

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.*

Chapter I

Contents

I	Foreword	2
II	Common Instructions	3
III	Mandatory Part	5
III.1	Technical considerations	5
III.2	Ex00: Heroes Never Die	5
III.3	Ex01: Try me	5
IV	Bonus Part	—
V	Submission and peer-evaluation	—

Chapter II

Foreword

Irregular conjugation

Chapter III

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

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.1

Exercise 00: Heroes Never Die



Exerise 00

Heroes Never Die

Turn-in directory : ex00/

Files to turn in : signal_practice.c

Allowed function : signal

- Write a function to make the program not to stop when the SIGFPE is raised.
- The program compiled 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 (*pfn)(int));`
```

- We will use this function to raise signal in the program and to test your function.


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

 answer = life / everything;
}
```
```

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

Chapter III.2

Exercise 01:Try me

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

There is **Try-Catch** keyword to handle **Exception** in C++.

Here is an example handle an error by try-catch keyword.

```
```c++
int i = 42;
try {
 i /= 0;
} catch (const std::overflow_error& ex) {
 std::cout << ex.what() << std::endl;
}
```
```

In C, We have no keyword to handle an error in this way. But that doesn't mean that it is impossible to handle an error with exception in C.

So, do it yourself to write Try.

Moulinette will mass your code during the runtime. Preparing for this, write a function meet the bellow requirements :

1. Your function must not be crashed under the following conditions.

- Segmentation fault
 - Divided by zero
 - Broken pipe
 - Trying to free a pointer was not allocated
 - Bus error
2. Using `longjmp()`, your function must return the signal which was handled in your function.
 3. You must add a line as below :

```
`extern jmp_buf env`
```



In this exercise, the tester uses `setjmp()`. So, you do not consider when you use `setjmp()` in your code.

4. Here's how it should be prototyped :

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


Chapter III.3

Exercise 02 : Catch Me If You Can



Exercise 02

Catch Me If You Can

Turn-in directory : ex02/

Files to turn in : catch_me.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 global variable is allowed in this exercise.
- The global variable's type must be **jmp_buf**.
- Using static variable is forbidden.
- Here's how it should be prototyped :

```
- `int catch_me(int (*try_action)(void), void (*catch_action)(int))`
```

- 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 always smooth.

When **try_run** function runs, one of the below conditions would be occurred during the runtime.

- Segmentation fault
 - Divided by zero
 - Broken pipe
 - Trying to free a pointer was not allocated
 - Bus error
-
- If one of the mentioned conditions occurred, the program must not crashed or exited, then should run a function passed by a function pointer as argument of **catch_action**
 - The signo of the signal was raised must be passed as argument of **catch_action**, if this case occurred, your **catch_action** must returns '-1'.

If everything went right, you can catch output as bellow with this tester code.

Tester Code :

```
***
void catch_function(int sig)
{
    if (sig == SIGFPE)
        puts("Caught SIGFPE :)");
}

int try_function(void)
{
    int answer = 42;
    answer /= (answer ^ answer);
    return answer;
}

int main(void)
{
    return catch_me(&try_function, &catch_function);
}
***
```

Output :

```
```bash
$./catch_me
Caught SIGFPE :)
$ echo $?
255
```
```