# Analyse Syntaxique et Compilation, TP n° 2 : Menhir

Pour effectuer votre TP, il faut récupérer le répertoire `/ens/meyera/Public/Compil/tp2`. Pour chaque exercice, vous pourrez prendre les fichiers de ce répertoire comme point de départ pour travailler.

### Exercice 1 : Première spécification de grammaire Menhir.

Soit l'alphabet $T \equiv \{\mathbf{true}, \mathbf{false}, \mathbf{id}, \vee, \wedge, (, )\}$. Voici une grammaire décrivant la syntaxe des expressions booléennes et utilisant $T$ pour ensemble de terminaux :

$$\begin{array}{rcl} e & ::= & \mathbf{id} \\ & | & \mathbf{true} \\ & | & \mathbf{false} \\ & | & e \vee e \\ & | & e \wedge e \\ & | & (e) \end{array}$$

1. Spécifier l'ensemble des terminaux à l'aide de la directive `%token` dans le fichier `parser.mly`.
2. Modifier le fichier `lexer.mll` pour reconnaître cet ensemble de non terminaux lors de l'analyse lexicale. Pour $\wedge$, on prendra par exemple "`/\`". Pour représenter les identificateurs de variables (terminal **id**), on pourra prendre par exemple le langage `['a'-'z']+`. N'oubliez pas d'ignorer les espaces et les retours à la ligne.
3. Modifier le fichier `parser.mly` en mimant la définition ci-dessus.
4. Lire attentivement le fichier `parser.conflicts` généré par `menhir`. Affecter des priorités raisonnables aux règles de production pour résoudre ces conflits. Pour cela, utiliser les directives `%left`, `%right` ou `%nonassoc` (voir la documentation pour comprendre leur fonctionnement).

### Exercice 2 : Un conflit classique.

On ajoute à $T$ les terminaux $\{\mathbf{if}, \mathbf{then}, \mathbf{else}, =\}$. On définit une syntaxe pour les instructions d'un langage très simple :

$$\begin{array}{rcl} i & ::= & \mathbf{if}\ e\ \mathbf{then}\ i \\ & | & \mathbf{if}\ e\ \mathbf{then}\ i\ \mathbf{else}\ i \\ & | & (i) \\ & | & \mathbf{id} = e \end{array}$$

1. Modifier le fichier `lexer.mll` pour qu'il tienne compte des nouveaux terminaux.
2. Rajouter la définition de $i$ dans le fichier `parser.mly`.
3. Comprendre le conflit détecté par `menhir` en donnant un exemple d'entrée qui peut être analysée de deux manières différentes.
4. Implémenter deux spécifications de grammaire correspondant à ces deux variantes.

**Exercice 3 : Une grammaire LR(2).**

On modifie ainsi la spécification de l'exercice précédent :

$$\begin{array}{lcl} i & ::= & \ldots \\ & | & ma \\ ma & ::= & a \\ & | & a \wedge ma \\ a & ::= & \textbf{id} = e \end{array}$$

1. Faire les modifications nécessaires dans le fichier `parser.mly` pour implémenter telle quelle cette définition.
2. Modifier la structure de la grammaire pour qu'elle définisse le même langage mais soit LR(1) et donc adaptée à `menhir`.
3. Revenir à la grammaire initiale et utiliser le mot-clé `%inline` pour obtenir le même résultat que celui de la question 2.