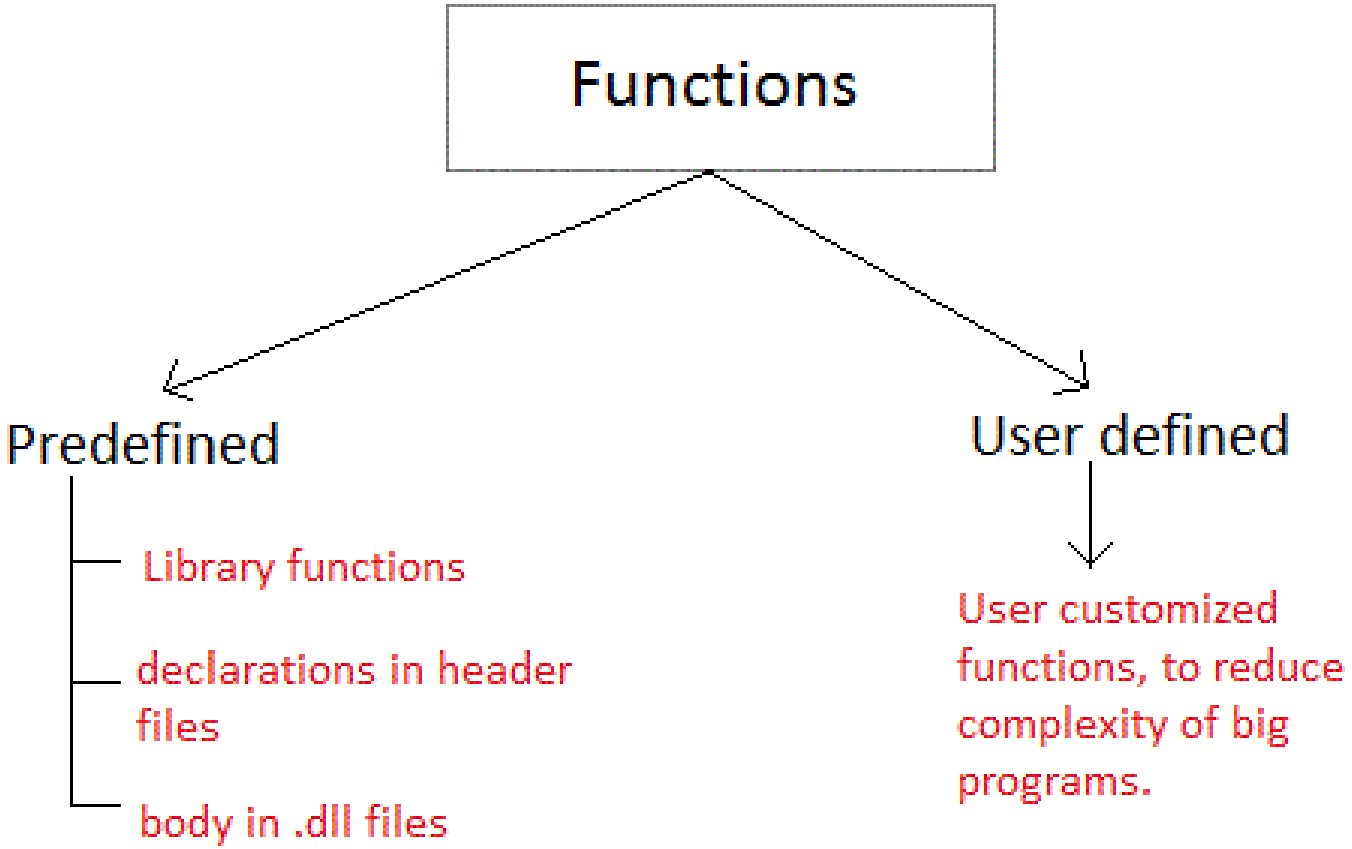
Introduction to Functions in C Language

A **function** is a block of code that performs a particular task.

There are many situations where we might need to write same line of code for more than once in a program. This may lead to unnecessary repetition of code, bugs and even becomes boring for the programmer.

So, C language provides an approach in which a programmer can declare and define a group of statements once in the form of a function and it can be called and used whenever required. These functions defined by the user are also know as **User-defined Functions**

Categories of functions in C



C functions can be classified into two categories,

1. **Predefined functions**
2. **User-defined functions**

Library functions Vs User-defined functions

Library functions are those functions which are already defined in C library, example printf(), scanf(), strcat(), strcopy(), strcomp() etc. You just need to include appropriate header files to use these functions. These are already declared and defined in C libraries.

A **User-defined functions** on the other hand, are those functions which are defined by the user at the time of writing program. These functions are made for code reusability and for saving time and space.

Benefits of Using Functions

1.It provides modularity to your program's structure.

2.It makes your code reusable. You just have to call the function by its name to use it, wherever required.

3.In case of large programs with thousands of code lines, debugging and editing becomes easier if you use functions.

4.It makes the program more readable and easy to understand.

Parts of the Function

Function declaration consists of 4 parts.

1. returntype
2. function name
3. parameter list
4. terminating semicolon

returntype

When a function is declared to perform some sort of calculation or any operation and is expected to return some value, in such cases, a return statement is added at the end of function body.

Return type specifies the type of value(int, float, char, double) that function is expected to return to the program which called the function.

Note: In case a function doesn't return any value, the return type would be void.

functionName

Function name is an identifier and it specifies the name of the function. The function name is any valid C identifier and therefore must follow the same naming rules like other variables in C language.

parameter list

The parameter list declares the type and number of arguments that the function expects when it is called. Also, the parameters in the parameter list receives the argument values when the function is called. They are often referred as **formal parameters**.

functionbody

The function body contains the declarations and the statements(algorithm) necessary for performing the required task. The body is enclosed within curly braces { ... } and consists of three parts.

1. **local** variable declaration(if required).
2. **function statements** to perform the task inside the function.
3. a **return** statement to return the result evaluated by the function(if return type is void, then no return statement is required).

Function Declaration/ Prototyping

Like any variable or an array, a function must also be declared before its used. Function declaration informs the compiler about the function name, parameters to accept, and its return type. The actual body of the function can be defined separately. It's also called as **Function Prototyping**.

General syntax for function declaration is

```
returntype functionName(type1 parameter1, type2 parameter2,...);

int calc(int a, int b);
```

Function definition Syntax

Just like in the example above, the general syntax of function definition is,

```
returntype functionName(type1 parameter1, type2 parameter2,...)
 {
 // function body goes here
}
```

The first line *returntype* **functionName(type1 parameter1, type2 parameter2,...)** is known as **function header** and the statement(s) within curly braces is called **function body**.

Note: While defining a function, there is no semicolon(;) after the parenthesis in the function header, unlike while declaring the function or calling the function.

Calling a function

When a function is called, control of the program gets transferred to the function.

```
Int main(){

functionName(argument1, argument2,...);

}
```

Passing Arguments to a function

Arguments are the values specified during the function call, for which the formal parameters are declared while defining the function.

Sample Function call Code

```
#include<stdio.h>
void sampleFunction(); // function declaration
int main() {
        printf("I am the main Function here...\n");
        sampleFunction(); //function call
    return 5;
}
void sampleFunction(){ // function definition
        printf("\nI am Sample function..\n");   }
```

Output:

I am the main Function here...

I am Sample function..

Types of User-defined Functions in C

There can be 4 different types of user-defined functions, they are:

1.Function with no arguments and no return value

2.Function with no arguments and a return value

3.Function with arguments and no return value

4.Function with arguments and a return value

Functions

With Arguments

- declared and defined with parameter list
- values for parameter passed during call
- Eg :

```
// declaration
int sum (int x, int y);
//call
sum(10, 20);
```

Without Arguments

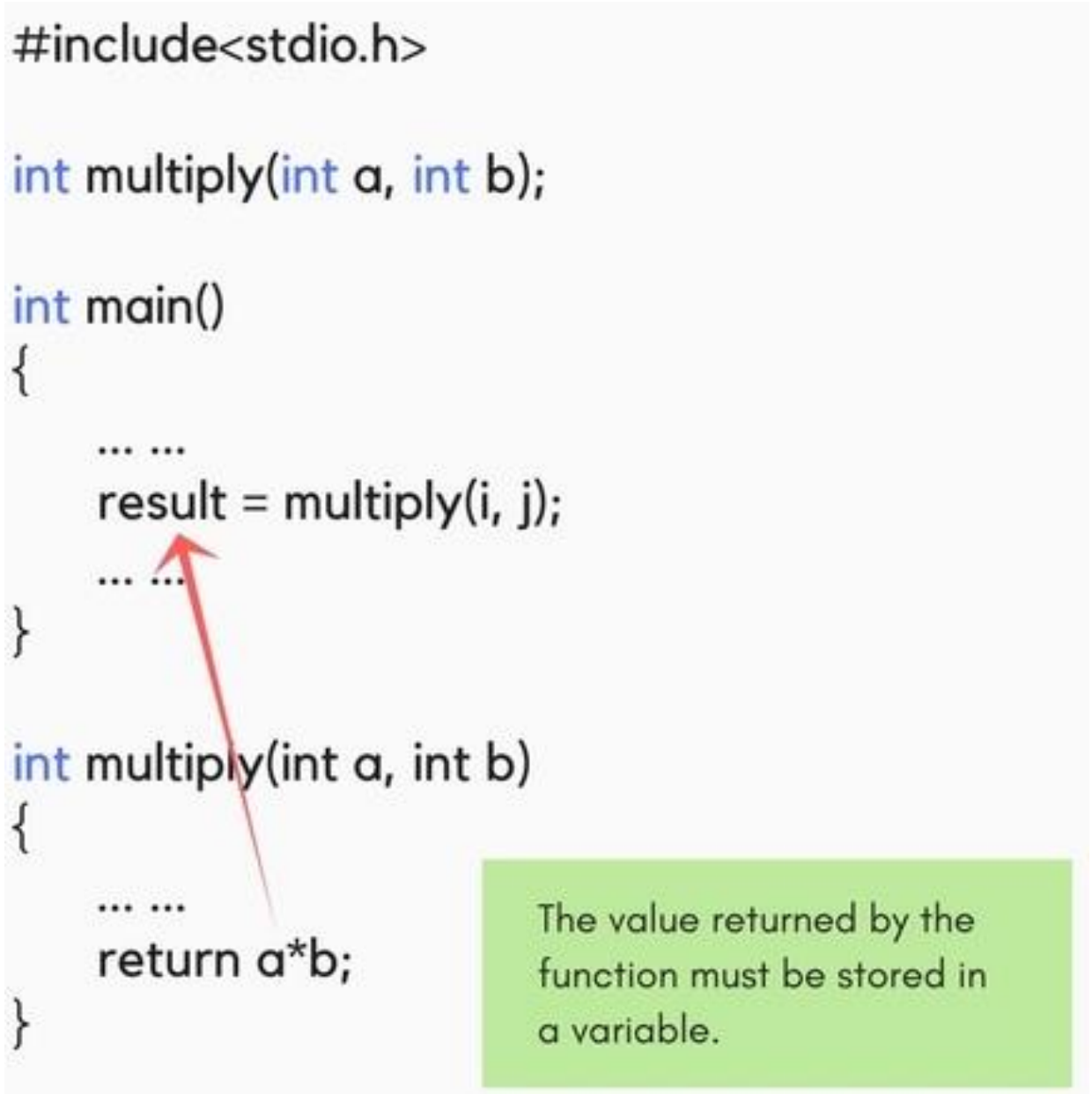
- No parameters included.
- No value passed during function call
- Eg :

```
// declaration
int display();
// call
display();
```

Returning a value from function

A function may or may not return a result. But if it does, we must use the return statement to output the result.

return statement also ends the function execution, hence it must be the last statement of any function. If you write any statement after the return statement, it won't be executed.



The datatype of the value returned using the return statement should be same as the return type mentioned at function declaration and definition. If any of it mismatches, you will get compilation error.

Function with no arguments and no return value

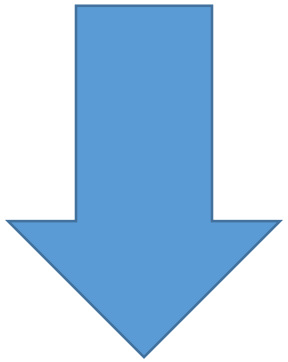
Such functions can either be used to display information or used to completely dependent on user inputs.

Below is an example of a function, which takes 2 numbers as input from user, and display which is the greater number.

```c
#include<stdio.h>
void greatNum(); // function declaration
int main() {
greatNum(); // function call
return 0;
}
void greatNum() /*function definition*/ {
	int i, j;
	printf("Enter 2 numbers that you want to compare...");
	scanf("%d%d", &i, &j);
	if(i > j) {
		printf("The greater number is: %d", i);
	}
	else {
	printf("The greater number is: %d", j);
	}
}
```

Function with no arguments but with a return value +

The program below is a modification of the above example to make the function greatNum() return the number which is greater amongst the 2 input numbers.



```c
#include<stdio.h>
int greatNum(); // function declaration
int main() {
	int result; result = greatNum(); // function call
	printf("The greater number is: %d", result);
	return 0;
}
int greatNum(){// function definition
	int i, j, greaterNum;
	printf("Enter 2 numbers that you want to compare...");
	scanf("%d%d", &i, &j);
	if(i > j) {
		greaterNum = i;
	}
	else {
		greaterNum = j;
	} // returning the result
	return greaterNum;
}
```

Function with arguments and no return value

We are using the same function as example again and again, to demonstrate that to solve a problem there can be many different ways.

This time, we have modified the above example to make the function greatNum() take two int values as arguments, but it will not be returning anything.

Function with arguments and no return value

```c
#include<stdio.h>
void greatNum(int a, int b); // function declaration
int main(){
	int i, j;
	printf("Enter 2 numbers that you want to compare...");
	scanf("%d%d", &i, &j); greatNum(i, j); // function call
	return 0;
}
void greatNum(int x, int y) // function definition {
	if(x > y) {
	printf("The greater number is: %d", x);
	}
	 else {
		printf("The greater number is: %d", y);
	}
}
```

Function with arguments and a return value

This is the best type, as this makes the function completely independent of inputs and outputs, and only the logic is defined inside the function body.

Function with arguments and a return value

```c
#include<stdio.h>
int greatNum(int a, int b); // function declaration
int main() {
	int i, j, result;
	printf("Enter 2 numbers that you want to compare...");
	 scanf("%d\n%d", &i, &j);
	result = greatNum(i, j); // function call
	printf("The greater number is: %d", result);
	return 0;
}
int greatNum(int x, int y){// function definition
	if(x > y) {
		return x;
	} else {
		return y;
	}
}
```

Example

Let's write a simple program with a main() function, and a user defined function to multiply two numbers, which will be called from the main() function.

```
#include<stdio.h>
int multiply(int a, int b); // function declaration
int main() {
       int i, j, result;
       printf("Please enter 2 numbers you want to multiply...");
       scanf("%d%d", &i, &j);
      result = multiply(i, j); // function call
      printf("The result of muliplication is: %d", result);
       return 0;
}
int multiply(int a, int b) {
return (a*b); // function definition, this can be done in one line
}
```

Nesting of Functions

C language also allows nesting of functions i.e to use/call one function inside another function's body. We must be careful while using nested functions, because it may lead to infinite nesting.

```
function1() {
// function1 body here
function2(); // function1 body
here
}
```

Nesting of Functions

If function2() also has a call for function1() inside it, then in that case, it will lead to an infinite nesting. They will keep calling each other and the program will never terminate.

Not able to understand? Lets consider that inside the main() function, function1() is called and its execution starts, then inside function1(), we have a call for function2(), so the control of program will go to the function2().

But as function2() also has a call to function1() in its body, it will call function1(), which will again call function2(), and this will go on for infinite times, until you forcefully exit from program execution.

Nested Functions

```
#include<stdio.h>
void function1();
int function2(); // function declaration
int main() {
		printf("I am the main function...\n\n");
		function1(); // function call
		return 0;
}
void function1(){
   printf("I am Function One...\n\n");
   function2(); //function Call
}
int function2(){
   printf("I am Function Two...\n\n");
   function1();
return 1;
}
```

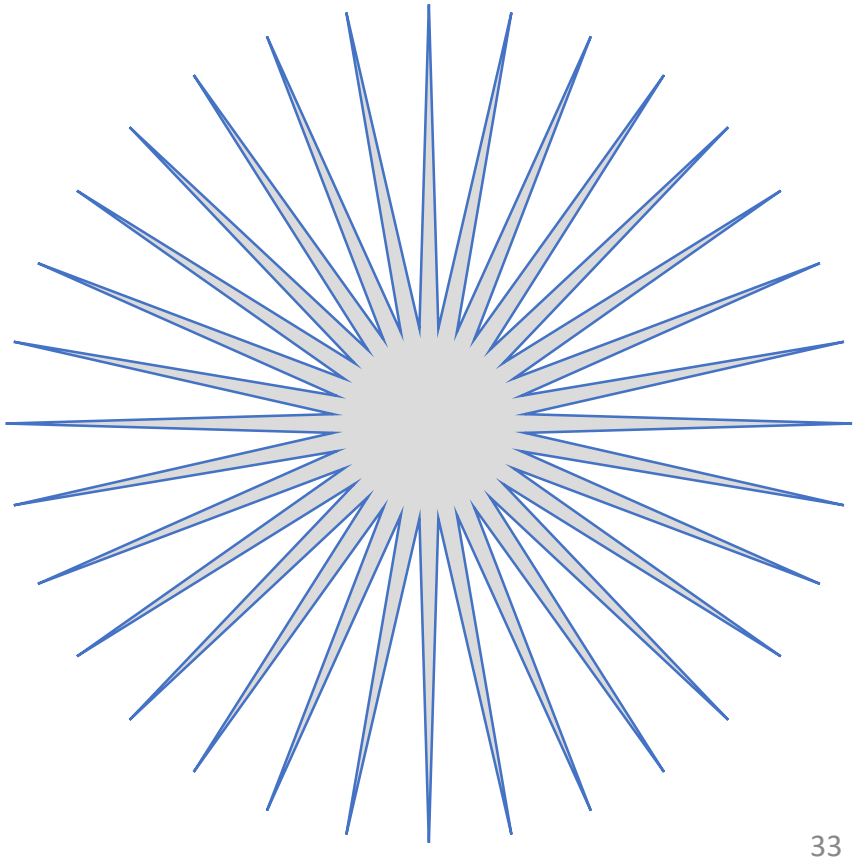
Summary

In summary therefore, we looked into; -

1. Introduction to Functions
2. Types of Functions

Types of Function calls next lecture

Thank you for you attention



References

Functions in C. Studytonight.com. (n.d.). Retrieved November 4, 2021, from https://www.studytonight.com/c/user-defined-functions-in-c.php.