Compilation et Analyse Lexicale
TD 4 - Correction

Licence 3 Informatique (2022-2023)

Jairo Cugliari, Guillaume Metzler
Institut de Communication (ICOM)
Université de Lyon, Université Lumière Lyon 2

jairo.cugliari@univ-lyon2.fr guillaume.metzler@univ-lyon2.fr

Exercice 1

Vous pouvez utiliser la machine virtuelle dockerizée que nous avons construit, ou lancer l'application Debian qui émule un système sous Linux avec la distribution homonyme. À la première exécution de l'application, il faudra créer un utilisateur et renseigner un mot de passe.

Une fois dans le système, vous avez le droit d'installer les modules complémentaires nécessaires pour l'édition, compilation et audit du code. En particulier, vous aurez besoin d'un éditeur (`nano` ou `vim`), d'un compilateur (`gcc`) et du déboguer `gdb`.

1. Écrire la fonction `plus` qui renvoie la somme de deux arguments.

```
long plus(long x, long y) {
  return x + y;
}
```

2. Compléter le code source afin d'obtenir un programme exécutable. Vérifiez que votre programme produit le résultat attendu si on initialise les variables avec de valeurs que vous choisirez. Enlevez de votre code tout affichage pour avoir un binaire le plus simple possible.

   Voici une version pour les tests, avec affichages pour vérifier que les résultats sont corrects.

```
#include <stdio.h>
#include <stdlib.h>

```

```c
long plus(long x, long y) {
  return x + y;
}

void sumstore(long x, long y, long *dest)
{
  long t = plus(x, y) ;
  *dest  = t ;
}

int main(int argc, char *argv[]) {
    long x = atoi(argv[1]) ;
  long y = atoi(argv[2]) ;

  long z ;
  sumstore(x, y, &z);
  printf("%ld + %ld => %ld\n", x, y, z);

  return(0);
}
```

On peut compiler ce code et vérifier que tout se passe bien

```
gcc -Og sum.c -o sum && ./sum
```

3. Obtenir le dump du fichier exécutable. Gardez-le dans un fichier.

   Il suffit de diriger la sortie de l'utilitaire `objdump` vers un fichier avec l'opérateur `>`.

```
objdump -d sum > sum.dump
```

4. Ouvrir l'exécutable avec `gdb`.

   Pour ouvrir gdb avec un exécutable, on utilise la commande :

```
gdb sum
```

   Vous pouvez de manière alternative lancer le debuggeur puis, définir le fichier sur lequel vous voulez travailler avec la commande

```
file sum
```

5. Désassemblez le code avec la commande `disassemble main`. Comparez l'affichage avec la section principale du fichier que vous avez créé dans 2.

   L'affichage coïncide sur les instructions, avec un élément additionnel. Nous avons maintenant les adresses mémoire (relatives) ce qui nous permettra explorer la mémoire utilisée en temps réel.

6. Exécutez le code à l'intérieur de gdb en utilisant la commande `run`. Comprenez-vous ce qui se passe ? Pourquoi il n'y a aucun affichage ? Peut-on dire que l'ordinateur n'a rien fait ?

L'exécution du programme se passe sans accroche. Nous ne voyons rien sur l'écran, car nous avons éliminé tout affichage. Nous ne pouvons pas dire que rien ne se passe, par exemple si on avait fait une erreur de programmation en laissant une fuite de mémoire, le programme aurait pu bloquer la machine.

7. Utilisez la commande `breakpoint plus` pour placer un breakpoint au niveau de la fonction `main`.

   Toutes les commandes de `gdb` peuvent être renseignées de manière extensive ou par un raccourci. Ainsi, au lieu d'écrire `breakpoint main` nous pouvons utiliser

   ```
   b main
   ```

   Le déboguer informe que le point d'arrêt est fixé. Puis, si on exécute le programme (commande `run` ou tout simplement `r`). L'exécution sera arrêtée au moment où le registre `%rip` (*instruction pointer*) pointera vers l'adresse mémoire signalé par le point d'arrêt.

8. Affichez l'état des registres avant l'exécution du code (commande `info registers`).

   Les registres ne contiennent pas d'information de la fonction avant son exécution.

9. Placez maintenant un breakpoint au niveau de la fonction `plus`.

   ```
   b plus
   ```

10. Examinez l'état de registres. Pouvez-vous identifier comment sont passés les arguments de la fonction ?

    Une fois exécuté la fonction, il faut avancer (avec `n` ou `next`) jusqu'au point d'arrêt dans plus (rappelez vous que nous avons 2 points d'arrêt, un au niveau de main et l'autre au niveau de plus).

11. Vous pouvez examiner l'état d'une adresse mémoire avec la commande `examine *ADDRESSE`.