ALGORITHMIC Project

Dr_Quine

42 Staff pedago@staff.42.fr

Summary: *This project is about making you discover the recursion theorem of Kleene!*

Contents

Chapter I

Foreword

Chapter II

Introduction

A quine is a computer program (a kind of metaprogram) whose output and source code are identical. As a challenge or for fun, some programmers try to write the shortest quine in a given programming language.

The operation that consist of simply opening the source file and displaying it is considered cheating. More generally, a program that uses any data entry cannot be considered a valid quine. A trivial solution is a program whose source code is empty. Indeed, the execution of such a program produces for most languages no output, that is to say the source code of the program.

Chapter III

Objectives

This project invites you to confront the principle of self-reproduction and the problems that derive from it. It is a perfect introduction to more complex projects, particularly malware projects.

For the curious ones, I strongly recommend that you watch everything related to fixed points!

Chapter IV

General Instructions

- This project will be corrected by humans only. You're allowed to organise and name your files as you see fit, but you must follow the following rules.

- You must use C and submit a Makefile.

- Your Makefile must compile the project and must contain the usual rules. It must recompile and re-link the program only if necessary.

- You have to handle errors carefully. In no way can your program quit in an unexpected manner (Segmentation fault, bus error, double free, etc).

- You can ask your questions on the forum, on slack...

Chapter V

Mandatory part

For this project, you will have to recode three different programs, each with different properties.

The first program will have the following characteristics:

- The executable must be named `Colleen`.

- When executed, the program must display on the standard output an output identical to the source code of the file used to compile the program.

- The source code must contain at least:

  - A main function.
  - Two different comments.
  - One of the comments must be present in the main function.
  - One of the comments must be present outside of your program.
  - Another function in addition to the main function (which of course will be called)

Here is a usage example:

```
$> ls -al
total 12
drwxr-xr-x 2 root root 4096 Feb  2 13:26 .
drwxr-xr-x 4 root root 4096 Feb  2 13:26 ..
-rw-r--r-- 1 root root  647 Feb  2 13:26 Colleen.c
$> clang -Wall -Wextra -Werror -o Colleen Colleen.c; ./Colleen > tmp_Colleen ; diff tmp_Colleen
 Colleen.c
$> _
```

For the second program:

- The executable must be named `Grace`.

- When executed, the program writes in a file named `Grace_kid.c` the source code of the file used to compile the program.

- The source code must strickly contain:

    ○ No main declared.
    ○ Three defines only.
    ○ One comment.

- The program will run by calling a macro.

Here is a usage example:

```
$> ls -al
total 12
drwxr-xr-x 2 root root 4096 Feb  2 13:30 .
drwxr-xr-x 4 root root 4096 Feb  2 13:29 ..
-rw-r--r-- 1 root root 362 Feb  2 13:30 Grace.c
$> clang -Wall -Wextra -Werror -o Grace Grace.c; ./Grace ; diff Grace.c Grace_kid.c
$> ls -al
total 24
drwxr-xr-x 2 root root 4096 Feb  2 13:30 .
drwxr-xr-x 4 root root 4096 Feb  2 13:29 ..
-rwxr-xr-x 1 root root 7240 Feb  2 13:30 Grace
-rw-r--r-- 1 root root  362 Feb  2 13:30 Grace.c
-rw-r--r-- 1 root root  362 Feb  2 13:30 Grace_kid.c
$> _
```

For the last program:

- The executable must be named `Sully`.

- When executed the program writes in a file named `Sully_X.c`. The X will be an interger given in the source. Once the file is created, the program compiles this file and then runs the new program (which will have the name of its source file).

- Stopping the program depends on the file name: if the integer X is set to 0, the resulting program is not executed.

- An integer is therefore present in the source of your program and will have to evolve by decrementing every time you create a source file from the execution of the program.

- You have no constraints on the source code, apart from the integer that will be set to 5 at first.

Here is a usage example:

```
$> clang -Wall -Wextra -Werror ../Sully.c -o Sully ; ./ Sully
$> ls -al | grep Sully | wc -l
13
$> diff ../Sully.c Sully_0.c
1c1
< int i = 5;
---
> int i = 0;
$> diff Sully_3.c Sully_2.c
1c1
< int i = 3;
---
> int i = 2;
$> _
```

A comment will look like:

```
$> nl comment.c
 1   /*
 2       This program will print its own source when run.
 3   */
```

A program without a declared main will look like:

```
$> nl macro.c
 1   #include
 2   #define FT(x)int main(){ /* code */ }
     [..]
 5   FT(xxxxxxxx)
```

Using advanced macros is strongly recommended for this project.

For the smarty pants (or not)... just reading the source and
displaying it is considered to be cheating. The use of argv/argc
is also considered cheating.

Chapter VI

Bonus part

> ⚠️ We will look at your bonuses if and only if your mandatory part is EXCELLENT. This means that your must complete the mandatory part, beginning to end, and your error management must be flawless, even in cases of twisted or bad usage. If that's not the case, your bonuses will be totally IGNORED.

The only Bonus accepted during p2p is to have redone this project entirely in the language of your choice.

> ℹ️ In case of a language without define/macro, you will naturally have to adapt the program accordingly.

Chapter VII

Submission and peer-evaluation

Submit your work on your `GiT` repository as usual. Only the work on your repository will be graded.