This course has already ended.

« 3 Data types and Variables Course materials 5 Round feedback »

ELEC-A7100 / 1. Basics / 4 Functions

First round tasks as a `ZIP file.`

Functions¶

The C programs are organized in functions. A function contains a single logical part of the program, and it can be called from other functions of the program (or from within the function itself). Use of functions enables reusing code: in a well designed program any particular part of program logic only needs to be implemented once in a single function, that is then called from other parts of the code.

Function has four main components: **name**, **return value**, **argument declarations**, and **body of the function**. Below is an example of a simple function definition of function 'square' that multiplies argument 'base' by itself and returns it as the result of the function (lines 1-5). Below the 'square' function definition there is the main function that has three calls to the square function (lines 9-11).

```
1  #include <stdio.h>
2
3  int square(int base)
4  {
5      int res = base * base;
6      return res;
7  }
8
9  int main(void)
10 {
11     int val = square(3);  // val becomes 9
12     int val2 = square(val * 2);  // val2 becomes 18 * 18
13     int val3 = square(square(val));  // val3 becomes (9*9) * (9*9)
14
15     printf("val2: %d  val3: %d\n", val2, val3);
16     return 0;
17 }
```

Line 1 above starts with the data type of the function return value (here ***'int'***) that can be one of the data types introduced above (there are also other data types that can be used, but more about those later). Then comes the function name, ***'square'***. The function arguments are listed inside parenthesis. There can be multiple arguments separated with comma, but here we only have one parameter, ***'base'*** that has int data type. Each argument must have data type and a name. It is also possible that function does not have any arguments. In such case void is used to represent an empty argument list. This is the case with the main function on line 7).

After the function return value type, name, and arguments is the definition of the ***'function body'***, inside curly brackets. The ***'square'*** function body is on lines 2 to 6, and the 'main' function body is on lines 8 to 17. As discussed earlier, the program execution always starts from the main function, and each function body consists of one or more statements that make the program logic.

The function arguments work like any other local variables inside the function implementation. This can be seen in the ***'square'*** function body, where the ***'base'*** argument is used in the expression that multiplies the given argument by itself, and stores the result to variable named ***'res'***.

The result of the function is indicated by the return statement (on line 4 of the ***'square'*** function). When program encounters the return statement, it exits the function, and returns to the point of the code from where the function was called. At the same time, value of the expression following the return statement is passed to the caller of the function. For example, when the main function calls the ***'square'*** function for the first time on line 9, the value of variable ***'val'*** is set based on the return result of the function. In this case it will be 9. The return statement can be in any part of the function definition, and there can be multiple return statements in single function definition.

The main function calls the ***'square'*** function repeatedly three times with different parameters. As can be seen, the parameter can be any expression, not just a constant value. In such case, the expression is evaluated first, and when the result is known, it is passed to the function as an argument. Because a function call can be as part of any expression, a function call can contain another function call as a parameter, as can be seen on line 11. In such case the inner function result is evaluated first, and the result is passed as argument to the outer function. In this case we happen to use the ***'square'*** function itself as parameter. What will be the value of 'val3' at the end of the program execution?

It is important to note that the local variables declared inside a function definition are only visible inside the function (or more generally, inside the block statement marked with curly brackets). Therefore variable ***'res'*** declared in function ***'square'*** cannot be used in the main function, nor can variable ***'val'*** be used inside the ***'square'*** function. The only way to pass information between the functions is by using the arguments, or the return value. (It should be noted, though, that the C language allows declaring variables outside the functions, in which case they are global, and visible to all functions. Use of global variables is discouraged, however, unless there is a very good and well-justified reason for that).

Function does not need to have any return value. In such case ***void*** is used in place of the return type on the function definition. In such function, the return statement does not provide any value, and can be omitted from the function. Functions can have several return statements, to force early exit from the function under some given conditions, and in such case return statement can be useful also when there is no return value for the function.

Task 1.3: PowerThree¶

Add a new function namely ***powerthree*** to the above program which calculates cube of an integer. After ***powerthree*** function implementation, declare a new variable ***val4*** where you place the value of powerthree (13). Use the following print statement format to print the new value ***val4*** along with the already values that are being printed in the main function.

```
"val2: %d  val3: %d  val4: %d\n"
```

Note that there needs to be two spaces between the values and names.

Points 10 / 10 My submissions 1 Deadline Tuesday, 8 June 2021, 19:59 To be submitted alone

This course has been archived (Saturday, 31 December 2022, 20:00).

Power Three

Select your files for grading

main.c

Choose File No file chosen

Submit

Task 1.4: Fraction Sum¶

Define function **fracsum** that calculates the sum of two fraction numbers, and gets four integers as parameter in the following way: the first two parameters are the numerator and denominator of the first number, and the last two parameters are the numerator and denominator of the second number. The function returns sum of these numbers as a floating point number.

Points 15 / 15 My submissions 1 Deadline Tuesday, 8 June 2021, 19:59 To be submitted alone

This course has been archived (Saturday, 31 December 2022, 20:00).

Fraction Sum

Select your files for grading

source.c

Choose File No file chosen

Submit

Task 1.5: Own Program¶

The objective of this task is to learn how to write, compile and test a program in your own environment. After completing the following assignment, submit it here.

Implement a program that prints three lines of the following style:

```
January
February
March
```

A new line character must be printed after printing the last row. Output must be in the same order as the sample output given above.

Implement a program (main-function), in the file ***main.c***. Test the program first on your local machine, and while it seems to work, return it below as an attachment.

Points 25 / 25 My submissions 2 Deadline Tuesday, 8 June 2021, 19:59 To be submitted alone

This course has been archived (Saturday, 31 December 2022, 20:00).

Intro

Select your files for grading

main.c

Choose File No file chosen

Submit

Task 1.6: Vector function¶

In this task, you will learn how to write a function that has a few parameters. In addition, you will have to call some of the math library functions that are present in a specific header files.

Implement function titled **vectorlength(double x, double y, double z)** that calculates the length of the given three-dimensional euclidean vector. The function gets three arguments that represent the different components in the three-dimensional space. The function should return the length of the vector. All numbers should be **double**-precision floating point numbers.

If you have forgotten about vector mathematics, you can find additional information in the Wikipedia. You will need to calculate square root that is not part of C's basic operators, but there is a sqrt function in that math library that you can use. With pow function you can calculate power functions. See the detailed function specifications from the linked manual pages. To use sqrt, pow functions, include **math.h** header. Use **-lm** option in end of the compilation command to link math library like below:

```
gcc -std=c99 -Wall -g -o main source.c -lm
```

The given exercise templates have this set up already.

Implement vector function in the `source.c` file. When your function works properly, upload the program below and run your program.

Points 25 / 25 My submissions 1 Deadline Tuesday, 8 June 2021, 19:59 To be submitted alone

This course has been archived (Saturday, 31 December 2022, 20:00).

Vector

Select your files for grading

source.c

Choose File No file chosen

Submit

« 3 Data types and Variables Course materials 5 Round feedback »