

TD3 – Analyse syntaxique

Exercice 3.1

Écrire une grammaire algébrique sur le monoïde libre $\{a, b\}^*$ qui reconnaît le langage $a^n b^n$.

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	include C
.c	programme C
.o	binaire objet
.l	Lex (Flex)
.y	yacc (Bison)
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 $a^k b^k$. 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) `td3.4.1` tel que la fonction `yylex()` renvoie la constante `NUMBER` pour l'analyse d'un nombre entier et sinon, le caractère courant.

Par exemple l'analyse de `458+98*(45+8)` devra renvoyer successivement les valeurs 0, 43, 0, 42, 40, 0, 43, 0, 41 où la constante `NUMBER` vaut 0.

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 :

$$\begin{aligned} E &\rightarrow E + T \\ E &\rightarrow E - T \\ E &\rightarrow T \\ T &\rightarrow T * F \\ T &\rightarrow T / F \\ T &\rightarrow F \\ F &\rightarrow \text{NUMBER} \\ F &\rightarrow (E) \end{aligned}$$

Écrire un programme lex (flex) `td3.5.lex.1` 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 :

$$\begin{aligned} E &\rightarrow T + E \\ E &\rightarrow T - E \\ E &\rightarrow T \\ T &\rightarrow F * T \\ T &\rightarrow F / T \\ T &\rightarrow F \\ F &\rightarrow \text{NUMBER} \\ F &\rightarrow (E) \end{aligned}$$

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 :

$E \rightarrow E + E$
 $E \rightarrow E - E$
 $E \rightarrow E * E$
 $E \rightarrow E / E$
 $E \rightarrow \text{NUMBER}$
 $E \rightarrow (E)$

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 3.8

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

10/2/2 doit afficher 2