



# Piscine C++ - d02m

Mehdi “Drax” Ait-Bachir [ait-ba\\_m@epitech.eu](mailto:ait-ba_m@epitech.eu)

*Abstract: Ce document est le sujet du d02m*

# Table des matières

I	Remarques générales	2
II	Exercice 0	3
III	Exercice 1	5
IV	Exercice 2	7
V	Exercice 3	9
VI	Exercice 4	11
VII	Exercice 5	15


# Chapitre I

## Remarques générales

- Si vous faites la moitié des exercices car vous avez du mal, c'est normal. Par contre, si vous faites la moitié des exercices par flemme et vous tirez à 14h, vous AUREZ des surprises. Ne tentez pas le diable.
- Les noms de fichiers qui vous sont imposés doivent être respectés A LA LETTRE, de même que les noms de fonctions.
- Les répertoires de rendus sont ex00, ex01, ...
- Lisez attentivement les exemples, ils peuvent requérir des choses que le sujet ne dit pas...
- Lisez ENTièrement le sujet d'un exercice avant de le commencer !
- RÉFLECHISSEZ. Par pitié.
- Notez bien qu'aucun de vos fichiers ne doit contenir de fonction "main" sauf si le contraire est explicite. Nous utiliserons notre propre fonction "main" pour compiler et tester votre code.

# Chapitre II

## Exercice 0

	Exercice : 00	points : 3
Add Mul - Pointeurs basiques		
Répertoire de rendu: (DÉPOT SVN - piscine_cpp_d02m-promo-login_x)/ex00		
Compilateur : gcc	Flags de compilation: -Wall -Wextra -Werror	
Makefile : Non	Règles : n/a	
Fichiers a rendre : mul_div.c		
Remarques : n/a		
Fonctions Interdites : Aucune		

- Add Mul 4 Params (Fichier mul\_div.c)

Créer la fonction `add_mul_4param`, qui a le prototypage suivant :

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

Cette fonction additionne les deux paramètres '`first`' et '`second`' et stocke le resultat dans l'int pointé par '`add`'.

Elle stocke également le résultat de la multiplication de '`first`' par '`second`' dans l'int pointé par '`mul`'.

- Add Mul 2 Params (Fichier mul\_div.c)

Créer la fonction `add_mul_2param`, qui a le prototypage suivant :

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

Cette fonction additionne les int pointés par '`first`' et '`second`', elle multiplie également les int pointés par '`first`' et '`second`'.


- Le résultat de l'addition est stocké dans l'int pointé par '**first**'.
- Le résultat de la multiplication est stocké dans l'int pointé par '**second**'.

Voici un exemple de main avec la sortie attendue :

```
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 $>
```

# Chapitre III

## Exercice 1

	Exercice : 01	points : 3
Mem Ptr - Pointeurs et mémoire		
Répertoire de rendu: (DÉPOT SVN - piscine_cpp_d02m-promo-login_x)/ex01		
Compilateur : gcc	Flags de compilation: -Wextra -Werror -Wall	
Makefile : Non	Règles : n/a	
Fichiers a rendre : mem_ptr.c		
Remarques : La structure 't_str_op' est dans le fichier 'mem_ptr.h' qui vous est fourni.		
Fonctions Interdites : Aucune		

- add\_str (Fichier mem\_ptr.c)

Créer la fonction add\_str qui a le prototype suivant :

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

Cette fonction concatène les chaînes de caractères 'str1' et 'str2'. Le chaîne résultante sera stockée dans le pointeur pointé par 'res'.

L'espace mémoire nécessaire pour stocker la nouvelle chaîne ne sera pas préalloué dans 'res'.

- add\_str\_struct (Fichier mem\_ptr.c)

Créer la fonction 'add\_str\_struct' qui a le prototype suivant :

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

Cette fonction possède le même comportement que la fonction add\_str. Elle conca-


tène les chaînes '**str1**' et '**str2**' contenues dans la structure pointée par **str\_op**. La chaîne résultante devra être stockée dans le champ '**res**' de la structure pointée par '**str\_op**'.

Voici un exemple de main avec la sortie attendue :

```
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 $>
```

# Chapitre IV

## Exercice 2

	Exercice : 02	points : 4
Tab to 2dTab - Pointeurs et mémoire		
Répertoire de rendu: (DÉPOT SVN - piscine_cpp_d02m-promo-login_x)/ex02		
Compilateur : gcc	Flags de compilation: -Wextra -Werror -Wall	
Makefile : Non	Règles : n/a	
Fichiers a rendre : tab_to_2dtab.c		
Remarques : n/a		
Fonctions Interdites : Aucune		

Créer la fonction `tab_to_2dtab` qui a le prototype suivant :

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

Cette fonction prend en paramètre un tableau de nombres entiers nommé '`tab`'. Elle crée un tableau à 2 dimensions, contenant '`length`' nombre de lignes et '`width`' nombre de colonnes, à partir de '`tab`'.

Le tableau à 2 dimensions est stocké dans le pointeur pointé par '`res`'. L'espace nécessaire pour stocker le tableau à deux dimensions ne sera pas alloué au préalable dans '`res`'.

Voici un exemple de main avec la sortie attendue :


```
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 $>
```

# Chapitre V

## Exercice 3

	Exercice : 03	points : 5
Func Ptr - Pointeurs sur fonctions		
Répertoire de rendu: (DÉPOT SVN - piscine_cpp_d02m-promo-login_x)/ex03		
Compilateur : gcc	Flags de compilation: -Wextra -Werror -Wall	
Makefile : Non	Règles : n/a	
Fichiers a rendre : func_ptr.h, func_ptr.c		
Remarques : t_action est définie dans le fichier 'func_ptr_enum.h' qui vous est fourni		
Fonctions Interdites : Aucune		

Fonctions (Fichiers `func_ptr.c` et `func_ptr.h` )

Créer les fonctions suivantes :

- `void print_normal(char *str);`

Qui affiche la chaîne de caractères '`str`' passée en parametre, suivie d'un retour à la ligne.

- `void print_reverse(char *str);`

Qui affiche la chaîne de caractères '`str`' passée en parametre, à l'envers suivie d'un retour à la ligne.

- `void print_upper(char *str);`

Qui affiche la chaîne de caractères '`str`' passée en paramètre, en transformant toutes les lettres minuscules en lettres majuscules, suivie d'un retour a la ligne.

- `void print_42(char *str);`

Qui affiche la chaine de caractères "42" suivie d'un retour a la ligne.



Conseil : utilisez 'printf' OU 'write' pour afficher mais pas les deux en même temps!

Pointeurs sur fonction (Fichiers `func_ptr.c` et `func_ptr.h`)

Vous devez inclure le fichier '`func_ptr_enum.h`' dans votre fichier '`func_ptr.h`'.  
Créer la fonction `do_action` qui a le prototype suivant :

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

Cette fonction exécute une action en fonction du paramètre '`action`'.

- Si '`action`' vaut '`PRINT_NORMAL`' elle appelle la fonction '`print_normal`' en lui envoyant '`str`' en parametre.
- Si '`action`' vaut '`PRINT_REVERSE`' elle appelle la fonction '`print_reverse`' en lui envoyant '`str`' en parametre.
- Si '`action`' vaut '`PRINT_UPPER`' elle appelle la fonction '`print_upper`' en lui envoyant '`str`' en parametre.
- Si '`action`' vaut '`PRINT_42`' elle appelle la fonction '`print_42`' en lui envoyant '`str`' en parametre.


Evidemment vous DEVEZ utiliser les pointeurs sur fonctions, les series de 'if/else if', les 'switch', etc. sont interdits.

Voici un exemple de main avec la sortie attendue :

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

# Chapitre VI

## Exercice 4

	Exercice : 04	points : 5
Cast Mania - Comprendre et maitriser les casts		
Répertoire de rendu: (DÉPOT SVN - piscine_cpp_d02m-promo-login_x)/ex04		
Compilateur : gcc	Flags de compilation: -Wextra -Werror -Wall	
Makefile : Non	Règles : n/a	
Fichiers a rendre : add.c, div.c, castmania.c		
Remarques : Toutes les structures et énumérations sont définies dans le fichier 'castmania.h' qui vous est fourni		
Fonctions Interdites : Aucune		

Divisions (Fichier 'div.c' )

Implémenter les fonctions suivantes :

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

Effectue une division entière (ou euclidienne) de 'a' et de 'b' et renvoie le résultat. Si 'b' vaut 0 la fonction renvoie 0.

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

Effectue une division décimale de 'a' et de 'b' et renvoie le résultat. Si 'b' vaut 0 la fonction renvoie 0.

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

La fonction prend en argument un pointeur sur structure de type `t_div` et qui, selon la valeur du champ `'div_type'`, effectue une division entière ou décimale. Le champ `'div_op'` est un pointeur générique. Il pointe sur une structure de type `t_integer_op` si `'div_type'` vaut `'INTEGER'` ou sur une structure de type `t_decimale_op` si `'div_type'` vaut `'DECIMALE'`.

Les deux entiers `'a'` et `'b'` à utiliser sont les champs de la structure pointée par `'div_op'`.

Dans les deux cas il faudra stocker le résultat dans le champ `'res'` de la structure pointée par `'div_op'`.

Additions (Fichier `'add.c'`)

Implémenter les fonctions suivantes :

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

Effectue l'addition de `'a'` et de `'b'` et renvoie le résultat.

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

Effectue l'addition de la valeur absolue de `'a'` et de la valeur absolue de `'b'` et renvoie le resultat.

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

La fonction prend en argument un pointeur sur structure de type `t_add` et qui, selon la valeur de `'add_type'`, effectue une addition normale ou absolue.

Les deux entiers `'a'` et `'b'` à utiliser sont des champs de la structure `'add_op'`.

Dans les deux cas il faudra stocker le résultat dans le champ `'res'` de la structure `'add_op'`.

CastMania (Fichier `'castmania.c'`)

Implémenter les fonctions suivantes :

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

Effectue une addition ou une division selon la valeur de `'instruction_type'` . Dans tout les cas `'data'` pointera sur une structure de type `'t_instruction'` .

- Si `'instruction_type'` vaut `'ADD_OPERATION'` , il faut effectuer une addition grâce a la fonction `'exec_add'` . Dans ce cas le champ `'operation'` de la structure pointée par `'data'` , pointera sur une structure de type `'t_add'` .
- Si `'instruction_type'` vaut `'DIV_OPERATION'` , il faut effectuer une division grace a la fonction `'exec_div'` . Dans ce cas le champ `'operation'` de la structure pointée par `'data'` , pointera sur une structure de type `'t_div'` .
- Pour toute autre valeur de `'instruction_type'` , ne rien faire.
- Si le champ `'output_type'` de la structure `'data'` vaut `'VERBOSE'` , il faut afficher le résultat `'res'` de l'opération.

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

Effectue une action selon la valeur de `'instruction_type'` .

- Si `'instruction_type'` vaut `'PRINT_INT'` , `'data'` pointera sur une variable de type `'int'` qu'il faudra afficher.
- Si `'instruction_type'` vaut `'PRINT_FLOAT'` , `'data'` pointera sur une variable de type `'float'` qu'il faudra afficher.
- Sinon il faudra executer une opération grace à la fonction `'exec_operation'` , en lui passant en paramètre `'instruction_type'` et `'data'` .

Voici un Main de test avec la sortie attendue :


```
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 $>
```

# Chapitre VII

## Exercice 5

Un fichier 'ptr\_tricks.h' d'exemple vous est fourni

	Exercice : 05	points : 2
Pointer Master - [Achievement] Rouleau compresseur des pointeurs		
Répertoire de rendu: (DÉPOT SVN - piscine_cpp_d02m-promo-login_x)/ex05		
Compilateur : gcc	Flags de compilation: -Wextra -Werror -Wall	
Makefile : Non	Règles : n/a	
Fichiers a rendre : ptr_tricks.c		
Remarques : n/a		
Fonctions Interdites : Aucune		

Pointer Arithmetic (Fichier ptr\_tricks.c )

- Créer la fonction `get_array_nb_elem` qui a le prototype suivant :

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

Cette fonction prend en paramètre deux pointeurs sur int qui pointent à deux endroit différents d'un même tableau d'int.

La fonction renvoie le nombre d'éléments du tableau contenus entre les deux pointeurs.

... (Fichier ptr\_tricks.c )

- Créer la fonction `get_struct_ptr` qui a le prototype suivant :

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

`t_whatever` étant définie de la manière suivante :



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

Les '...' veulent dire que l'on peut mettre n'importe quel champ dans la structure 's\_whatever' avant et après le champ 'member' .

Un exemple de structure 's\_whatever' est donné dans le fichier 'ptr\_tricks.h' .

La fonction 'get\_struct\_ptr' prend donc en paramètre un pointeur sur le champ 'member' d'une structure 's\_whatever' et doit tout simplement renvoyer un pointeur sur cette structure.

Voici un main de test pour chacune des fonctions avec la sortie attendue :

```
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 $>
```