TD 6 Compilation

L3 INFO, Univ Lumière Lyon 2

2022 – 2023

Dans la dernière partie du cours, nous allons étudier le travail fait par le compilateur pour convertir le code source en code assembleur. Le but de la séance d'aujourd'hui c'est de se familiariser avec une partie des notions liées à l'analyse lexicale, à l'analyse syntactique et à l'analyse sémantique.

Exercice 1 Regardez la vidéo [1], en particulier le processus expliqué dans la partie '*Translating Source Code to Machine Code*' (dès la minute `3:38`). Vous aurez une vision d'ensemble du travail qu'on fera dans ce TP. Puis, installez dans votre séance de travail les outils `flex` et `bison` (que nous utiliserons jeudi prochain).

Exercice 2 (*Analyse lexicale*) Nous allons utiliser le générateur d'analyseur lexical `flex`. L'outil prend en entrée la définition des unités lexicales pour produire un programme `C` capable de reconnaitre ces unités et les transformer selon les règles données. Considérez cet exemple de fichier d'entrée

```
%%
[0-9]+  printf("?");
.       ECHO;
```

1. Après la ligne de début des règles (marquée par les caractères %%), la première colonne du fichier contient des expressions régulières. Que représentent elles ? La deuxième colonne contient les tokens de remplacement. Que feront ces règles ?
2. Créez un fichier de texte plat avec le contenu ci-dessus. Nommez le fichier `cacher_chiffres.l`.
3. Utilisez la commande `flex cacher_chiffres.l`. Elle produit un fichier dans votre répertoire de travail. Quel est le format de ce format ? Quel est son contenu ?
4. Compilez `gcc -o cacher_chiffres lex.yy.c -ll` pour obtenir le parseur. Notez l'option -ll à la fin de la ligne, elle permet le linkage vers la librairie fournie par `flex`.
5. Exécutez votre parseur. Il traduira les lignes que vous donnerez selon les règles que nous avons définies.

Exercice 3 (*Analyse lexicale*) Cette définition du parseur produit un résultat différent. La définition contient trois section. Une première où on définit de variables globales, la section du milieu que vous avez déjà travaillé dans l'exercice 2, puis une section finale avec une redéfinition de la fonction `main`.

```
%{
int numChars = 0, numWords = 0, numLines = 0;
%}
%%
\n {numLines++; numChars++;}
[^ \t\n]+ {numWords++; numChars += yyleng;}
. {numChars++;}
%%
int main() {
    yylex();
    printf("%d\t%d\t%d\n", numChars, numWords, numLines);
}
```

Obtenez le parseur associé, puis utilisez un fichier de texte plat de votre choix pour évaluer ce qui fait votre parseur.

Exercice 4 (*Analyse lexicale*)

```
%%
a*b     printf("1");
(a|b)*b printf("2");
c*      printf("3");
```

Étant donné les jetons suivants et les expressions régulières qui leur sont associées, montrez quelle sortie est produite lorsque cet analyseur `flex` est exécuté sur les chaînes de caractères suivantes :

1. aaabccabbb
2. cbbbbac
3. cbabc

1. Cette vidéo sert aussi à évaluer vos acquis du cours de Compilation. À la fin du cours, vous devriez être en mesure d'expliquer à quelqu'un qui n'a pas suivi le cours toutes les étapes mentionnées dans cette vidéo. Si ce n'est pas le cas, n'hésitez pas à retravailler le cours et à poser de questions à l'équipe enseignante