

Compilation et Analyse Lexicale

TD 2 - 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

Préparation d'une VM linux pour le TP

Nous allons travailler avec de machines virtuelles afin de simuler le comportement d'un système de type Unix (nous utiliserons la distribution de GNU Linux connue sous le nom de Debian¹).

Vous avez plusieurs options pour travailler sur ce TD. L'option la plus simple, c'est d'utiliser le client de virtualisation Virtual Box avec l'image que nous mettons à disposition sur moodle. Cependant, comme le client n'est pas installé dans les machines des salles informatiques, nous allons préférer la solution passant par de conteneurs de virtualisation. En particulier, nous allons travailler sur Docker. Voici les premiers pas techniques, une fois que le client Docker Desktop est installé sous MS Windows.

- Ouvrir Docker Desktop
- Modifier la variable %PATH pour inclure le chemin de docker
- Redémarrer l'ordinateur
- Dans l'invite de commandes, faire :

```
1 docker build -t l3info_compil .
```

¹Debian est réputé pour être un système d'exploitation universel, autrement dit, il devrait être possible de le faire contrôler n'importe quel système allant d'ordinateur jusqu'un frigo connecté ou un ballon d'eau chaude intelligent.

- Dans Docker Desktop, vous trouverez l'image créée dans l'onglet images. Appuyez sur Run pour démarrer votre machine virtuelle (laissez les options par défaut). Il faudra bien noter l'étiquette générée de manière aléatoire.
- Dans l'invite de commandes, lancez votre image avec la commande

```
1 docker exec -i -t \${ETIQUETTE_IMAGE} bash
```

L'invite de commande marque maintenant `rootxxxx` où `xxxx` correspond à une valeur aléatoire, vous êtes l'utilisateur `root` dans la machine virtuelle.

Exercice 1

Rappelez-vous que la commande `man machin` affiche la page d'aide pour la commande `machin`. Appuyez sur la touche `:q` pour sortir de la page d'aide.

1. Connectez-vous à votre machine virtuelle en utilisant la commande `ssh`. Vous êtes l'utilisateur `root` de la machine. En cas de besoin (ou préférence), vous pouvez utiliser l'image de MV disponible sur moodle², ou d'un container docker qu'il faut créer en suivant les instructions du cours.
2. Obtenez le répertoire de travail actuel avec la commande `pwd` (*present working directory*).
3. Déplacez vous vers répertoire temporaire du système de fichiers (commande `cd -change directory`), puis créez un dossier nommé `pickabou` à l'aide de la commande `mkdir` (*make directory*).
4. Affichez le contenu du dossier (commande `ls` pour *list*).
5. Affichez la chaîne de caractères "coucou" à l'aide de la commande `echo`.
6. Tapez la commande suivante `echo "machin" > fichier`. Expliquez le résultat (affichez le contenu du dossier et de fichiers présents).
7. Redémarrez votre machine virtuelle.
8. Vérifiez que les dossiers et fichiers que vous avez créés dans le répertoire `/tmp` n'y sont plus.

²Cette machine est à utiliser avec un logiciel de virtualisation comme VirtualBox

Exercice 2

Voici les caractères selon l'encodage ASCII (**American Standard Code for Information Interchange**). Les listes sont en hexa et décimale.

1. Pouvez-vous identifier les codages ? Lequel est hexadécimale et lequel est décimale ?

Le codage de gauche correspond à un codage en hexadécimal et celui de droite à un codage en décimal.

2. Identifiez le code ASCII pour les lettres du mot "pickabou" en hexadécimal et en décimal.

Le code ASCII en hexadécimale du mot *pickabou* nous donne :

70 69 63 6B 61 62 6F 75.

Le code ASCII en hexadécimale du mot *pickabou* nous donne :

112 105 99 107 97 98 111 117.

3. Obtenez l'encodage binaire du mot.

On va repartir du code en hexadécimale pour cela. Souvenez-vous que chaque caractère du code hexadécimale peut être représenté sur 4 bits, ce qui nous donne la représentation binaire suivante :

$\underbrace{0111\ 0000}_p \underbrace{0110\ 1001}_i \underbrace{0110\ 0011}_c \underbrace{0110\ 1011}_k \underbrace{0110\ 0001}_a \underbrace{0110\ 0010}_b \underbrace{0110\ 1111}_o \underbrace{0111\ 0101}_u.$

4. Vérifiez votre résultat avec la commande `xxd`.

Pensez à créer un fichier *test.txt* qui contient le mot en question puis effectuer la commande suivante dans votre terminal :

```
1 xxd -b test.txt
```

.

ou encore

```
1 echo -e "pickabou" | xxd -b
```

Vous pouvez également vérifier que votre code en hexadécimal est le bon.

- Utilisez la commande `man ascii` pour consulter la page d'aide contenant ces tables (tout en bas de la page), et une autre avec l'encodage en binaire. Ces trois tables sont très utiles en pratique.
- Pouvez-vous dire pourquoi il n'y a pas de colonne 0 et 1 sur la table à gauche ?

Les premiers caractères du code ASCII (plus précisément les 31 premières valeurs) sont des caractères dits de contrôle (les deux autres valeurs sont des espaces).

```
1      2 3 4 5 6 7      30 40 50 60 70 80 90 100 110 120
2      -----
3      0:  0 @ P ' p      0:  ( 2 < F P Z d n x
4      1:  ! 1 A Q a q      1:  ) 3 = G Q [ e o y
5      2:  " 2 B R b r      2:  * 4 > H R \ f p z
6      3:  # 3 C S c s      3:  ! + 5 ? I S ] g q {
7      4:  $ 4 D T d t      4:  " , 6 @ J T ^ h r |
8      5:  % 5 E U e u      5:  # - 7 A K U _ i s }
9      6:  & 6 F V f v      6:  $ . 8 B L V ' j t ~
10     7:  ' 7 G W g w      7:  % / 9 C M W a k u DEL
11     8:  ( 8 H X h x      8:  & 0 : D N X b l v
12     9:  ) 9 I Y i y      9:  ' 1 ; E O Y c m w
13     A:  * : J Z j z
14     B:  + ; K [ k {
15     C:  , < L \ l |
16     D:  - = M ] m }
17     E:  . > N ^ n ~
18     F:  / ? 0 _ o DEL
```

L'encodage ASCII n'utilise que la moitié d'un octal pour représenter un ensemble relativement réduit de caractères utilisés dans le monde. Beaucoup de place reste disponible, ce qui a conduit à l'existence de plusieurs encodages. Aujourd'hui un seul et unique standard international semble se dégager. Il s'agit d'Unicode, développé par le Consortium Unicode, avec un système très ingénieux basé en un nombre de bits extensible (utf-8, utf-16, ...)

Unicode est un super ensemble d'ASCII, c'est-à-dire qu'on peut récupérer en Unicode les mêmes caractères ASCII en utilisant les mêmes codes.

Pour représenter les caractères en Unicode, on utilise souvent comme notation un code de la forme $U+$ $\underbrace{1043}_{\text{code h\u00e9xa}}$. Dans l'invite de commandes, nous rempla\u00e7ons le pr\u00e9fixe $U+$ par $\backslash U$.

- Utilisez la commande `echo -e "\U41"` pour obtenir le caract\u00e8re ASCII qui correspond \u00e0 la position 41 (comparez avec l'écriture avec celle utilis\u00e9e pour l'encodage ASCII).

La commande avec le drapeau `-e` modifie le comportement d'`echo` (essayez la commande sans le drapeau) afin de convertir le code h\u00e9xa entre guillemets.

```
1 echo -e "\textbackslash" echo -e "\U41"
```

8. Cherchez sur internet le code hexadécimal de deux ou trois émojis et obtenez leur affichage dans la console.

```
1 echo -e "\U1F9C9"
```

Exercice 3

1. Trouvez le fichier nommé batD dans votre machine virtuelle (utilisez la commande `locate nom_de_fichier`)

Il suffit de faire

```
1 locate batD
```

Vous obtiendrez le chemin absolu vers le fichier. Il suffit maintenant de changer de répertoire avec `cd` puis le copie/collé du chemin absolu.

2. Utilisez la commande `file` pour déterminer le type de fichier.

Pour obtenir de l'information sur un fichier pour lequel il n'y a pas d'extension ou on doute de sa véracité, on fait

```
1 file batD
```

et on obtient

```
batD: PNG image data, 1280 x 720, 8-bit/color RGBA, non-interlaced
```

Il s'agit d'un fichier image en format PNG.

3. Modifiez le nom du fichier avec la commande `cp source destination` pour rajouter une extension au fichier.

L'extension habituelle pour un fichier PNG c'est `.png` !

```
1 cp batD batD.png
```

On peut vérifier que tout se passe bien avec `ls`.

4. Créez une copie cachée du fichier (pour rappel, les fichiers cachés commencent par un point de leur nom, i.e. `.fichier`).

```
1 cp batD.png .batD_secret
```

Nous devons maintenant faire `ls -a` (drapeau `-a` pour `all`) car les fichiers cachés ne sont pas affichés avec `ls` tout seul.

Exercice 4

Utilisez l'éditeur `nano` pour écrire un fichier contenant le code qui suit

```
1 #include <stdio.h>
2
3 int main( ) {
4     int x = 3 ;
5     int y = 2 ;
6     int accum = 0;
7
8     accum = x + y ; //sum(x, y);
9     printf("%i + %i => %i\n", x, y, accum);
10
11     return(0);
12 }
```

1. Enregistrez le fichier sous le nom de `acc.c`. Obtenez une première compilation du fichier, en mettant comme nom du fichier binaire de sortie `acc`.

```
1 g++ acc.c -o acc
```

2. Observez la table de liens du programme (commande `ldd`).

```
1 ldd acc
```

Le fichier binaire est dynamiquement lié, cad. qu'il utilise de bibliothèques qui doivent être dans le système (vs dans le fichier lui-même). D'autres fichiers peuvent ne pas avoir de lien dynamique, au coup d'avoir une taille plus importante.

Sur Mac, vous pourrez utiliser la commande équivalente suivante

```
1 otool -L acc
```

3. Exécutez le programme. Il se peut que vous devriez changer les droits du fichier pour le rendre exécutable avec la commande `chmod +x acc`.

```
1 chmod +x acc
2 ./acc
```

4. Obtenez le dump du fichier binaire avec `objdump` et gardez le résultat dans un fichier `acc_dump`. Identifiez la section `main` du code assembleur.

```
1 objdump -d acc
2 objdump -d acc > objdump_acc.txt
```

Les sections sont identifiées par un nom commençant par un point. La section qui correspond au point d'entrée de votre programme est donc `.main`. Nous essaierons de comprendre les instructions qui sont dans cette section. Elles correspondent à un encodage précis de points d'opérations (**opcodes**) du processeur. Chaque constructeur de processeur livre sa propre liste de opcodes, en conséquence, si vous

compilez le même programme sur une machine avec une architecture différent (e.g. un Mac avec le processeur ARM, un système embarqué ou une vieille machine en 32b) vous trouverez du code assembleur différent. L'assembleur, c'est le langage le plus proche du processeur que nous pouvons lire 'aisément'.

5. Obtenez des versions alternatives du fichier binaire en modifiant les options d'optimisation du compilateur (drapeaux `-O[0-2|g]`). Faites bien attention à nommer les différentes versions de manière à pouvoir vous repérer par la suite.

Le drapeau `-O` contrôle le niveau d'optimisation du binaire obtenu. Plus le niveau est haut, plus rapide devrait (en théorie) être l'exécutable, car on utilise plus les propriétés spécifiques du processeur disponible.

```
1 g++ -O0 acc.c -o acc_00
2 g++ -O1 acc.c -o acc_01
3 g++ -O2 acc.c -o acc_02
4 g++ -Og acc.c -o acc_debug
```

Le code assembleur peut devenir de plus en plus difficile à lire avec les optimisations plus avancées. En général, lors de la face de débogage du code, nous utiliserons l'optimisation `-Og` afin d'obtenir davantage d'informations pour un code qui sera relativement plus lent.

6. Comparez la taille de chaque fichier binaire (commande `ls`) ainsi que les temps d'exécution.
7. Utilisez le drapeaux `-E` pour obtenir la version intermédiaire de la compilation qui correspond à la précompilation. Nommez le fichier résultant **acc.i**.
8. Utilisez le drapeaux `-S` pour obtenir la version intermédiaire de la compilation qui correspond à l'assemblage. Nommez le fichier résultant **acc.i**.