



Piscine C++ - d01

Le C, la vie, l'univers et le reste

Zoltān Konarzewski konarz_z@epitech.eu

Abstract: Ce document est le sujet du d01.

Table des matières

I	Préambule	2
II	Exercice 0	3
III	Exercice 1	6
IV	Exercice 2	10
V	Exercice 3	16
VI	Exercice 4	19
VII	Exercice 5	21
VIII	Exercice 6	24
IX	Exercice 7	26
X	Exercice 8	32

Chapitre I

Préambule



La correction des jours de piscine sera effectuée sur un dump officiel (Fedora 14 x86_64, GCC 4.5.1). L'équipe Koala rejette toute responsabilité pour des problèmes de correction si vous n'utilisez pas le même système.

Si votre dump n'est pas conforme, prenez immédiatement contact avec le Bocal afin de le redumper.

Pour toute la durée de cette piscine, votre rendu se fera via le système de rendus du Koalab. Vous aurez à votre disposition un dépôt SVN par journée, sur lequel vous devrez rendre tous vos exercices, et qui sera fermé en écriture à l'heure exacte de fin de l'activité, NTP epitech faisant foi.

La documentation concernant ces dépôts sera jointe à chaque sujet. Veillez à la lire rigoureusement et entièrement, et veillez surtout à vous familiariser immédiatement avec la procédure de rendu, car si vous venez vous plaindre à 10 minutes de la fin que "ça marche pas", vous ne trouverez que rires et quolibets.


Chaque jour, les dépôts de rendus seront créés entre 9h et 11h du matin (entre 15h et 16h pour les activités de l'après-midi). Un mail vous sera automatiquement envoyé à la création de vos dépôts, il est donc parfaitement inutile de tenter d'y accéder avant de l'avoir reçu.

En outre, veuillez noter que comme vous ne saurez pas à quels horaires les moulinettes intermédiaires seront exécutées, il est de votre responsabilité de commit régulièrement votre travail sur le dépôt SVN si vous souhaitez recevoir des corrections intermédiaires ...

Si d'aventure votre question ne trouvait pas réponse dans la documentation qui vous est fournie, prenez contact avec un assistant de votre salle.

Chapitre II

Exercice 0

	Exercice : 00	points : 2
Follow the white rabbit		
Répertoire de rendu: (DÉPOT SVN - piscine_cpp_d01-promo-login_x)/ex00		
Compilateur : gcc	Flags de compilation: -Wall -Wextra -Werror -std=gnu99	
Makefile : Non	Règles : n/a	
Fichiers a rendre : white_rabbit.c		
Remarques : n/a		
Fonctions Interdites : Aucune		

Assise sur le talus, Alice n'avait rien à faire. Elle se demandait si le plaisir de tresser une guirlande de pâquerettes vaudrait la peine de se lever et d'aller cueillir lesdites pâquerettes, lorsque, brusquement, un Lapin Blanc, aux yeux roses passa en courant près d'elle. Ceci n'avait rien de particulièrement remarquable ; et Alice ne trouva pas non plus tellement bizarre d'entendre le Lapin se dire à mi-voix : "Oh mon Dieu ! Oh mon Dieu ! Je vais être en retard !". Cependant, lorsque le Lapin tira bel et bien une montre de la poche de son gilet, regarda l'heure, et se mit à courir de plus belle, Alice se dressa d'un bond, car, tout à coup, l'idée lui était venue qu'elle n'avait jamais vu de lapin pourvu d'une poche de gilet, ni d'une montre à tirer de cette poche. Dévorée de curiosité, elle traversa le champ en courant à sa poursuite, et eut la chance d'arriver juste à temps pour le voir s'enfoncer comme une flèche dans un énorme terrier placé sous la haie. Un instant plus tard, elle y pénétrait à son tour, sans se demander une seule fois comment diable elle pourrait bien en sortir.

Après une longue errance dans les méandres de ce terrier, Alice rencontra un Koala qui lui dit : "Salut toi ! Qu'est-ce que tu fais là ? Toi aussi, tu cherches le Poney Rose ?". Alice lui répondit qu'elle ne cherchait pas un poney rose mais qu'elle suivait le Lapin Blanc. "Ah mais j'te connais bien l'ami Lapin, lui répondit le Koala, j'ai même vu passer y'a pas cinq minutes de ça ! Il avait l'air drôlement pressé l'ami Lapin Blanc !". Alice demanda au Koala de lui indiquer dans quelle direction était parti le lapin. Sans hésitation, le Koala tendit un doigt vers sa gauche et s'écria : "Par là !!", mais il se ravisa aussitôt et pointa la direction opposée : "Euh non... J crois qu'c'était plutôt par là...". Après avoir montré une douzaine de directions différentes, le Koala finit par dire : "Hmm... En fait, j'crois que j'suis paumé...". Alice était désespérée. Elle était perdue dans un immense

terrier et avait perdu la trace du Lapin Blanc. En la voyant dans cet état, le Koala eut pitié d'elle et dit : "T'en fais pas petite! On va le retrouver ton ami l'Lapin Blanc. Regarde ce que j'ai là". Il sorti alors de sa poche de gilet (oui, lui aussi avait un gilet avec une poche!) un dé a 37 faces et le donna à Alice. Tandis qu'Alice contemplait le dé, le Koala montra à Alice chacune des faces. "Ça c'est la face 1, on la reconnaît parce que y'a le nombre 1 écrit dessus! Et ça c'est la face 2" et ainsi de suite jusqu'à la face 37...

Le Koala dit ensuite à Alice : "C'que t'as dans la main, c'est un dé magique! Il faut qu't'y fasse super attention! Mais avec ça, tu pourras retrouver l'Lapin Blanc! Maintenant, écoute-moi bien, sois très attentive, je vais t'expliquer comment t'en servir. À chaque fois qu'tu n'sais plus par où aller, lance le dé. Suivant le résultat, tu sauras par où est parti l'Lapin Blanc. Si le dé indique un multiple de 11 et qu'il fait beau, fais une sieste! T'aurais tort d'pas en profiter! Et tu peux être sûre que l'Lapin en f'ra autant. Par contre, s'il fait pas beau, relance le dé. S'il faisait beau et qu't'as fait une sieste, quand tu t'éveilles, lance le dé. S'il fait toujours beau et que le dé te dit de r'faire une sieste, c'est qu'tu t'es réveillée trop tôt! Si jamais l'dé te donne 37, c'est qu't'as trouvé l'Lapin Blanc. Oublie jamais qu'après avoir bu un thé, faut toujours continuer tout droit. Quand l'dé te donne un nombre plus grand que 24 et qu'l'triple de c'nombre vaut septante-huit ou 146, ça veut dire qu'l'dé s'est trompé, tu dois faire demi-tour. Si l'résultat vaut quatre ou 5, pars à gauche, y'a une chenille qui fume une pipe et fait des ronds. Elle est un peu folle mais elle est marrante! Si t'obtiens un 15, va tout droit. Quand le dé te donne 12, t'as pas d'chance, c'est un jet pour rien, faut qu'tu r'commence... Quand l'résultat vaut 13, va à droite. Ça marche aussi quand l'dé te donne 34 ou plus. Faut qu't'ailles à gauche si l'dé te sort un six, un 17 ou un 28. Un 23 veut dire qu'il est 22h! C'est l'heure du thé. Trouve une table et commande un thé au citron. Si t'aimes pas l'citron, prend un thé vert. Au fait, oublie pas d'compter tous tes résultats! Quand tu les additionnes et que tu trouves 421, c'est que t'as trouvé l'Lapin Blanc. Tu lui passeras l'bonjour de ma part tant que t'y es. Au fait, le truc de la somme des lancers, ça marche aussi si ça fait au moins trois cent nonante sept. Quand tu lances l'dé, si tu peux diviser l'résultat par 8 et qu'y'a pas de reste, fais demi-tour. J'aime pas le chiffre 8, il est pourri. Quand tu trouves un résultat qui vaut le double ou le triple de 5, continue tout droit, t'es sur la bonne voie! La somme est capricieuse! Si quand tu lances l'dé elle t'dit qu't'a trouvé l'ami Lapin, faut quand même que tu fasses c'que t'a dit l'dé! Si l'dé t'dit d'aller à gauche, et bah tu vas à gauche et l'Lapin Blanc sera là-bas. Si l'dé te dit droite, bah tu vas pas à gauche mais à droite! Enfin, t'as compris quoi. T'en fais pas, ça va bien se passer! Par contre, méfie-toi, si jamais l'dé t'sort un truc entre dix-huit et 21, pars tout de suite à gauche! Sinon, tu vas tomber sur la Reine des Cartes. Elle est complètement felée celle la! Elle va jamais t'laisser partir. Crois-moi, entre 18 et 21 inclus, tu pars à gauche fissa!!! Hé! T'sais quoi? Si l'dé donne un, regarde sur ta tête, y'à l'Lapin Blanc! Ca va? T'as tout retenu? Tu vois, c'pas si compliqué!". Alice avait la tête toute retournée avec tous ces nombres, mais elle lanca le dé et partit à la recherche du Lapin Blanc. Alors qu'elle s'éloignait du Koala, il lui cria "Ah j'oubliais! S'tu sais pas quoi faire, relance le dé!"

Écrivez une fonction `follow_the_white_rabbit` prototypée ainsi :

```
1 int follow_the_white_rabbit(void);
```

Cette fonction devra suivre le parcours d'Alice. Le lancer de dé se fera avec la fonction `random(3)`. Quand Alice doit aller à gauche, vous afficherez sur la sortie standard :

```
1 gauche
```

Si elle doit aller à droite, vous afficherez :

```
1 droite
```

Si Alice doit continuer tout droit, vous afficherez :

```
1 devant
```

Si elle doit faire demi-tour, vous afficherez :

```
1 derriere
```

Enfin, quand Alice trouve le Lapin Blanc, vous devrez afficher :

```
1 LAPIN !!!
```

La fonction devra retourner la somme de tous les lancers du dé.

Toutes les sorties se font sur la sortie standard et doivent être suivies d'un retour à la ligne.


Vous ne devez rendre qu'UNE SEULE fonction. Ne rendez pas de fonction `main`, la koalinette s'en chargera pour vous. De même, vous ne devez pas appeler `srandom(3)` dans votre fonction.

Exemple :

```
1 $>./follow_the_white_rabbit | cat -e
2 gauche$
3 droite$
4 droite$
5 devant$
6 derriere$
7 derriere$
8 LAPIN !!!$
```

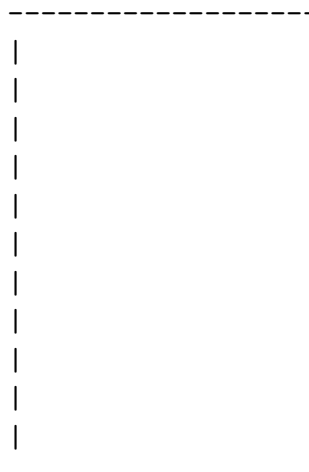
Chapitre III

Exercice 1

	Exercice : 01	points : 2
Éponge de Menger		
Répertoire de rendu: (DÉPOT SVN - piscine_cpp_d01-promo-login_x)/ex01		
Compilateur : gcc	Flags de compilation: -Wextra -Werror -Wall -std=gnu99	
Makefile : Oui	Règles : all, clean, fclean, re	
Fichiers a rendre : menger.c, main.c		
Remarques : n/a		
Fonctions Interdites : atoi		

L'éponge de Menger est une figure fractale basée sur des carrés. Son principe est simple. Il consiste a découper un carré en 9 carrés identiques. Celui du centre est "vide". Aux 8 carrés restants, on applique le même procédé.

Ainsi, le carré suivant (supposons que c'est un carré) :



se découpe de la manière suivante :

	0,0		1,0		2,0			

	0,1		1,1		2,1			

	0,2		1,2		2,2			

Le carré (1,1) est marqué comme vide et les 8 carrés restants sont marqués comme pleins. À chaque étape, on applique le même procédé aux carrés pleins.

En marquant les carrés vides par des espaces, et les pleins par des #, on a :

Niveau 0

	#	#	#		#	#	#	
	#	#	#		#	#	#	
	#	#	#		#	#	#	

	#	#	#		#	#	#	
	#	#	#		#	#	#	
	#	#	#		#	#	#	

	#	#	#		#	#	#	
	#	#	#		#	#	#	
	#	#	#		#	#	#	

Niveau 1

	#	#	#		#	#	#		#	#	#
	#	#	#		#	#	#		#	#	#
	#	#	#		#	#	#		#	#	#

	#	#	#						#	#	#
	#	#	#						#	#	#
	#	#	#						#	#	#

	#	#	#		#	#	#		#	#	#
	#	#	#		#	#	#		#	#	#
	#	#	#		#	#	#		#	#	#

Niveau 2

	#	#	#		#	#	#		#	#	#	
	#		#		#		#		#		#	
	#	#	#		#	#	#		#	#	#	

	#	#	#						#	#	#	
	#		#						#		#	
	#	#	#						#	#	#	

Écrivez un programme 'menger' qui, pour chaque niveau, affiche la taille et la position du carré vide ainsi que celles des sous-carrés. Toutes les valeurs doivent être affichées sur 3 chiffres et séparées par un espace.

Votre programme prendra en argument la taille du carré original ainsi que le nombre de niveaux désirés.

- La taille des carrés est TOUJOURS une puissance de 3.
- La profondeur est TOUJOURS inférieure ou égale à cette puissance de 3.

```
1 > ./menger square_size level
```



Exemples de sortie attendue :

```
1 $> ls
2 Makefile main.c menger menger.c
3
4 $> ./menger 3 1
5 001 001 001
6
7 $> ./menger 9 1
8 003 003 003
9
10 $> ./menger 9 2
11 003 003 003
12 001 001 001
13 001 001 004
14 001 001 007
15 001 004 001
16 001 004 007
17 001 007 001
18 001 007 004
19 001 007 007
20
21 $> ./menger 27 3 | head -n 29
22 009 009 009
23 003 003 003
24 001 001 001
25 001 001 004
26 001 001 007
27 001 004 001
28 001 004 007
29 001 007 001
30 001 007 004
31 001 007 007
32 003 003 012
33 001 001 010
34 001 001 013
35 001 001 016
36 001 004 010
37 001 004 016
38 001 007 010
39 001 007 013
40 001 007 016
41 003 003 021
42 001 001 019
43 001 001 022
44 001 001 025
45 001 004 019
46 001 004 025
47 001 007 019
48 001 007 022
49 001 007 025
```

```
50 003 012 003  
51 $>
```

Chapitre IV

Exercice 2

	Exercice : 02	points : 2
Le format BMP		
Répertoire de rendu: (DÉPOT SVN - piscine_cpp_d01-promo-login_x)/ex02		
Compilateur : gcc	Flags de compilation: -Wextra -Werror -Wall -std=gnu99	
Makefile : Non	Règles : n/a	
Fichiers a rendre : bitmap.h, bitmap_header.c		
Remarques : n/a		
Fonctions Interdites : Aucune		

Nous allons nous intéresser au format BMP.

Le format BMP est composé de trois éléments obligatoires :

- un en-tête du fichier (Bitmap file header);
- un en-tête de l'image (Bitmap information header);
- le codage de l'image.

L'en-tête du fichier est composé de 5 champs :

- un nombre magique dont la valeur doit être 0x424D sur 2 octets;
- la taille du fichier sur 4 octets;
- un champ réservé de 2 octets dont la valeur doit être 0;
- un autre champ réservé de 2 octets dont la valeur doit être 0;
- l'adresse dans le fichier où commence le codage de l'image (l'offset).

Différentes versions de l'en-tête de l'image existent. La plus répandue (pour des raisons

de rétro-compatibilité) est composée de 11 champs :

- la taille de l'en-tête (40 octets dans notre cas) sur 4 octets;
- la largeur de l'image sur 4 octets signés;
- la hauteur de l'image sur 4 octets signés;
- le nombre de niveaux de couleurs utilisés sur 2 octets;
- le nombre de bits par pixel sur 2 octets (les valeurs peuvent être 1, 2, 4, 8, 16 et 32);
- la méthode de compression utilisée sur 4 octets, prend la valeur 0 quand il n'y a pas de compression;
- la taille de l'image sur 4 octets (différent de la taille du fichier);
- la résolution horizontale de l'image sur 4 octets signés;
- la résolution verticale de l'image sur 4 octets signés;
- la taille de la palette de couleurs (0 dans notre cas) sur 4 octets;
- le nombre de couleurs importantes utilisées sur 4 octets, prend la valeur 0 quand toutes les couleurs sont importantes.

Tous les champs de ces deux en-têtes sont non-signés sauf contre-indication.

Dans un fichier `bitmap.h`, créez 2 structures `t_bmp_header` et `t_bmp_info_header` qui représentent respectivement l'en-tête du fichier et l'en-tête de l'image.

La structure `t_bmp_header` devra avoir les membres suivants :

- `magic;`
- `size;`
- `__app1;`
- `__app2;`
- `offset;`

La structure `t_bmp_info_header` devra avoir les membres suivants :

- `size;`
- `width;`
- `height;`
- `planes;`

- bpp;
- compression;
- raw_data_size;
- h_resolution;
- v_resolution;
- palette_size;
- important_colors.

Tous les membres de ces deux structures auront pour type un des types suivants :

- int16_t;
- uint16_t;
- int32_t;
- uint32_t;
- int64_t;
- uint64_t;

Ces types sont définis dans `stdint.h`.

Dans un fichier `bitmap_header.c`, écrivez ensuite deux fonctions `make_bmp_header` et `make_bmp_info_header` qui initialiseront tous les membres des structures.

Dans la mesure où toutes les images que nous allons créer sont carrées, la largeur de l'image sera égale à sa hauteur.

La fonction `make_bmp_header` est prototypée ainsi :

```
1 void make_bmp_header(t_bmp_header * header, size_t size);
```

Ses paramètres sont :

- header : un pointeur sur la structure `t_bmp_header` à initialiser.
- size : la longueur d'un des cotés de l'image.

La fonction `make_bmp_info_header` est ainsi prototypée :

```
1 void make_bmp_info_header(t_bmp_info_header * header, size_t size);
```

Ses paramètres sont :

- header : un pointeur sur la structure `t_bmp_info_header` à initialiser.
- size : la longueur d'un des cotés de l'image.

Quelques remarques sur les images que nous allons créer :

- le nombre de niveaux de couleurs est toujours 1;
- le nombre de bits par pixels est toujours 32;
- il n'y a pas de compression;
- les résolutions horizontales et verticales valent toujours 0;
- la taille de la palette de couleurs est toujours 0;
- toutes les couleurs de nos images sont importantes.



Si vous êtes sur une machine en little endian (comme c'est le cas sur les architectures intel), le membre magic dans `t_bmp_header` devrait avoir ses 2 octets inversés dans l'initialisation. En effet, les octets d'un nombre sont inversés dans la représentation mémoire sur une machine en little endian.



Par défaut, le compilateur applique du padding sur les structures.

```
1 $> cat padding.c
2 #include <stdlib.h>
3 #include <stdio.h>
4
5 struct foobar
6 {
7     char foo[2];
8     int bar;
9 };
10
11 int main(void)
12 {
13     printf('%zu\n', 2 * sizeof(char) + sizeof(int));
14     printf('%zu\n', sizeof(struct foobar));
15     return EXIT_SUCCESS;
16 }
17 }
```

```

18
19 $> gcc padding.c && ./a.out
20 6
21 8
22 $>
23 $>

```

La taille de la structure est supérieure à la somme de la taille de ses membres. Le compilateur a aligné les membres de la structure `foobar`. Pour éviter ce comportement, il faut donner un attribut à la structure.

L'attribut `packed` dit explicitement au compilateur de ne pas appliquer de padding à la structure.

```

1 $> cat padding.c
2 #include <stdlib.h>
3 #include <stdio.h>
4
5 struct __attribute__((packed)) foobar
6 {
7     char foo[2];
8     int bar;
9 };
10
11 int main(void)
12 {
13     printf("%zu\n", 2 * sizeof(char) + sizeof(int));
14     printf("%zu\n", sizeof(struct foobar));
15     return EXIT_SUCCESS;
16 }
17 $> gcc padding.c && ./a.out
18 6
19 6
20 $>
21 $>

```

La taille de la structure est égale à la somme de la taille de ses membres.

Exemple d'un main qui crée une image de 32x32 pixels entièrement blanche :

```

1 $> cat main.c
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <fcntl.h>
5
6 #include "bitmap.h"

```


```

7
8 int main(void)
9 {
10     t_bmp_header header;
11     t_bmp_info_header info;
12     int d;
13     uint32_t pixel = 0x00FFFFFF;
14
15     make_bmp_header(&header, 32);
16     make_bmp_info_header(&info, 32);
17
18     /* Ah le gros degueulasse qui verifie pas ses valeurs de retour !!! */
19     d = open("32px.bmp", O_CREAT | O_TRUNC | O_WRONLY, 0644);
20     write(d, &header, sizeof(header));
21     write(d, &info, sizeof(info));
22     for (size_t i = 0; i < 32 * 32; ++i)
23         write(d, &pixel, sizeof(pixel));
24     close(d);
25     return EXIT_SUCCESS;
26 }
27 $> hexdump -C 32px.bmp | head -n 6
28 00000000 42 4d 36 10 00 00 00 00 00 00 36 00 00 00 28 00 |BM6.....6...(.|
29 00000010 00 00 20 00 00 00 20 00 00 00 01 00 20 00 00 00 |.. ... ..|
30 00000020 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
31 00000030 00 00 00 00 00 00 ff ff ff 00 ff ff ff 00 ff ff |.....|
32 00000040 ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff |.....|
33 *
34 $>
35 $>

```


Chapitre V

Exercice 3

	Exercice : 03	points : 2
Dessine moi un carré		
Répertoire de rendu: (DÉPOT SVN - piscine_cpp_d01-promo-login_x)/ex03		
Compilateur : gcc	Flags de compilation: -Wextra -Werror -Wall -std=gnu99	
Makefile : Non	Règles : n/a	
Fichiers a rendre : drawing.h, drawing.c		
Remarques : n/a		
Fonctions Interdites : Aucune		

Nous allons maintenant voir comment remplir les images que nous avons créées.

Le contenu d'une image dans le format BMP se trouve dans la troisième section du fichier : le codage de l'image. L'image est stockée comme une série de lignes. Toutes les lignes sont stockées consécutivement. Ainsi, on peut considérer une image BMP comme un tableau à 2 dimensions dont chaque pixel est une case du tableau. L'origine de ce tableau se situe dans le coin inférieur gauche de l'image.

Dans les images avec 32 bits par pixel, chaque pixel est stocké sur 4 octets (normal, à priori). Dans notre cas, le premier octet sera toujours à 0. Les trois octets restant représentent respectivement les composantes rouge, verte et bleue (RGB) de chaque pixel. Voici quelques exemples de pixels avec leur couleur :

Noir		0x00000000
Blanc		0x00FFFFFF
Rouge		0x00FF0000
Vert		0x0000FF00
Bleu		0x000000FF
Jaune		0x00FFFF00

Dans un fichier `drawing.h`, créez un type `t_point` composé de deux entiers non-signés.

Ses deux membres sont :

- x;
- y.

Le membre **x** représente l'abscisse d'un point d'un plan et **y** son ordonnée.

Dans un fichier **drawing.c**, écrivez une fonction **draw_square** qui reçoit un tableau à deux dimensions représentant une image et qui dessine un carré d'une taille donnée à une position donnée.

Son prototype est le suivant :

```
1 void draw_square(uint32_t ** img, t_point * orig, size_t size, uint32_t
2 color);
```

Ses paramètres sont :

- **img**, un tableau à deux dimensions qui représente l'image;
- **orig**, les coordonnées du coin inférieur gauche du carré;
- **size**, la taille d'un côté du carré;
- **color**, la couleur du carré.

Cette fonction devra être prototypée dans **drawing.h**.


Exemple d'une fonction main qui réutilise les fonctions de l'exercice précédent et qui génère une image cyan de 64 pixels de côté avec un carré rouge dans le quart inférieur gauche.

```
1 $> cat main.c
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <fcntl.h>
5 #include <string.h>
6
7 #include "drawing.h"
8 #include "bitmap.h"
9
10 int main(void)
11 {
12     t_bmp_header header;
13     t_bmp_info_header info;
14     unsigned int *buffer;
15     unsigned int **img;
16     t_point p = {0, 0};
```

```
17     size_t size = 64;
18     int d;
19
20     /* Creation d'un tableau a deux dimensions. */
21     buffer = malloc(size * size * sizeof(*buffer));
22     img = malloc(size * sizeof(*img));
23     memset(buffer, 0, size * size * sizeof(*buffer));
24     for (size_t i = 0; i < size; ++i)
25         img[i] = buffer + i * size;
26
27     make_bmp_header(&header, size);
28     make_bmp_info_header(&info, size);
29
30     draw_square(img, &p, size, 0x0000FFFF);
31     p.x = 10;
32     p.y = 10;
33     draw_square(img, &p, 22, 0x00FF0000);
34
35     d = open("square.bmp", O_CREAT | O_TRUNC | O_WRONLY, 0644);
36     write(d, &header, sizeof(header));
37     write(d, &info, sizeof(info));
38     write(d, buffer, size * size * sizeof(*buffer));
39     close(d);
40     return EXIT_SUCCESS;
41 }
42 $>
```

Chapitre VI

Exercice 4

	Exercice : 04	points : 4
Dessine moi une éponge		
Répertoire de rendu: (DÉPOT SVN - piscine_cpp_d01-promo-login_x)/ex04		
Compilateur : gcc	Flags de compilation: -Wextra -Werror -Wall -std=gnu99	
Makefile : Oui	Règles : all, clean, fclean, re	
Fichiers a rendre : drawing.h, drawing.c, bitmap.h, bitmap_header.c, menger.c, main.c		
Remarques : n/a		
Fonctions Interdites : Aucune		

Maintenant que vous savez créer des fichiers bitmaps et y dessiner des carrés, vous avez tous les éléments pour dessiner une face d'une éponge de Menger.

Écrivez un programme menger_face qui génère une image d'une taille donnée qui représente une face d'une éponge de Menger à un niveau donné.

Votre programme prendra en argument le nom du fichier a créer, la taille d'un côté de l'image et le niveau de l'éponge de Menger voulu. Si le nombre d'argument est incorrect, vous devrez retourner une valeur différente de 0 et afficher sur la sortie d'erreur le message suivant suivi d'un retour à la ligne.

```
1 menger_face file_name size level
```

Les trois arguments reçus sont les suivants :

- le nom du fichier à créer;
- la taille d'un côté de l'image;
- le niveau de profondeur désiré.

Les carrés pleins doivent être dessinés en noir. Les carrés vides doivent être gris. Leur couleur dépend du niveau de profondeur courant. Chaque composante doit valoir $0xFF$ divisé par le nombre de niveaux restants plus un. Ainsi, les plus petits carrés vides de l'éponge seront toujours blancs.



Une couleur est grise lorsque ses trois composantes sont égales.


Exemple pour une éponge de profondeur totale 3 :

Niveau	Couleur		

1	255 / 3	0x00555555	
2	255 / 2	0x007F7F7F	
3	255 / 1	0x00FFFFFF	

Chapitre VII

Exercice 5

	Exercice : 05	points : 3
Doit y'avoir une belle vue de là haut		
Répertoire de rendu: (DÉPOT SVN - piscine_cpp_d01-promo-login_x)/ex05		
Compilateur : gcc	Flags de compilation: -Wextra -Werror -Wall -std=gnu99	
Makefile : Non	Règles : n/a	
Fichiers a rendre : pyramid.c		
Remarques : n/a		
Fonctions Interdites : Aucune		

Vous êtes coincé au sommet d'une pyramide. Chaque pièce de cette pyramide donne accès à deux pièces voisines de l'étage inférieur.

1	0
2	1 2
3	3 4 5
4	6 7 8 9

Ainsi, de la pièce 0, on peut accéder aux pièces 1 et 2. À partir de la pièce 2, on accède aux pièces 4 et 5. De la pièce 4, on accède aux pièces 7 et 8.

Vous ne possédez que le plan de la pyramide où vous êtes. Ce plan indique la distance entre chaque pièce :

1	0
2	7 4
3	2 3 6
4	8 5 9 3

Depuis le sommet, il y a 7 mètres jusqu'à la pièce de gauche et 4 seulement jusqu'à la pièce de droite.

Votre but est de trouver le chemin le plus court pour sortir de la pyramide. Dans notre exemple, ce serait :

```

1           0 + 4 + 3 + 5
2      soit
3           12

```

Dans un fichier `pyramid.c`, écrivez une fonction `pyramid_path` dont le prototype est le suivant :

```

1 int pyramid_path(int size, int ** map);

```

Cette fonction retourne la longueur du chemin parcouru pour sortir de la pyramide. Elle prend en paramètre :

- `size`, la hauteur de la pyramide;
- `map`, un tableau à deux dimensions contenant le plan de la pyramide.

Dans l'exemple précédent, le paramètre `map` serait ainsi déclaré :

```

1 {
2     {0},
3     {7, 4},
4     {2, 3, 6},
5     {8, 5, 9, 3}
6 };

```



Vous ne DEVEZ PAS rendre de fonction main.

Plan d'une pyramide plus complexe :

```


1           00
2         95 64
3       17 47 82
4     18 35 87 10
5   20 04 82 47 65
6 19 01 23 75 03 34
7 88 02 77 73 07 63 67
8 99 65 04 28 06 16 70 92

```

```
9      41 41 26 56 83 40 80 70 33
10     41 48 72 33 47 32 37 16 94 29
11     53 71 44 65 25 43 91 52 97 51 14
```


Chapitre VIII

Exercice 6

	Exercice : 06	points : 3
Collège Foo Foo Foo		
Répertoire de rendu: (DÉPOT SVN - piscine_cpp_d01-promo-login_x)/ex06		
Compilateur : gcc	Flags de compilation: -Wextra -Werror -Wall -std=gnu99	
Makefile : Non	Règles : n/a	
Fichiers a rendre : ex_6.h		
Remarques : n/a		
Fonctions Interdites : Aucune		

Écrivez le fichier `ex_6.h` de telle sorte que le code suivant compile et affiche la sortie attendue.

```

1 $>cat main.c
2 #include <stdlib.h>
3 #include <stdio.h>
4
5 #include "ex_6.h"
6
7 int main(void)
8 {
9     t_foo foo;
10
11     foo.bar = 0;
12     foo.foo.foo = 0xCAFE;
13     printf("%d\n", sizeof(foo) == sizeof(foo.foo));
14     printf("%d\n", sizeof(foo.foo.bar.foo) == sizeof(foo.foo.foo));
15     printf("%d\n", sizeof(foo.bar) == 2 * sizeof(foo.foo.bar));
16     printf("%d\n", sizeof(foo.foo.foo) == sizeof(foo.foo.bar.bar));
17     printf("%08X\n", foo.bar);
18     return EXIT_SUCCESS;
19 }
20 $>ls
21 ex_6.h main.c

```


```
22 $>gcc -Wall -Wextra -Werror -std=c99 main.c
23 $>./a.out
24 1
25 1
26 1
27 1
28 0000CAFE
29 $>
30 $>
```



Vous ne DEVEZ PAS rendre le fichier main.c

Chapitre IX

Exercice 7

	Exercice : 07	points : 3
Koalatchi		
Répertoire de rendu: (DÉPOT SVN - piscine_cpp_d01-promo-login_x)/ex07		
Compilateur : gcc	Flags de compilation: -Wextra -Werror -Wall -std=gnu99	
Makefile : Non	Règles : n/a	
Fichiers a rendre : koalatchi.c		
Remarques : n/a		
Fonctions Interdites : Aucune		

Nous allons nous intéresser au Koalatchi, un ancêtre réputé du tamagotchi.

Pour les incultes, le Koalatchi est un Koala virtuel dont le propriétaire doit prendre soin afin que le Koalatchi devienne un être surpuissant, prêt à dominer le monde.

Cependant, comme nous sommes feignants et qu'élever un Koala est une tâche longue et éprouvante (même s'il s'agit d'un Koala virtuel), nous allons utiliser la magnifique API (Application Programming Interface) dont ses Créateurs ont doté le Koalatchi. Cette API, par l'intermédiaire de messages, va nous permettre de connaître et de répondre aux besoins de notre Koalatchi.

Chaque message est composé d'une en-tête de 4 octets et peut éventuellement comporter une chaîne de caractères.

Un message peut être de 3 types :

- Request, a lieu quand le Koala veut demander quelque chose à son maître, ou alors quand le maître veut demander à son Koala de faire une action particulière;
- Notification, a lieu quand le Koala veut informer son maître d'une chose qu'il fait, et réciproquement;

- Error, a lieu quand le Koala se retrouve dans une situation impossible (qui peut notamment engendrer la mort du Koala...).

Chaque message a un domaine d'application particulier. Ces domaines sont :

- Nutrition;
- Entertainment;
- Education.

Cependant, un message ne peut avoir qu'un seul type et un seul domaine d'application.

L'en-tête d'un message est composé de la façon suivante :

- 1 octet spécifiant le type du message. Il peut prendre les valeurs suivantes :
 - Notification : valeur 1
 - Request : valeur 2
 - Error : valeur 4
- 1 octet spécifiant le domaine d'application du message. Il peut prendre les valeurs suivantes :
 - Nutrition : valeur 1;
 - Entertainment : valeur 2;
 - Education : valeur 4;
- 2 octets décrivant le message par une valeur unique.

Voici, par domaine d'application, la liste de tous les messages possibles :

- Nutrition :
 - Notification
 - Eat : le Maître nourrit son Koala
+ Valeur des 2 derniers octets : 1
 - Defecate : le Koala défèque
+ Valeur des 2 derniers octets : 2

- Request
 - Hungry : le Koala a faim
 - + Valeur des 2 derniers octets : 1
 - Thirsty : le Koala a soif
 - + Valeur des 2 derniers octets : 2
- Error
 - Indigestion : le Koala a une indigestion
 - + Valeur des 2 derniers octets : 1
 - Starving : le Koala meurt de faim
 - + Valeur des 2 derniers octets : 2
- Entertainment :
 - Notification
 - Ball : le Koala joue avec une balle
 - + Valeur des 2 derniers octets : 1
 - Bite : le Koala mord son Maitre (ça defoule!)
 - + Valeur des 2 derniers octets : 2
 - Request
 - NeedAffection : le Koala demande de l'affection à son Maitre
 - + Valeur des 2 derniers octets : 1
 - WannaPlay : le Koala ou le Maitre a envie de jouer
 - + Valeur des 2 derniers octets : 2
 - Hug : le Maitre et le Koala se font un calin
 - + Valeur des 2 derniers octets : 3
 - Error
 - Bored : le Koala s'ennuit à mourrir
 - + Valeur des 2 derniers octets : 1
- Education :

- Notification :
 - TeachCoding : le Maitre enseigne au Koala à coder
+ Valeur des 2 derniers octets : 1
 - BeAwesome : le Maitre apprend au Koala à etre AWESOME
+ Valeur des 2 derniers octets : 2
- Request :
 - FeelStupid : le Koala se sent stupide et veut apprendre
+ Valeur des 2 derniers octets : 1
- Error :
 - BrainDamage : la Koala a un tel mal de crane qu'il voit des flamants roses.
+ Valeur des 2 derniers octets : 1

Dans un fichier `koalatchi.c`, définissez la fonction `prettyprint_message` dont le prototype est le suivant :

```
1 int prettyprint_message(uint32_t header, char const * content);
```

Ses paramètres sont :

- header, l'en-tête du message;
- content, le contenu du message.

Cette fonction devra afficher le détail d'un message de manière humainement lisible selon le format suivant :

```
1 TYPE DOMAINE ACTION [CONTENU]
```

Si le paramètre `content` est `NULL`, alors rien n'est affiché après l'action.

Si le message est valide, la fonction retourne 0, sinon elle retourne 1. Dans le cas où un message est invalide, la fonction devra afficher :

```
1 Invalid message.
```

Toutes les sorties se font sur la sortie standard et devront être suffixées d'un retour à la ligne.



L'utilisation des unions est INTERDITE!!!

Vous devez utiliser au moins deux des operateurs suivants :

```
1 << >> & | ^ ~
```

Voici un exemple de main :

```
1 $> cat main.c
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <stdint.h>
5
6 int prettyprint_message(uint32_t, char const *);
7
8 int main(void)
9 {
10     prettyprint_message(0x00COFFEE, "Needed !");
11
12     prettyprint_message(0x02010001, "\"Kreog !\"");
13     prettyprint_message(0x01010001, "Eucalyptus");
14     prettyprint_message(0x01010002, "\"CACA !\"");
15     prettyprint_message(0x01010001, "Keytronic");
16     prettyprint_message(0x04010001, NULL);
17
18     /* Dark voodoo incantations to resurect the Koala. */
19
20     prettyprint_message(0x02020001, NULL);
21     prettyprint_message(0x01040002, NULL);
22     prettyprint_message(0x01020002, "\"KREOG !!!\"");
23     prettyprint_message(0x01040001, "Brainfuck");
24     prettyprint_message(0x04040001, "\"Dark Moon of the side...\"");
25
26     return 0;
27 }
28 $> gcc -Wall -Wextra -Werror -std=gnu99 main.c koalatchi.c
29 $> ./a.out | cat -e
30 Invalid message.$
31 Request Nutrition Hungry "Kreog !"$$
32 Notification Nutrition Eat Eucalyptus$
33 Notification Nutrition Defecate "CACA !"$$
34 Notification Nutrition Eat Keytronic$
35 Error Nutrition Indigestion$
36 Request Entertainment NeedAffection$
37 Notification Education BeAwesome$
```


```
38 Notification Entertainment Bite "KREOG !!!"$  
39 Notification Education TeachCoding Brainfuck$  
40 Error Education BrainDamage "Dark Moon of the side..."$  
41 $>  
42 $>
```



Vous ne DEVEZ PAS rendre de main

Chapitre X

Exercice 8

	Exercice : 08	points : 3
Log		
Répertoire de rendu: (DÉPOT SVN - piscine_cpp_d01-promo-login_x)/ex08		
Compilateur : gcc	Flags de compilation: -Wextra -Werror -Wall -std=gnu99	
Makefile : Non	Règles : n/a	
Fichiers a rendre : log.h, log.c		
Remarques : n/a		
Fonctions Interdites : Aucune		

Les logs ont une part très importante dans l'informatique. Ils permettent de garder des traces de tout ce qui est fait. Un simple coup d'oeil dans /var/log montre la quantité d'informations qui sont gardées, soit par des applications diverses, soit par le système d'exploitation lui-même.

Nous allons écrire quelques fonctions de log. Ces fonctions devront permettre de choisir où iront les messages que nous allons logger. Il devra donc être possible d'écrire sur la sortie standard, la sortie d'erreur ou encore dans un fichier de notre choix. Par défaut, l'affichage se fera sur la sortie d'erreur. Si un programme loggue des messages dans un fichier, chaque message devra être ajouté à la fin du fichier.

De plus, chaque message aura un niveau de log associé. Ces niveaux sont repris de syslog(3) et sont :

- Error
- Warning
- Notice
- Info
- Debug

Tous ces niveaux seront définis dans une énumération nommée `LogLevel`.

Par ailleurs, il devra être possible de choisir le niveau maximum de log désiré. Par exemple, si le niveau maximum voulu est `Warning`, seuls les messages notés `Error` ou `Warning` devront être loggués. Par défaut, seuls les messages d'erreur devront être loggués.

Les messages devront tous être formatés de la manière suivante :

```
1 Date [Niveau]: Message
```

La date devra être formatée de la même manière que celle renvoyée par `ctime(3)`.

Dans un fichier `log.h`, écrivez une enum `LogLevel` qui contient tous les énumérateurs définis précédemment.

Dans un fichier `log.c`, implémentez les fonctions suivantes :

```
1 enum LogLevel get_log_level(void);
2 enum LogLevel set_log_level(enum LogLevel);
3 int set_log_file(char const *);
4 int close_log_file(void);
5 int log_to_stdout(void);
6 int log_to_stderr(void);
7 void log_msg(enum LogLevel, char const * fmt, ...);
```

La fonction `get_log_level` retourne le niveau de log courant.

La fonction `set_log_level` détermine le niveau de log à utiliser. Ce niveau est reçu en paramètre. Si le niveau de log souhaité n'existe pas, le niveau courant est conservé. Elle retourne le niveau de log courant à la fin de l'appel.

La fonction `set_log_file` permet de choisir le fichier dans lequel les messages vont être écrits. Elle prend en paramètre le nom du fichier cible. Si un autre fichier était déjà ouvert, elle le ferme préalablement. Elle retourne 0 quand il n'y a pas d'erreur, 1 sinon.

La fonction `close_log_file` ferme le fichier de log courant s'il y en a un et remet la sortie de log sur la sortie d'erreur. Si aucun fichier n'était ouvert, cette fonction ne fait rien. Elle retourne 0 quand il n'y a pas d'erreur, 1 sinon.

La fonction `log_to_stdout` règle la sortie des messages sur la sortie standard. Si un fichier était déjà ouvert, elle le ferme préalablement. Elle retourne 0 quand il n'y a pas d'erreur, 1 sinon.

La fonction `log_to_stderr` règle la sortie des messages sur la sortie d'erreur. Si un fichier était déjà ouvert, elle le ferme préalablement. Elle retourne 0 quand il n'y a pas d'erreur, 1 sinon.

La fonction `log_msg` écrit un message sur la sortie attendue. Elle prend en argument le niveau de log du message, une chaîne de formatage printf-like et des arguments variadiques. Si le niveau de log demandé n'existe pas, cette fonction ne fait rien.

La destination des messages ainsi que le niveau de log courant doivent être conservés dans des variables globales qui ne doivent pas être accessibles par d'autres fonctions que celles définies dans l'unité de compilation `log.c`.

Quelques lectures conseillées : `fopen(3)`, `fprintf(3)`, `vfprintf(3)`, `ctime(3)` et `stdarg(3)`.

Voici un main d'exemple pour vous aider :

```
1 $>ls
2 log.c log.h main.c
3 $>cat main.c
4 #include <stdio.h>
5 #include <stdlib.h>
6
7 #include "log.h"
8
9 int main(void)
10 {
11     set_log_file("out.log");
12     set_log_level(Debug);
13     log_msg(Debug, "This is debug\n");
14     log_msg(42, "This should not be printed\n");
15     log_msg(Warning, "This is a warning\n");
16     set_log_level(Warning);
17     log_msg(Info, "This is info\n");
18     log_msg(Error, "KREOG !\n");
19     close_log_file();
20     return EXIT_SUCCESS;
21 }
22 $>gcc -Wall -Wextra -Werror -std=c99 main.c log.c
23 $>./a.out
```

```
24 $>ls
25 a.out log.c log.h main.c out.log
26 $>cat out.log
27 Tue Dec 7 01:08:19 2010 [Debug]: This is debug
28 Tue Dec 7 01:08:19 2010 [Warning]: This is a warning
29 Tue Dec 7 01:08:19 2010 [Error]: KREOG !
30 $>./a.out && cat out.log
31 Tue Dec 7 01:08:19 2010 [Debug]: This is debug
32 Tue Dec 7 01:08:19 2010 [Warning]: This is a warning
33 Tue Dec 7 01:08:19 2010 [Error]: KREOG !
34 Tue Dec 7 01:11:09 2010 [Debug]: This is debug
35 Tue Dec 7 01:11:09 2010 [Warning]: This is a warning
36 Tue Dec 7 01:11:09 2010 [Error]: KREOG !
37 $>
```



Vous ne DEVEZ PAS rendre de fonction main