Flex

Pour compiler le fichier `exemple.lex` :

```
# flex exemple.lex
# gcc lex.yy.c -lfl
```

On obtient alors un exécutable `a.out` qu'il faut lancer par `./a.out`.
Par défaut, le programme généré attend que l'on tape au clavier le texte à traiter. Pour quitter le programme, il faut entrer au clavier la commande *fin de fichier*, c'est à dire : Contrôle + D en début de ligne (attention à ne pas taper Contrôle + C qui interromprait le programme sans le laisser terminer son travail).
Il est possible de nourrir le programme à partir d'un fichier texte en utilisant la commande `./a.out < mon-fichier.txt`.

La documentation du logiciel Flex est disponible en utilisant les commandes `man flex` et `info flex`.

Exercice 1 :

Écrire un lexeur `flex` qui supprime les espaces redondants : la sortie du programme sera le texte d'entrée où chaque séquence d'espaces est remplacée par un seul espace.
Modifier ce lexeur pour que tous les espaces apparaissant en début de ligne soient supprimés.

Exercice 2 :

Écrire un lexeur `flex` qui compte le nombre de mots, de nombres et de signes de ponctuation dans un texte.
Attention, dans `12 345,45 euros`, il n'y a qu'un seul nombre et aucun signe de ponctuation !

Exercice 3 :

Écrire un lexeur `flex` capable de reconnaître les entités suivantes :
- les nombres (séquences de chiffres) ;
- les mot-clefs : `begin`, `end` ;
- les identificateurs (séquences de chiffres et de lettres commançant par une lettre, et qui ne sont pas des mot-clefs) ;
- les opérateurs : $+$, $-$, $*$ et $**$.

Les espaces, tabulations et retours à la ligne sont des séparateurs. Toute autre séquence de caractères qui ne forme pas une entité est considérée comme une erreur.
Le lexeur affichera à l'écran la liste des entités reconnues : “nombre 12”, “identificateur abc”, “erreur”, etc.

Exercice 4 :

Une chaîne de caractères est délimitée par des guillemets `"`. À l'intérieur de la chaîne, il est possible de taper des caractères `"` à condition de les précéder de \ (pour ne pas les confondre avec le délimiteur de fin de chaîne).
Ajouter au lexeur précédent la gestion des chaînes de caractères.

Exercice 5 :

En langage C, les commentaires sont encadrés entre `/*` et `*/`. Ils ne peuvent pas être imbriqués.
Ajouter au lexeur précédent la gestion des commentaires C.
Quelle est la sortie générée par `bla /* blo /* bli */ blu */ bligerc` , par `/** toto **/` ?

Exercice 6 :

Refaire l'exercice précédent, mais en gérant les commentaires imbriqués (utiliser les *start conditions*).
Si la fin du texte d'entré est atteinte alors qu'un commentaire n'est pas terminé, on affichera une erreur.

Exercice 7 :

Écrire en `flex` un macro-processeur qui permet de définir des abréviations.
Une ligne de la forme `.def toto du texte` définit une nouvelle macro `toto` associée au texte `du texte`. Dans la suite du texte, chaque occurrence du mot `toto` sera remplacée par le contenu de la macro : `du texte`. (On pourra, pour cela, stocker les couples (macro,texte de remplacement) dans des tableaux C de chaînes de caractères.)

Exercice 8 :

Écrire en `flex` un macro-processeur qui permet l'inclusion de fichiers.
Une ligne de la forme `.include blha` sera remplacée par le contenu du fichier `blha`. Le fichier `blha` peut également contenir des directives d'inclusion. (On pourra, pour cela, gérer une pile de `YY_BUFFER_STATE` et utiliser la commande `yy_switch_to_buffer`.)