# miniRE Dash

*Summary: THIS document is the subject for the miniRE Dash @42seoul.eduthon*

*version: 1.0*

# Contents

# Chapter I

# Instructions

- In this subject, you will practice finding the **pattern** you want using ***regular expressions***.
- The **pattern** should not be too long. Make your pattern efficiently. If you submit a pattern that is too long and complicate, the test will **KO** even if your answer is correct.
- The turn in file must be a single, compilable file.

# Chapter II
# Foreword

# Chapter III

## Exercice 00 : Eleven number checker

| | Exercise 00 |
|---|---|
| eleven_number_checker | |
| Turn-in directory : *ex00/* | |
| File to turn in : eleven_number_checker.c | |
| Allowed functions : write | |

- Create a program that check input string is ***phone number.***
- ***phone number*** does not contain country code, hyphen or anything, except number. JUST plane 11 digit of number.
- Here's how it should be prototyped :

```
void     eleven_number_checker(char *input);
```

Example :

```
$>./eleven_number_checker 01012345678
OK$
$>./eleven_number_checker 0101234o678
KO$
```

# Chapter IV

## Exercice 01 : Phone number checker

| | Exercise 01 |
|---|---|
| phone_number_checker | |
| Turn-in directory : *ex01/* | |
| File to turn in : phone_number_checker.c | |
| Allowed functions : regcomp, regexec, regerror, regfree, write | |

- Rewrite a program that check input number is ***valid phone number***.
- Use <regex.h> header's function. We provide example code with basic usage of Regex functions.
- In Regex manner, ***valid phone number*** starts with three-digit of 01[0-9], followed by two four-digit numbers, hyphen in between.
- Here's how it should be prototyped :

```
void     phone_number_checker(char *input);
```

Examples in next page

Example :

```
$> ./example
Error
$> ./example arg1 arg2
Error
$> ./example "010-4242-a242"
KO
$> ./example "010-4242-4010-4242-4242"
KO
$> ./example "010-4242-4242"
010-4242-4242
$> ./example "010-424-4242"
010-424-4242
$> ./example "010-4242-4242010-2424-2424"
010-4242-4242
010-2424-2424
$> ./example " 010-4242-4242    010-4242-4242"
010-4242-4242
010-4242-4242
```

# Bonus

# Chapter V

## Exercice 02 : Simple E-mail validator

| | Exercise 02 |
|---|---|
| email_validator | |
| Turn-in directory : *ex02/* | |
| File to turn in : simple_email_validator.c | |
| Allowed functions : regcomp, regexec, regerror, regfree, write | |

- Write a program to check an input is ***valid E-mail***
- ***Valid E-mail*** is divided into two parts. Before the '@' character is the **ID** part. After the '@' is **Domain** part
- **ID** contains uppercase and lowercase letters and numbers, except 4, 2, s, e, o, u, or l.
- **Domain** contains uppercase and lowercase letters and numbers, but only lowercase two or three letters after '.'(dot sign).
- Here's how it should be prototyped :

```
void     simple_email_validator(char *input);
```

$$(A^c)^c = A$$

Example :

```
$> ./example
Error
$> ./example arg1 arg2
Error
$> ./example "benene31@42seoul.kr"
KO
$> ./example "banana42@42seoul.kr"
KO
$> ./example "@42seoul.kr"
KO
$> ./example "banana@42seoul.Kr"
KO
$> ./example "  banana31@42seoul.kr  "
banana31@42seoul.kr
```

# Chapter VI

## Exercice 03 :

## Push swap instruction validator

|  |  |
|---|---|
|  | Exercise 02 |
| ps_instruction_validator | |
| Turn-in directory : *ex02/* | |
| File to turn in : pushswap_instruction_validator.c | |
| Allowed functions : regcomp, regexec, regerror, regfree, write | |

- Push_swap has 11 operations, "pa, pb, sa, sb, ss, ra, rb, rr, rra, rrb, rrr".
- Write a program to check instructions are valid. We will test your code with random operations, some of them are invalid form.
- Instructions must be separated by a '\n' and nothing else.
- Here's how it should be prototyped :

```
void    pushswap_instruction_validator(char *input);
```

This is pipe and, yes. This is a hint.

# Chapter VII

## Exercice 04 : Snake to Camel

| | |
|---|---|
| | Exercise 03 |
| sanke_toCamel | |
| Turn-in directory : *ex03/* | |
| File to turn in : sanke_ToCamel.c | |
| Allowed functions : regcomp, regexec, regerror, regfree | |

- Write a program to substitute a valid ***snake case*** input to ***camel case.***
- ***Camel case*** always starts with uppercase letter.
- Here's how it should be prototyped :

```
void     snake_ToCamel(char *input);
```

I think everyone knows what a snake case and a camel case are, but in case anyone doesn't, I prepared this.