



C++ Pool - d02m

Koalab koala@epitech.eu

Abstract: This document is the subject of d02m.

Contents

I	GENERAL RULES	2
II	Exercise 0	4
III	Exercise 1	6
IV	Exercise 2	8
V	Exercise 3	10
VI	Exercise 4	13
VII	Exercise 5	17

Chapter I

GENERAL RULES


- READ THE GENERAL RULES CAREFULLY !!
 - You will have no possible excuse if you end up with a 0 because you didn't follow one of the general rules
- GENERAL RULES :
 - If you do half the exercises because you have comprehension problems, it's okay, it happens. But if you do half the exercises because you're lazy, and leave at 2PM, you WILL have problems. Do NOT tempt the devil.
 - Every function implemented in a header, or unprotected header, means 0 to the exercise.
 - The imposed filenames must be followed TO THE LETTER, as well as functions names.
 - Turn-in directories are ex00, ex01, ..., exN
 - Read the examples CAREFULLY. They might require things the subject doesn't say ...
 - Read ENTIRELY the subject of an exercise before you start it !
 - THINK. Please.
 - THINK. For Odin's sake.
- COMPILATION OF THE EXERCISES :



- The Koalinette compiles your code with the following flags : `-W -Wall -Werror`
- To avoid compilation problems with the Koalinette, include every required headers in your headers.
- Note that none of your files must contain a `main` function. We will use our own to compile and test your code.
- This subject may be modified up to 4h before turn-in time. Refresh it regularly !
- The turn-in dirs are `(piscine_cpp_d02m/exN)` , N being the exercise number

Chapter II

Exercise 0

	Exercise : 00	points : 3
Add Mul - Basic pointers		
Turn-in directory: (piscine_cpp_d02m)/ex00		
Compiler: gcc	Compilation flags: -Wall -Wextra -Werror	
Makefile: No	Rules: n/a	
Files to turn in : mul_div.c		
Remarks : n/a		
Forbidden functions : None		

- Add Mul 4 Params (File mul_div.c)

Create the function `add_mul_4param`, with the following prototype :

```
1 void add_mul_4param(int first, int second, int *add, int *mul);
```

This function calculates the sum of the two parameters 'first' and 'second' and stores the result in the integer pointed by 'add'.

This function also stores the result of the multiplication of 'first' and 'second' in the integer pointed by 'mul'.

- Add Mul 2 Params (File mul_div.c)

Create the function `add_mul_2param`, with the following prototype :

```
1 void add_mul_2param(int *first, int *second);
```

This function adds the integers pointed by 'first' and 'second', it also multiplies the integers pointed by 'first' and 'second'.


- The result of the addition is stored in the integer pointed by 'first'.
- The result of the multiplication is stored in the integer pointed by par 'second'.

Here is an example of a main function with the expected output :

```
1 int main(void)
2 {
3     int first;
4     int second;
5     int add_res;
6     int mul_res;
7
8     first = 5;
9     second = 6;
10
11     add_mul_4param(first, second, &add_res, &mul_res);
12     printf('%d + %d = %d\n', first, second, add_res);
13     printf('%d * %d = %d\n', first, second, mul_res);
14
15     add_res = first;
16     mul_res = second;
17     add_mul_2param(&add_res, &mul_res);
18     printf('%d + %d = %d\n', first, second, add_res);
19     printf('%d * %d = %d\n', first, second, mul_res);
20
21     return (0);
22 }
23
24 $> ./a.out
25 5 + 6 = 11
26 5 * 6 = 30
27 5 + 6 = 11
28 5 * 6 = 30
29 $>
```

Chapter III

Exercise 1

	Exercise : 01	points : 3
Mem Ptr - Pointers and memory		
Turn-in directory: (piscine_cpp_d02m)/ex01		
Compiler: gcc	Compilation flags: -Wextra -Werror -Wall	
Makefile: No	Rules: n/a	
Files to turn in : mem_ptr.c		
Remarks : The structure 't_str_op' is in the provided file 'mem_ptr.h'.		
Forbidden functions : None		

- add_str (File mem_ptr.c)

Create the add_strfunction, with the following prototype :

```
1 void add_str(char *str1, char *str2, char **res);
```

This function concatenates the strings 'str1' and 'str2'. The resulting string is stored in the pointer pointed by 'res'.

The needed memory space WILL NOT be preallocated in 'res'.

- add_str_struct (File mem_ptr.c)

Create the add_str_structfunction, with the following prototype :

```
1 void add_str_struct(t_str_op *str_op)
```

This function has the same behaviour as the add_strfunction. It concatenates the strings 'str1' and 'str2' contained in the structure pointed by str_op. The


resulting string has to be stored in the 'res' field of the structure pointed by 'str_op'.

Here is an example of a main function with the expected output :

```
1 int main(void)
2 {
3     char *str1 = "Salut, ";
4     char *str2 = "ca marche !";
5     char *res;
6     t_str_op str_op;
7
8
9     add_str(str1, str2, &res);
10
11     printf("%s\n", res);
12
13     str_op.str1 = str1;
14     str_op.str2 = str2;
15     add_str_struct(&str_op);
16
17     printf("%s\n", str_op.res);
18
19     return (0);
20 }
21
22 $> ./a.out
23 Salut, ca marche !
24 Salut, ca marche !
25 $>
```


Chapter IV

Exercise 2

	Exercise : 02	points : 4
Tab to 2dTab - Pointers and memory		
Turn-in directory: (piscine_cpp_d02m)/ex02		
Compiler: gcc	Compilation flags: -Wextra -Werror -Wall	
Makefile: No	Rules: n/a	
Files to turn in : tab_to_2dtab.c		
Remarks : n/a		
Forbidden functions : None		

Create the `tab_to_2dtab` function, with the following prototype :

```
1 void tab_to_2dtab(int *tab, int length, int width, int ***res);
```

This function takes an array of integers as its first parameter named '`tab`'. It creates a bidimensional array of '`length`' lines and '`width`' columns from '`tab`'.

The bidimensional array has to be stored in the pointer pointed by '`res`'. The necessary memory space will not be allocated in '`res`' before calling the function.


Here is an example of a main function with the expected output :

```
1 int main(void)
2 {
3     int **tab_2d;
4     int tab[42] = {0, 1, 2, 3, 4, 5,
5                   6, 7, 8, 9, 10, 11,
6                   12, 13, 14, 15, 16, 17,
7                   18, 19, 20, 21, 22, 23,
8                   24, 25, 26, 27, 28, 29,
9                   30, 31, 32, 33, 34, 35,
10                  36, 37, 38, 39, 40, 41};
11
12     tab_to_2dtab(tab, 7, 6, &tab_2d);
```

```
13
14 printf('tab2[%d] [%d] = %d\n', 0, 0, tab_2d[0][0]);
15 printf('tab2[%d] [%d] = %d\n', 6, 5, tab_2d[6][5]);
16 printf('tab2[%d] [%d] = %d\n', 4, 4, tab_2d[4][4]);
17 printf('tab2[%d] [%d] = %d\n', 0, 3, tab_2d[0][3]);
18 printf('tab2[%d] [%d] = %d\n', 3, 0, tab_2d[3][0]);
19 printf('tab2[%d] [%d] = %d\n', 4, 2, tab_2d[4][2]);
20
21 return (0);
22 }
23
24
25 $> ./a.out
26 tab2[0][0] = 0
27 tab2[6][5] = 41
28 tab2[4][4] = 28
29 tab2[0][3] = 3
30 tab2[3][0] = 18
31 tab2[4][2] = 26
32 $>
```

Chapter V

Exercise 3

	Exercise : 03	points : 5
Func Ptr - Function pointers		
Turn-in directory: (piscine_cpp_d02m)/ex03		
Compiler: gcc	Compilation flags: -Wextra -Werror -Wall	
Makefile: No	Rules: n/a	
Files to turn in : func_ptr.h, func_ptr.c		
Remarks : t_action is defined in the provided file 'func_ptr_enum.h'		
Forbidden functions : None		

Functions (Files func_ptr.c and func_ptr.h)

Create the following functions :

- void print_normal(char *str);

Displays the string 'str' which is the first parameter, followed by a newline.

- void print_reverse(char *str);

Displays the string 'str' which is the first parameter, reversed and followed by a newline.

- void print_upper(char *str);

Displays the string 'str' which is the first parameter, with every lowercase letter converted to uppercase, followed by a newline.

- void print_42(char *str);

Displays the string "42" followed by a newline.



Hint : Use 'printf' OR 'write' to display the strings but not both at the same time !

Function pointers (Files `func_ptr.c` and `func_ptr.h`)

You have to include the file '`func_ptr_enum.h`' in your file '`func_ptr.h`'.
Create the function `do_action` with the following prototype :

```
1 void do_action(t_action action, char *str);
```

This function executes an action depending on the parameter '`action`'.

- If the value of '`action`' is '`PRINT_NORMAL`' the '`print_normal`' function has to be called with '`str`' as its parameter.
- If the value of '`action`' is '`PRINT_REVERSE`' the '`print_reverse`' function has to be called with '`str`' as its parameter.
- If the value of '`action`' is '`PRINT_UPPER`' the '`print_upper`' function has to be called with '`str`' as its parameter.
- If the value of '`action`' is '`PRINT_42`' the '`print_42`' function has to be called with '`str`' as its parameter.

Of course, you HAVE to use function pointers, chaining 'if/else if', the 'switch' statement, etc. are forbidden.


Here is an example of a main function with the expected output :

```
1 int main(void)
2 {
3     char *str = "J'utilise les pointeurs sur fonctions !";
4
5     do_action(PRINT_NORMAL, str);
6     do_action(PRINT_REVERSE, str);
7     do_action(PRINT_UPPER, str);
8     do_action(PRINT_42, str);
9
10    return (0);
11 }
12
13 $>./a.out | cat -e
14 J'utilise les pointeurs sur fonctions !$
15 ! snoitcnof rus srutniop sel esilitu'J$
```

```
16 J'UTILISE LES POINTEURS SUR FONCTIONS !$  
17 42$
```

Chapter VI

Exercise 4

	Exercise : 04	points : 5
Cast Mania - Understanding and mastering casts		
Turn-in directory: (piscine_cpp_d02m)/ex04		
Compiler: gcc	Compilation flags: -Wextra -Werror -Wall	
Makefile: No	Rules: n/a	
Files to turn in : add.c, div.c, castmania.c		
Remarks : Every structure and enumeration are defined in the provided file 'castmania.h'		
Forbidden functions : None		

Divisions (File 'div.c')

Implement the following functions :

- 1 `int integer_div(int a, int b);`

Does a euclidian division between 'a' and 'b' and returns the result. If the value of 'b' is 0, the function returns 0.

- 1 `float decimale_div(int a, int b);`

Does a decimal division between 'a' and 'b' and returns the result. If the value of 'b' is 0, the function returns 0.

- 1 `void exec_div(t_div *operation);`

The first and only parameter of this function is a pointer on a structure of type `t_div` . Depending on the value of the field `'div_type'` the function does an integral or decimal division.

The field `'div_op'` is a generic pointer. It points towards a structure of type `t_integer_op` if the value `'div_type'` is `'INTEGER'` or towards a structure of type `t_decimale_op` if the value of `'div_type'` is `'DECIMALE'` .

The two integers `'a'` and `'b'` to use are the fields of the structure pointed by `'div_op'` .

In both cases, the result has to be stored in the `'res'` field of the structure pointed by `'div_op'` .

Additions (File `'add.c'`)

Create the following functions :

- 1 `int normal_add(int a, int b);`

Calculates the sum of `'a'` and `'b'` and returns the result.

- 1 `int absolute_add(int a, int b);`

Calculates the sum of the absolute value of `'a'` and the absolute value of `'b'` and returns the result.

- 1 `void exec_add(t_add *operation);`

The function takes a pointer to a structure of type type `t_add` as its only argument and depending on the value of `'add_type'` will do a normal or absolute addition.

The two integers `'a'` and `'b'` are both fields of the `'add_op'` structure. In both cases, the result has to be stored in the `'res'` field of the `'add_op'` structure.

CastMania (File `'castmania.c'`)

Implement the following functions :

•
1 `void exec_operation(t_instruction_type instruction_type, void *data);`

Executes an addition or a division depending on the value of 'instruction_type' . In each case, 'data' will point on a structure of type 't_instruction' .

- If the value of 'instruction_type' is 'ADD_OPERATION' , you have to do an addition using the 'exec_add' function.
In this precise case, the 'operation' field of the structure pointed by 'data' will point to a structure of type 't_add' .

- If the value of 'instruction_type' is 'DIV_OPERATION' , you have to do a division using the 'exec_div' function.

In this case the field 'operation' of the structure pointed by 'data' , will point to a structure of type 't_div' .

- For any other value of 'instruction_type' , nothing has to be done.

- If the 'output_type' field of the 'data' structure has 'VERBOSE' as a value, the result of the operation has to be displayed.

•
1 `void exec_instruction(t_instruction_type instruction_type, void *data);`

Executes an action depending on the value of 'instruction_type' .

- If the value of 'instruction_type' is 'PRINT_INT' , 'data' will point to an 'int' that has to be displayed.
- If the value of 'instruction_type' is 'PRINT_FLOAT' , 'data' will point to a 'float' that has to be displayed.
- Else the 'exec_operation' function should be called, with 'instruction_type' and 'data' as parameters.

Here is an example of a main function with the expected output :

```
1 int main(void)
2 {
3     int i = 5;
```



```


4 float f = 42.5;
5 t_instruction inst;
6 t_add add;
7 t_div div;
8 t_integer_op int_op;
9
10 printf('Affiche i : ');
11 exec_instruction(PRINT_INT, &i);
12 printf('Affiche f : ');
13 exec_instruction(PRINT_FLOAT, &f);
14
15 printf('\n');
16
17 int_op.a = 10;
18 int_op.b = 3;
19
20 add.add_type = ABSOLUTE;
21 add.add_op = int_op;
22
23 inst.output_type = VERBOSE;
24 inst.operation = &add;
25
26 printf('10 + 3 = ');
27 exec_instruction(ADD_OPERATION, &inst);
28
29 printf('En effet 10 + 3 = %d\n\n', add.add_op.res);
30
31 div.div_type = INTEGER;
32 div.div_op = &int_op;
33
34 inst.operation = &div;
35
36 printf('10 / 3 = ');
37 exec_instruction(DIV_OPERATION, &inst);
38
39 printf('En effet 10 / 3 = %d\n\n', int_op.res);
40
41 return (0);
42 }
43
44 $> ./a.out
45 Affiche i : 5
46 Affiche f : 42.500000
47
48 10 + 3 = 13
49 En effet 10 + 3 = 13
50
51 10 / 3 = 3
52 En effet 10 / 3 = 3
53
54 $>

```

Chapter VII

Exercise 5

An exemple file ('ptr_tricks.h') is provided

	Exercise : 05	points : 2
Pointer Master - [Achievement] Pointer Steamroller		
Turn-in directory: (piscine_cpp_d02m)/ex05		
Compiler: gcc	Compilation flags: -Wextra -Werror -Wall	
Makefile: No	Rules: n/a	
Files to turn in : ptr_tricks.c		
Remarks : n/a		
Forbidden functions : None		

Pointer Arithmetic (File ptr_tricks.c)

- Create the `get_array_nb_elem` function with the following prototype :

```
1 int get_array_nb_elem(int *ptr1, int *ptr2);
```

This function takes 2 pointers to int as parameters, each one of these 2 pointers point to a different location of the same array of int. This function returns the number of elements of the array located between both pointers.

... (File ptr_tricks.c)

- Create the `get_struct_ptr` function with the following prototyp:

```
1 t_whatever *get_struct_ptr(int *member_ptr);
```

`t_whatever` is defined this way :

```
1     typedef struct s_whatever
2     {
3         ...
4         int member;
5         ...
6     } t_whatever;
```

'...' means that any field could be inserted in the structure 's_whatever' before and after the 'member' field.

An example of the 's_whatever' structure is given in the 'ptr_tricks.h' file.

The function 'get_struct_ptr' takes as its only parameter a pointer on the 'member' field of a structure 's_whatever' and simply returns a pointer to this structure.

Here is an example of a main function with the expected output :

```
1 int main(void)
2 {
3     int tab[1000] = {0};
4     int nb_elem;
5
6     nb_elem = get_array_nb_elem(&tab[666], &tab[708]);
7
8     printf("Il y a %d elements entre l'element 666 et 708\n", nb_elem);
9
10    return (0);
11 }
12
13 $> ./a.out
14 Il y a 42 elements entre l'element 666 et 708
15 $>
```

```
1 int main(void)
2 {
3     t_whatever test;
4     t_whatever *ptr;
5
6     ptr = get_struct_ptr(&test.member);
7
8     if (ptr == &test)
9         printf("Ca marche !\n");
10
11    return (0);
12 }
13
```

```
14 $>./a.out  
15 Ca marche !  
16 $>
```