Travaux Dirigés Compilation: Feuille 4

Informatique 2ème année. ENSEIRB 2018/2019

—David Janin & Myriam Desainte-Catherine—

Tous les fichiers ou répertoire décrits se trouve sous `~janin/TP-Compilation`.

▸**Exercice 1. Grammaire ETF** *(comprendre les interactions LEX/YACC). On trouve dans le répertoire* `ETF` *une mise en oeuvre complète de la grammaire ETF via une spécification de son analyseur lexical dans* `calculette.l`*, une spécification de son analyseur syntaxique dans* `calculette.y` *et une spécification des dépendances associées et des commandes à lancer dans* `Makefile`*.*

(0) En observant le `Makefile`*, observer quels sont les fichiers générés par* `lex`*,* `bison` *et* `gcc`*. Dans quels langages sont-il ? Quels sont les dépendances entre les fichiers C générés ?*

(1) Analyseur syntaxique (`calculette.y`*) : quels sont les terminaux et non-terminaux de la grammaire ? Comment sont-ils déclarés ? Quelles sont les règles de grammaire ? le start symbol ?*

(2) Analyseur lexical (`calculette.l`*) : comment sont spécifiés les langages associés aux terminaux ? Quel peut-être le rôle des variables prédéfinies* `yytext` *et* `yylval` *?*

(3) Compiler (avec `make`*) et lancer l'exécutable* `parse` *produit. Il attend, via la fonction* `yyparse()`*, une expression arithmétique terminé par "fin de fichier" (*`CTRL D`*) sur* `stdin`*. Essayer avec quelques exemples simples.*

(4) Repérer les `printf` *dans les analyseurs lexicaux et syntaxiques. Sur l'entrée "1+2*3" quel* `printf` *affiche quelle sortie ? Reconstituer à l'aide de ces sorties la construction de l'arbre de dérivation associé et l'entrée et les actions des deux analyseurs,*

(5) Modifier les actions sémantiques pour numéroter les affichages à partir de 1. Pour cela, on pourra utiliser une variable globale placée juste après les `include` *dans le fichier calculette.y*

(6) Ajouter des actions sémantiques pour faire les calculs lorsque tous les arguments sont des constantes. Indications, on pourra utiliser les attributs des terminaux (produit dans `calculette.l` *via la variable* `yylval`*) et non terminaux (produit dans* `calculette.y` *via des actions sémantiques adéquates). On affichera à la fin du calcul "Le résultat est ...".*

▸**Exercice 2. Grammaire EEP**, *la grammaire de Pascal. Repertoire* `EEP`*. Pour vérifier qu'on a compris, même questions (4) et (6) comme ci-dessus.*

▸**Exercice 3. Une petite calculatrice** *(répertoire* `Var-Cal`*). La première partie de cette exercice vise à comprendre le code proposé.*

(1.1) Quel est le rôle joué par la table des chaines ? Quel est la fonction du type `sid` *?*

(1.2) Quel est la structure de la table des symboles ? Quel est son rôle ?

(1.3) Quel est le sens de la déclaration

```
%union {
int val;
char * sid;
}
```

dans les spécifications `YACC`*. Où sont déclarés les types des attributs des terminaux ? des non terminaux ? Où sont-il calculés ?*

(1.4) Une fois le code compilé et lancé, dans quelles conditions les variables peuvent-elles être utilisées en partie gauche d'affectation ? En partie droite d'affectation ? Où, quand et comment sont lues ou mémorisés les valeurs des variables ?

La seconde partie de l'exercice consiste à étendre notre calculatrice au type flottant avec la vérification de type associée.

(2.1) Ajouter le terminal `FLOAT` *(dans quels fichiers) et la définition associée des constantes décimales (dans quel fichier).*

(2.2) Etendre le type des valeurs de symboles en conséquences, en ajoutant notament un champs `symb_type` *permettant d'expliciter le type du symbole.*

(2.3) Ajouter aux actions sémantiques des règles de vérification de type en suivant la convention usuelle selon laquelle `int` *est un sous-type de* `float`*.*

(2.4) Combiner les questions précédentes pour obtenir la calculatrice souhaitée ainsi étendue.

La troisième partie de cet exercice consiste à remplacer l'évaluation (c'est-à-dire l'interprétation) de ce code de calculatrice, par sa compilation en code 3 adresses.

(3.1) Remplacer les actions d'évaluation par des actions de production de code afin de produire du code C à trois adresses d'instructions de l'une des formes suivantes `ri = x` *(lecture mémoire)* `y = rj` *(écriture mémoire) ou* `ri = rj op rk` *(opération) ou* `rf1 = (float) ri3` *(conversion explicite).*

(3.2) Veillez à ajouter toutes les déclarations de variables nécessaires dans le code produit. On distinguera soigneusement les déclarations de variables "utilisateurs" des déclarations de "registres", qui seront typés `int` *ou* `float`*. Toutes les conversions de types devront être explicitées.*

(3.3) Proposer quelques exemples de tests de votre code.

(3.4) (correction/robustesse) Dans le cas où des noms de registres internes se trouvent dans le code source, votre code cible est-il correct ? Si ce n'est pas le cas, comment y remédier ?