Bison

Pour compiler le fichier `exemple.y` :

```
# bison -v exemple.y
# gcc exemple.tab.c
```

Le fichier `exemple.y` doit contenir les définitions des fonctions :
- `int yylex()` qui renvoie le prochain caractère à traiter ;
- `void yyerror(char* s)` appelée en cas d'erreur de syntaxe.

La documentation du logiciel Bison est disponible en utilisant les commandes `man bison` (options de ligne de commande uniquement) et `info bison` (documentation détaillée).

Exercice 9 :

Écrire un programme `bison` qui lit une ligne de texte et affiche `OK` sur l'écran si la ligne peut être générée par la grammaire $\begin{cases} list \rightarrow \texttt{(}in\texttt{)} \mid \texttt{a} \\ in \rightarrow list \mid in\texttt{,}list \end{cases}$

Il faudra définir une fonction `yylex` qui fournit à `bison` le prochain caractère (on pourra utiliser la fonction `getchar`).

Exercice 10 :

Écrire un parseur qui accepte les mots générés par la grammaire $\begin{cases} list \rightarrow DEBUT\ in\ FIN \mid ID \\ in \rightarrow list \mid in\ SEPARE\ list \end{cases}$ où :
- le terminal $DEBUT$ peut être le caractère `(` ou le mot-clef `begin` ;
- le terminal FIN peut être le caractère `)` ou le mot-clef `end` ;
- le terminal $SEPARE$ peut être `, . ;` ou `:` ;
- le terminal ID est un identificateur quelconque ;
- les espaces, tabulations et retours à la ligne sont ignorés.

Cette fois, la fonction `yylex`, complexe, sera générée automatiquement par `flex`.

Pour compiler un couple `bison.y`, `flex.lex` :

```
# bison -d -v bison.y
# flex flex.lex
# gcc lex.yy.c bison.tab.c -lfl
```

Exercice 11 :

Soit la grammaire suivante pour la *notation polonaise inversée* des expressions mathématiques :
$expr \rightarrow NOMBRE \mid expr\ expr\ + \mid expr\ expr\ - \mid expr\ expr\ * \mid expr\ expr\ /$

Écrire un parseur `bison` pour ces expressions.

Modifier le parseur pour qu'il calcule la valeur de l'expression et qu'il l'affiche sur l'écran. (Pour cela, il faut associer une *valeur entière* au terminal NOMBRE *via* `yylval` et au non terminal *expr* *via* des actions sémantiques de la forme `$$=$1+$2`.)

Exercice 12 :

Soit la grammaire $expr \rightarrow expr + expr \mid expr * expr \mid NOMBRE \mid (expr)$.
Montrer qu'elle est ambiguë. Comment `bison` indique-t-il cela ?

Pour lever l'ambiguïté, on décide que les $+$ et $*$ associent à gauche et que $*$ a priorité sur $+$.
Écrire un parseur `bison` sans conflit pour cette grammaire en utilisant des directives de précédence et d'associativité. Le parseur calculera et affichera sur l'écran la valeur de l'expression entrée.

Exercice 13 :

On étend la grammaire de l'exercice précédent à
$expr \rightarrow expr + expr \mid expr - expr \mid expr * expr \mid -expr \mid NOMBRE \mid (expr)$.

Le $-$ unaire est prioritaire sur $+$, $-$, et $*$ alors que $*$ est prioritaire sur le $-$ binaire. De plus, tous les opérateurs binaires associent à gauche.
Refaire l'exercice précédent pour cette nouvelle grammaire.

Exercice 14 :

Soit la grammaire suivante qui gère des expressions arithmétiques et chaînes de caractères :

$$\begin{cases} iexpr \rightarrow NOMBRE \mid iexpr + iexpr \mid iexpr * iexpr \mid \texttt{len}(sexpr) \mid (iexpr) \\ sexpr \rightarrow CHAINE \mid sexpr\texttt{@}sexpr \mid \texttt{eval}(iexpr) \mid (sexpr) \end{cases}$$

où $CHAINE$ est une chaîne de caractères entre `"`, `@` est l'opérateur de concaténation de chaînes, `len(l)` est la longueur de la chaîne `l` et `eval(e)` est le résultat de l'évaluation de l'expression `e`

Écrire un parseur qui évalue une expression chaîne de caractères et affiche sa valeur sur l'écran. (On associera des valeurs de type différent selon que l'expression est entière ou chaîne de caractères grâce au mot-clefs `%union` et `%type`.)

Essayer le parseur sur l'exemple : `"2+3 vaut " @ eval(2+3) @ " et la chaîne \"titi\" a pour longueur " @ eval(len("titi"))`.

Exercice 15 :

Soit le fragment de grammaire suivant :

$$\begin{array}{lcl} inst & \rightarrow & \texttt{if}\ expr\ \texttt{then}\ inst \\ & \mid & \texttt{if}\ expr\ \texttt{then}\ inst\ \texttt{else}\ inst \\ & \mid & ID := expr \end{array}$$

Montrer que cette grammaire est ambiguë.
Comment `bison` gère-t-il cette ambiguïté ?