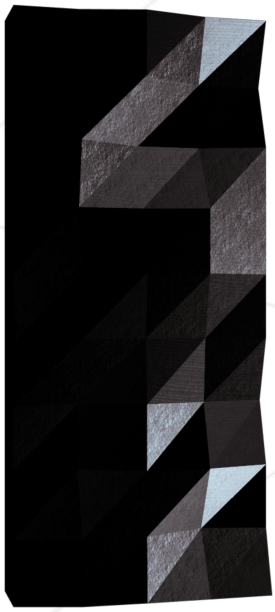

Dr_Quine

Project ALGORITHME

Résumé: Ce projet consiste à vous faire découvrir le Théorème de récursion de Kleene !

Table des matières

I	**Préambule**	**2**
II	**Introduction**	**3**
III	**Objectifs**	**4**
IV	**General Instructions**	**5**
V	**Partie obligatoire**	**6**
VI	**Partie bonus**	**10**
VII	**Rendu et peer-évaluation**	**11**

Chapitre I

Préambule

Chapitre II

Introduction

Un quine est un programme informatique (une sorte de métaprogramme) dont la sortie et le code source sont identiques. À titre de défi ou d'amusement, certains programmeurs essaient d'écrire le plus court quine dans un langage de programmation donné.

L'opération qui consiste à ouvrir le fichier source et à l'afficher est considérée comme une tricherie. Plus généralement, un programme qui utilise une quelconque entrée de données ne peut être considéré comme un quine valide. Une solution triviale est un programme dont le code source est vide. En effet, l'exécution d'un tel programme ne produit pour la plupart des langages aucune sortie, c'est-à-dire le code source du programme.

Chapitre III

Objectifs

Ce projet vous invite à vous confronter au principe d'auto-reproduction et des problématiques qui en découlent. Il s'agit d'une parfaite introduction aux projets plus complexes, notamment les projets malware.

Pour les curieux, je vous conseille vivement de regarder tout ce qui est lié aux points fixes !

Chapitre IV

General Instructions

- Ce projet ne sera corrigé que par des hummains
- Vous devez le coder en `C/ASM` et fournir un Makefile.
- Votre `Makefile` doit compiler le projet en utilisant les regles classiques. Il doit recompiler et re-linker seulement si necessaire.
- Vous devez gérer les erreurs de façon raisonnée. En aucun cas votre programme ne doit quitter de façon inattendue (Segmentation fault, etc).
- You can ask your questions on the forum, on slack...

Chapitre V

Partie obligatoire

Pour ce projet, vous aller devoir coder trois programmes différents, possédant chacun des propriétés différentes. Chacun des programmes devra à la fois être codé en C et en Assembleur, et respectivement rendu dans un dossier nommé C et ASM, chacun des dossiers possédant son propre Makefile contenant les règles usuelles.

La réalisation de la partie C est suffisante pour la validation de ce projet, cependant nous vous encourageons très fortemment à effectuer la partie en Assembleur pour la suite des projets de cette branche.

Le premier programme aura les caractéristiques suivantes :

- L'exécutable se nomme `Colleen`.
- Lors de son exécution, le programme doit afficher sur la sortie standard un output identique au code source du fichier utilisé pour compiler ce même programme.
- Le code source doit comporter au minimum :
  - Une fonction main.
  - Deux commentaires différents.
  - Un des commentaires doit être présent dans la fonction main.
  - Un des commentaires doit être présent en dehors des fonctions de votre programme.
  - Une fonction en plus de la fonction main principale (qui sera bien entendu appelée)

Voici un exemple d'utilisation :

```
$> 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
$> _
```

Pour le second programme :

- L'exécutable se nomme `Grace`.
- Lors de son exécution, le programme écrit dans un fichier nommé `Grace_kid.c`/`Grace_kid.s` le code source du fichier utilisé pour compiler ce même programme.
- Le code source doit comporter strictement :
  - Aucun main déclaré.
  - Trois defines uniquement.
  - Un commentaire.
- Le programme se lancera à l'appel d'une macro.

Voici un exemple d'utilisation :

```
$> 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
$> _
```

Pour le dernier programme :

- L'exécutable se nomme `Sully`.
- Lors de son exécution le programme écrit dans un fichier nommé `Sully_X.c`/`Sully_X.s`. Le X sera alors un entier donné dans la source. Une fois le fichier créé, le programme compile ce fichier puis exécute le nouveau programme (qui aura le nom de son fichier source).
- L'arrêt du programme se fait en fonction du nom du fichier : si l'entier X est à 0, le programme résultant n'est pas exécuté.
- Un entier est donc présent dans la source de votre programme et devra évoluer en se décrémentant à chaque création d'un fichier source depuis l'exécution du programme.
- Vous n'avez aucune contrainte au niveau du code source, mis à part l'entier qui sera défini à 5 dans un premier temps.

Voici un exemple d'utilisation :

```
$> 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;
$> _
```

Un commentaire sera de type :

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

Un programme sans main déclaré sera de type :

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

Utiliser des macros avancées est fortement recommandé pour ce projet.

Pour les malins : se contenter de lire la source et de l'afficher est considéré comme de la triche. L'utilisation d'argv/argc est considérée comme de la triche aussi.

Chapitre VI

Partie bonus

Les bonus ne seront comptabilisés que si votre partie obligatoire est PARFAITE. Par PARFAITE, on entend bien évidemment qu'elle est entièrement réalisée, et qu'il n'est pas possible de mettre son comportement en défaut, même en cas d'erreur aussi vicieuse soit-elle, de mauvaise utilisation, etc ... Concrètement, cela signifie que si votre partie obligatoire n'est pas validée, vos bonus seront intégralement IGNORÉS.

Le seul bonus accepté en soutenance est d'avoir refait le projet intégralement dans le langage de votre choix.

Dans le cas d'un langage sans define/macro, il faudra bien entendu adapter le programme en conséquence.

Chapitre VII

Rendu et peer-évaluation

- Rendez-votre travail sur votre dépot `GiT` comme d'habitude. Seul le travail présent sur votre dépot sera évalué en soutenance.