Travaux Dirigés Compilation: TP1

Informatique 2ème année. ENSEIRB-MATMECA 2014/2015

——

Une fiche de présentation de yacc/bison est disponible par le document `http://moodle.ipb.fr/mod/resource/view.php?id=16447` sur moodle.

Une documentation plus exhaustive est disponible par le document `http://moodle.ipb.fr/mod/resource/view.php?id=16448` concernant `lex` et `yacc`.

Utilisation de lex/flex

L'outil `lex` est un outil qui permet de générer des analyseurs lexicaux. On lui donne un fichier d'entrée, dont la syntaxe est décrite après, et il construit à partir d'expressions régulières, un automate les reconnaissant. Pour chaque expression régulière, on peut déclencher une action, qui sera décrite par du code C. La fonction générée par `lex` qui fait l'analyse lexicale s'appelle `yylex()`. Quand l'analyseur lexical est associé à un analyseur syntaxique, l'action déclenchée à chaque expression régulière consiste à retourner le lexème reconnu.

Un fichier d'entrée pour `lex` suit le squelette suivant. On utilise le suffixe `.l` pour ce type de fichiers :

```
%{
#include <stdio.h>
... code C optionnel ...
%}
Nommage d'expression regulieres
%%
Liste des expressions regulières / actions
%%
... code C optionnel ...
```

La liste des expressions régulières à reconnaitre, avec leur action, consiste en une liste de lignes de la forme :

```
expression-reguliere    { code C a executer quand elle est reconnue }
```

Dans le code C associé à une action régulière, la variable prédéfinie `char *yytext` contient la chaine de caractères correspondant à l'expression reconnue.

```
%{
#include <stdio.h>
int num_lines = 0, num_chars = 0;
%}
%%
\n      {  ++num_lines; ++num_chars; }
.       { ++num_chars; }

%%

int main() {
  yylex();
  printf( "# of lines = %d, # of chars = %d\n",
          num_lines, num_chars );
}
```

L'ordre des expressions est important : `lex` cherche à reconnaitre l'expression régulière correspondant au mot le plus long, puis en cas d'égalité, prend la première expression apparaissant dans le texte.

On peut donner des noms à des expressions régulières de la façon suivante, avec une définition par ligne. Par exemple,

```
DIGIT [0-9]
```

définit un chiffre. Ce nom peut être utilisé dans les autres expressions régulières en entourant le nom d'accolades : `{DIGIT}+` reconnait les nombres.

▸**Exercice 1.** *Prise en main de lex*

1. *Recopier l'exemple précédent, qui compte le nom de lignes et de caractères d'un fichier. Compiler le fichier avec*

   ```
   lex monfichier.l
   ```

   puis compiler avec gcc le fichier `lex.yy.c` *généré en un binaire* `count`*, avec l'option* `-ll` *(pour lex) ou* `-lfl` *(pour flex). Pour l'utiliser, faire* `count < monfichier`*.*
2. *Ecrire un fichier de description lex qui affiche chaque nombre à virgule flottante d'un fichier. Pour cela, on utilisera la variable prédéfinie* `yytext`*, déclarée comme tableau de* `char` *et qui contient à chaque action la chaine de caractères reconnue.*

Utilisation de yacc/bison

Un fichier yacc `.y` décrit une grammaire algébrique. La syntaxe est la suivante :

```
%{
 ... code C initial ...
%}
 ... declaration des lexemes et des types ...
%%
... regles de grammaire ...
%%
... Code C optionnel
```

La syntaxe de chacune des parties est exposée sur la fiche de présentation.

On prendra en exemple la grammaire du projet. Dans les sources du projet, il y a un fichier lex `scanner.l` pour la reconnaissance des lexèmes, un fichier yacc `parser.y` pour la grammaire, utilisant les lexèmes précédemment reconnus.

▸**Exercice 2.** *Utilisation et modification de la grammaire*

1. *Télécharger les sources du projet qui vous sont fournies, compiler les avec* `make`*. Lancer l'analyseur sur l'un des exemples fournis.*
2. *Ajouter le mot clé* `do` *dans l'analyseur lexical* `scanner.l` *, le déclarer comme token dans* `parser.y` *puis modifier la grammaire pour ajouter une boucle* `do .. while(..)` *similaire à celle du C. Tester que ces instructions sont effectivement bien reconnues.*

▸**Exercice 3.** *Actions sémantiques*

1. *3 lexèmes ont des valeurs définies passées comme chaines de caractères. A quoi correspondent-ils ?*
   Ces valeurs sont passées par une variable globale, `yylval` *dont le type est une union définie par* `%union` *dans* `parser.y`*. Chaque lexème doit déclarer quel champ de l'union il utilise par une ligne*
   `%token <nomduchamp> nomdulexeme`
   De même, les variables de la grammaire peuvent avoir un attribut qui est l'un des champs de l'union, et ils sont déclarés par une ligne
   `%type <nomduchamp> nomdevar`
2. *Les actions sémantiques peuvent être définies par du code C entre accolades à la fin de chaque règle. Les symboles de grammaire, lexèmes et variables sont accessibles ainsi que leurs attributs par des variables spéciales dans ces actions :* `$$` *pour le symbole à gauche de la règle,* `$n` *pour le nième. Ainsi, la règle*
   $E_1 \rightarrow E_2 + E_3$ $\qquad E_1.v = E_2.v + E_3.v$
   s'écrirait en yacc :

   ```
   E : E '+' E          { $$.v = $1.v + $3.v ;  }
   ```

   Tous les symboles sont comptés, y compris '+'.
   Ajouter des actions sémantiques dans `parser.y` *pour afficher la valeur de ces 3 lexèmes à chaque fois qu'ils apparaissent.*

3. *Modifier l'union pour que la valeur des lexèmes entiers et flottants soit des nombres entiers et flottants. Modifier également la façon dont ces valeurs sont créés dans* `scanner.l` *à l'aide des fonctions* `atoi` *et* `atof`*.*
4. *Définir un codage en C pour représenter les types des identificateurs (parmi int, float, int *, float *, les tableaux à une dimension et les fonctions).*
5. *Ajouter les actions sémantiques pour définir un attribut* `type` *pour la variable* `declarator`*, partout où elle est utilisée.*
6. *Déclarer et utiliser une variable globale* `int level` *comptant le niveau d'imbrication de bloc dans lequel se situe les instructions qu'on reconnait. On incrémentera/décrémentera ce compteur à chaque changement de bloc.*
7. *On utilisera par la suite la table de hachage proposée par la libc (*`man hcreate` *pour la documentation et un exemple). Pour chaque déclaration de variable, stocker dans la table son nom, son type et le niveau où elle est déclarée. On affichera par ailleurs ces informations pour vérifier leur correction.*