Compilation

TP 1

Dans le fichier `/etc/network/interfaces`, on définit... des interfaces !

Une interface consiste en une adresse IP, un masque de sous-réseau, une adresse de diffusion et une adresse de réseau. Ces informations sont spécifiées par l'utilisateur ou détectées automatiquement.

— L'interface **loopback**
Le premier interlocuteur de la machine, c'est elle-même. Et pour se parler à elle-même, elle peut utiliser tout simplement le réseau. On déclare l'interface ainsi : `iface lo inet loopback`

— L'interface **dhcp**
Lorsque l'interface obtient sa configuration grâce à un serveur dhcp, on déclare l'interface ainsi : `iface mon_interface inet dhcp`
Par défaut, quand l'interface se montera, elle va envoyer une requête dhcp, le serveur lui répondra en lui donnant une adresse IP, un masque de sous-réseau, une adresse de diffusion, une passerelle et des serveurs DNS.

— L'interface **static**
Parfois il n'est pas pertinent ou pas possible de se servir d'un serveur DHCP. Soit parce que c'est trop lent, soit parce qu'il n'y en a pas, soit parce qu'il ne sait pas donner des adresses fixes à ses clients, ...
Voici un exemple de configuration pour un réseau local

```
iface maison inet static
  address 192.168.0.1
  netmask 255.255.255.0
  gateway 192.168.0.254
```

S'il n'est pas forcément souhaitable de démarrer automatiquement une interface filaire au démarrage quand celle-ci ne sert que rarement, il est en revanche nécessaire de démarrer certaines interfaces avec le système. Le démarage automatique se fait de la façon suivante : `auto mon_interface`

Un fichier `/etc/network/interfaces` respecte donc une grammaire de la forme :

1. $IS \rightarrow I\ IS\$$
2. $IS \rightarrow \Lambda$
3. $I \rightarrow auto\ id$
4. $I \rightarrow iface\ id\ inet\ T$
5. $T \rightarrow loopback$
6. $T \rightarrow dhcp$
7. $T \rightarrow static\ address\ ip\ netmask\ ip\ gateway\ ip$

Un exemple de fichier est :

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet dhcp

iface maison inet static
  address 192.168.0.1
  netmask 255.255.255.0
  gateway 192.168.0.254
```

Exercice 1 : Analyse lexicale L'outil `ocamllex` (https://caml.inria.fr/pub/docs/manual-ocaml/lexyacc.html) permet de réaliser des analyseurs lexicaux. À une expression régulière est associée une action : quand un mot appartenant au langage de l'expression régulière est reconnu, l'action est effectuée. Attention à l'ordre des règles : quand un mot appartient au langage de plusieurs expressions expressions régulières, c'est la première règle qui est choisie.

1. Donner les expressions régulières pour les noms d'interface et les adresses IP ;
2. Compléter le fichier `lexer.mll` pour réaliser un analyseur lexical pour les fichiers `/etc/network/interfaces`. L'action consiste à renvoyer une valeur de type `token` ;
3. Vérifier que les tests unitaires présents dans `main.ml` fonctionnent pour les fichiers fournis ;
4. Écrire d'autres fichiers de tests pour l'analyseur lexical réalisé et les tester :
   `Tp1.Main.analyse_lexicale "test.txt";;`

Exercice 2 : Analyse syntaxique Nous souhaitons maintenant réaliser un analyseur syntaxique en utilisant le langage OCaml et l'outil Menhir (http://gallium.inria.fr/~fpottier/menhir/).

Menhir étant conçu pour réaliser une analyse syntaxique combinée avec une analyse sémantique, il faudra dans un premier temps accepter une syntaxe un peu particulière (cf sources fournies).

1. Dans le fichier `dune`, décommenter la partie liée à Menhir ;
2. Dans le fichier `lexer.mll`, supprimer ou commenter le type `token` et à la place ajouter `open Parser` : lors du couplage des analyseurs lexical et syntaxique, les tokens sont déclarés dans l'analyseur syntaxique ;
3. Dans le fichier `parser.mly`, compléter les tokens (faire attention à garder des noms compatibles avec l'ancien type `token` ou modifier l'analyseur lexical) ;
4. Compléter les règles de production de la grammaire ;
5. Dans le fichier `main.ml`, décommenter la fonction `analyse_syntaxique` et les tests associés ;
6. Vérifier que les tests unitaires fonctionnent pour les fichiers fournis ;
7. Écrire différents fichiers tests pour l'analyseur syntaxique réalisé :
   `Tp1.Main.analyse_syntaxique "test.txt";;`

Exercice 3 : Analyse sémantique (grammaires attribuées) Nous voulons maintenant réaliser un traitement sur le fichier : vérifier que toutes les interfaces lancées automatiquement (`auto`) ont bien été déclarées (`iface`). Pour cela, nous nous appuyons sur les attributs et actions sémantiques proposés par Menhir.

1. Modifier le fichier `parser.mly` de façon à ce qu'un couple de listes (liste des interfaces déclarées, listes des interfaces lancées automatiquement) remonte de la phase d'analyse ;
2. Dans le fichier `main.ml`, décommenter la fonction `analyse_semantique` et les tests associés ;
3. Vérifier que les tests unitaires fonctionnent pour les fichiers fournis ;
4. Écrire d'autres fichiers de tests pour l'analyseur sémantique réalisé :
   `Tp1.Main.analyse_semantique "test.txt";;`