KOALA

B3- C++ Pool

B-PAV-242

Day 01

C, Life, the Universe and everything else

v1.8

Day 01

C, Life, the Universe and everything else

repository name:	cpp_d01
repository rights:	ramassage-tek
language:	C++
group size:	1

- Your repository must contain the totality of your source files, but no useless files (binary, temp files, obj files,...).
- All the bonus files (including a potential specific Makefile) should be in a directory named *bonus*.
- Error messages have to be written on the error output, and the program should then exit with the 84 error code (0 if there is no error).

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, leads to 0 to the exercise.
Every class must possess a constructor and a destructor.

Every output goes to the standard output, and will be ended by a newline, unless specified otherwise.

Any use of "friend" will result in the grade -42 no questions asked.

To avoid compilation problems, please include necessary files within your headers.

Please note that none of your files must contain the main function except when explicitly asked. We will use our own main function to compile and test your code.

Exercise 00

Follow the White Rabbbit

repository subdir:	/ex00
compilation:	gcc -Wall -Wextra -Werror -std=gnu99
files to turn in:	white_rabbit.c
forbidden functions:	none
points:	2

Sitting on top of the hill, Alice was bored to death, wondering if making a chaplet of flowers was worth getting up and actually gathering said flowers.
And then, suddenly, a White Rabbit with pink eyes passed by, running like a madman. This was actually not really worth mentioning, and Alice wasn't much more puzzled to hear the Rabbit mumbling: "Oh my god, Oh my god! I'm going to be late!".

However, as soon as the Rabbit pulled a pocket watch out of his vest, looked at the time, and ran even faster, Alice was on her feet in a heartbeat. She suddenly realized that she had never seen either a rabbit with a vest pocket, nor a watch to pull out of it.
Dying to know more, she ran through the fields in the rabbit's wake, and was lucky enough to be right on time to see it rushing into a huge burrow, under the bushes.

Not a moment later was she already inside, not even wondering how the hell she would get out of there.
After drifting around for a long, long time, in the maze of the burrow's walls, Alice met a Koala whose words, more or less, were these: "Hey there ye! Wot's you doin' here? Lookin' fer the pink pony as well?".
"Not a pink pony but a white rabbit," Alice answered. "Aaah, but I know him well, th'old rabbit friend," the Koala retorted. "I even saw him not five minutes ago! 'Looked like he was in a hell of a hurry, th'old rabbit friend!".

Alice asked the Koala to show her the direction the Rabbit was heading. Without hesistating, the Koala pointed to his left and blurted out: "Thatta way!!", before suddenly pausing and pointing to the opposite direction.
"Err, nay... I think it wos rather thatta way...". After having pointed to a dozen different directions, the Koala finally admitted, "Hmm... Actually, I think I may well be lost."

Alice was in despair.
She was lost in a huge burrow, and was off the trail of the white rabbit. When he saw her in this state, the Koala took pity on her, and told her: "Dun' worry there gal, we'll find your friend th'White Rabbit.
"Look what I got here."
He immediately took a 37-sided dice out of his vest pocket (yes, he also has a vest pocket) and handed it to Alice.
He showed each of the sides to Alice.
"This is the first side - yeh can tell cause o'the number 1 written on it. And this is side 2", and so on until reaching the 37^{th} one.

The Koala then told Alice: "What yeh got in yer hand, it's a magic die!
Yeh must take real good care of it! But it'll help yeh find the White Rabbit! Now, listen well, open yer ears, I'll explain to yeh how yeh must use it.
Each time yeh don't know where t'go anymore, throw the die. Th'result'll tell yeh which direction the White Rabbit took.
Although, if the die gives yeh a multiple of 11 and the weather is nice, y'should always take a nap - might as well enjoy the sun. 'Cause yeh can be sure the White Rabbit'll do the same.
But if the weather is crap, better throw the die again. Same thing when you wake up after the nap, 'f'course. If the weather is still nice and it tells ye to nap again, you woke up waaay too early!

If you ever get a 37, then you found the White Rabbit. Never forget that after a cup of tea, you should always go straight ahead. When you get a side that's higher than 24 and that three times this side gives you seven-times-ten-and-eight or 146, means the die was wrong, y'should turn and head back.
If the result is four or 5, go left, there's a caterpillar here that smokes his pipe all day long and blows smoke rings. Bit crazy he is, but quite funneh! With a 15, straight ahead with you. When the die says 12, ye're out of luck, that was for naught and yeh have to throw again."

"When th'result is 13, head to yer right. Also works with 34 or more.
Left it is, if the die says six, 17 or 28. A 23 means it's 10pm! Time for a cup of tea. Find a table, and order a lemon tea. If you don't like lemon, green tea is fine enough. Oh right, never forget to count all your results!
When you add'em and yeh find 421, yeh found the White Rabbit. Say hi for me, while you're at it. Oh, I need to tell you: that counting the results things, that also works with three-times-hundred-and-ninety-seven at least. Whenever you get a result that you can divide by 8 and get a round result with nothing left, just head back where you came from. That number 8 is crappy, I don't like it. When the result is twice or three times 5, keep going ahead, yer on the good way!
The sum though is quite a bugger, if the die tells you you found the White Rabbit, y'still need to do what the die told you to do! If it tells yeh t'go left, well y'go left and th'White Rabbit'll be there. If it tells right, then you don't go left, you go right! Well, you got it anyway. Dun worry, everything'll be fine.
Ah, still, be careful if the die tells you sumthing between 18 and 21, go left right away! Otherwise you'll end up meeting the Queen of Hearts. She's completely nuts, she'd never let you leave. Really, between 18 and 21 included, left as fast as yeh can! Hey, know what ? If you ever get a 1, look on top of yer head, means the Rabbit is here!
Got it? Remember all that? Y'see, th'nots so complicated."

Alice was head over heels with all those numbers, but she rolled the dice and went behind the White Rabbit. While she was fading away, the Koala yelled "Ah, I forgot! Whenever y'don't know what teh do, just throw the die again!"

Write a function called **follow_the_white_rabbit** with the following prototype:

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

This function must follow Alice's journey. You will use random(3) to simulate the dice being rolled. When Alice must go left, print the following on the standard output: `gauche`.

If she must go right, display `droite`.
If she must keep going straight ahead, display `devant`.
If she must head back, display `derriere`.
When she finds the White Rabbit, display `LAPIN !!!`.

The function must return the sum of all the results the die has given up to this point.
Every output will be on the standard output and will be followed by a new line.
You must provide ONE FUNCTION only. Do not provide a main function, we will take care of it.
It will also take care of srandom(3), so you must NOT call it yourself.

Here is an example:

Terminal

```
~/B-PAV-242>  ./follow_the_white_rabbit | cat -e
gauche$
droite$
droite$
devant$
derriere$
derriere$
LAPIN !!!$
```

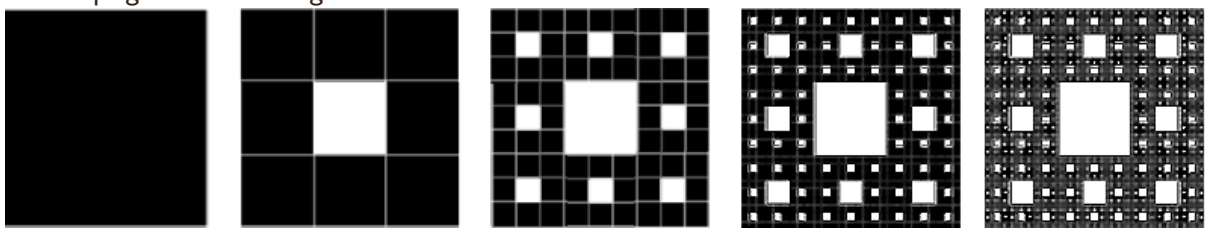
Exercise 01

The Menger Sponge

repository subdir:	/ex01
compilation:	via Makefile (rules: all, clean, fclean, re) (flags: -Wall -Wextra -Werror -std=gnu99)
files to turn in:	menger.c, main.c
forbidden functions:	atoi
points:	2

The Menger Sponge is a fractal curve that is based on squares.
The idea is simple: one square is to be separated into 9 smaller, identical squares. The middle one is removed.
The process is applied to the 8 other squares.

The first steps give the following result:



Write a program called **menger** which, for each level, displays the size, abscissa and ordinate of each empty square.
Every value must be displayed over 3 digits and be separated by one space.
Your program will take the size of the original square and the required number of levels as arguments.

The size of the square is ALWAYS a power of 3.
The depth is ALWAYS less than or equal to this power of 3.

Here are some examples:

Terminal

```
~/B-PAV-242> ls
Makefile main.c menger menger.c
~/B-PAV-242> ./menger 3 1
001 001 001
~/B-PAV-242> ./menger 9 1
003 003 003
~/B-PAV-242> ./menger 9 2
003 003 003
001 001 001
001 001 004
001 001 007
001 004 001
001 004 007
001 007 001
001 007 004
001 007 007
```

Terminal

```
~/B-PAV-242> ./menger 27 3 | head -m 29
009 009 009
003 003 003
001 001 001
001 001 004
001 001 007
001 004 001
001 004 007
001 007 001
001 007 004
001 007 007
003 003 012
001 001 010
001 001 013
001 001 016
001 004 010
001 004 016
001 007 010
001 007 013
001 007 016
003 003 021
001 001 019
001 001 022
001 001 025
001 004 019
001 004 025
001 007 019
001 007 022
001 007 025
003 012 003
```

Exercise 02

The BMP Format

repository subdir:	/ex02
compilation:	gcc -Wall -Wextra -Werror -std=gnu99
files to turn in:	bitmap.h, bitmap_header.c
forbidden functions:	none
points:	2

Let's study the BMP format for a while.
This format is composed of 3 mandatory elements:

- a file header (''Bitmap file header''),
- an image header (''Bitmap information header''),
- the encoded image.

The file header contains five fields:

- magic number, the value of which must be 0x424D on 2 bytes,
- the file size on 4 bytes,
- a reserved field of 2 bytes, holding the 0 value,
- another reserved field of 2 bytes, the value of which must be 0,
- the address where the encoded image begins in the file (its offset).

The image header exists in many different versions. The most usual (for backward compatibility reasons) is made of 11 fields:

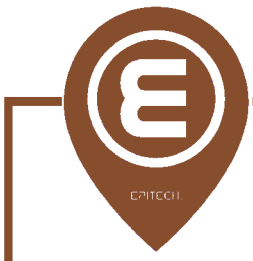
- the header size on 4 bytes (40 bytes in our case),
- the image's width on 4 signed bytes,
- the image's height on 4 signed bytes,
- the number of entries used in the color palette, on 2 bytes,
- the number of bits per pixel on 2 byts (possible values are 1,2,4,8,16 and 32),
- the compression method used on 4 bytes, set to 0 if there is no compression,
- the image's size on 4 bytes (which never equals the file's size),
- the image's horizontal resolution on 4 signed bytes,
- the image's vertical resolution on 4 signed bytes,
- the size of the color palette (0 in our case) on 4 bytes,
- the number of important colors used on 4 bytes. The value should be 0 when all the colors are equally important.

Unless otherwise specified, all the fields in those headers are unsigned.

In a bitmap.h file, create two **t_bmp_header** and **t_bmp_info_header** structures which represent the file header and image header.
The **t_bmp_header** structure must have the following fields:

- magic,
- size,
- _app1,
- _app2,
- offset.

The **t_bmp_info_header** structure must have the following fields:



- size,
- width,
- height,
- planes,
- bpp,
- compression,
- raw_data_size,
- h_resolution,
- v_resolution,
- palette_size,
- important_colors.

Each of these fields will be one of the following types:

- int16_t,
- uint16_t,
- int32_t,
- uint32_t,
- int64_t,
- uint64_t.

Those types are defined in **stdint.h**

In a file named **bitmap_header.c**, write two functions: **make_bmp_header** and **make_bmp_info_header**, which will initialize every member of the structures.
Since we will only create square-shaped images, an image's width will always be equal to its height.

The **make_bmp_header** function has the following signature:

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

header is a pointer to the **t_bmp_header** structure to initialize and *size* is the length of one of the image's sides.

The **make_bmp_info_header** has the following signature:

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

header is a pointer to the **t_bmp_info_header** structure to initialize and *size* is the length of one of the image's sides.

A few words about the pictures we're going to create:

- the number of entries in the color palette is always 1,
- the number of bits per pixel is always 32,
- there is no compression,
- the horizontal and vertical resolutions are always equal to 0,
- the size of the color palette is always 0.

All of our images' colors are important.

If you are on a little endian computer (or any Intel architecture, for example), the magic number's 2 bytes in t_bmp_header should have their order reversed.

Watch out! The compiler always applies padding on structures, unless otherwise specified.
Thus, the structure is bigger than the sum of the size of all of its members.
An attribute should be applied to the structure in order to avoid this behavior: the **packed** attribute explicitly asks the compiler not to apply padding to the following structure:

```c
// padding.c
#include <stdlib.h>
#include <stdio.h>

struct                          foobar
{
        char                    foo[2]
        int                     bar;
};

struct __attribute__((packed))  foobarP
{
        char                    foo[2];
        int                     bar;
};

int                             main(void)
{
        printf("total size: %zu bytes\n", 2*sizeof(char)+sizeof(int));
        printf("foobar size: %zu bytes\n", sizeof(struct foobar));
        printf("foobarP size: %zu bytes\n", sizeof(struct foobarP));
        return EXIT_SUCCESS;
}
```

Terminal

```
~/B-PAV-242> gcc padding.c && ./a.out
total size:  6 bytes
foobar size:  8 bytes
foobarP size:  6 bytes
```

The compiler aligned the foobar structure's fields.

Here is an example of a main function, which creates a 32x32 image that is entirely white:

```c
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include "bitmap.h"

int                             main(void)
{
        t_bmp_header            header;
        t_bmp_info_header       info;
        int                     d;
        uint32_t                pixel = 0x00FFFFFF;

        make_bmp_header(&header, 32);
        make_bmp_info_header(&info, 32);
        /* Not checking your return values is bad bad bad */
        d = open ("32px.bmp", O_CREAT | O_TRUNC | O\WRONLY, 0644);
        write(d, &header, sizeof(header));
        write(d, &info, sizeof(info));
        for (size_t i=0; i < 32 * 32; ++i)
                write(d, &pixel, sizeof(pixel));
        close(d);
        return EXIT_SUCCESS;
}
```

Exercise 03

Draw me a square

repository subdir:	/ex03
compilation:	gcc -Wall -Wextra -Werror -std=gnu99
files to turn in:	drawing.h, drawing.c
forbidden functions:	none
points:	2

The time has come to fill in the pictures that we've just created.
The image's actual content, as specified in the BMP format, can be found in the third section of the file: the encoded image.
It is stored as a set of lines, each line following the previous one.
We can consider that a BMP image is a two-dimensional array, with each of the array's elements being a pixel of our image. The origin of this array is located in the bottom-left corner of the image.
When an image has the 32 bits-per-pixel attribute, each pixel is encoded on 4 bytes (which is all well and good). In our situation, the first byte will always be equal 0. The next three bytes will represent the Red, Green and Blue (RGB) components of the pixel.

Here are a few examples:

Black	0x00000000
White	0x00FFFFFF
Red	0x00FF0000
Green	0x0000FF00
Blue	0x000000FF
Yellow	0x00FFFF00

In a **drawing.h** file, create a **t_point** type composed of two unsigned integers: x and y.

The x field represents the x-axis position of a point in a plane, and the y field its y-axis position.

In a file named **drawing.c**, write a **draw_square** function, which takes a two-dimensional array that represents an image as a parameter.
It will draw a square of a given size to that position, and be prototyped as follows:

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

- *img* is a two-dimensional array that represents the picture,
- *orig* is the position of the bottom-left corner of the square,
- *size* is the size of one of the square's sides,
- *color* is the color of the square to be drawn.

Write its prototype in a file named **drawing.h**.

Here is an instance of a main function, which reuses functions from the previous exercise and generates a cyan-colored 64x64 image, with a red square in the bottom-left quarter:

```c
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>

#include "drawing.h"
#include "bitmap.h"

int                             main(void)
{
        t_bmp_header            header;
        t_bmp_info_header       info;
        unsigned int            *buffer;
        unsigned int            **img;
        t_point                         p = {0,0};
        size_t                  size = 64;
        int                             d;

        /*Creates a two-dimensional array. */
        buffer = malloc(size * size * sizeof(*buffer));
        img    = malloc(size * sizeof(*img));
        memset(buffer, 0, size * sizeof(*buffer));
        for (size_t i = 0; i < size; ++i)
                img[i] = buffer + i * size;

        make_bmp_header(&header, size);
        make_bmp_info_header(&info, size);

        draw_square(img, &p, size, 0x0000FFFF);
        p.x = 10;
        p.y = 10;
        draw_square(img, &p, 22, 0x00FF0000);

        d = open("square.bmp", O_CREAT | O_TRUNC | O_WRONLY, 0644);
        write(d, &header, sizeof(header));
        write(d, &info, sizeof(info));
        write(d, buffer, size * size * sizeof(*buffer));
        close(d);
        return EXIT_SUCCESS;
}
```

Exercise 04

Draw me a sponge

repository subdir:	/ex04
compilation:	via Makefile (rules: all, clean, fclean, re) (flags: -Wall -Wextra -Werror -std=gnu99)
files to turn in:	drawing.h, drawing.c, bitmap.h, bitmap_header.c, menger.c, main.c
forbidden functions:	none
points:	4

You now have all the elements you need to draw a side of Menger's Sponge.
Write a **menger_face** program, which generates an image of a given size, that shows the side of a Menger's Sponge at a given depth.
This program will take the following as arguments: the name of the file to create, the size of the image's sides and the required depth for Menger's Sponge.

If the number of arguments is incorrect, you must return a value different than 0 and print the following message on the standard error output: "*menger_face file_name size level*\n".

Full squares will be black and empty squares will be gray. The colors will actually be closely related to the current depth of the Sponge.
Each component of the color will be equal to 0xFF, divided by the remaining depth level plus one. Therefore, the Sponge's smallest empty squares will always be white.

A color is considered gray whenever its three components have the same value.

For a total depth of 3, we would get:

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

Exercise 05

It must be nice from up there

repository subdir:	/ex05
compilation:	-Wall -Wextra -Werror -std=gnu99
files to turn in:	pyramid.c
forbidden functions:	none
points:	3

You are stuck at the top of a pyramid. Each room leads to two neighboring rooms on the lower floor.
The only thing in your possession is the map of the pyramid you're stuck in. It displays the distance between each room and the one above it:

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

From the top, there's 7 meters to reach the left room and only 4 to reach the room on the right.

Your goal is to find the shortest way to exit the pyramid. In our example, the shortest path out would be:

```
0 + 4 + 3 + 5
which equals 12
```

In a file named **pyramid.c**, write a function named **pyramid_path** with the following prototype:

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

This function returns the total distance traveled to get out of the pyramid, Its parameters are the height of the pyramid and a two-dimensional array containing the map.
In the previous example, the "map" parameter would be declared as follows:

Terminal

```
~/B-PAV-242> cat map
{
  {0}
   {7, 4},
  {2, 3, 6},
  {8, 5, 9, 3}
};
```

Here's a more complicatd pyramid:

```
                00
               95 64
              17 47 82
             18 35 87 10
            20 04 82 47 65
           19 01 23 75 03 34
          88 02 77 73 07 63 67
        99 65 04 28 06 16 70 92
      41 41 26 56 83 40 80 70 33
    41 48 72 33 47 32 37 16 94 29
  53 71 44 65 25 43 91 52 97 51 14
```

Exercise 06

Fook this, seriously

repository subdir:	/ex06
compilation:	-Wall -Wextra -Werror -std=gnu99
files to turn in:	ex_6.h
forbidden functions:	none
points:	2

Write the **ex_6.h** file needed for the following code to compile and print the expected output:

```c
#include <stdlib.h>
#include <stdio.h>

#include "ex_6.h"

int             main(void)
{
        t_foo   foo;

        foo.bar = 0;
        foo.foo.foo = 0xCAFE;
        printf("%d\n", sizeof(foo) == sizeof(foo.foo));
        printf("%d\n", sizeof(foo.foo.bar.foo) == sizeof(foo.foo.foo));
        printf("%d\n", sizeof(foo.bar) == 2 * sizeof(foo.foo.bar));
        printf("%d\n", sizeof(foo.foo.foo) == sizeof(foo.foo.bar.bar));
        printf("%08X\n", foo.bar);
        return EXIT_SUCCESS;
}
```

Terminal

```
~/B-PAV-242> ls
ex_6.h main.c
```

Terminal

```
~/B-PAV-242> gcc -Wall -Wextra -Werror -std=c99 main.c && ./a.out
1
1
1
1
0000CAFE
```

You must NOT provide the main.c file

Exercise 07

Koalatchi

repository subdir:	/ex07
compilation:	-Wall -Wextra -Werror -std=gnu99
files to turn in:	koalatchi.c
forbidden functions:	none
points:	3

We are now going to study the Koalatchi, well-known ancestor of the tamagotchi.
For the uncultured swine among us, a Koalatchi is a virtual Koala. Its owner must take care of it so that the Koalatchi might become an overpowered being, able to take over the world.
The only problem is our underlying laziness, and raising a Koala is a long and arduous task (even when it's a virtual one).
We're going to use the wonderful API (Application Programming Interface) that its creators have given the Koalatchi. This API, which uses pre-determined messages, will allow us to acknowledge and take care of our Koalatchi's needs. Each message is composed of a 4-byte header and can possibly contain a character string.

There are three types of messages:

- **request** occurs when the Koala wants to ask its owner something, or when the owner wants his/her Koala to perform a specific action,
- **notification** occurs when a Koala wants to inform its owner about something it just did and vice-versa,
- **error** occurs when the Koala encounters an impossible situation (which can induce various hazards, such as death).

Each message has a specific domain of application: **Nutrition**, **Entertainment** or **Education**.
However, a message can only have a single type and a single domain of application.
The message header is composed as follows:

- the message's type on 1 byte. The available values are 1 (notification), 2 (request) and 4 (error),
- the domain of application on 1 byte. The available values are 1 (nutrition), 2 (entertainment) and 4 (education),
- a unique value describing the message on 2 bytes.

Here are, for each domain, all the possible messages that can be emitted:

Nutrition	notification	Eat	owner feeds Koala	1
	notification	Defecate	Koala defecates	2
	request	Hungry	Koala is hungry	1
	request	Thirsty	Koala is thirsty	2
	error	Indigestion	Koala has indigestion	1
	error	Starving	Koala is starving	2

Entertainment	notification	Ball	Koala plays with a ball	1
	notification	Bite	Koala bites owner	2
	request	NeedAffection	Koala needs love and kindness from owner	1
	request	WannaPlay	Koala wants owner to play	2
	request	Hug	Koala and owner cuddle	3
	error	Bored	Koala is bored to death	1

Education	notification	TeachCoding	owner teaches Koala how to code	1
	notification	BeAwesome	owner teaches Koala how to be awesome	2
	request	FeelStupid	Koala feels stupid and craves knowledge	1
	error	BrainDamage	Koala has such a headache it can see flamingos	1

In a file named **koalatchi.c**, write a function called **prettyprint_message** with the following prototype:

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

Its paramaters are the message header and the content of the message.

This function must display every detail of the message in a human-readable way, by using the following format:

```
TYPE DOMAIN ACTION [CONTENT]
```

If the parameter content is NULL, do not print anything after the action.
If the message is valid, the function returns 0. Otherwise, it will return 1.
If a message is not valid, the function must print "*Invalid message.*". You should print everything on the standard output, followed by a newline.

Using unions is FORBIDDEN

You must use at least two of the following operators: *«, », &, |, ^, ~*.

Here is an example of a main function:

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

int prettyprint_message(uint32_t, char const *);

int main(void)
{
        prettyprint_message(0x00C0FFEE, "Needed!");

        prettyprint_message(0x02010001, "\"Kreog!\"");
        prettyprint_message(0x01010001, "Eucalytpus");
        prettyprint_message(0x01010002, "\"POO!\"");
        prettyprint_message(0x01010001, "Keytronic");
        prettyprint_message(0x04010001, NULL);

        /* Dark voodoo incantations to ressurect the Koala. */
        prettyprint_message(0x02020001, NULL);
        prettyprint_message(0x01040002, NULL);
        prettyprint_message(0x01020002, "\"KREOG!!!\"");
        prettyprint_message(0x01040001, "Brainfuck");
        prettyprint_message(0x04040001, "\"Dark Moon of the side...\"");

        return 0;
}
```

Terminal

```
~/B-PAV-242> gcc -Wall, -Wextra, -Werror -std=gnu99 main.c koalatchi.c && ./a.out | cat -e
Invalid message.$
Request Nutrition Hungry "Kreog!"$
Notification Nutrition Eat Eucalytpus$
Notification Nutrition Defecate "POO!"$
Notification Nutrition Eat Keytronic$
Error Nutrition Indigestion$
Request Entertainment NeedAffection$
Notification Education BeAwesome$
Notification Entertainment Bite "KREOG!!!"$
Notification Education TeachCoding Brainfuck$
Error Education BrainDamage "Dark Moon of the side..."$
```

Exercise 08

Log

repository subdir:	/ex08
compilation:	-Wall -Wextra -Werror -std=gnu99
files to turn in:	log.h, log.c
forbidden functions:	none
points:	2

Logs play a very important role in computer science. Thanks to them, you can keep a record of everything that has been done. If you look at /var/log, it shows the amount of information that is being kept by various operations or by the operating system itself.
We are going to write down a few logging functions. They should make it possible to choose where the messages we want to log will be sent. It must be possible to print on the standard output, on the error output or even in a file of your choosing. The error output should be the default choice. If a program logs messages in a file, each message should be appended at the end of the file.

Furthermore, each message will be associated with a log level. These levels are inspired by syslog(3) and are as follows:

- Error
- Warning
- Notice
- Info
- Debug

All of these levels will be defined in an enumeration named **LogLevel**.

It must be possible to choose the maximum desired log level.
For example, if the maximum desired level is Warning, the only messages that should actually be logged are Error and Warning-level messages. The default behavior will set the maximum desired level to Error, and therefore, will only log Error-level messages.
Messages will be formatted as follows:

```
Date [Level]: Message
```

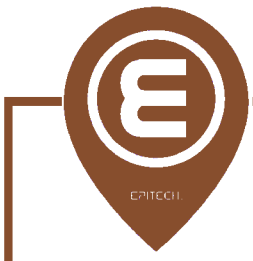
The date should have the same format as the one returned by ctime(3). You must obtain the system time with a call to time(2).
In a file named **log.h**, write a **LogLevel** enum that contains all of the previously-defined enumerators.
In a file named **log.c**, implement the following functions:

```
enum LogLevel   get_log_level(void);
enum LogLevel   set_log_level(enum LogLevel);
int             set_log_file(char const *);
int             close_log_file(void);
int             log_to_stdout(void);
int             log_to_stderr(void);
void            log_msg(enum LogLevel, char const *fmt, ...);
```

The **get_log_level** function returns the current log level.

The **set_log_level** function defines the log level to be used. This level is passed as its parameter. If the requested log level does not exist, the current levl is left unchanged.
This function returns the current log level at the end of the function call.

The **set_log_file** function allows the user to provide the name of the file where the messages will be written. The file's name is passed as its parameter. If another file was previously open, it will be closed beforehand. The function



returns 0 if it's successful, and in all other cases, 1.

The **close_log_file** function closes the current log file (if it exists) and resets the log output to the error output. If no file was previously open, this function returns without doing anything. It returns 0 if no errors occurred, and in all other cases, it returns 1.

The **log_to_stdout** sets the log output to the standard output. If a file was previously open, the functions close it beforehand. It returns 0 if no errors occurred, and in all other cases, it returns 1.

The **log_to_stderr** function sets the log output to the error output. If a file was previously open, the functions close it beforehand. It returns 0 if no errors occurred, and in all other cases, it returns 1.

The **log_msg** function writes a message on the previously-set log output. Its parameters are the log level of the message, which is a printf-like format string, and variadic arguments. If the required log level does not exist, this function returns with no further action.
The messages' destinations, as well as the current log levels, must be stored in global variables that CANNOT be accessible through functions that were not defined in the log.c compilation unit.

man fopen(3), fprintf(3), vfprintf(3), ctime(3), time(2), starg(3)

Here is an example of a main function:

```c
#include <stdio.h>
#include <stdlib.h>
#include "log.h"

int main(void)
{
        set_log_file("out.log");
        set_log_level(Debug);
        log_msg(Debug, "This is debug\n");
        log_msg(42, "This should not be printed\n");
        log_msg(Warning, "This is a warning\n");
        set_log_level(Warning);
        log_msg(Info, "This is info\n");
        log_msg(Error, "KREOG!\n");
        close_log_file();
        return EXIT_SUCCESS;
}
```

Terminal

```
~/B-PAV-242> ls
log.c log.h main.c
~/B-PAV-242> gcc -Wall -Wextra -Werror -std=c99 main.c log.c && ./a.out
~/B-PAV-242> ls
a.out log.c log.h main.c out.log
~/B-PAV-242> cat out.log
Tue Dec 7 01:08:19 2010 [Debug]:  This is debug
Tue Dec 7 01:08:19 2010 [Warning]:  This is a warning
Tue dec 7 01:08:19 2010 [Error]:  KREOG!
~/B-PAV-242> ./a.out && cat out.log
Tue Dec 7 01:08:19 2010 [Debug]:  This is debug
Tue Dec 7 01:08:19 2010 [Warning]:  This is a warning
Tue Dec 7 01:08:19 2010 [Error]:  KREOG!
Tue Dec 7 01:08:19 2010 [Debug]:  This is debug
Tue Dec 7 01:08:19 2010 [Warning]:  This is a warning
Tue Dec 7 01:08:19 2010 [Error]:  KREOG!
```