Université Bordeaux
Licence, module Compilation, 2017/2018

TD3 – Analyse syntaxique

Exercice 3.1

Après avoir rappelé pourquoi il est impossible d'écrire une expression régulière qui reconnaît le langage $\{a^n b^n \mid n \geq 0\}$, écrire une grammaire algébrique sur le monoïde libre $\{a, b\}^*$ qui le reconnaît.

Exercice 3.2

Écrire un document Makefile qui permet de l'automatiser la compilation en tenant compte des suffixes suivants :

Suffixe	type de fichier
.h	*header* C
.c	code C
.l	code lex (Flex)
.y	code yacc (Bison)
.o	binaire objet
aucun	exécutable
.in	input
.out	output

Exemple de session de compilation :

```
yacc --file-prefix=anasynt -d anasynt.y
cc -std=c89 -c -o anasynt.o anasynt.c
lex -o analex.c analex.l
cc -std=c89 -c -o analex.o analex.c
cc -o anasynt anasynt.o analex.o
./anasynt < test.in > test.out
```

Exercice 3.3

Écrire un programme yacc (bison) `td3.3.y` qui reconnaît le langage $\{a^n b^n \mid n \geq 0\}$. Pour cela, on utilisera les implémentations suivantes des fonctions `yylex()`, `yyerror()` et `main()` :

```
int yylex(){
  return getchar();
}

int yyerror(char *s){
  fprintf( stderr, "*** ERROR: %s\n", s );
  return 0;
}

int main(int argn, char **argv){
```

```
  yyparse();
  return 0;
}
```

Exercice 3.4

Écrire un programme lex (flex) `td3.4.l` où la fonction `yylex()` renvoie la constante `NUMBER` égale à zéro pour la lecture d'un nombre entier et sinon, le code ISO-8859-1 du caractère courant.
Par exemple une analyse de `458+98*(45+8)` par appels successifs de `yylex()` devra renvoyer successivement les valeurs 0, 43, 0, 42, 40, 0, 43, 0, 41.
Exemple d'utilisation de la fonction `yylex()` :

```
int main(){
 int c;
 while ((c=yylex())!='\n')
  printf("%d\n", c);
}
```

Exercice 3.5

Soit la grammaire (dite *ETF*) suivante :

E → E + T
E → E - T
E → T
T → T * F
T → T / F
T → F
F → NUMBER
F → (E)

Écrire un programme lex (flex) `td3.5.lex.l` et un programme `td3.5.yacc.y` qui permet d'analyser une expression arithmétique suivant cette grammaire où les opérandes sont des entiers.

Exercice 3.6

Soit la grammaire suivante :

E → T + E
E → T - E
E → T
T → F * T
T → F / T
T → F
F → NUMBER
F → (E)

Reconnaît-elle le même langage que la précédente ? Permet-elle de réaliser les mêmes dérivations gauches que la précédente ? Présente-t-elle un avantage, un inconvénient par rapport à celle-là ?

Exercice 3.7

Soit la grammaire suivante dite *EEE* :

E → E + E
E → E - E
E → E * E
E → E / E
E → NUMBER
E → (E)

Reconnaît-elle le même langage que la précédente ? Permet-elle de réaliser les mêmes dérivations gauches que la précédente ? Présente-t-elle un progrès technique par rapport à celle-là ?

Exercice 3.8

Reprendre la grammaire *ETF* et ajouter des attributs entiers pour réaliser l'affichage du calcul arithmétique sur des entiers.

$5 + 8/2/2$ doit afficher 7

Exercice 3.9

Reprendre la grammaire *EEE* et ajouter des relations de priorité entre les opérateurs et des attributs entiers pour réaliser exactement le même résultat.