

Exercise 5-1: calc_func.c

Maximilian Fernaldy - C2TB1702

Exercise 5-1 Practice your own function: calc_func.c

16

- Modify the sample program add_func2.c (with prototype declaration) and create a program calc_func.c that adds the three functions of
 - Subtraction function: sub_func
 - Multiplication function: multi_func
 - Division function: div_func
- Also, in the main function, execute the following calculation without using arithmetic operators.
 - $Y = (2.0 + 1.0) \times 6.0 / 1.5 - 3.0$
- However, the variable of each function is double type.
- The function should be defined after the main function and a prototype declaration should be used.

$$y = (2.0 + 1.0) \times 6.0 / 1.5 - 3.0$$

[Hint]

For example, if $y = 2.0 \times 3.0 + 4.0$, it is calculated as follows.
 $y = \text{add_func}(\text{multi_func}(2.0, 3.0), 4.0)$

Since we are going to work with divisions, we should change the variables to type `float` in order to avoid unwanted truncation. Then, we should create prototype declarations for all the functions we're going to use after the includes:

```
#include <stdio.h>

// Function declarations
float add_func(float x, float y);
float sub_func(float a, float b);
float multi_func(float a, float b);
float div_func(float a, float b);
```

Then, we can proceed with the main function. We simply initialize a `float result` and use the functions accordingly. Since functions work from inside to the outside, it is useful to think about arithmetic operations from the last operation first. For example, the last operation that we want to do in this exercise is subtracting 3 from everything else that has been calculated before. So, the outermost function should be `sub_func(..., 3.0)`. Then, the second to last operation is dividing everything that has been calculated before by `1.5`. We should then use `div_func()` inside the brackets of `sub_func()` like so `sub_func(div_func(..., 1.5), 3.0)`. We repeat this with the

operation that should be performed before it, and the operation before it, and so on, until we reach the first operation that should be performed. Doing so, we will get this line:

```
float result = sub_func(div_func(multi_func(add_func(2.0, 1.0), 6.0), 1.5), 3.0);
```

Now to define the functions. They are very simple arithmetic operations, so each of them simply takes two floats and does whatever operation is needed. The general form of the functions should look like this:

```
float operation_func(float a, float b) {  
    return a <operator> b;  
}
```

Compiling and running the program gives us the following output:

```
max@victus16:~/repos/pip/lec05/ex5-1
-`
.o+`
`ooo/
`+oooo:
`+oooooo:
-+ooooooo:
`/!:-!++oooo+:
`/++++/+++++++:
`/+++++++:
`/+++ooooooooooooo/`
./oooooooooo+ooooo+
.ooooooo-`-/ooooo+
-ooooooo. :ssssso.
:ooooooo/ ooooo+++
/ooooooooo/ +sssooo/-
`/ooooooo+/- -:/+oooo+
`+sso+:-` `./+oso:
`++:.` `-/+/
`/

max@victus16
-----
OS: Arch Linux x86_64
Host: Victus by HP Laptop 16-e0xxx
Kernel: 6.5.9-arch2-1
Uptime: 1 hour, 18 mins
Packages: 756 (pacman), 6 (flatpak)
Shell: bash 5.2.15
Resolution: 1920x1080, 1920x1080
WM: i3
Theme: vimix-dark-doder [GTK2/3]
Icons: breeze-dark [GTK2/3]
Terminal: alacritty
CPU: AMD Ryzen 7 5800H with Radeon Graphics (16) @ 4.463GHz
GPU: NVIDIA GeForce RTX 3060 Mobile / Max-Q
GPU: AMD ATI Radeon Vega Series / Radeon Vega Mobile Series
Memory: 3965MiB / 15323MiB

max ex5-1 $ ./calc_func
9.000000
max ex5-1 $ █
```