Fiche de TD-TP no. 1

Exercice 1 : *gcc.*
- Expliquez la fonction des options `-c`, `-S`, `-E` du compilateur `gcc`. Pour chaque option, proposer un cas d'utilisation.
- Expliquez la fonction des outils `cpp`, `as`, et `ld`.
- Proposez un diagramme de flux pour le compilateur `gcc`. Proposez une suite de commandes, qui eclaircissent ce flux, dont le résultats est équivalent à `gcc helloworld.c`.

Exercice 2 : *Analyse lexicale, syntaxique, traduction, évaluation et typage.* Exercice no. 2.6 de Amadio.

Exercice 3 : *Typage.* Montrez qu'ils existent des expressions du langage `lse` qui s'évaluent, mais qui ne peuvent pas être typés.

Exercice 4 : *Syntaxe abstraite.* Proposez des structures de donnés en C, aptes à représenter la syntaxe abstraite du langage `lse`.

Expressions regulières

Exercice 5. Nommez au moins dix programmes qui utilisent les expressions régulières.

Exercice 6. Rappelez la définition, informelle et formelle, d'expression régulière. Proposez des expression régulières (sur l'alphabet $\{a, b\}$) pour les langages suivants :
- le langage des mots avec un seul b,
- le langage des mots avec un seul b, et au moins un a,
- le langage des mots où tout a est suivi par b, et tout b est suivi par a.

Exercice 7. Proposez des expression régulières, sur l'alphabet ASCII, pour denoter :
- l'ensemble des mots clès du langage `lse`,
- l'ensemble des identificateurs (variables) du langage `lse`,
- l'ensemble des constantes entières du langage `lse`.

Exercice 8 : *Prise en main de l'outil lex-flex.* Complétez le ficher `lexeur.lex` suivant. La commande `lex lexeur.lex ; cc lexeur.c` engendrera un programme `a.out` qui, en faisant l'analyse lexicale d'un fichier `fichier.lse`, affichera à l'écran la catégorie des lexèmes rencontrés.

```
/* Section definitions */
%option noyywrap

%%
        /* Debut section regles */


%%
/* Section code additionnel */
int
main(int argc, char* argv[])
{
  if ( argc > 1 ) {
    yyin = fopen( argv[1], "r" );
    yylex();
```

```
    }
    fclose(yyin);
    exit(0);
}
```

Exercice 9. Améliorez l'exercice 8 afin qu'un message d'erreur apparaisse si le fichier en entrée contient un caractère qui ne peut pas être prolongé à un lexème du langage `lse`. Le message d'erreur produira le numéro de ligne où l'erreur se produit, le numero de ce caractère sur la ligne, et le caractère lui même.

Exercice 10. Améliorez encore l'exercice 8 afin que le message d'erreur affiche non pas un seul caractère, mais la plus longue suite – contiguë, non espacée – de caractères qui ne sont pas des début de lexèmes. Par exemple, la suite `; ;` donnera lieu à un seul message d'erreur au lieu de deux messages. Le message affichera le nombre (la colonne) de début et le nombre de fin de cette suite dans la ligne.

Proposition pour d'autres exercices (en TP)

Exercice 11. Écrivez un programme `virt.c` qui soit une implémentation de la machine virtuelle présentée en cours. La commande `virt fichier.lse` exécutera la séquence d'instructions `lse` contenue dans `fichier`. Après avoir exécuté une instruction `return`, elle affichera à l'écran le contenu de la case $M[0]$.

Exercice 12 : *Prise en main de yacc-bison.* Modifiez le fichier `lexeur.lex` de l'exercice 8 et intégrez-le avec un fichier `parseur.y` qui fasse l'analyse syntaxique d'un fichier contenant un programme `lse`. On produira un programme `lseparseur` qui affichera à l'écran si un programme est syntaxiquement correcte ou non.

Exercice 13. Ajoutez des actions au fichier `parseur.y` de l'exercice 12 pour générer votre premier compilateur. Le programme affichera sur la sortie standard une traduction d'un programme `lse` dans le langage de la machine virtuelle.

Exercice 14. Dans cet exercice nous nous proposons d'écrire un assembleur pour la machine virtuelle.
Modifiez votre exercice 11 afin que la machine virtuelle lise un fichier binaire (codez les instructions de la machine par des entiers).
Utilisez l'outil lex pour générer un traducteur du langage symbolique de la machine virtuelle vers son codage par entiers.