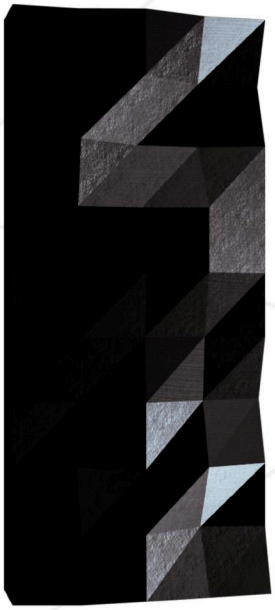

ft_exception

Handle Your exception without exception

Summary:
This is a project to about how to handle an error in C
with simple things from Signal, Long-jump in place of if-conditions.

Contents

Chapter I

Foreword

Chapter II

Common Instructions

- Your project must be written in C.
- All heap allocated memory space must be properly freed when necessary. No leaks will be tolerated.
- If the subject requires it, you must submit a Makefile which will compile your source files to the required output with the flags -Wall, -Wextra and -Werror, use cc, and your Makefile must not relink
- Your Makefile must at least contain the rules $(NAME), all, clean, fclean and re.
- We encourage you to create test programs for your project even though this work won't have to be submitted and won't be graded. It will give you a chance to easily test your work and your peers' work. You will find those tests especially useful during your defence. Indeed, during defence, you are free to use your tests and/or the tests of the peer you are evaluating.
- Submit your work to your assigned git repository. Only the work in the git repository will be graded. If Deepthought is assigned to grade your work, it will be done after your peer-evaluations. If an error happens in any section of your work during Deepthought's grading, the evaluation will stop.

Chapter III

Mandatory Part

Chapter III.1

Technical considerations

- except as allowed, declaring global variables is forbidden.
- If you need helper functions to split a more complex function, define them as static functions. This way, their scope will be limited to the appropriate file.
- Place all your files at the correct directory if it is mentioned in this subject.
- Turning in unused files is forbidden.
- Every .c files must compile with the flags -Wall -Wextra -Werror

Chapter III.2

Exercise 00: Heroes Never Die

	Exerise 00
Heroes Never Die	
Turn-in directory : ex00/	
Files to turn in : signal_practice.c	
Allowed function : signal, exit	

- Write a function to make the program not to stop when the SIGFPE is raised.
- The program complied with your function must exits, returning 42 as exit code, when the signal SIGFPE is raised.
- Here's how it should be prototyped :

```
void register_fpe(void);
```

- We will use this function to raise the signal in order that testing your function.

```
void raise_fpe(void)
{
    const int life = 42;
    const int zero = 0;
    int answer;

    answer = life / zero;
}
```

- Do not afraid if your function is too short. This is just simple practice. This is a simple step to remind the signals once more before the next step.

Chapter III.3

Exercise 01:Try me

	Exercise 01
Try me	
Turn-in directory : ex01/	
Files to turn in : try_me.c	
Allowed function : signal, longjmp	

Write a function meet the bellow requirements :

1. Here's how it should be prototyped :

```
void register_catch(void);
```

2. By calling this function, Your program must not be crashed and exited even if the below happened.
    1. Dividing by zero.
    2. Segmentation fault : the tester will raise the signal by running this code to test your function.

```
void raise_segv(void)
{
    const char *ptr = NULL;
    ptr[42] = '*'; // Occur segmentation fault
}
```

- Due to use a variable ‘jmp_buf’ in main function of the tester, write the below line in your .c file.

```
extern jmp_buf env;
```

- We will run this code to test your submission. Try to print a right value in **catch** block if a signal raised in **try** block.

```
#include <setjmp.h>

jmp_buf env;

void main(void)
{
    register_catch(); // Your function

    int k = setjmp(env);
    if (k == 0)
    {
        // <<try>>
        raise_segv(); // or raise_fpe();
    }
    else
    {
        // <<catch>>
        printf("Signal no %d is caught safely :)\n", k);
    }
}
```

Chapter III.4

Exercise 02 : Catch Me If You Can

	Exercise 02
Catch Me If You Can	
Turn-in directory : ex02/	
Files to turn in : safe_replace.c	
Allowed function : signal, longjmp, setjmp	

Now, it is time to your own C-based **Try-Catch** keyword.

With this exercise, you will understand a little bit more how to use setjmp() and longjmp() and it's correlation.

- Just only one grobal variable is allowed in this exercise.
- Write a function that replace a certain character in a string to other character. The function must be prototyped as follow:

```
int safe_replace(char* str, char oldchar, char newchar)
```

Your **catch_me** function must returns a value returned from a function passed by a function pointer as parameter of **try_action**.

But watch out! Life isn't alway smooth. Your function will be runned in a terrible spaghetti code.

When it runs, **one of the bellow conditions would be occurred during the runtime.**

- Segmentation fault, Bus error

- If one of the mentioned conditions occurred, the program must not be crashed or exited, then, if your function fail to replace the string due to the error, your function must returns 0.
- If your function successes to replace the string without an error, it must returns 1.
- **Your function must not handle other errors except the mentioned errors.**

Did you restore the signal handler after running your function?
If not, what would be happened?

Inspection a string with if-condition isn't good idea.

Tester Main

```c
int main(void)
{
    char hello[] = "Hello world";

    // Segmentation fault
    printf("result 1: %d\n", safe_replace(NULL, 'o', '?'));

    // Bus error
    printf("result 2: %d\n", safe_replace("Hello", 'o', '?'));

    // Success
    printf("result 3: %d\n", safe_replace(hello, 'o', '?'));
}
```

Output :

```bash
$ ./safe_replace
0
0
1
```

Chapter IV

Additional Tasks

- Implementation of ‘Throw’ keyword with raise()
- Running stackTrace using backtrace() when an error occurred.
- Implementation of exception blocks : try, catch, throw, finally …