9. Functions

- Function is a block of code used to perform a specific task
- Every C program has at least one function **main()**.
- Functions can call other functions and/or itself.
- Functions improve **reusability** and **readability**.
    - Using functions, large programs can be divided into smaller modules
    - Functions can be called any number of times

Types of functions in C programming:

- **Standard library functions**
    - Examples: printf(), scanf(), pow(), sqrt(), ...
    - Declared in header files (stdio.h, math.h, ...)
- **User defined** functions - Created by the user

Every function has:

- Function **declaration** (function **prototype**)
- Function **definition**
- Function **call**

Function Definition

- Contains the block of code required to perform a specific task
- Consists of a **function header** and a **function body**

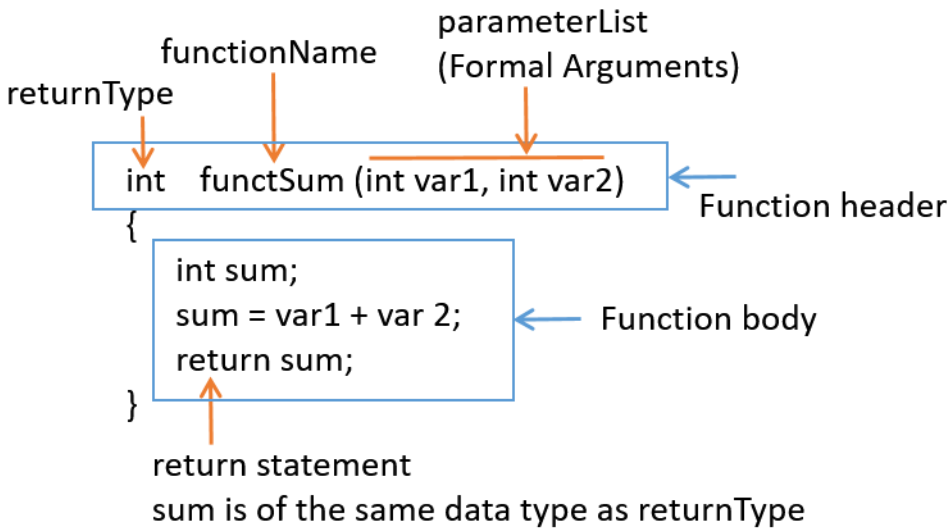
Syntax:

```
returnType    functionName ( parameterList )
{
    body of the function
}
```

where:

- **returnType -** Data type of the value the function returns.
    - If function does not return a value, the returnType is **void**.
- **functionName** - Function name is an identifier and must be unique.
- **parameterList –** Comma-separated list of parameters (their types and names) also known as formal parameters (formal arguments). Parameter list is optional.
    - Example: int a, int b, float c
- **body of the function –** Contains all statements to be executed whenever a function call is made.
    - Includes return statement if function's returnType is not void. Return value must be of the same data type as returnType.
    - Function does not return a value if returnType is void.

Example:



Function Declaration/Prototype

- Declaration of a function
- Should be identical to **function header** with the exception of semicolon at the end
- Informs the compiler about a function's name, return type, and parameters.
- If the function definition is written after main then we need function declaration (prototype) before main. If the function definition is written above main, there is no need to have function prototype.

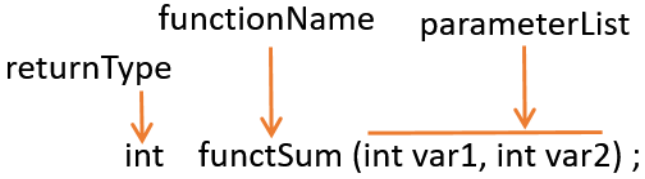
Syntax:

returnType functionName (parameterList);

where:

- **returnType** – Data type of the value the function returns. If function does not return a value, the returnType is **void**.
- **functionName** - Function name is an identifier and must be unique.
- **parameterList** – Defines the data type, order, and number of parameters.
    - Parameter list is optional
    - Parameter names can be avoided in function prototype, only their data types are required (example: int funcSum(int, int);)

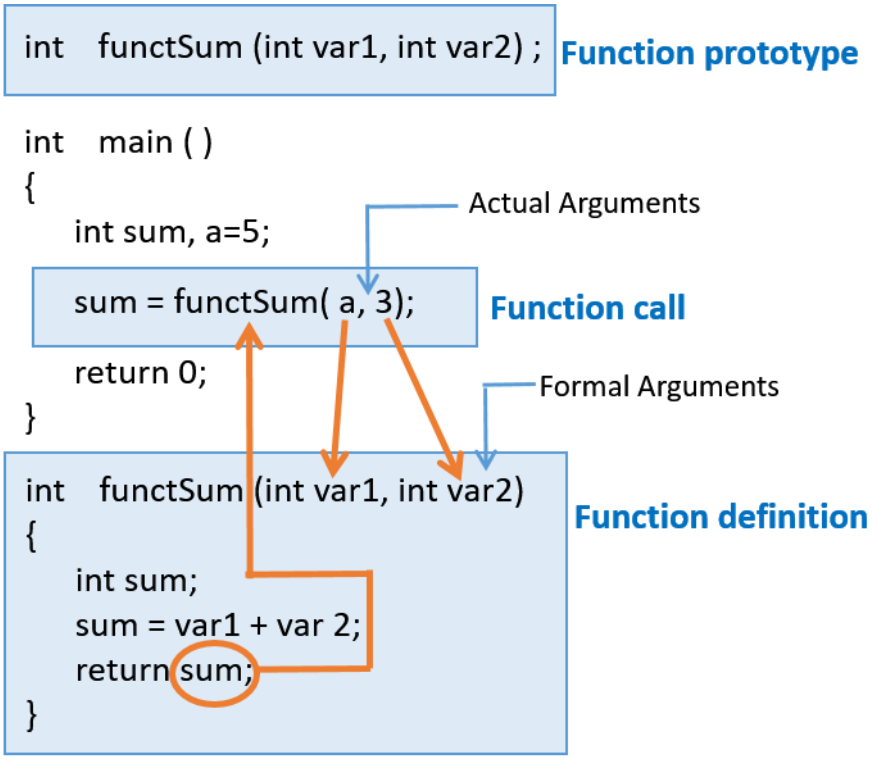
Examples:



Function Call

- When a function is called:
  - The control of the program is transferred to the function definition.
  - The function name must match exactly the name of the function in the function prototype and function header.
  - The number and data type of parameters "passed" to the function must match the number and data type of parameters from the function prototype's parameter list and function header's parameter list.
  - If a function returns a value, the function call must be assigned to a variable in order to capture returned value. The data type of a variable used to store returned value must match the function return type.

Example:



Types of functions

Function Type (based on data flow between functions)	**Syntax**

Accepting parameters
Returning value
	Function definition:
	`returnType funcName( type1 par1, type2 par2, ... )`
	`{`
	`  returnType result;`
	`  statements;`
	`  return result;`
	`}`

	Function prototype: `returnType funcName( type1, type2, ... );`

	Function call:
	`returnType var;`
	`type1 actual_param1;`
	`type2 actual_param2;`
	`...`
	`var = funcName (actual_param1, actual_param2, ... );`

Accepting parameters
Not returning value
	Function definition:
	`void funcName( type1 par1, type2 par2, ... )`
	`{`
	`  statements;`
	`}`

	Function prototype: `void funcName( type1, type2, ... );`

	Function call:
	`type1 actual_param1;`
	`type2 actual_param2;`
	`...`
	`funcName (actual_param1, actual_param2, ... );`

Not accepting parameter
Not returning value
	Function definition:
	`void funcName( )`
	`{`
	`  statements;`
	`}`

	Function prototype: `void funcName( );`
	Function call: `funcName ( );`

Not Accepting parameter
Returning value
	Function definition:
	`returnType funcName( )`
	`{`
	`  returnType result;`
	`  statements;`
	`  return result;`
	`}`

	Function prototype: `returnType funcName( );`

	Function call:
	`returnType var;`
	`var = funcName ( );`

Functions vs. Macros ("Differences between macros and functions in C Programming")

Macro	Function
Macro is preprocessed	Function is compiled
No type checking	Type checking is done
Speed of execution is faster	Speed of execution is slower
Useful where small code appears many time	Useful where large code appears many time
Macro does not check for compile errors	Function checks for compile errors

Macros are created using the #define preprocessor directive.

Examples:

```
#define COLLEGE printf("*****SENECA COLLEGE*****")
#define CUBE(x) x*x*x
```

References

- Tan, H.H., and T.B. D'Orazio. *C Programming for Engineering & Computer Science*. USA: WCB McGRaw-Hill. 1999. Print.
- "C working of function - C Programming." *Learn Programming Language Step By Step*. N.p., 24 Nov. 2013. Web. 28 Mar. 2017. <http://www.c4learn.com/c-programming/c-working-of-function/>.
- Tutorialspoint.com. "C Functions." *Www.tutorialspoint.com*. N.p., n.d. Web. 28 Mar. 2017. <https://www.tutorialspoint.com/cprogramming/c_functions.htm>.
- "C – Argument, return value." *Fresh2refresh.com*. N.p., n.d. Web. 28 Mar. 2017. <http://fresh2refresh.com/c-programming/c-function/c-function-arguments-and-return-values/>.
- "Difference between macro and function in C Programming - C Programming." *Learn Programming Language Step By Step*. N.p., 15 Nov. 2013. Web. 12 Apr. 2017. <http://www.c4learn.com/c-programming/difference-between-macro-and-function/>.