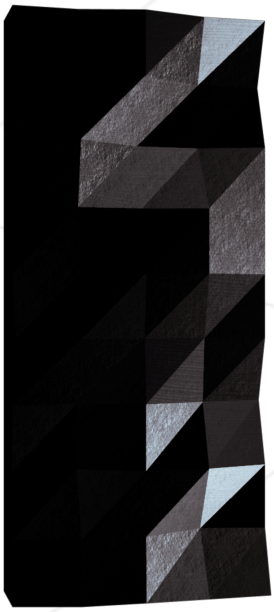

C Bootcamp

Day 08

Staff WeThinkCode_ info@wethinkcode.co.za

Summary: This document is the subject for Day08 of the C Bootcamp @ WeThinkCode_.

Contents

Chapter I

Instructions

- Only this page will serve as reference: do not trust rumors.
- Watch out! This document could potentially change up to an hour before submission.
- Make sure you have the appropriate permissions on your files and directories.
- You have to follow the submission procedures for every exercise.
- Your exercises will be checked and graded by your fellow classmates.
- On top of that, your exercises will be checked and graded by a program called Moulinette.
- Moulinette is very meticulous and strict in its evaluation of your work. It is entirely automated and there is no way to negotiate with it. So if you want to avoid bad surprises, be as thorough as possible.
- Moulinette is not very open-minded. It won't try and understand your code if it doesn't respect the Norm. Moulinette relies on a program called `Norminator` to check if your files respect the norm. TL;DR: it would be idiotic to submit a piece of work that doesn't pass `Norminator`'s check.
- These exercises are carefully laid out by order of difficulty - from easiest to hardest. We `will not` take into account a successfully completed harder exercise if an easier one is not perfectly functional.
- Using a forbidden function is considered cheating. Cheaters get `-42`, and this grade is non-negotiable.
- If ft_putchar() is an authorized function, we will compile your code with our `ft_putchar.c`.
- You'll only have to submit a main() function if we ask for a program.

- Moulinette compiles with these flags: -Wall -Wextra -Werror, and uses `gcc`.
- If your program doesn't compile, you'll get `0`.
- You cannot leave any additional file in your directory than those specified in the subject.
- Got a question? Ask your peer on the right. Otherwise, try your peer on the left.
- Your reference guide is called `Google / man / the Internet / ....`
- Check out the "C Bootcamp" part of the forum on the intranet.
- Examine the examples thoroughly. They could very well call for details that are not explicitly mentioned in the subject...
- By Odin, by Thor ! Use your brain !!!

Chapter II

Foreword

Here's how to connect an XBOX ONE:

```
1. Connect the power adapter into a plug, depending on the country,
you might need an adapter.
2. Connect the HDMI connector to the HDMI port on the console.
3. Connect the HDMI connector on the HDMI port on your huge HDTV or monitor.

optional :
4. If you have a great 7.1 sound system use the optical cable to connect it.
In some very specific cases, a 5.1 sound system will also be accepted.
```

Of course to connect a PS4 it's much more simpler !

Chapter III

Exercise 00 : ft.h

	Exercise 00
ft.h	
Turn-in directory : $ex00/$	
Files to turn in : `ft.h`	
Allowed functions : `None`	
Notes : `n/a`	

- Create your `ft.h` file.
- It contains all prototypes of your `libft.a` functions.

Chapter IV

Exercise 01 : ft_boolean.h

	Exercise 01
	ft_boolean.h
Turn-in directory : *ex*01/	
Files to turn in : `ft_boolean.h`	
Allowed functions : `None`	
Notes : `n/a`	

- Create a `ft_boolean.h` file. It'll compile and run the following main appropriately :

```
#include "ft_boolean.h"

void        ft_putstr(char *str)
{
        while (*str)
                write(1, str++, 1);
}

t_bool        ft_is_even(int nbr)
{
        return ((EVEN(nbr)) ? TRUE : FALSE);
}

int                main(int argc, char **argv)
{
        (void)argv;
        if (ft_is_even(argc - 1) == TRUE)
                ft_putstr(EVEN_MSG);
        else
                ft_putstr(ODD_MSG);
        return (SUCCESS);
}
```

- This program should display

```
I have an even number of arguments.
```

- ou

```
I have an odd number of arguments.
```

- followed by a line break when adequate.

Chapter V

Exercise 02 : ft_abs.h

	Exercise 02
ft_abs.h	
Turn-in directory : $ex02/$	
Files to turn in : `ft_abs.h`	
Allowed functions : `None`	
Notes : `n/a`	

- Create a macro `ABS` which replaces its argument by it absolute value :

```
#define ABS(Value)
```

Chapter VI

Exercise 03 : ft_point.h

	Exercise 03
ft_point.h	
Turn-in directory : $ex03/$	
Files to turn in : `ft_point.h`	
Allowed functions : `None`	
Notes : `n/a`	

- Create a file `ft_point.h` that'll compile the following main :

```
#include "ft_point.h"

void        set_point(t_point *point)
{
        point->x = 42;
        point->y = 21;
}

int                main(void)
{
        t_point                point;

        set_point(&point);
        return (0);
}
```

Chapter VII

Exercise 04 : ft_param_to_tab

	Exercise 04
ft_param_to_tab	
Turn-in directory : *ex*04/	
Files to turn in : `ft_param_to_tab.c, ft_stock_par.h`	
Allowed functions : `ft_split_whitespaces, ft_show_tab, malloc`	
Notes : `n/a`	

- Create a function that stores the program's arguments within an array structure and that returns the address of that array's first box.
- All elements of the array must be processed, including `av[0]`.
- Here's how it should be prototyped :

```
struct s_stock_par *ft_param_to_tab(int ac, char **av);
```

- The structure array should be allocated and its last box shall contain 0 in its `str` element to point out the end of the array.

- The structure is defined in the `ft_stock_par.h` file, like this :

```
typedef struct s_stock_par
{
  int  size_param;
  char *str;
  char *copy;
  char **tab;
}            t_stock_par;
```

  - `size_param` being the length of the argument ;
  - `str` being the address of the argument ;
  - `copy` being the copy of the argument ;
  - `tab` being the array returned by `ft_split_whitespaces`.

- We'll test your function with our `ft_split_whitespaces` and our `ft_show_tab` (next exercise). Take the appropriate measures for this to work !

Chapter VIII

Exercise 05 : ft_show_tab

	Exercise 05
ft_show_tab	
Turn-in directory : $ex05/$	
Files to turn in : `ft_show_tab.c, ft_stock_par.h`	
Allowed functions : `ft_putchar`	
Notes : `n/a`	

- Create a function that displays the content of the array created by the previous function.
- Here's how it should be prototyped :

```
void ft_show_tab(struct s_stock_par *par);
```

- The structure is defined in the `ft_stock_par.h` file, like this :

```
typedef struct s_stock_par
{
  int  size_param;
  char *str;
  char *copy;
  char **tab;
}            t_stock_par;
```

- For each box, we'll display (one element per line):
  - the argument
  - the size
  - each word (one per line)

- We'll test your function with our `ft_param_to_tab` (previous exercise). Take the appropriate measures for this to work !