

Projet de automate

Nom Francais: Abigail

Nom Chinoise: 魯君一

No.517261910033

Content

Projet de automate

Content

1.Introduction

2.Analyse lexicale

2.1.ExPLICATION générale

Definition de variables

Tableau des codes d'identification

Idee

2.2.Fonctions

2.3.Difficulté & Solution

3.Analyse Syntaxique

3.1.ExPLICATION générale

3.2.Fonctions

3.3.Difficulté & Solution

3.4.Limitation

3.5. Resultat

4.Analyse Semantique

5.Compilation

5.1.ExPLICATION générale

5.2.Limitation

5.3.Resultat

6.Execution

6.1.ExPLICATION générale

6.2.Difficulté & Solution

6.3.Limitation

README

Structure de dossier

Comment exécuter le programme

Analyse Lexicale

Analyse Syntaxique

Analyse Semantique

Compilation

Execution

1. Introduction

Les commentaires dans le code sont en chinois pour faciliter la programmation, mais les variables et fonctions les plus importantes sont expliquées en français dans le rapport.

Les noms de variables et les rapports d'erreurs dans le code, et les journaux sont en anglais, ceci parce que le français contient des caractères spéciaux qui ne sont pas pratiques à utiliser.

Le rapport et le README sont écrits en français.

Avant de lancer l'ensemble du projet, j'ai défini dans `analyse_lexical.h` un certain nombre de variables globales.

```
char targetfile[64];//Stockage du nom du fichier cible( par exemple: Zpile.txt)

//Vous pouvez voir leur fonction à partir des noms des variables suivantes
int numStatus=0;
int numStatusFinal=0;
int numTransitions=0;
int numStacks=0;

STATUS statusList[10];//Stocke les informations sur l'état de l'automate (nom, index,
adresse dans VM.txt)
char triggerList[10];//Stockage des caractères dont l'état de stimulus a changé
int index_status_initial=0;
int index_status_final[10];
TRANSITION transList[10];//Stocke toutes les changement de l'états définies dans le
fichier cible
```

Ces variables sont très importantes et leur manipulation est effectuée pendant toutes les phases du projet.

Je les ai définies globalement car cela évite de passer des variables dans des phases adjacentes, par exemple, je peux introduire et utiliser ces variables en écrivant `#include "analyse_lexical.h"` au début de `analyse_syntax.h`

Puisque l'analyse lexicale doit être faite avant l'analyse syntaxique, et que l'analyse lexicale et l'analyse syntaxique doivent être faites avant l'analyse sémantique... J'ai choisi d'envelopper le code qui met en œuvre ces analyses dans des fonctions, écrites dans un fichier d'en-tête qui peut être facilement appelé à d'autres étapes.

Le débogage des fonctions se fera au moyen d'un programme de test dans le dossier `src`.

2. Analyse lexicale

2.1. Explication générale

Definition de variables

Les variables globales suivantes ne seront utilisées que dans la section d'analyse lexicale

```
char str[1024];//Stocke la chaîne de caractères lue à partir du fichier cible (pas de commentaires, pas de coupures d'espace).
int strlength=0;

char ch;//Caractère actuel
int ch_index_in_str=0;//Index du caractère actuel dans str
int syn;// Code d'identification du caractère courant

char token[20];//Mot réservé actuel de str
int ch_index_in_token=0;//Pointeur actuel du Mot réservé

char chinese_token[4];//Caractères chinois courants et caractères spéciaux récupérés dans str
int ch_index_in_chinesetoken=0;//Pointeur actuel du chinese_token
```

- J'ai fait quelques erreurs avec le nommage de `chinese_token`, après de nombreux tests, j'ai réalisé que les caractères spéciaux → doivent être sauvegardés avec 3 char et un '\0', comme les caractères chinois, mais je n'ai pas trouvé de meilleur nom, donc j'ai opté pour ce nommage, donc vous pouvez voir que il y a un '→' dans la liste:

```
static char * chinese[12]={"零","一","二","三","四","五","六","七","八","九","十","→"};
```

- J'aurais aimé implémenter la reconnaissance des états nommés avec des caractères français spéciaux, mais j'ai trouvé cela très délicat et j'ai été contraint par des contraintes de temps de ne pas implémenter cette fonctionnalité.
- Si un nombre est utilisé comme nom d'un état et d'un trigger, il ne sera pas reconnu comme un id, ce problème sera corrigé dans `analyse_syntax`.

Tableau des codes d'identification

Caractère	Code d'identification	Caractère	Code d'identification
Automate	1	[23
etats	2]	24
initial	3	{	25
final	4	}	26
transitions	5	,	27
id	10	'	28
digit	11	"	29
(21		
)	22	=	31
		→	32

Ideé

D'abord, je lis le contenu du fichier dans str et j'élimine le contenu du commentaire en identifiant le début et la fin du commentaire et en réécrivant le bit de drapeau.

Je vérifie ensuite les caractères **un par un**, d'abord pour voir si la forme `int` du caractère est **négative** (car un nombre négatif signifie que le caractère et les deux caractères suivants représentent un caractère spécial) ; ensuite pour déterminer si le caractère représente une lettre, un chiffre ou un symbole.

Si le caractère représente une lettre, vérifiez qu'il ne forme pas, avec les caractères qui le suivent, un mot réservé.

- Si c'est le cas, on lui attribue l'identifiant correspondant.
- Dans le cas contraire, le caractère est considéré comme un id (l'id est le nom que l'utilisateur donne à l'état ou au trigger) et se voit attribuer l'identifiant correspondant.

Si un caractère et ses deux suivants représentent des caractères spéciaux, il y a deux possibilités.

- Le caractère spécial est un caractère chinois et doit donc recevoir l'identifiant de id: 10 , car les exigences du projet indiquent que les caractères chinois ne seront utilisés que pour la dénomination des états
- Le caractère est →, on lui donne donc l'identifiant correspondant.

Je fais cela pour faciliter l'étape suivante de l'analyse syntaxique, et aussi pour lancer une erreur lexicale si je tombe sur un caractère qui n'a pas été défini.

2.2.Fonctions

`isletter()`:Cette fonction est utilisée pour identifier si un caractère est une lettre ou non.

`isDigit()`:Cette fonction est utilisée pour identifier si un caractère est un nombre ou non.

`isBoundary()`: Cette fonction est utilisée pour identifier si un caractère est un symbole ou non.

`readin_without_comment()`: Cette fonction est utilisée pour supprimer les commentaires et les espaces du fichier cible et stocker les informations utiles dans str.

`analyse_lexical()`: C'est la principale fonction utilisée pour l'analyse lexicale.

2.3. Difficulté & Solution

J'ai des difficultés à analyser les caractères spéciaux et les mots réservés.

Premièrement, les caractères spéciaux et les mots réservés sont représentés par des combinaisons de plusieurs caractères, ce qui m'a obligé à trouver une nouvelle logique de jugement.

Deuxièmement, les caractères spéciaux sont représentés d'une manière très spéciale. En C, les caractères spéciaux semblent être conservés sous forme de `string`, puisque les caractères spéciaux sont représentés par trois caractères et un '`\0`'.

Enfin, j'ai choisi de traiter les caractères spéciaux et les mots réservés comme des `string`, sauvegardées avec `static char * table[]`, et j'ai utilisé `strcmp` pour les comparer avec ce qui était lu.

3. Analyse Syntaxique

3.1. Explication générale

En analyse syntaxique, mon idée principale est la suivante:

- Il faut d'abord analyser un caractère (ou un caractère spécial/mot réservé) avec la fonction `analyse_lexical` pour obtenir l'identifiant, puis analyser si cet identifiant apparaît à la bonne position, et s'il est correct, stocker l'information sur l'automate porté par ce caractère dans la variable globale correspondante, ou lancer un avertissement d'erreur de syntaxe s'il est incorrect.
- Continuez ensuite à analyser le caractère suivant avec la fonction `analyse_lexical` et répétez ce qui précède jusqu'à ce que la fin de `str` soit lue.

Malgré la simplicité de l'idée d'implémentation, la quantité de travail impliquée est énorme et on peut écrire un code extrêmement complexe et difficile à comprendre si l'on ne fait pas attention. Pour rendre la structure du code plus claire, j'ai encapsulé l'analyse des chaînes locales dans des fonctions plus petites et les ai ensuite appelées dans les fonctions d'analyse pour les chaînes plus grandes... jusqu'à la fonction principale `analyse_synatax()`.

3.2. Fonctions

`errorinfo()`: Cette fonction est conçue pour lancer un avertissement d'erreur de syntaxe

`analyse_syntax()`: Cette fonction est la principale fonction d'analyse syntaxique.

- `etats_statement()`: Cette fonction est conçue pour analyser des instructions comme `etats = ["一", "二", "三"]`

- `st()`: Cette fonction est conçue pour analyser des instructions comme "—"
- `initialstatement()`: Cette fonction est conçue pour analyser des instructions comme `initial = 0`
- `final_statement()`: Cette fonction est conçue pour analyser des instructions comme `final =[1]`
- `transitions_statement()`: Cette fonction est conçue pour analyser des instructions comme:

```
transitions=[(0 → 0, `a`, (→, `a`)),
(0 → 1, `b`, (`a`, →), (→, `b`)),
(1 → 1, `b`, (`a`, →), (→, `b`)),
(1 → 2, `c`, (), (`b`, → )),
(2 → 2, `c`, (), (`b`, →))]
```

- `trans()`: Cette fonction est conçue pour analyser des instructions comme `(0 → 1, `b`, (`a`,
→), (→, `b`))`
 - `srcdst()` Cette fonction est conçue pour analyser des instructions comme `0 → 1`
 - `trig()` Cette fonction est conçue pour analyser des instructions comme ``b``
 - `st_action2()` et `st_action1()`: Cette fonction est conçue pour analyser des instructions
comme `(`a`, →)` et `et()`

`print_ syn_result()` :Cette fonction est conçue pour imprimer les résultats de l'analyse syntaxique.

3.3.Difficulté & Solution

- J'ai été inspiré par cet [article](#) en termes de réflexion globale
- Après de nombreux bugs, J'ai trouvé un moyen très simple de convertir un char en int ASCII
correspondant:

```
index_status_initial=ch-'0';//Ici index_status_initial est de type int, ch est de
type char
```

3.4.Limitation

1. Puisqu'il y a tellement d'endroits pour lancer des avertissements d'erreur de syntaxe, je n'ai pas eu l'énergie d'écrire la cause de chacun d'entre eux, donc j'ai simplement défini une fonction de lancement d'avertissement, `errorinfo()` , qui est appelée chaque fois qu'un avertissement est nécessaire.
2. Mon code comporte trop de couches de structures de jugement if-else imbriquées, ce qui entraîne une mauvaise lisibilité. J'ai réfléchi à la manière de résoudre ce problème, et en faisant des recherches sur le web, j'ai découvert que ce problème est courant en `java` et qu'il peut être résolu en écrivant [garde](#) , mais j'ai été contraint par le manque d'énergie de mener à bien la tâche de refactoring du code.

Plus tard, je prévois de lire quelques livres pour m'aider à écrire un code plus lisible, moins répétitif et plus standard.

3.5. Resultat

Lorsque la pile est vide, printf imprime le \0 à l'intérieur, un symbole non reconnu par le système d'encodage UTF-8, et donc un rappel que `testsyn_result.log` n'est pas reconnu

`zpile.txt` :

```
Status: 1
Status: 2
Status: 3
Status: Init
numStacks: 0
index_status_initial: 3
index_status_final: 0
index_status_final: 1
index_status_final: 2
from:3, to:0, trigger:0,
stack1:•, action1:0, stack2:•, action2:0

from:3, to:1, trigger:1,
stack1:•, action1:0, stack2:•, action2:0

from:3, to:2, trigger:2,
stack1:•, action1:0, stack2:•, action2:0

from:0, to:1, trigger:1,
stack1:•, action1:0, stack2:•, action2:0

from:0, to:2, trigger:2,
stack1:•, action1:0, stack2:•, action2:0

from:1, to:0, trigger:0,
stack1:•, action1:0, stack2:•, action2:0

from:1, to:2, trigger:2,
stack1:•, action1:0, stack2:•, action2:0

from:2, to:0, trigger:0,
stack1:•, action1:0, stack2:•, action2:0

from:2, to:1, trigger:1,
stack1:•, action1:0, stack2:•, action2:0
```

`upile.txt` :

```
Status: —
Status: —
Status: —
numStacks: 1
index_status_initial: 0
index_status_final: 1
from:0, to:0, trigger:a,
stack1:a, action1:1, stack2:•, action2:0

from:0, to:1, trigger:b,
stack1:a, action1:-1, stack2:•, action2:0

from:2, to:1, trigger:b,
stack1:a, action1:-1, stack2:•, action2:0

from:0, to:2, trigger:c,
stack1:•, action1:0, stack2:•, action2:0

from:2, to:2, trigger:c,
stack1:•, action1:0, stack2:•, action2:0

from:1, to:1, trigger:b,
stack1:a, action1:-1, stack2:•, action2:0
```

Dpile.txt:

```
Status: A
Status: B
Status: C
numStacks: 2
index_status_initial: 0
index_status_final: 2
from:0, to:0, trigger:a,
stack1:a, action1:1, stack2:•, action2:0

from:0, to:1, trigger:b,
stack1:a, action1:-1, stack2:b, action2:1

from:1, to:1, trigger:b,
stack1:a, action1:-1, stack2:b, action2:1

from:1, to:2, trigger:c,
stack1:•, action1:0, stack2:b, action2:-1

from:2, to:2, trigger:c,
stack1:•, action1:0, stack2:b, action2:-1
```

4.Analyse Semantique

Cette partie est très simple, il suffit de vérifier que les valeurs des variables globales obtenues après l'analyse syntaxique ne sont pas en conflit les unes avec les autres

5.Compilation

5.1.ExPLICATION générale

Dans cette section, je dois sortir les informations que j'ai obtenues précédemment dans un certain ordre, je dois donc regrouper les transitions en fonction de leur état de départ, ce qui nécessite que je puisse obtenir d'autres informations sur une transition en fonction de son index, ce qui serait très simple si j'utilisais C++ : je pourrais définir une classe de transitions qui contient de telles fonctions de requête. Mais en C, il n'y a pas de concept de classe, donc j'ai défini quelques fonctions pour le faire :

```
search_address_given_index (int)
search_index_given_address(int)
search_name_given_index(int)
search_name_given_address(int)
```

Puisque ce qui est écrit dans VM.txt et TS.txt doit être l'adresse de l'état dans VM.txt, et non l'index qu'il a obtenu lorsqu'il a été défini, j'ai besoin de savoir à quelle position un état doit être écrit dans VM.txt, ce qui est compliqué, donc j'écris VM.txt la première fois en écrivant l'index de l'état, puis en stockant sa position actuelle(avec fonction `transition_info_from (int)`), puis j'écris VM.txt à nouveau, cette fois avec l'état à son adresse(avec fonction `update_transition_info_from(int)`).

5.2.Limitation

Bien qu'initialement le `trigger`, `target_trigger_stack` puisse être défini sur un caractère spécial ou plusieurs caractères, seul un caractère commun unique tel que a,b,c,1,2,3 est pris en compte dans la sortie de la VM.txt.

5.3.Resultat

```
zpiel.txt:
```

```

0 3 3
6 11 16 // l'adress dans VM.txt de l'etats 1,2,3 est 6 11 16
2 49 11 50 16 // A partir de l'etats 1
2 48 6 50 16 // A partir de l'etats 2
2 48 6 49 11 // A partir de l'etats 3
3 48 6 49 11 50 16 // A partir de l'etats Init

```

Upile.txt :

```

1 0 1
17 // l'adress dans VM.txt de l'etats 2 est 17
3 97 4 97 1 98 17 97 -1 99 22 0 0 // A partir de l'etats 2
1 98 17 97 -1 // A partir de l'etats 2
2 98 17 97 -1 99 22 0 0 // A partir de l'etats 3

```

Dpile.txt :

```

2 0 1
30 // l'adress dans VM.txt de l'etats C est 30
2 97 4 97 1 0 0 98 17 97 -1 98 1 // A partir de l'etats A
2 98 17 97 -1 98 1 99 30 0 0 98 -1 // A partir de l'etats B
1 99 30 0 0 98 -1 // A partir de l'etats C

```

6. Execution

6.1. Explication générale

Dans cette section, je dois faire en sorte que l'automate exécute une chaîne d'entrée donné par l'utilisateur, consistant par trigger, et voir si l'automate peut atteindre l'état d'acceptation avec cette chaîne. Dans cette section, j'utilise la structure myStack définie précédemment:

```

myStack * stack1;
myStack * stack2;

```

J'analyse chaque caractère un par un, et si le caractère peut déplacer l'état actuel vers l'état suivant, je saute à cet état dans `vminfo[]`, puis j'analyse le caractère suivant et je répète ce qui précède, en envoyant un message d'erreur s'il n'y a pas d'état vers lequel sauter, ou si la pile est déjà vide mais que la pile doit être vidée ensuite, ou si on ne peut pas aller à l'état d'acceptation en fonction de la chaîne lue.

J'ai écrit deux fonctions, une qui imprime des informations de debug `execute()` et une qui ne le fait pas `execute_without_debug()`. Dans `main.c`, je contrôle quelle fonction est utilisée en définissant la macro `debug` :

```
#ifdef debug
    execute();
#else
    execute_without_debug();
#endif
```

j'ai écrit les règles dans le `Makefile`:

```
ifeq ($(debug),yes)
DEBUG := -D debug
endif
```

Il suffit d'ajouter : `debug:=yes` à la compilation pour activer le mode de debug, lisez la section README ci-dessous pour plus d'informations.

- `flag_exist` est utilisé pour vérifier si l'état suivant peut être atteint à partir de l'état actuel en fonction du trigger d'entrée.

6.2. Difficulté & Solution

Après avoir écrit le programme et l'avoir compilé, gcc lance une erreur.

```
Segmentation fault (core dumped)
```

Cette erreur est généralement causée par un pointeur pointant vers une mauvaise zone de mémoire, ce qui est une erreur fatale et difficile à dépanner, et comme je ne pouvais pas configurer le debugger de VScode, je ne pouvais la dépanner qu'en insérant des `printf` dans le code, et après une après-midi de debug, j'ai finalement trouvé que la cause de l'erreur était que je n'avais pas alloué de mémoire pour `stack1` et `stack2` !

Ainsi, après avoir ajouté ces deux lignes de code, le programme fonctionne bien.

```
stack1=(struct myStack*)malloc(sizeof(struct myStack));
stack2=(struct myStack*)malloc(sizeof(struct myStack));
```

6.3. Limitation

En raison de contraintes de temps, je n'ai pas pu séparer le processus de compilation du processus d'exécution, ce que je n'aurais pas dû pouvoir faire. Théoriquement, VM.txt et TS.txt sont équivalents aux fichiers cibles binaires compilés par le programme, et j'aurais pu exécuter l'automate sur la base des seules informations contenues dans ces deux fichiers.

README

- J'ai installé une machine virtuelle `Ubunt20.04LTS` dans `Vmware 15.5Pro` sur un ordinateur `Win10`, et j'ai utilisé `Remote-SSH` de `Visual Studio Code` pour accéder à cette machine virtuelle à distance depuis un autre `Macbook` pour des travaux de développement.
- J'ai compilé avec `GCC` et debug avec `GDB`.
- Environment:

```
abigail@ubuntu:~/Desktop/automate$ cat /proc/version
Linux version 5.11.0-43-generic (buildd@lcy02-amd64-036) (gcc (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0
, GNU ld (GNU Binutils for Ubuntu) 2.34) #47~20.04.2-Ubuntu SMP Mon Dec 13 11:06:56 UTC 2021
abigail@ubuntu:~/Desktop/automate$ gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-linux-gnu/9/lto-wrapper
OFFLOAD_TARGET_NAMES=nvptx-none:hsa
OFFLOAD_TARGET_DEFAULT=1
Target: x86_64-linux-gnu
Configured with: ../src/configure -v --with-pkgversion='Ubuntu 9.3.0-17ubuntu1~20.04' --with-bugurl=file:///usr/share/doc/gcc-9/README.Bugs --enable-languages=c,ada,c++,go,brig,d,fortran,objc,obj-c++,gm2 --prefix=/usr --with-gcc-major-version-only --program-suffix=-9 --program-prefix=x86_64-li
nux-gnu- --enable-shared --enable-linker-build-id --libexecdir=/usr/lib --without-included-gettext
--enable-threads=posix --libdir=/usr/lib --enable-nls --enable-clocale=gnu --enable-libstdcxx-deb
ug --enable-libstdcxx-time=yes --with-default-libstdcxx-abi=new --enable-gnu-unique-object --disab
le-vtable-verify --enable-plugin --enable-default-pie --with-system-zlib --with-target-system-zlib
=auto --enable-objc-gc=auto --enable-multiarch --disable-werror --with-arch-32=i686 --with-abi=m64
--with-multilib-list=m32,m64,mx32 --enable-multilib --with-tune=generic --enable-offload-targets=
nvptx-none=/build/gcc-9-HskZEa/gcc-9-9.3.0/debian/tmp-nvptx/usr,hsa --without-cuda-driver --enable
-checking=release --build=x86_64-linux-gnu --host=x86_64-linux-gnu --target=x86_64-linux-gnu
Thread model: posix
gcc version 9.3.0 (Ubuntu 9.3.0-17ubuntu1~20.04)
abigail@ubuntu:~/Desktop/automate$ gdb -v
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
abigail@ubuntu:~/Desktop/automate$
```

Structure de dossier

```
abigail@ubuntu:~/Desktop/automate$ tree
.
├── bin
│   ├── compile
│   ├── Dpile.txt
│   ├── main
│   ├── testlex
│   ├── testsem
│   └── testsyn
│       ├── Upile.txt
│       └── Zpile.txt
├── include
│   ├── analyse_lexical.h
│   ├── analyse_semantic.h
│   ├── analyse_syntax.h
│   ├── compile_automate.h
│   ├── executeur.h
│   ├── mystack.h
│   └── transition.h
└── log
    ├── del.sh
    ├── testsem.log
    ├── testsyn.log
    ├── testsyn_result.log
    ├── TS.txt
    └── VM.txt
.
├── Makefile
└── src
    ├── compile.c
    ├── main.c
    ├── testlex.c
    ├── testsem.c
    └── testsyn.c

4 directories, 27 files
```

Le dossier `bin` contient les **fichiers exécutables compilés** et les fichiers `Dpile.txt`, `Upile.txt` et `Zpile.txt`.

Le dossier `include` contient les **fichiers d'en-tête** utilisés dans le projet, dont

- `mystack.h` définit une **structure de pile** et des fonctions associées à appeler lorsque nous rencontrons un automate de pile
- `transition.h` définit une **structure de transition** et une **structure d'état** pour stocker les informations lues dans `Dpile.txt`, `Upile.txt` et `Zpile.txt`.
- Les autres fichiers d'en-tête définissent les fonctions nécessaires à chaque processus de compilation et execution.

Le dossier `log` contient les **journaux** générés lors de l'exécution de l'exécutable, dont le nom vous indiquera quelle phase a généré le journal

Le dossier `src` contient des programmes C écrits pour **tester l'effet des fonctions** à chaque étape de la compilation et execution.

Comment exécuter le programme

Exécutez `make all` dans le dossier `automate` pour compiler tous les fichiers exécutables .

Exécutez `make all debug:=yes` dans le dossier `automate` pour compiler tous les fichiers exécutables et dans le mode debug.

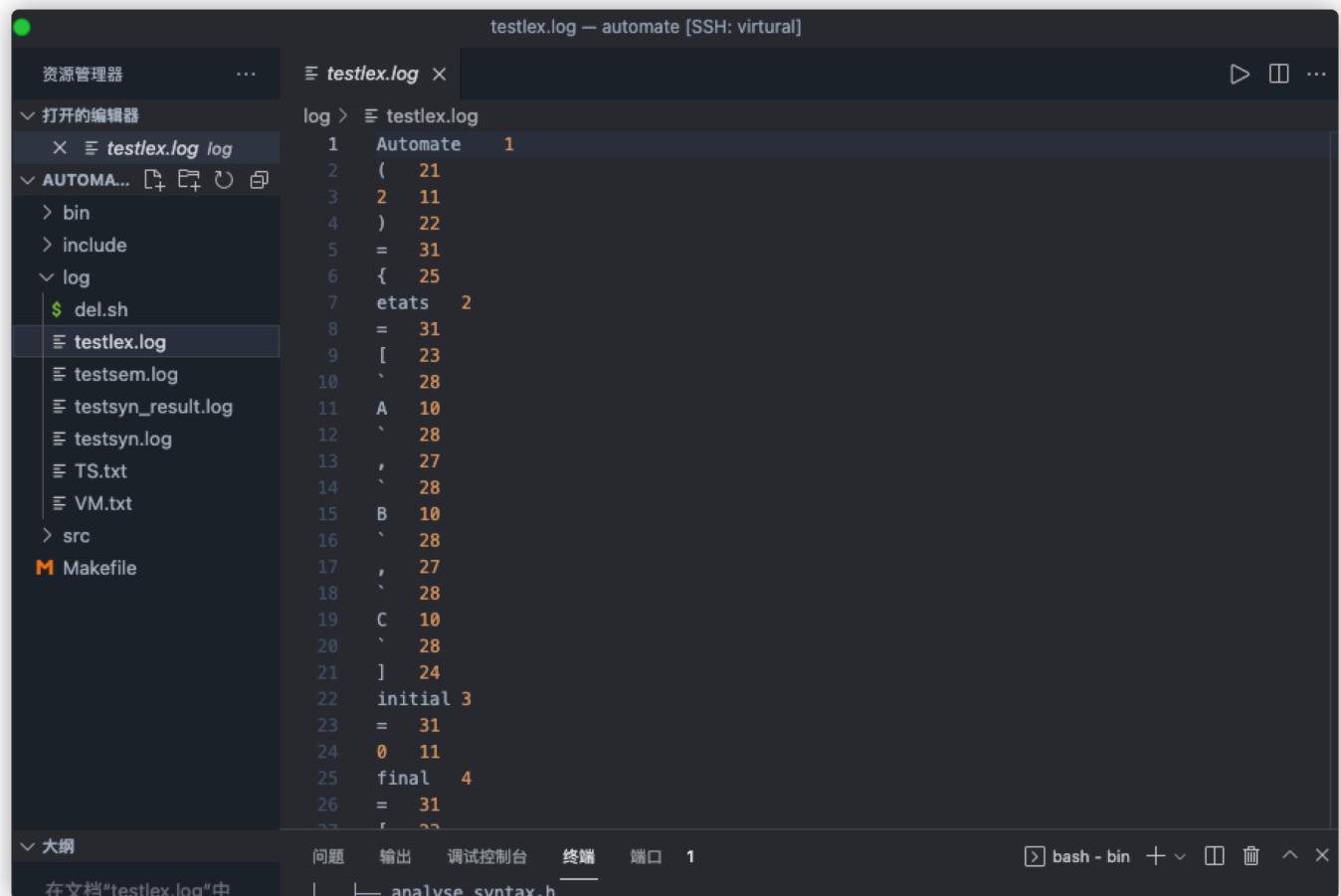
Exécuter `make clean` pour effacer tous les fichiers exécutables.

Analyse Lexicale

1. Exécutez `make testlex` dans le dossier `automate`.
2. Ouvrez le dossier `bin` et exécutez `./testlex`
3. Entrez le nom du fichier que vous souhaitez faire analyser, ici , je utilise `Dpile.txt`
4. Ouvrez le dossier `log` pour voir les journaux générés.

```
abigail@ubuntu:~/Desktop/automate$ make testlex
gcc -Wall -o bin/testlex src/testlex.c -I./include
abigail@ubuntu:~/Desktop/automate$ cd bin
abigail@ubuntu:~/Desktop/automate/bin$ ./testlex
Enter the name of the file that you want to compile:Dpile.txt
Analyse lexical finish!
Read testlex.log for more information
abigail@ubuntu:~/Desktop/automate/bin$
```

Le fichier `testlex.log` généré à partir de l'analyse lexicale de `Dpile.txt`.



The screenshot shows a terminal window titled "testlex.log — automate [SSH: virtual]". The window displays the contents of the "testlex.log" file, which is a log of a lexical analysis process. The log includes line numbers, tokens, and their corresponding states. The terminal interface includes a sidebar for navigating through files and a bottom status bar showing the current tab and other terminal controls.

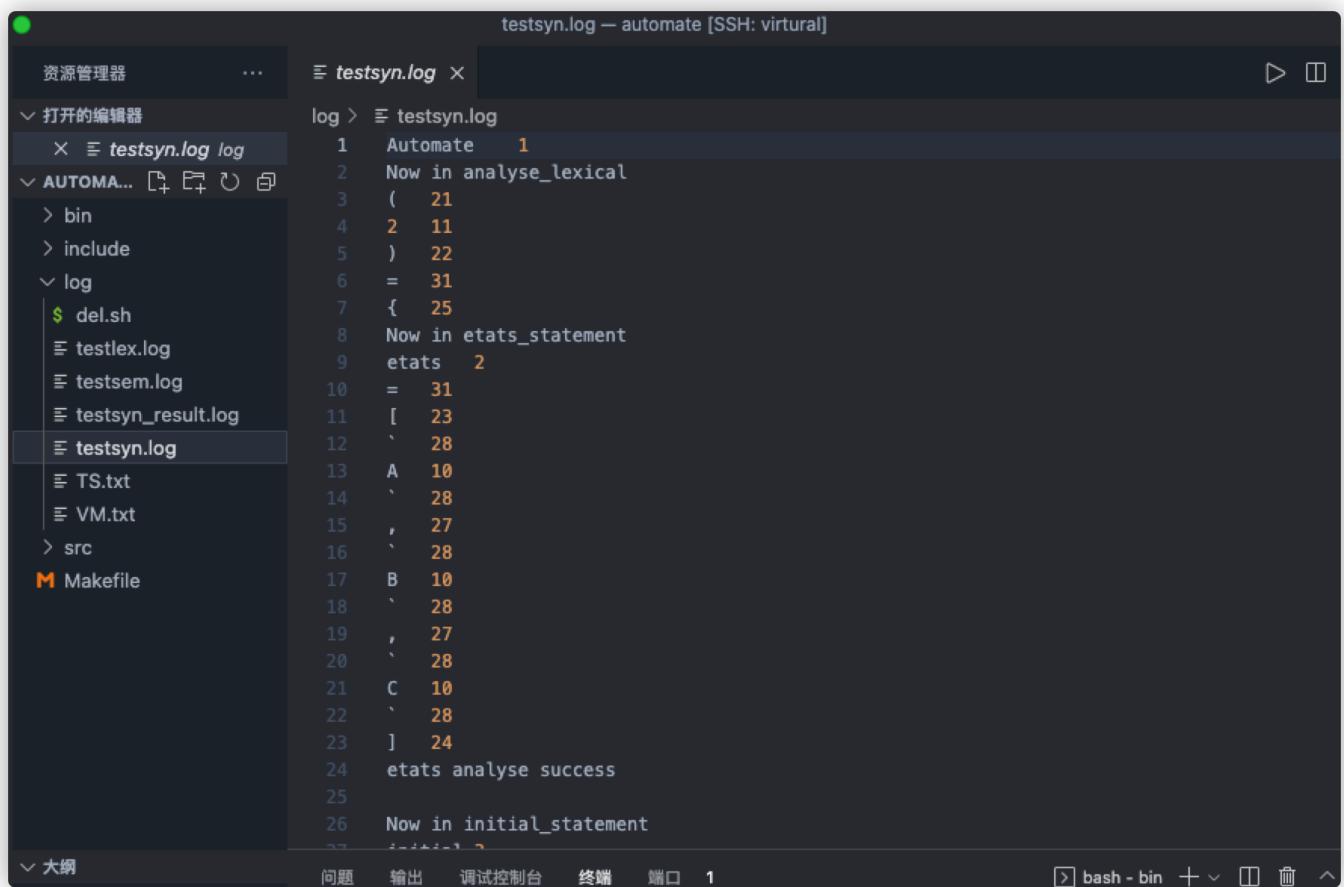
```
1 Automate 1
2 ( 21
3 2 11
4 ) 22
5 = 31
6 { 25
7 etats 2
8 = 31
9 [ 23
10 ` 28
11 A 10
12 ` 28
13 , 27
14 ` 28
15 B 10
16 ` 28
17 , 27
18 ` 28
19 C 10
20 ` 28
21 ] 24
22 initial 3
23 = 31
24 0 11
25 final 4
26 = 31
27 ` 27
```

Analyse Syntaxique

1. Exécutez `make testsyn` dans le dossier `automate`.
2. Ouvrez le dossier `bin` et exécutez `./testsyn`
3. Entrez le nom du fichier que vous souhaitez faire analyser, ici , je utilise `Dpile.txt`
4. Ouvrez le dossier `log` pour voir les journaux générés.

```
abigail@ubuntu:~/Desktop/automate$ make testsyn
gcc -Wall -o bin/testsyn src/testsyn.c -I./include
abigail@ubuntu:~/Desktop/automate$ cd bin
abigail@ubuntu:~/Desktop/automate/bin$ ./testsyn
Enter the name of the file that you want to compile:Dpile.txt
Analyse syntax finish!
Read testsyn.log and testsyn_result.log for more information
abigail@ubuntu:~/Desktop/automate/bin$
```

Le fichier `testsyn.log` généré à partir de l'analyse syntaxique de `Dpile.txt`.



```
testsyn.log — automate [SSH: virtual]
资源管理器 ... log > testsyn.log
打开的编辑器
× testsyn.log log
AUTOMA... bin include log del.sh testlex.log testsem.log testsyn_result.log
testsyn.log
1 Automate 1
2 Now in analyse_lexical
3 ( 21
4 2 11
5 ) 22
6 = 31
7 { 25
8 Now in etats_statement
9 etats 2
10 = 31
11 [ 23
12 ` 28
13 A 10
14 ` 28
15 , 27
16 ` 28
17 B 10
18 ` 28
19 , 27
20 ` 28
21 C 10
22 ` 28
23 ] 24
24 etats analyse success
25
26 Now in initial_statement
```

Le fichier `testsyn_result.log` généré à partir de l'analyse syntaxique de `Dpile.txt`.

```

testsyn_result.log — automate [SSH: virtual]
资源管理器 ... ≡ testsyn.log ≡ testsyn_result.log ×
打开的编辑器 log > ≡ testsyn_result.log
≡ testsyn.log log
× ≡ testsyn_result.lo...
AUTOMATE [SSH: VIRTUR...
> bin
> include
< log
$ del.sh
≡ testlex.log
≡ testsem.log
≡ testsyn_result.log
≡ testsyn.log
≡ TS.txt
≡ VM.txt
> src
M Makefile

1 Status: A
2 Status: B
3 Status: C
4 numStacks: 2
5 index_status_initial: 0
6 index_status_final: 2
7 from:0, to:0, trigger:a,
8 stack1:a, action1:1, stack2: , action2:0
9
10 from:0, to:1, trigger:b,
11 stack1:a, action1:-1, stack2:b, action2:1
12
13 from:1, to:1, trigger:b,
14 stack1:a, action1:-1, stack2:b, action2:1
15
16 from:1, to:2, trigger:c,
17 stack1: , action1:0, stack2:b, action2:-1
18
19 from:2, to:2, trigger:c,
20 stack1: , action1:0, stack2:b, action2:-1
21
22

```

问题 输出 调试控制台 终端 端口 1 bash - bin + ×

Analyse Semantique

- Exécuter `make testsem` dans le dossier `automate`.
- Ouvrez le dossier `bin` et exécutez `./testsem`
- Entrez le nom du fichier que vous souhaitez faire analyser, ici, je utilise `Dpile.txt`
- Ouvrez le dossier `log` pour voir les journaux générés.

```

abigail@ubuntu:~/Desktop/automate$ make testsem
gcc -Wall -o bin/testsem src/testsem.c -I./include
abigail@ubuntu:~/Desktop/automate$ cd bin
abigail@ubuntu:~/Desktop/automate/bin$ ./testsem
Enter the name of the file that you want to compile:Dpile.txt
Analyse semantic finish!
Read testsem.log for more information
abigail@ubuntu:~/Desktop/automate/bin$ 

```

Le fichier `testsem.log` généré à partir de l'analyse semantique de `Dpile.txt`.

```
testsem.log — automate [SSH: virtual]
资源管理器 ... ⌂ testsyn_result.log ⌂ testsem.log ×
打开的编辑器 log > ⌂ testsem.log
    1 No semantic error!
    2
AUTOMATE [SSH: VIRTUR... > bin
> include
log
$ del.sh
⠄ testlex.log
⠄ testsem.log
⠄ testsyn_result.log
⠄ testsyn.log
⠄ TS.txt
⠄ VM.txt
> src
M Makefile
```

Compilation

1. Exécuter `make compile` dans le dossier `automate`.
2. Ouvrez le dossier `bin` et exécutez `./compile`
3. Entrez le nom du fichier que vous souhaitez faire analyser, ici, je utilise `Dpile.txt`
4. Ouvrez le dossier `log` pour voir les journaux générés.

```
abigail@ubuntu:~/Desktop/automate$ make compile
gcc -Wall -o bin/compile src/compile.c -I./include
abigail@ubuntu:~/Desktop/automate$ cd bin
abigail@ubuntu:~/Desktop/automate/bin$ ./compile
Enter the name of the file that you want to compile:Dpile.txt

Analyse lexical and analyse syntax finish!
Read testsyn.log and testsyn_result.log for more information

Analyse semantic finish!
Read testsem.log for more information

Start to compile!
Compile finish!
Read VM.txt and TS.txt for more information

abigail@ubuntu:~/Desktop/automate/bin$
```

Le fichier `VM.txt` généré à partir du compilation de `Dpile.txt`.

VM.txt — automate [SSH: virtual]

```
1 2 0 1 30 2 97 4 97 1 0 0 98 17 97 -1 98 1 2 98 17 97 -1 98 1 99 30 0 0 98 -1 1 99 30 0 0 98
```

资源管理器 ... testsyn_result.log VM.txt

打开的编辑器

VM.txt log

AUTOMA... bin include log

- bin
- include
- log
 - del.sh
 - testlex.log
 - testsem.log
 - testsyn_result.log
 - testsyn.log
 - TS.txt
- VM.txt
- src

Makefile

问题 输出 调试控制台 终端 端口 1

bash - bin + ×

Le fichier `TS.txt` généré à partir du compilation de `Dpile.txt`.

TS.txt — automate [SSH: virtual]

```
1 name:A ,index:0 ,address:4
2 name:B ,index:1 ,address:17
3 name:C ,index:2 ,address:30
4
```

资源管理器 ... testsyn_result.log TS.txt

打开的编辑器

TS.txt log

AUTOMA... bin include log

- bin
- include
- log
 - del.sh
 - testlex.log
 - testsem.log
 - testsyn_result.log
 - testsyn.log
 - TS.txt
- VM.txt
- src

Makefile

问题 输出 调试控制台 终端 端口 1

bash - bin + ×

Execution

1. Exécuter `make main` dans le dossier `automate`.
 - o Exécuter `make main debug:=yes` dans le dossier `automate` si vous voulez être en mode debug
2. Ouvrez le dossier `bin` et exécutez `./main`
3. Entrez le nom du fichier que vous souhaitez faire analyser, ici, je utilise `Dpile.txt`

Le mot d'entrée est: `abc`

```
abigail@ubuntu:~/Desktop/automate$ make main debug:=yes
gcc -D debug -Wall -o bin/main src/main.c -I./include
abigail@ubuntu:~/Desktop/automate$ cd bin
abigail@ubuntu:~/Desktop/automate/bin$ ./main
Enter the name of the file that you want to compile: Dpile.txt

Start to compile!
Compile finish!

Enter the string for the VM: abc

-> Status: A Stack1: Empty Stack2: Empty
a -> Status: A Stack1: a Stack2: Empty
b -> Status: B Stack1: Empty Stack2: b
c -> Status: C Stack1: Empty Stack2: Empty

The string is accepted!
```

Le mot d'entrée est: `aaabbc`

```
abigail@ubuntu:~/Desktop/automate/bin$ ./main
Enter the name of the file that you want to compile: Dpile.txt

Start to compile!
Compile finish!

Enter the string for the VM: aaabbc

-> Status: A Stack1: Empty Stack2: Empty
a -> Status: A Stack1: a Stack2: Empty
a -> Status: A Stack1: aa Stack2: Empty
a -> Status: A Stack1: aaa Stack2: Empty
b -> Status: B Stack1: aa Stack2: b
b -> Status: B Stack1: a Stack2: bb
c -> Status: C Stack1: a Stack2: b

Error: The stack is not empty in the end!
The string is refused!
```

Je utilise `Zpile.txt`, Le mot d'entrée est: `1112211`

```
abigail@ubuntu:~/Desktop/automate/bin$ ./main
Enter the name of the file that you want to compile: Zpile.txt

Start to compile!
Compile finish!

Enter the string for the VM: 1112211
-> Status: Init
1 -> Status: 2

The string is not recognized!
The string is refused!
abigail@ubuntu:~/Desktop/automate/bin$ 
```

Je utilise `Zpile.txt`, Le mot d'entrée est: `1212`

```
abigail@ubuntu:~/Desktop/automate/bin$ ./main
Enter the name of the file that you want to compile: Zpile.txt

Start to compile!
Compile finish!

Enter the string for the VM: 1212
-> Status: Init
1 -> Status: 2
2 -> Status: 3
1 -> Status: 2
2 -> Status: 3
The string is accepted!
abigail@ubuntu:~/Desktop/automate/bin$ 
```

Je utilise `Upile.txt`, Le mot d'entrée est: `aabbbaa`:

```
abigail@ubuntu:~/Desktop/automate/bin$ ./main
Enter the name of the file that you want to compile: Upile.txt

Start to compile!
Compile finish!

Enter the string for the VM: aabbbaa
-> Status: — Stack1: Empty
a -> Status: — Stack1: a
a -> Status: — Stack1: aa
b -> Status: __ Stack1: a
b -> Status: __ Stack1: Empty
b -> Status: __
Error : Stack 1 empty !
abigail@ubuntu:~/Desktop/automate/bin$ 
```