PLT Projet

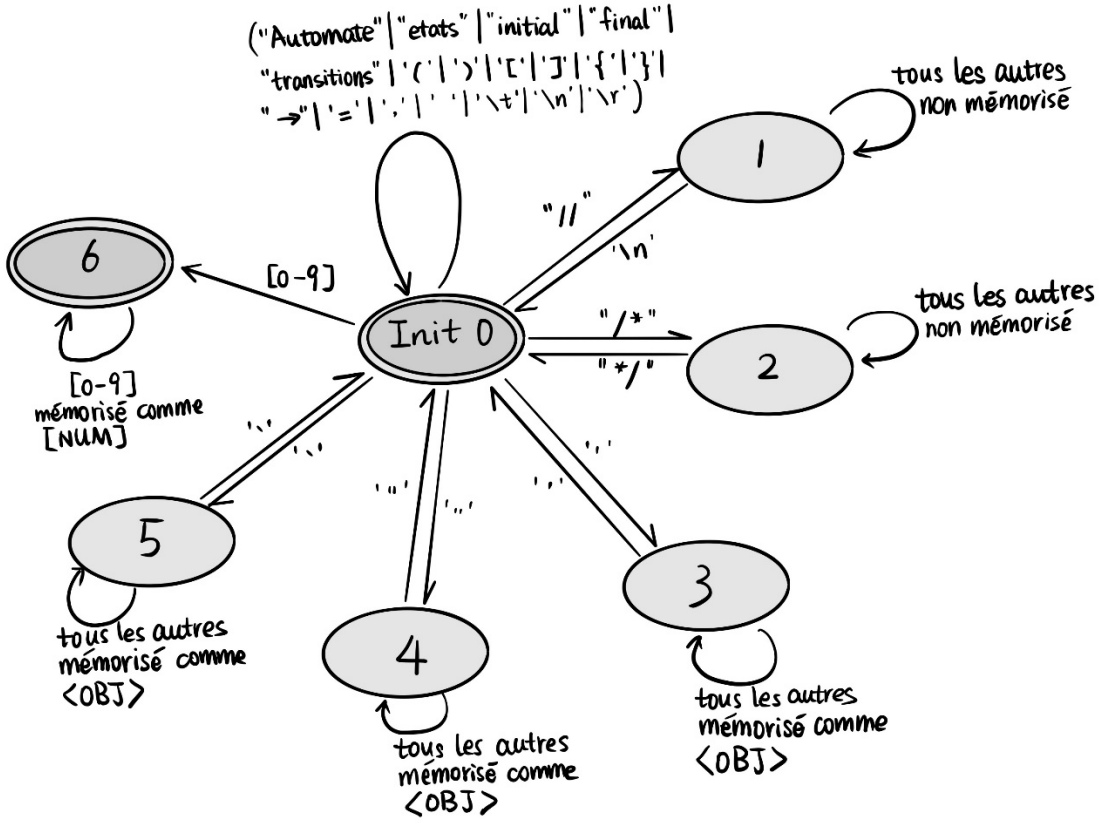
Charlie 516260910011

1. Analyse Lexicale

On a défini 18 lexèmes :

- <NONE>
- <KW_AUTOMATE> ::= "Automate"
- <KW_ETATS> ::= "etats"
- <KW_INITIAL> ::= "initial"
- <KW_FINAL> ::= "final"
- <KW_TRANSITIONS> ::= "transitions"
- <SEPAR> généré avant les mots clés
- <PAREN_L> ::= "("
- <PAREN_R> ::= ")"
- <BRACKET_L> ::= "["
- <BRACKET_R> ::= "]"
- <BRACE_L> ::= "{"
- <BRACE_R> ::= "}"
- <ARROW> ::= "→"
- <EQUAL> ::= "="
- <COMMA> ::= ","
- <OBJ> ::= ’.*?’|”.*?”|`.*?`
- <NUM> ::= [0-9]+

On utilise l’automate pour l’analyse lexicale.



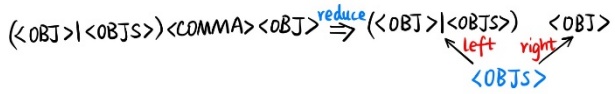
2. Analyse Syntaxique

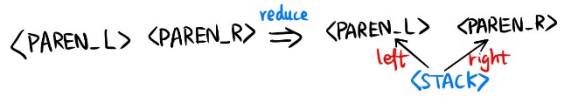
On a défini une grammaire BNF comme suivantes :

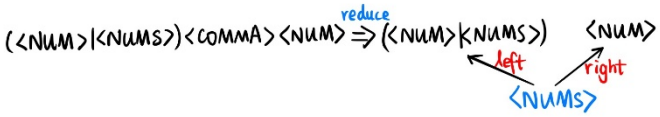
- <OBJS> ::= (<OBJ>|<OBJS>) <COMMA> <OBJ>
- <NUMS> ::= (<NUM>|<NUMS>) <COMMA> <NUM>
- <NTONS> ::= <NUM> <ARROW> <NUM>
- <EDGE> ::= <NTON> <COMMA> <OBJ>
- <STACK> ::= (<PAREN_L> <ARROW> <COMMA> <OBJ> <PAREN_R> |
  <PAREN_L> <OBJ> <COMMA> <ARROW> <PAREN_R> |
  <PAREN_L> <COMMA> <PAREN_R>)
- <EANDS> ::= (<EDGE>|<EANDS>) <COMMA> <STACK>
- <TRAN_UNIT> ::= (<PAREN_L> (<EDGE>|<EANDS>) <PAREN_R> |
- <TRAN_UNIT> <COMMA> <TRAN_UNIT>)
- <BLOCK> ::= (<KW_ETATS> <EQUAL> <BRACKET_L> (<OBJ>|<OBJS>) <BRACKET_R> |
  <KW_INITIAL> <EQUAL> <NUM> |
  <KW_FINAL> <EQUAL> <BRACKET_L> <NUM>|<NUMS> <BRACKET_R> |
  <KW_TRANSITIONS> <EQUAL> <BRACKET_L> <TRAN_UNIT> <BRACKET_R> |
  <BLOCK> <BLOCK>)
- <AUTOMATE> ::= <KW_AUTOMATE> <PAREN_L> <NUM> <PAREN_R> <EQUAL> <BRACE_L>
  <BLOCK> <BRACE_R>

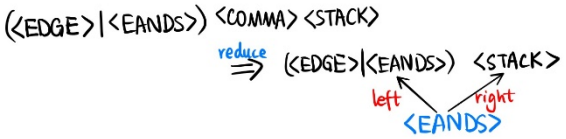
On utilise l'arbre binaire et LR(0) pour générer une arbre syntaxique.

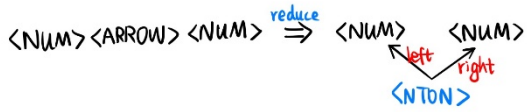
Les règles de "reduce" opération sont représentées comme suivantes :

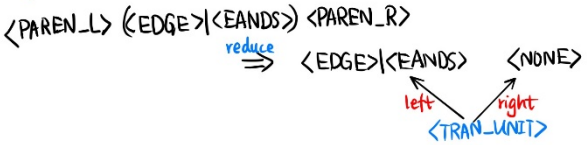


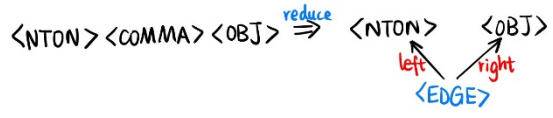


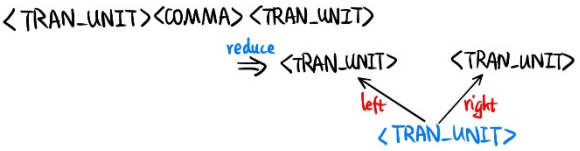


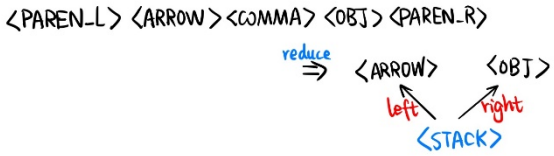


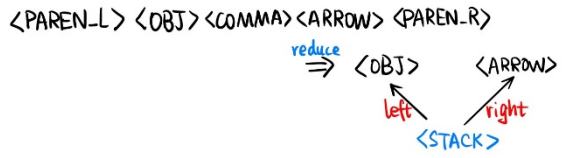


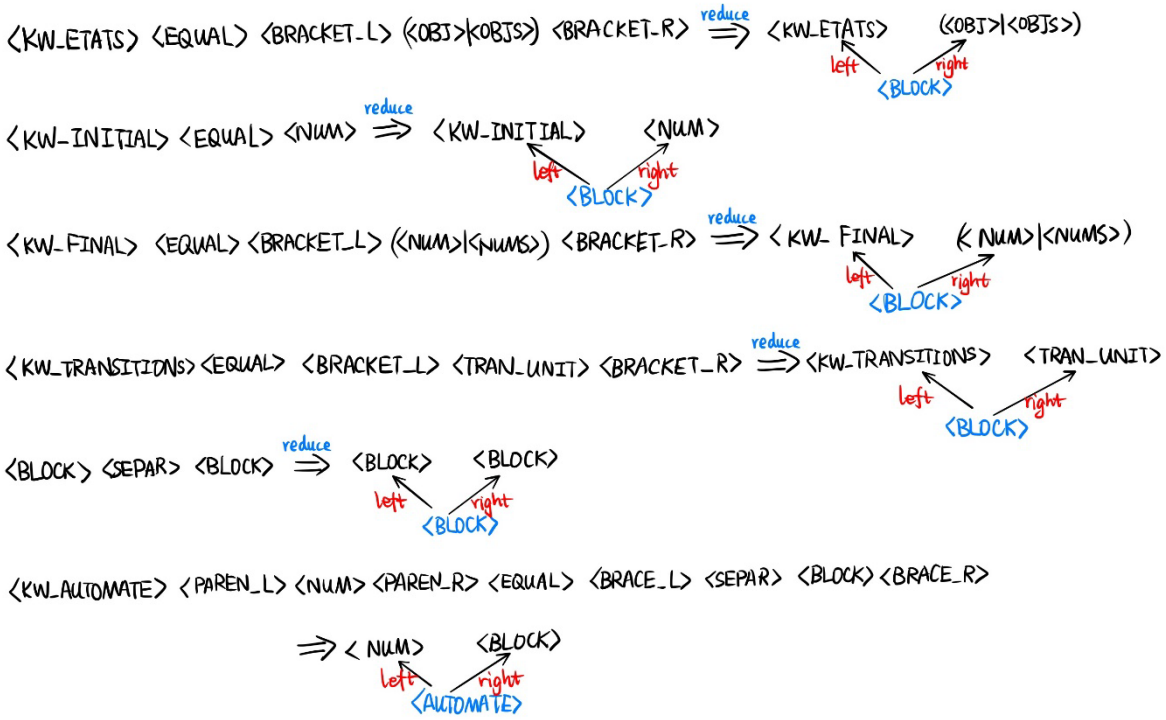












En fin de l'analyse syntaxique, s'il reste un seul lexème qui est <AUTOMATE>, alors ce document est correct syntaxique.

3. Analyse Sémantique

Dans cette partie, on extrait d'abord l'information de l'arbre syntaxique. Puis on le fait analyse sémantique. Il y a totalement 7 vérifications décrit suivant.

- <OBJ> dans <STACK> doit être un seul caractère.
- <NUM> dans <TRAN_UNIT> doit être inférieur au nombre de stack.
- Fonction *info_complete* est désigné pour vérifier est-ce que toutes les quatre parties (etats, initial, final, transitions) sont données.
- Fonction *valid_index* est désigné pour vérifier est-ce que les indices dans final et transitions sont correctes.
- Fonction *unique_edge* est désigné pour vérifier est-ce que on a définir un bord en double.
- Fonction *existe_path* est désigné pour vérifier est-ce qu'il existe des chemins du sommet initial à des sommets finals.
- Fonction *stack_iobal* est désigné pour vérifier est-ce que chaque pile a des opérations de push et à la fois des opérations de pop.
-

4. Compilation et Exécution

Le compilateur prendra un fichier .txt et écrira dans un fichier VM.csv la machine virtuelle et dans un autre fichier symtable.txt la table des symboles. L'exécuteur prendra le fichier VM.txt et sera capable de reconnaître ou pas un mot saisi ai clavier.

5. Construire

Pour obtenir ***compile_automate*** et ***Executeur***, il faut seulement utiliser un command : make

```
(base) zhchu@DESKTOP-3SPP7NP:/mnt/e/CCode/C/PLT$ make
cc -c compile_automate.c
cc -c analyseur_lexical.c
cc -c analyseur_syntaxique.c
cc -c analyseur_semantique.c
cc -c util.c
gcc -Wall -Werror -pedantic -g -fsanitize=address -o compile_automate compile_automate.
o analyseur_lexical.o analyseur_syntaxique.o analyseur_semantique.o util.o
cc -c Executeur.c
gcc -Wall -Werror -pedantic -g -fsanitize=address -o Executeur Executeur.o util.o
rm -rf *.o
(base) zhchu@DESKTOP-3SPP7NP:/mnt/e/CCode/C/PLT$
```

6. compile_automate

On peut utiliser ***compile_automate*** à lire un fichier .txt.

```
(base) zhchu@DESKTOP-3SPP7NP:/mnt/e/CCode/C/PLT$ ./compile_automate Upile.txt
analyse lexicale terminée
analyse syntaxique terminée
extraction des informations de l'arbre de syntaxe terminée
analyse sémantique terminée5/5
VM généré
VM mémorisé
(base) zhchu@DESKTOP-3SPP7NP:/mnt/e/CCode/C/PLT$
```

Bien sûr, il y a beaucoup de modes présentés suivantes.

- -show_tokens : on représente le fichier .txt en lexème. (test mode de l'analyse lexicale)

```
(base) zhchu@DESKTOP-3SPP7NP:/mnt/e/CCode/C/PLT$ ./compile_automate -show_tokens Upile.
txt
analyse lexicale terminée
analyse syntaxique terminée
extraction des informations de l'arbre de syntaxe terminée
analyse sémantique terminée5/5
VM généré
VM mémorisé
_____________________________________________________________________
[LEXEMES]
ROOT KW_AUTOMATE PAREN_L NUM PAREN_R AUTOMATE BRACE_L SEPAR KW_ETATS BLOCK BRACKET_L OB
J OBJS OBJ OBJS OBJ BRACKET_R BLOCK KW_INITIAL BLOCK NUM BLOCK KW_FINAL BLOCK BRACKET_L
 NUM BRACKET_R BLOCK KW_TRANSITIONS BLOCK BRACKET_L TRAN_UNIT NUM NTON NUM EDGE OBJ EAN
DS PAREN_L ARROW STACK OBJ PAREN_R PAREN_R TRAN_UNIT TRAN_UNIT NUM NTON NUM EDGE OBJ EA
NDS PAREN_L OBJ STACK ARROW PAREN_R PAREN_R TRAN_UNIT TRAN_UNIT NUM NTON NUM EDGE OBJ E
ANDS PAREN_L OBJ STACK ARROW PAREN_R PAREN_R TRAN_UNIT TRAN_UNIT NUM NTON NUM EDGE OBJ
EANDS STACK PAREN_R PAREN_R TRAN_UNIT TRAN_UNIT NUM NTON NUM EDGE OBJ PAREN_R TRAN_UNIT
 TRAN_UNIT NUM NTON NUM EDGE OBJ EANDS PAREN_L OBJ STACK ARROW PAREN_R PAREN_R BRACKET_
R BRACE_R
(base) zhchu@DESKTOP-3SPP7NP:/mnt/e/CCode/C/PLT$
```

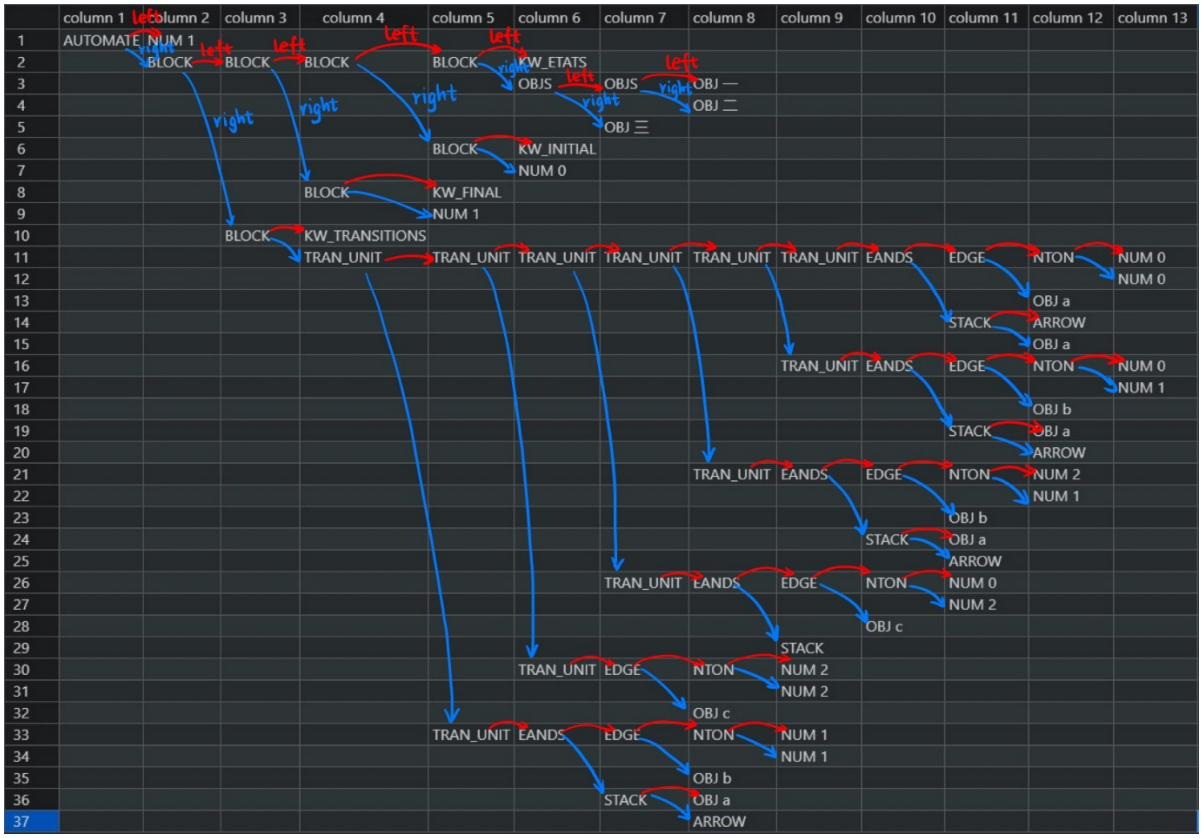
- -show_syntree : on représente l'arbre syntaxique dans le fichier syntree.csv. (test mode de l'analyse syntaxique)

```
(base) zhchu@DESKTOP-3SPP7NP:/mnt/e/CCode/C/PLT$ ./compile_automate -show_syntree Upile
.txt
analyse lexicale terminée
analyse syntaxique terminée
extraction des informations de l'arbre de syntaxe terminée
analyse sémantique terminée5/5
VM généré
VM mémorisé
_____________________________________________________________________
arbre syntaxique disponible sur "syntree.csv"
(base) zhchu@DESKTOP-3SPP7NP:/mnt/e/CCode/C/PLT$
```

Dans le fichier syntree.csv, pour un nœud, son fils nœud à gauche est dans la

colonne suivante en même ligne, son fils nœud à droite est dans la même colonne de son fils nœud à gauche mais dans la ligne dessous. On vous l'explique par un exemple suivant.

La flèche rouge est le fils nœud à gauche et la flèche bleue est le fils nœud à droite. Il faut noter que quelques nœuds (<TRAN_UNIT>) n'ont pas le fils nœud à droite parce que l'on n'a pas définir le fils nœud à droite dans le process de "reduce".



C'est évident que toutes les informations sont mémorisées dans les feuilles. Les nœuds internes ne portent pas des informations données dans le fichier .txt.

- -show_graph : on représente les informations extraites de l'arbre syntaxique. (test mode de l'analyse sémantique) S'il n'y a pas d'erreur sémantique, alors on peut extraire les informations favorablement, sinon, le compile_automate est terminé avec une information d'erreur représente sur l'écran.

```
(base) zhchu@DESKTOP-3SPP7NP:/mnt/e/CCode/C/PLT$ ./compile_automate -show_graph Upile.t
xt
analyse lexicale terminée
analyse syntaxique terminée
extraction des informations de l'arbre de syntaxe terminée
analyse sémantique terminée 5/5
VM généré
VM mémorisé
_____________________________________________________________

[GRAPHE INFO]
nombre des piles: 1
etats: `一` `二` `三`
initial: 0
final: 1
transitions:
de:  0, à:  0, char: a, opérations de pile: (PUSH  a)
de:  0, à:  1, char: b, opérations de pile: (POP   a)
de:  0, à:  2, char: c, opérations de pile: (NONE   )
de:  1, à:  1, char: b, opérations de pile: (POP   a)
de:  2, à:  1, char: b, opérations de pile: (POP   a)
de:  2, à:  2, char: c, opérations de pile: (NONE   )
(base) zhchu@DESKTOP-3SPP7NP:/mnt/e/CCode/C/PLT$
```

- -show_vm : on représente le VM généré. (test mode de compile_automate) On peut utiliser ce mode pour vérifier le VM.

```
(base) zhchu@DESKTOP-3SPP7NP:/mnt/e/CCode/C/PLT$ ./compile_automate -show_vm Upile.txt
analyse lexicale terminée
analyse syntaxique terminée
extraction des informations de l'arbre de syntaxe terminée
analyse sémantique terminée 5/5
VM généré
VM mémorisé
_____________________________________________________________________
[VM INFO]
table des symboles:
nom:     一, adresse:   5
nom:     二, adresse:  18
nom:     三, adresse:  23
Mémoire de VM: 1 5 1 18 3 3 97 5 97 1 98 18 97 -1 99 23 0 0 1 98 18 97 -1 2 98 18 97 -1
 99 23 0 0
(base) zhchu@DESKTOP-3SPP7NP:/mnt/e/CCode/C/PLT$
```

- -show_all : c'est le même de utiliser tous les quatre modes avants à la fois, (-show_tokens -show_syntree -show_graph -show_vm)
  Il faut noter que vous pouvez utiliser plusieurs modes, par exemple :

```
(base) zhchu@DESKTOP-3SPP7NP:/mnt/e/CCode/C/PLT$ ./compile_automate -show_graph -show_v
m Upile.txt
analyse lexicale terminée
analyse syntaxique terminée
extraction des informations de l'arbre de syntaxe terminée
analyse sémantique terminée 5/5
VM généré
VM mémorisé
_____________________________________________________________________
[GRAPHE INFO]
nombre des piles: 1
etats: `一` `二` `三`
initial: 0
final: 1
transitions:
de:  0, à:  0, char: a, opérations de pile: (PUSH  a)
de:  0, à:  1, char: b, opérations de pile: (POP   a)
de:  0, à:  2, char: c, opérations de pile: (NONE   )
de:  1, à:  1, char: b, opérations de pile: (POP   a)
de:  2, à:  1, char: b, opérations de pile: (POP   a)
de:  2, à:  2, char: c, opérations de pile: (NONE   )
_____________________________________________________________________
[VM INFO]
table des symboles:
nom:     一, adresse:   5
nom:     二, adresse:  18
nom:     三, adresse:  23
Mémoire de VM: 1 5 1 18 3 3 97 5 97 1 98 18 97 -1 99 23 0 0 1 98 18 97 -1 2 98 18 97 -1
 99 23 0 0
(base) zhchu@DESKTOP-3SPP7NP:/mnt/e/CCode/C/PLT$
```

7. Executeur

On peut utiliser ***Executeur*** pour executeur le VM mémorisé dans VM.txt et symtabel.csv.

```
(base) zhchu@DESKTOP-3SPP7NP:/mnt/e/CCode/C/PLT$ ./Executeur
VM reloaded
Donner le mot d'entrée: aacccbb
Le mot aacccbb est accepté !
(base) zhchu@DESKTOP-3SPP7NP:/mnt/e/CCode/C/PLT$
```

Si vous voulez le mode debug, vous peuvez utilser le command : -debug, ce qui est représenté dans la figure suivante.

```
(base) zhchu@DESKTOP-3SPP7NP:/mnt/e/CCode/C/PLT$ ./Executeur -debug
VM reloaded
Donner le mot d'entrée: aacccbb
  -> État : ⟶    Pile  1 : Vide
a -> État : ⟶    Pile  1 : a
a -> État : ⟶    Pile  1 : aa
c -> État : ⟶    Pile  1 : aa
c -> État : ☰    Pile  1 : aa
c -> État : ☰    Pile  1 : aa
b -> État : ☰    Pile  1 : a
b -> État : ☰    Pile  1 : Vide
Le mot aacccbb est accepté !
(base) zhchu@DESKTOP-3SPP7NP:/mnt/e/CCode/C/PLT$
```

Il y a aussi des exemples de Dpile.txt.

```
(base) zhchu@DESKTOP-3SPP7NP:/mnt/e/CCode/C/PLT$ ./compile_automate Dpile.txt
analyse lexicale terminée
analyse syntaxique terminée
l'extraction des informations de l'arbre syntaxique est terminée
analyse sémantique terminée 5/5
VM généré
VM mémorisé
(base) zhchu@DESKTOP-3SPP7NP:/mnt/e/CCode/C/PLT$ ./Executeur -debug
VM reloaded
Donner le mot d'entrée: aabbcc
  -> État : 0      Pile  1 : Vide     Pile  2 : Vide
a -> État : 0      Pile  1 : a        Pile  2 : Vide
a -> État : 0      Pile  1 : aa       Pile  2 : Vide
b -> État : 0      Pile  1 : a        Pile  2 : b
b -> État : 1      Pile  1 : Vide     Pile  2 : bb
c -> État : 1      Pile  1 : Vide     Pile  2 : b
c -> État : 2      Pile  1 : Vide     Pile  2 : Vide
Le mot aabbcc est accepté !
(base) zhchu@DESKTOP-3SPP7NP:/mnt/e/CCode/C/PLT$ ./Executeur -debug
VM reloaded
Donner le mot d'entrée: abbc
  -> État : 0      Pile  1 : Vide     Pile  2 : Vide
a -> État : 0      Pile  1 : a        Pile  2 : Vide
b -> État : 0      Pile  1 : Vide     Pile  2 : b
b -> Erreur : Pile 1 : Vide vs l'opération ('a', →)
Le mot abbc est refusé !
(base) zhchu@DESKTOP-3SPP7NP:/mnt/e/CCode/C/PLT$ ./Executeur -debug
VM reloaded
Donner le mot d'entrée: aabbc
  -> État : 0      Pile  1 : Vide     Pile  2 : Vide
a -> État : 0      Pile  1 : a        Pile  2 : Vide
a -> État : 0      Pile  1 : aa       Pile  2 : Vide
b -> État : 0      Pile  1 : a        Pile  2 : b
b -> État : 1      Pile  1 : Vide     Pile  2 : bb
c -> État : 1      Pile  1 : Vide     Pile  2 : b
Erreur : Pile 1 non vide
Le mot "aabbc" est refusé !
(base) zhchu@DESKTOP-3SPP7NP:/mnt/e/CCode/C/PLT$
```

8. Exemples des Erreurs

Il y a certain fichiers test.txt pour tester et chaque fichier a une erreur différente. Vous pouvez apprendre le règle de fichier donnée plus efficacement.

- test1.txt

```
(base) zhchu@DESKTOP-3SPP7NP:/mnt/e/CCode/C/PLT$ ./compile_automate test1.txt
erreur lexicale: mot inconnu "transition" in line 5
(base) zhchu@DESKTOP-3SPP7NP:/mnt/e/CCode/C/PLT$
```

- test2.txt

```
(base) zhchu@DESKTOP-3SPP7NP:/mnt/e/CCode/C/PLT$ ./compile_automate test2.txt
erreur lexicale: mot clés initial en double
(base) zhchu@DESKTOP-3SPP7NP:/mnt/e/CCode/C/PLT$
```

- test3.txt

```
(base) zhchu@DESKTOP-3SPP7NP:/mnt/e/CCode/C/PLT$ ./compile_automate test3.txt
analyse lexicale terminée
_____________________________________________________________
erreur syntaxique
[LEXEMES RESTANTS] KW_AUTOMATE PAREN_L NUM PAREN_R EQUAL BRACE_L SEPAR BLOCK SEPAR KW_T
RANSITIONS EQUAL BRACKET_L PAREN_L EANDS COMMA PAREN_L EANDS COMMA PAREN_L EANDS BRACKE
T_R BRACE_R
(base) zhchu@DESKTOP-3SPP7NP:/mnt/e/CCode/C/PLT$
```

- test4.txt

```
(base) zhchu@DESKTOP-3SPP7NP:/mnt/e/CCode/C/PLT$ ./compile_automate test4.txt
analyse lexicale terminée
analyse syntaxique terminée
_____________________________________________________________
erreur sémantique: n'acceptez que char plutôt qu'une chaîne dans la transition de 0 à 2
(base) zhchu@DESKTOP-3SPP7NP:/mnt/e/CCode/C/PLT$
```

- test5.txt

```
(base) zhchu@DESKTOP-3SPP7NP:/mnt/e/CCode/C/PLT$ ./compile_automate test5.txt
analyse lexicale terminée
analyse syntaxique terminée
_____________________________________________________________
erreur sémantique: le numéro de la pile dans la transition de 0 à 2 ne devrait pas excé
der 1
(base) zhchu@DESKTOP-3SPP7NP:/mnt/e/CCode/C/PLT$
```

- test6.txt

```
(base) zhchu@DESKTOP-3SPP7NP:/mnt/e/CCode/C/PLT$ ./compile_automate test6.txt
analyse lexicale terminée
analyse syntaxique terminée
l'extraction des informations de l'arbre syntaxique est terminée
_____________________________________________________________
erreur sémantique: informations finales manquantes
(base) zhchu@DESKTOP-3SPP7NP:/mnt/e/CCode/C/PLT$
```

- test7.txt

```
(base) zhchu@DESKTOP-3SPP7NP:/mnt/e/CCode/C/PLT$ ./compile_automate test7.txt
analyse lexicale terminée
analyse syntaxique terminée
l'extraction des informations de l'arbre syntaxique est terminée
_____________________________________________________________
erreur sémantique: index invalide 3 en final = [3, ... ]
(base) zhchu@DESKTOP-3SPP7NP:/mnt/e/CCode/C/PLT$
```

- test8.txt

```
(base) zhchu@DESKTOP-3SPP7NP:/mnt/e/CCode/C/PLT$ ./compile_automate test8.txt
analyse lexicale terminée
analyse syntaxique terminée
l'extraction des informations de l'arbre syntaxique est terminée
_____________________________________________________________
erreur sémantique: redéfinition de transition (0 → 1,`b`(`a`,→)) vs (0 → 1,`a`(`a`,→))
(base) zhchu@DESKTOP-3SPP7NP:/mnt/e/CCode/C/PLT$
```

- test9.txt

```
(base) zhchu@DESKTOP-3SPP7NP:/mnt/e/CCode/C/PLT$ ./compile_automate test9.txt
analyse lexicale terminée
analyse syntaxique terminée
l'extraction des informations de l'arbre syntaxique est terminée
_____________________________________________________________
erreur sémantique: aucun chemin disponible de 0 à 2
(base) zhchu@DESKTOP-3SPP7NP:/mnt/e/CCode/C/PLT$
```

- test10.txt

```
(base) zhchu@DESKTOP-3SPP7NP:/mnt/e/CCode/C/PLT$ ./compile_automate test10.txt
analyse lexicale terminée
analyse syntaxique terminée
l'extraction des informations de l'arbre syntaxique est terminée
_____________________________________________________________
erreur sémantique: pile 0 nécessite une opération pop
(base) zhchu@DESKTOP-3SPP7NP:/mnt/e/CCode/C/PLT$
```

9. Problèmes et Solutions

J’ai rencontré des problèmes mais j’ai tous les résolu. Par exemple, les fonctions *fwrite* et *fprintf* ont le même usage en général mais *fwrite* peut causer un petit problème sur des caractères chinoises dans un fichier .csv. Les autres problèmes sont petits et facile à résoudre, mais il faut un peut plus de temps. En fait, d’abord je pense qu’il faut seulement 3 jours pour finir ce projet. Mais le fait, c’est j’ai dépensé presque une semaine parce que j’ai dépensé beaucoup de temps à connaître les fonctions dans le langage C, par exemple, *fopen*, *fprintf*, *strncmp*, etc. En fait, les choses les plus valables j’ai appris, c’est CFG(BNF), FR(0) et comment utiliser Lex & YACC. Ce sont des outils populaires pour développer le compile et ils généreront deux fichiers dans le langage C, mais je ne peux pas les comprendre parce que dans les fichiers il y a beaucoup de 0/1 matrices. Ce projet vraiment bénéfique pour ma recherche dans NLP :).