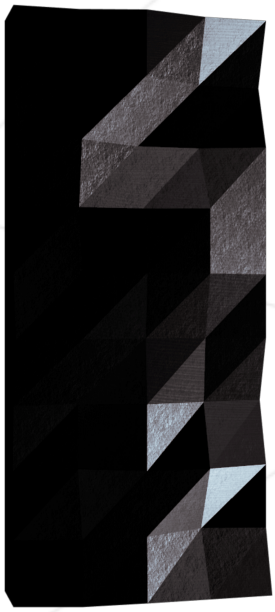

# C Bootcamp

## Day 07

Staff WeThinkCode_ info@wethinkcode.co.za

*Summary: This document is the subject for Day07 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 your right. Otherwise, try your peer on your 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 !!!

`Norminator must be launched with the -R CheckForbiddenSourceHeader flag. Moulinette will use it too.`

# Chapter II

# Foreword

Here is how to make pancakes.

```
In a large bowl, shift together the flour, baking powder, salt and sugar.
Make a well in the center and pour in the milk, egg and melted butter.
Mix until smooth.
Heat a lightly oiled griddle or frying pan over medium high heat.
Pour or scoop the batter onto the griddle, using approximately
1/4 cup for each pancake.
Turn cautiously to bake both sides.
```

We all love pancakes in the staff team.

# Chapter III

# Exercise 00 : ft_strdup

|  | Exercise 00 |
|---|---|
| ft_strdup | |
| Turn-in directory : $ex00/$ | |
| Files to turn in : `ft_strdup.c` | |
| Allowed functions : `malloc` | |
| Notes : `n/a` | |

- Reproduce the behavior of the function `strdup` (man strdup).
- Here's how it should be prototyped :

```
char    *ft_strdup(char *src);
```

# Chapter IV

# Exercise 01 : ft_range

|  | Exercise 01 |
|---|---|
| ft_range | |
| Turn-in directory : *ex*01/ | |
| Files to turn in : `ft_range.c` | |
| Allowed functions : `malloc` | |
| Notes : `n/a` | |

- Create a function `ft_range` which returns an array of`ints`. This `int` array should contain all values between `min` and `max`.
- `Min` included - `max` excluded.
- Here's how it should be prototyped :

```
int    *ft_range(int min, int max);
```

- If `min`́value is greater or equal to `max`'s value, a null pointer should be returned.

# Chapter V

# Exercise 02 : ft_ultimate_range

|  | Exercise 02 |
|---|---|
| ft_ultimate_range | |
| Turn-in directory : *ex*02/ | |
| Files to turn in : `ft_ultimate_range.c` | |
| Allowed functions : `malloc` | |
| Notes : `n/a` | |

- Create a function `ft_ultimate_range` which allocates and assigns an array of `ints`. This `int` array should contain all values between `min` and `max`.
- `Min` included - `max` excluded.
- Here's how it should be prototyped :

```
int    ft_ultimate_range(int **range, int min, int max);
```

- If the value of `min` is greater or equal to `max`'s value, `range` will point on NULL.
- The size of `range` should be returned (or 0 on error).

# Chapter VI

# Exercise 03 : ft_concat_params

|  | Exercise 03 |
|---|---|
| ft_concat_params | |
| Turn-in directory : *ex*03/ | |
| Files to turn in : `ft_concat_params.c` | |
| Allowed functions : `malloc` | |
| Notes : `n/a` | |

- Create a function that transforms arguments given as command-line into a single string of characters. Those arguments should be separated by a "\n".
- Here's how it should be prototyped :

```
char *ft_concat_params(int argc, char **argv);
```

# Chapter VII

# Exercise 04 : ft_split_whitespaces

|  | Exercise 04 |
|---|---|
| ft_split_whitespaces | |
| Turn-in directory : *ex*04/ | |
| Files to turn in : `ft_split_whitespaces.c` | |
| Allowed functions : `malloc` | |
| Notes : `n/a` | |

- Create a function that splits a string of characters into words.
- Separators are spaces, tabs and line breaks.
- This function returns an array where each box contains a character-string's address represented by a word. The last element of this array should be equal to 0 to emphasise the end of the array.
- There can't be any empty strings in your array. Draw the necessary conclusions.
- The given string can't be modified.
- Here's how it should be prototyped :

```
char **ft_split_whitespaces(char *str);
```

# Chapter VIII

# Exercise 05 : ft_print_words_tables

|  | Exercise 05 |
|---|---|
| ft_print_words_tables | |
| Turn-in directory : $ex05/$ | |
| Files to turn in : `ft_print_words_tables.c` | |
| Allowed functions : `ft_putchar` | |
| Notes : `n/a` | |

- Create a function that displays the content of the array you created in the last excercise's function.
- One word per line.
- Each word will be followed by a "\n", including the last one.
- This exercise will be compiled with your ft_split_whitespaces.c
- Watch out not to have `multiple define`.
- Here's how it should be prototyped :

```
void ft_print_words_tables(char **tab);
```

# Chapter IX

# Exercise 06 : ft_convert_base

|  | Exercise 06 |
|---|---|
| ft_convert_base | |
| Turn-in directory : *ex*06/ | |
| Files to turn in : `ft_convert_base.c` | |
| Allowed functions : `malloc, free` | |
| Notes : `n/a` | |

- Create a function that returns the result of the conversion of the string `nbr` from a base `base_from` to a base `base_to`. The string must have enough allocated memory. The number represented by `nbr` must fit inside an `int`.

- Here's how it should be prototyped :

```
char *ft_convert_base(char *nbr, char *base_from, char *base_to);
```

# Chapter X

# Exercise 07 : ft_split

|  | Exercise 07 |
|---|---|
| ft_split | |
| Turn-in directory : *ex*07/ | |
| Files to turn in : `ft_split.c` | |
| Allowed functions : `malloc` | |
| Notes : `n/a` | |

- Create a function that slits a string of character depending on another string of characters.
- You'll have to use each character from the string `charset` as a separator.
- The function returns an array where each box contains the address of a string wrapped between two separators. The last element of that array should equal to 0 to indicate the end of the array.
- There cannot be any empty strings in your array. Draw your conclusions accordingly.
- The string given as argument won't be modifiable.
- Here's how it should be prototyped :

```
char **ft_split(char *str, char *charset);
```