



République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université AMO de Bouira
Faculté des Sciences et des Sciences Appliquées
Département d'Informatique



Rapport de projet TP

Compilation

Specialite : L3 Systeme Informatique

Tokeniser, analyseur lexicale et analyseur syntaxique

Elaboré par:

- MEDDAHI Fares
- DAHMANI Abdelhak
- HAMANA Zineb

Contents

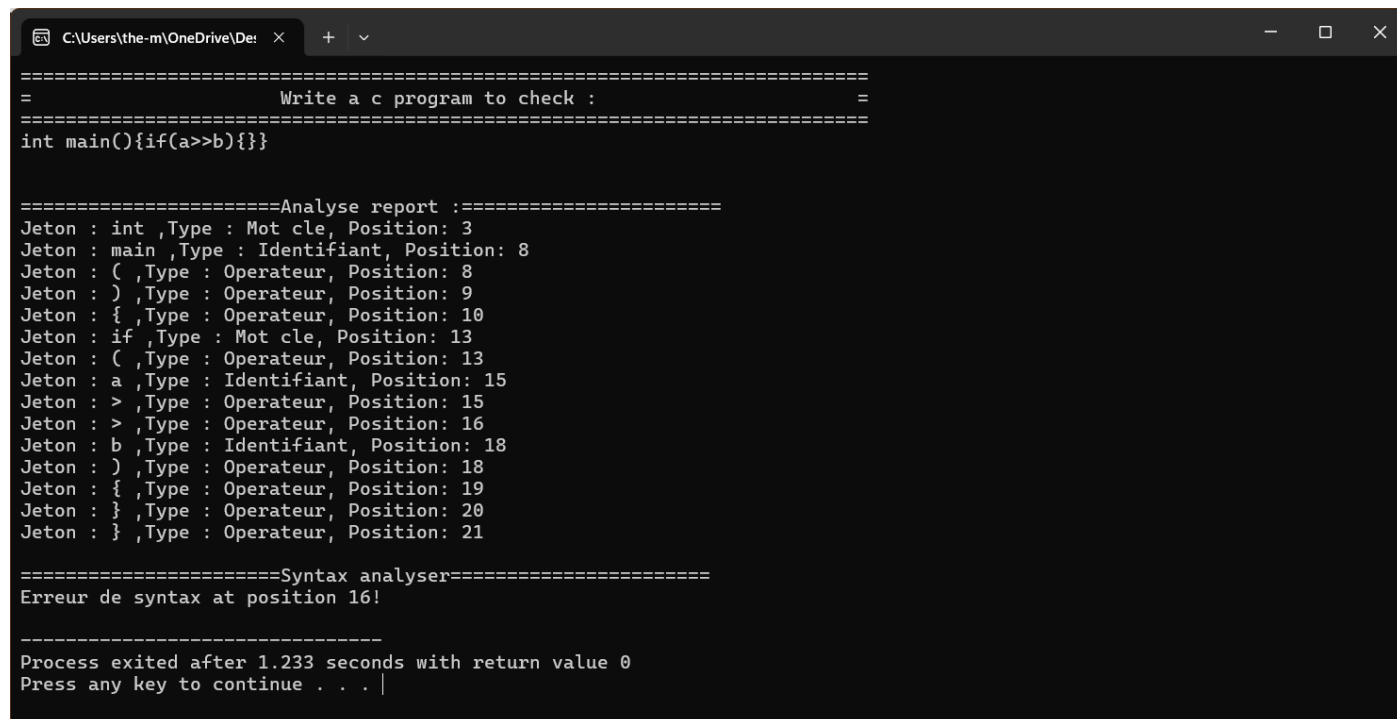
0.1	Introduction	1
0.2	L'analyseur l'exicale	2
0.2.1	La fonction tokeniser	2
0.2.2	Les automates, tables de transition et les fonctions	5
0.2.3	Exemples d'execution	10
0.2.4	Error handling examples	12
0.3	L'analyseur syntaxique	14
0.3.1	Grammaire	16
0.3.2	Table de transition	17
0.3.3	Exemples des erreurs de syntax	18
0.4	L'application de JavaSwing	20

0.1 Introduction

Ce rapport est rédigé pour expliquer le fonctionnement de notre programme d'analyse lexicale et syntaxique en langage C, ainsi que du programme d'interface graphique Java Swing.

L'utilisateur écrit un programme en C dans le terminal, et notre programme extrait les jetons, leurs types et leurs positions pour l'utilisateur, puis l'analyse lexicalement et vérifie s'il y a une erreur, en la gérant le cas échéant.

Dans l'application Java Swing, l'utilisateur saisit du code C dans un JTextArea, et l'application extrait automatiquement les jetons, leurs positions et leurs types pour les afficher à l'utilisateur. Lorsque l'utilisateur clique sur le bouton Vérifier, l'application vérifie s'il y a une erreur lexicale.



```
C:\Users\the-m\OneDrive\De: X + v
=====
= Write a c program to check : =
=====
int main(){if(a>>b){}}

=====Analyse report :=====
Jeton : int ,Type : Mot cle, Position: 3
Jeton : main ,Type : Identifiant, Position: 8
Jeton : ( ,Type : Operateur, Position: 8
Jeton : ) ,Type : Operateur, Position: 9
Jeton : { ,Type : Operateur, Position: 10
Jeton : if ,Type : Mot cle, Position: 13
Jeton : ( ,Type : Operateur, Position: 13
Jeton : a ,Type : Identifiant, Position: 15
Jeton : > ,Type : Operateur, Position: 15
Jeton : > ,Type : Operateur, Position: 16
Jeton : b ,Type : Identifiant, Position: 18
Jeton : ) ,Type : Operateur, Position: 18
Jeton : { ,Type : Operateur, Position: 19
Jeton : } ,Type : Operateur, Position: 20
Jeton : } ,Type : Operateur, Position: 21

=====Syntax analyser=====
Erreur de syntax at position 16!

-----
Process exited after 1.233 seconds with return value 0
Press any key to continue . . . |
```

0.2 L'analyseur l'exicale

En commençant par la fonction main, celle-ci demande à l'utilisateur d'écrire un programme en C, puis appelle la fonction tokeniser, appelle la fonction qui imprime le rapport des jetons, et ensuite appelle la fonction qui analyse le syntax du code.

0.2.1 La fonction tokeniser

La fonction tokeniser prend l'expression écrite par l'utilisateur et la divise en jetons. Chaque jeton possède une valeur, un type et une position. La fonction tokeniser vérifie également s'il y a une erreur lexicale pour chaque jeton et s'arrête immédiatement si une erreur est détectée.

À la fin de l'expression, la fonction tokeniser retourne un tableau de jetons (Token). Un Token est une structure qui contient la valeur, le type et la position du jeton.

```
typedef struct {
    char token[TOKEN_SIZE];
    char type[TOKEN_TYPE_SIZE];
    int position;
} Token;

Token *tokeniser(char *expression){
    Token *tokens = malloc(TOKEN_TABLE_SIZE * sizeof(Token));
    int i = 0, j = 0;
    while(expression[i] != '\0'){
        if(expression[i] == '>' || expression[i] == '<' || expression[i] == '=' || expression[i] == '!'){
            if(expression[i+1] == '='){
                char word[3];
                word[0] = expression[i];
                word[1] = expression[i+1];
                word[2] = '\0';
                Token t; strcpy(t.token, word); strcpy(t.type, TYPE_OPERATOR); t.position = i+2; tokens[j++] = t;
            } else {
                char word[2];
                word[0] = expression[i];
                word[1] = '\0';
                Token t; strcpy(t.token, word); strcpy(t.type, TYPE_OPERATOR); t.position = i++; tokens[j++] = t;
            }
        } else if(is_operator(expression[i])){
            char word[2];
            word[0] = expression[i];
            word[1] = '\0';
            Token t; strcpy(t.token, word); strcpy(t.type, TYPE_OPERATOR); t.position = i++; tokens[j++] = t;
        } else if(is_alphabetic(expression[i]) || expression[i] == '_'){
```

```

char word[TOKEN_SIZE];
int k = 0;
while(is_alphabetic(expression[i]) || is_numeric(expression[i]) || expression[i] == '_' ){
    word[k++] = expression[i++];
}
word[k] = '\0';
if(is_keyword(word)){
    Token t;strcpy(t.token,word);strcpy(t.type,TYPE_KEYWORD);t.position = i;tokens[j++] = t;
}else if(is_id(word)){
    Token t;strcpy(t.token,word);strcpy(t.type,TYPE_ID);t.position = i;tokens[j++] = t;
}else{
    Token t;strcpy(t.token,word);strcpy(t.type,TYPE_ERROR);t.position = i;tokens[j++] = t;
    save_error("Mot cle invalide");
    break;
}
}else if(is_numeric(expression[i])){
    char word[TOKEN_SIZE];
    int k = 0;
    while( is_numeric(expression[i]) || expression[i] == '.'){
        word[k++] = expression[i++];
    }
    word[k] = '\0';
    if(is_float(word) || is_int(word)){
        Token t;strcpy(t.token,word);strcpy(t.type,TYPE_LITERAL);t.position = i;tokens[j++] = t;
    }else{
        Token t;strcpy(t.token,word);strcpy(t.type,TYPE_ERROR);t.position = i;tokens[j++] = t;
        save_error("Literal numerique invalide");
        break;
    }
}else if(expression[i] == '\'){
    char word[TOKEN_SIZE];
    word[0] = '\';
    int k = 1;i++;
    while( expression[i] != '\ ' && expression[i] != '\0'){
        word[k++] = expression[i++];
    }
    word[k++] = '\ ' ;i++;
    word[k] = '\0';
    if(expression[i] == '\0' || !is_char(word) ){
        if(expression[i] == '\0') word[--k] = '\0';
        Token t;strcpy(t.token,word);strcpy(t.type,TYPE_ERROR);t.position = i;tokens[j++] = t;
        save_error("Literal caractere invalide");
        break;
    }
}

```

```

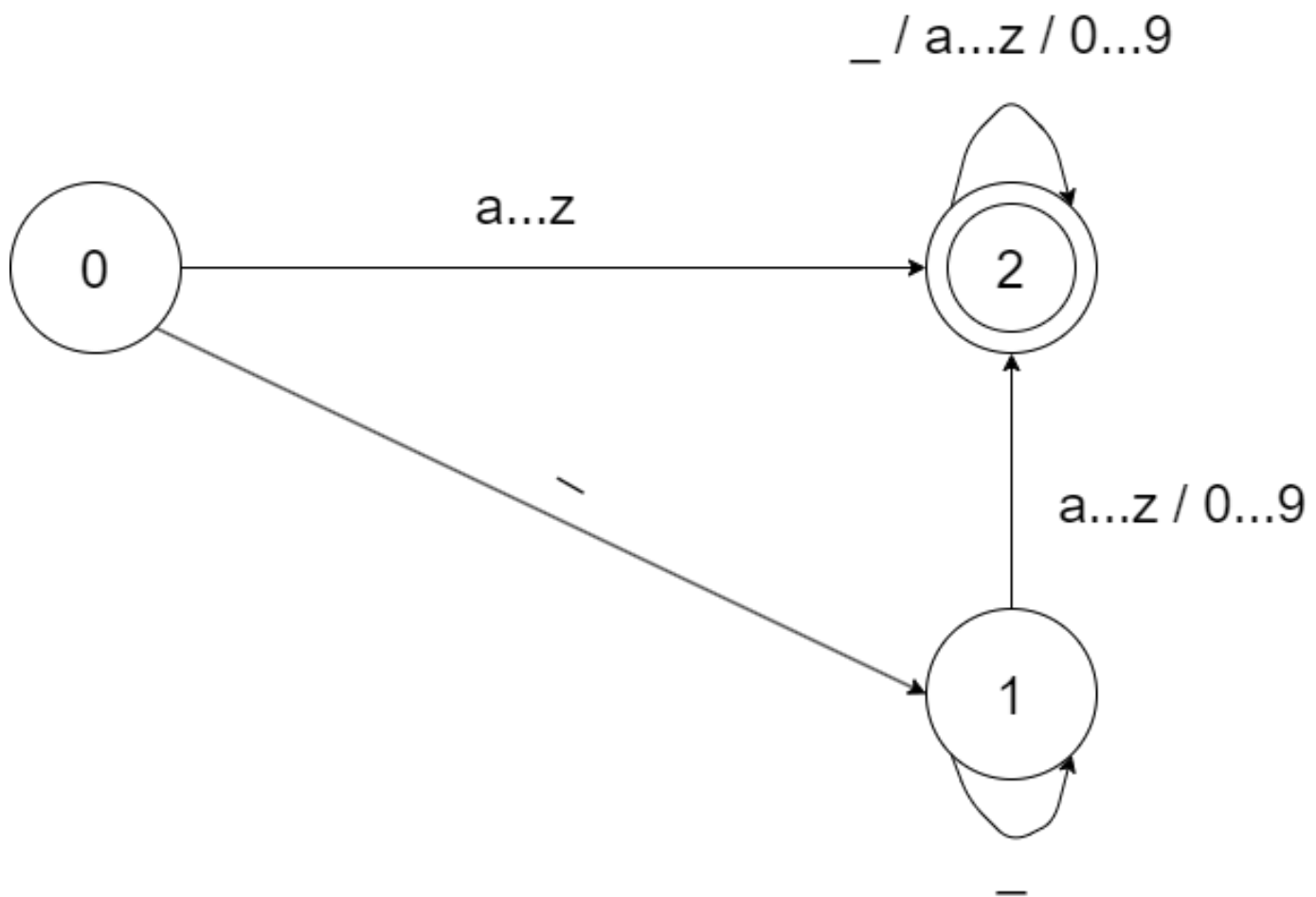
}else{
    Token t;strcpy(t.token,word);strcpy(t.type,TYPE_LITERAL);t.position = i;tokens[j++] = t;
}
}else if(expression[i] == '"'){
    char word[TOKEN_SIZE];
    word[0] = '"';
    int k = 1;i++;
    while( expression[i] != '"' && expression[i] != '\0'){
        word[k++] = expression[i++];
    }
    word[k++] = '"';i++;
    word[k] = '\0';
    if(expression[i] == '\0' || !is_str(word) ){
        word[--k] = '\0';
        Token t;strcpy(t.token,word);strcpy(t.type,TYPE_ERROR);t.position = i;tokens[j++] = t;
        save_error("Literal chaine de caractere invalide");
        break;
    }else{
        Token t;strcpy(t.token,word);strcpy(t.type,TYPE_LITERAL);t.position = i;tokens[j++] = t;
    }
}else if(is_space(expression[i])){
    i++;
}else{
    char word[2];
    word[0] = expression[i];
    word[1] = '\0';
    Token t;strcpy(t.token,word);strcpy(t.type,TYPE_ERROR);t.position = i;tokens[j++] = t;
    save_error("Caractere non reconnu");
    break;
}
}
Token t;strcpy(t.token,"$");strcpy(t.type,"$");t.position = i;tokens[j++] = t;
return tokens;
}

```

0.2.2 Les automates, tables de transition et les fonctions

L'identificateur

Pour vérifier si un jeton est un identificateur, nous avons utilisé cet automate et cette table de transition.



	-	a...z	0...9	error
0	1	2	3	3
1	1	2	2	3
2	2	2	2	3
3	3	3	3	3

```
int is_id(char *word){
    int tra_table[4][5] = {
{0, 1 , 2, 3, 3},
{1, 1 , 2, 2, 3},
{2, 2 , 2, 2, 3},
{3, 3 , 3, 3, 3}
};
int current_state = 0;
int i = 0,symbol_type;
while(current_state != 3 && word[i] !='\0'){
    if(word[i] == '_' ) symbol_type = 1;
```

```

else if(is_alphabetic(word[i])) symbol_type = 2;
else if(is_numeric(word[i])) symbol_type = 3;
else symbol_type = 4;
current_state = tra_table[current_state][symbol_type];
i++;
}
return current_state == 2 ? 1 : 0;
}

```

Le mot clé

Le mot-clé est un identificateur spécial, il n'est donc pas nécessaire d'utiliser un automate. Il suffit de vérifier si ce jeton appartient aux mots-clés du langage.

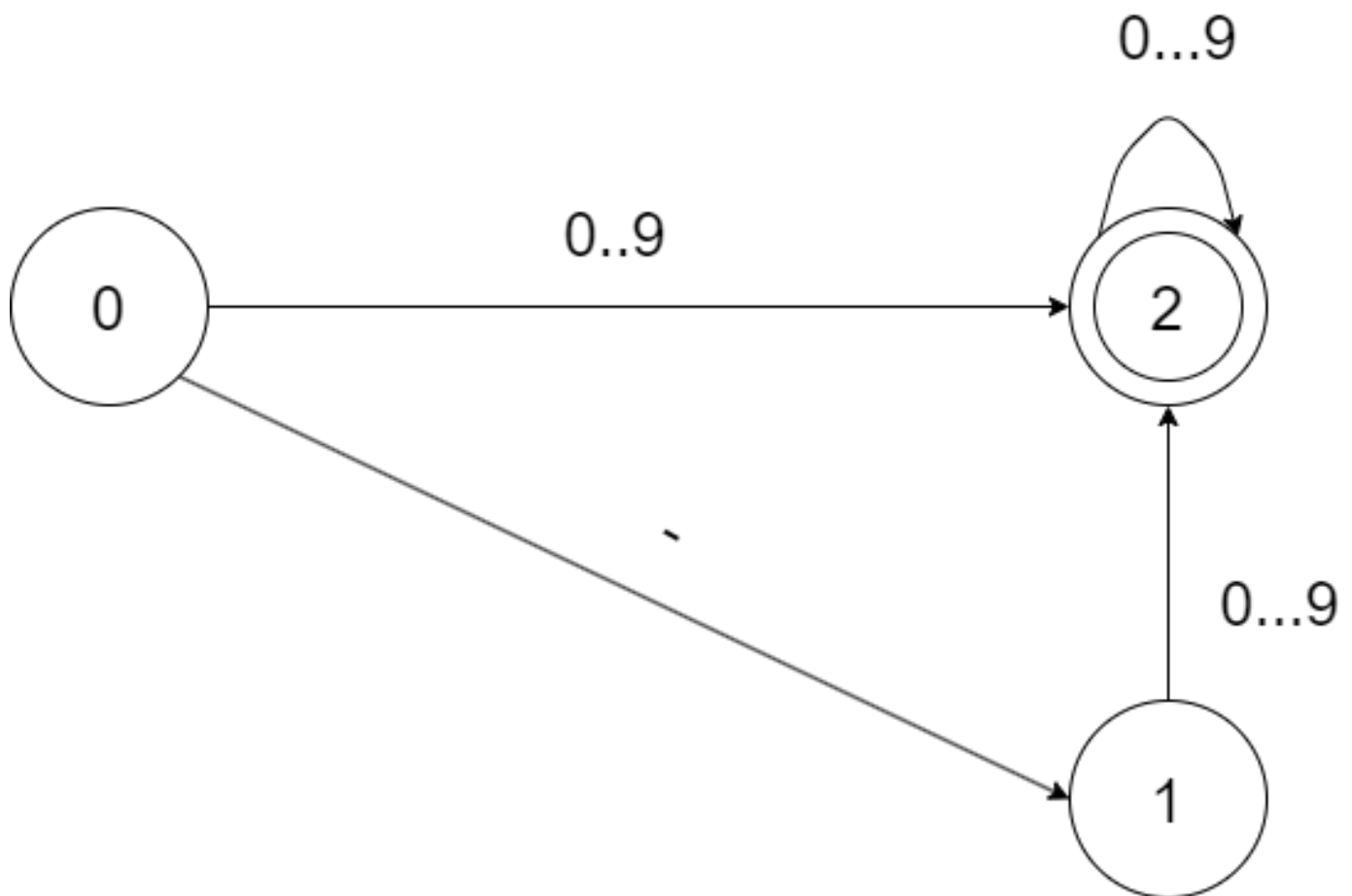
```

int is_keyword(char *keyword){
    return strcmp(keyword,"int") == 0
        || strcmp(keyword,"void") == 0
        || strcmp(keyword,"float") == 0
        || strcmp(keyword,"char") == 0
        || strcmp(keyword,"if") == 0
        || strcmp(keyword,"else") == 0
        || strcmp(keyword,"while") == 0
        || strcmp(keyword,"return") == 0
        || strcmp(keyword,"do") == 0 ? 1 : 0;
}

```


Les nombres entiers

Pour vérifier un nombre entier, qu'il soit négatif ou positif, nous avons suivi cet automate et cette table de transition.



	-	0..9	error
0	1	2	3
1	3	2	3
2	3	2	3
3	3	3	3

```

int is_int(char *word){
int tra_table[4][4] = {
  {0, 1 , 2, 3},
  {1, 3 , 2, 3},
  {2, 3 , 2, 3},
  {3, 3 , 3, 3}
};
int current_state = 0;
int i = 0,symbol_type;
while(current_state != 3 && word[i] !='\0'){
  if(word[i] == '-') symbol_type = 1;
  else if(is_numeric(word[i])) symbol_type = 2;
  else symbol_type = 3;
  current_state = tra_table[current_state][symbol_type];
}
}

```

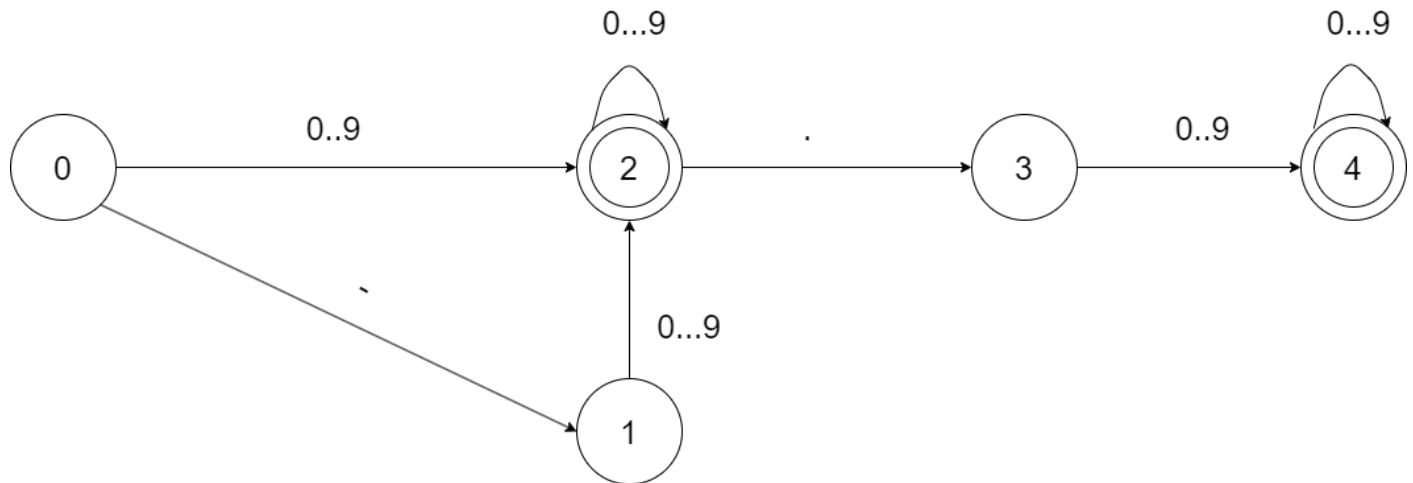
```

    i++;
}
return current_state == 2 ? 1 : 0;
}

```

Les nombres reels

Pour vérifier un nombre reels, qu'il soit négatif ou positif, nous avons suivi cet automate et cette table de transition.



	-	0..9	.	error
0	1	2	5	5
1	5	2	5	5
2	5	2	3	5
3	5	4	5	5
4	5	4	5	5
5	5	5	5	5

```

int is_float(char *word){
int tra_table[5][6] = {
    {0, 1 , 2, 5, 5},
    {1, 5 , 2, 5, 5},
    {2, 5 , 2, 3, 5},
    {3, 5 , 4, 5, 5},
    {4, 5 , 4, 5, 5},
    {5, 5 , 5, 5, 5}
};
int current_state = 0;
int i = 0,symbol_type;
while(current_state != 5 && word[i] !='\0'){
    if(word[i] == '-') symbol_type = 1;
    else if(is_numeric(word[i])) symbol_type = 2;
    else if(word[i] == '.') symbol_type = 3;
    else symbol_type = 4;

```

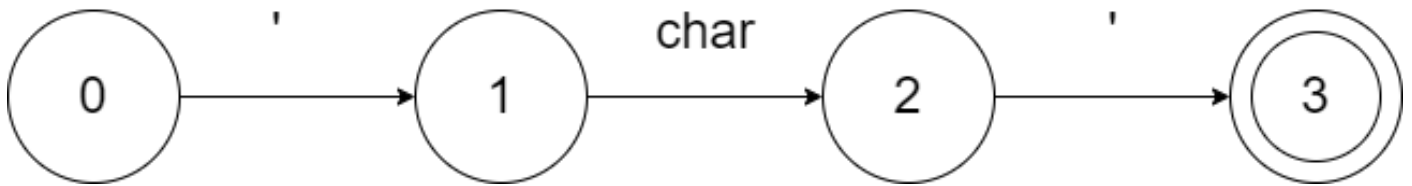
```

    current_state = tra_table[current_state][symbol_type];
    i++;
}
int _ = 3;
printf("%f",_);
return current_state == 2 || current_state == 4 ? 1 : 0;
}

```

Les caracteres

Pour vérifier un littéral de type char, nous avons utilisé l'automate et la table de transition suivants :



	'	char	error
0	1	4	4
1	4	2	4
2	3	4	4
3	4	4	4
4	4	4	4

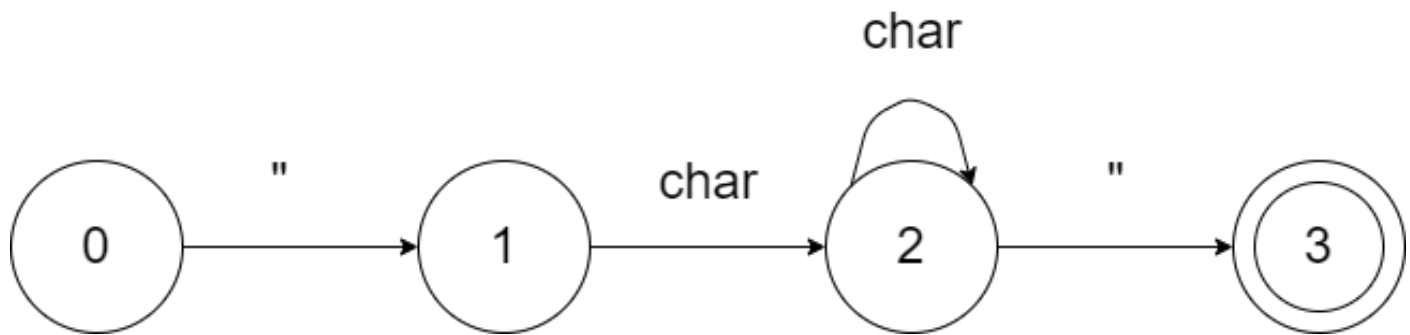
```

    int is_char(char *word){
int tra_table[5][4] = {
    {0, 1 , 4, 4},
    {1, 4 , 2, 4},
    {2, 3 , 4, 4},
    {3, 4 , 4, 4},
    {4, 4 , 4, 4}
};
int current_state = 0;
int i = 0,symbol_type;
while(current_state != 4 && word[i] !='\0'){
    if(word[i] == '\'' ) symbol_type = 1;
    else if(word[i] >= 0 && word[i] <= 127) symbol_type = 2;
    else symbol_type = 3;
    current_state = tra_table[current_state][symbol_type];
    i++;
}
return current_state == 3 ? 1 : 0;
}

```

Les chaines de caracteres

Pour les chaînes de caractères (Strings), nous avons utilisé l'automate et la table de transition suivants :



	"	char	error
0	1	4	4
1	4	2	4
2	3	2	4
3	4	4	4
4	4	4	4

```
int is_str(char *word){
int tra_table[5][4] = {
    {0, 1 , 4, 4},
    {1, 4 , 2, 4},
    {2, 3 , 2, 4},
    {3, 4 , 4, 4},
    {4, 4 , 4, 4}
};
int current_state = 0;
int i = 0,symbol_type;
while(current_state != 4 && word[i] !='\0'){
    if(word[i] == 34) symbol_type = 1;
    else if(word[i] >= 0 && word[i] <= 127) symbol_type = 2;
    else symbol_type = 3;
    current_state = tra_table[current_state][symbol_type];
    i++;
}
return current_state == 3 ? 1 : 0;
}
```

0.2.3 Exemples d'exécution

Exemple 01

```
int main(){}
```

//Output

=====Analyse report :=====

Jeton : int ,Type : Mot cle, Position: 3
Jeton : main ,Type : Identifiant, Position: 8
Jeton : (,Type : Operateur, Position: 8
Jeton :) ,Type : Operateur, Position: 9
Jeton : { ,Type : Operateur, Position: 10
Jeton : } ,Type : Operateur, Position: 11

Exemple 02

```
int fonction(int a, int b){return a;} int main(){float a = 3.14;}
```

//Output

=====Analyse report :=====

Jeton : int ,Type : Mot cle, Position: 3
Jeton : fonction ,Type : Identifiant, Position: 12
Jeton : (,Type : Operateur, Position: 12
Jeton : int ,Type : Mot cle, Position: 16
Jeton : a ,Type : Identifiant, Position: 18
Jeton : , ,Type : Operateur, Position: 18
Jeton : int ,Type : Mot cle, Position: 23
Jeton : b ,Type : Identifiant, Position: 25
Jeton :) ,Type : Operateur, Position: 25
Jeton : { ,Type : Operateur, Position: 26
Jeton : return ,Type : Mot cle, Position: 33
Jeton : a ,Type : Identifiant, Position: 35
Jeton : ; ,Type : Operateur, Position: 35
Jeton : } ,Type : Operateur, Position: 36
Jeton : int ,Type : Mot cle, Position: 41
Jeton : main ,Type : Identifiant, Position: 46
Jeton : (,Type : Operateur, Position: 46
Jeton :) ,Type : Operateur, Position: 47
Jeton : { ,Type : Operateur, Position: 48
Jeton : float ,Type : Mot cle, Position: 54
Jeton : a ,Type : Identifiant, Position: 56
Jeton : = ,Type : Operateur, Position: 57
Jeton : 3.14 ,Type : Litteraux, Position: 63
Jeton : ; ,Type : Operateur, Position: 63
Jeton : } ,Type : Operateur, Position: 64

0.2.4 Error handling examples

Caractere inconue

```
int main(){int a@bc = 3;}
//Output
=====Analyse report :=====
Jeton : int ,Type : Mot cle, Position: 3
Jeton : main ,Type : Identifiant, Position: 8
Jeton : ( ,Type : Operateur, Position: 8
Jeton : ) ,Type : Operateur, Position: 9
Jeton : { ,Type : Operateur, Position: 10
Jeton : int ,Type : Mot cle, Position: 14
Jeton : a ,Type : Identifiant, Position: 16
Jeton : @ ,Type : Erreur, Position: 16
Erreur @ : Caractere non reconnu a la position 16.
```

Erreur de litteraux

```
//Exemple 01:
float a = 3.14.5
//Output
=====Analyse report :=====
Jeton : float ,Type : Mot cle, Position: 5
Jeton : a ,Type : Identifiant, Position: 7
Jeton : = ,Type : Operateur, Position: 8
Jeton : 3.14.5 ,Type : Erreur, Position: 16
Erreur 3.14.5 : Literal numerique invalide a la position 16.
```

```
//Exemple 02:
char a = 'ab'
//Output
=====Analyse report :=====
Jeton : char ,Type : Mot cle, Position: 4
Jeton : a ,Type : Identifiant, Position: 6
Jeton : = ,Type : Operateur, Position: 7
Jeton : 'ab' ,Type : Erreur, Position: 13
Erreur 'ab' : Literal caractere invalide a la position 13.
```

```
//Exemple 03:
char a = 'a ;return 0;
//Output
=====Analyse report :=====
```

Jeton : char ,Type : Mot cle, Position: 4
Jeton : a ,Type : Identifiant, Position: 6
Jeton : = ,Type : Operateur, Position: 7
Jeton : 'a ;return 0 ,Type : Erreur, Position: 23
Erreur 'a ;return 0 : Literal caractere invalide a la position 23.

0.3 L'analyseur syntaxique

Notre analyseur syntaxique accepte :

- Les procedures et fonctions avec les parametres
- Dans chaque fonction ou procédure, un bloc d'instructions commence par "{" et se termine par "}".
- Les instructions acceptées sont :
 - Une declaration : `int abc = 20;`
 - Une Condition : `if(a==b){}else if(a != b){}else{}`
 - Une boucle : `while(a != 0)` ou `dowhile(a != 0);`
 - Une expression de return : `return a+b;`

Pour mieu comprendre voici cette exemple:

```
int somme(int a,int b){  
    return a+b;  
}
```

```
int main(){  
    int a = 5;  
    if(a>b){}  
    else if(a == b){}  
    else{  
        while(a!=0){}  
        do{}while(a <= 5);  
        return 0;  
}
```

//Output

=====Analyse report :=====

```
Jeton : int ,Type : Mot cle, Position: 3  
Jeton : somme ,Type : Identifiant, Position: 9  
Jeton : ( ,Type : Operateur, Position: 9  
Jeton : int ,Type : Mot cle, Position: 13  
Jeton : a ,Type : Identifiant, Position: 15  
Jeton : , ,Type : Operateur, Position: 15  
Jeton : int ,Type : Mot cle, Position: 19  
Jeton : b ,Type : Identifiant, Position: 21  
Jeton : ) ,Type : Operateur, Position: 21  
Jeton : { ,Type : Operateur, Position: 22  
Jeton : return ,Type : Mot cle, Position: 29  
Jeton : a ,Type : Identifiant, Position: 31  
Jeton : + ,Type : Operateur, Position: 31
```


Jeton : b ,Type : Identifiant, Position: 33
 Jeton : ; ,Type : Operateur, Position: 33
 Jeton : } ,Type : Operateur, Position: 34
 Jeton : int ,Type : Mot cle, Position: 39
 Jeton : main ,Type : Identifiant, Position: 44
 Jeton : (,Type : Operateur, Position: 44
 Jeton :) ,Type : Operateur, Position: 45
 Jeton : { ,Type : Operateur, Position: 46
 Jeton : int ,Type : Mot cle, Position: 50
 Jeton : a ,Type : Identifiant, Position: 52
 Jeton : = ,Type : Operateur, Position: 53
 Jeton : 5 ,Type : Litteraux, Position: 56
 Jeton : ; ,Type : Operateur, Position: 56
 Jeton : if ,Type : Mot cle, Position: 60
 Jeton : (,Type : Operateur, Position: 60
 Jeton : a ,Type : Identifiant, Position: 62
 Jeton : > ,Type : Operateur, Position: 62
 Jeton : b ,Type : Identifiant, Position: 64
 Jeton :) ,Type : Operateur, Position: 64
 Jeton : { ,Type : Operateur, Position: 65
 Jeton : } ,Type : Operateur, Position: 66
 Jeton : else ,Type : Mot cle, Position: 71
 Jeton : if ,Type : Mot cle, Position: 74
 Jeton : (,Type : Operateur, Position: 74
 Jeton : a ,Type : Identifiant, Position: 76
 Jeton : == ,Type : Operateur, Position: 79
 Jeton : b ,Type : Identifiant, Position: 81
 Jeton :) ,Type : Operateur, Position: 81
 Jeton : { ,Type : Operateur, Position: 82
 Jeton : } ,Type : Operateur, Position: 83
 Jeton : else ,Type : Mot cle, Position: 88
 Jeton : { ,Type : Operateur, Position: 88
 Jeton : } ,Type : Operateur, Position: 89
 Jeton : while ,Type : Mot cle, Position: 97
 Jeton : (,Type : Operateur, Position: 97
 Jeton : a ,Type : Identifiant, Position: 99
 Jeton : != ,Type : Operateur, Position: 101
 Jeton : 0 ,Type : Litteraux, Position: 102
 Jeton :) ,Type : Operateur, Position: 102
 Jeton : { ,Type : Operateur, Position: 103
 Jeton : } ,Type : Operateur, Position: 104
 Jeton : do ,Type : Mot cle, Position: 108
 Jeton : { ,Type : Operateur, Position: 108

```

Jeton : } ,Type : Operateur, Position: 109
Jeton : while ,Type : Mot cle, Position: 115
Jeton : ( ,Type : Operateur, Position: 115
Jeton : a ,Type : Identifiant, Position: 117
Jeton : <= ,Type : Operateur, Position: 120
Jeton : 5 ,Type : Litteraux, Position: 122
Jeton : ) ,Type : Operateur, Position: 122
Jeton : ; ,Type : Operateur, Position: 123
Jeton : return ,Type : Mot cle, Position: 130
Jeton : 0 ,Type : Litteraux, Position: 132
Jeton : ; ,Type : Operateur, Position: 132
Jeton : } ,Type : Operateur, Position: 133

```

=====Syntax analyser=====

Pas d'erreur de syntax

0.3.1 Grammaire

La grammaire utilisée par cet analyseur est :

```

S → type id(P) {B} S | #
P → type idF | #
F → ,type idF | #
B → DB | CB | LB | return E | #;
D → type id = E;
E → YX
X → +YX | -YX | *YX | /YX | %YX | #
Y → id | const | (E)
C → if(Q){B}A
A → else T | #
T → if(Q){B}A | {B}
L → while(Q){B} | do{B}while(Q);
Q → E cs E

```

0.3.2 Table de transition

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	type	id	const	()	{	}	return	if	else	while	do	+	-	*	/	%	;	cs	,	#
0	S	1																			2
1	P	3			4																4
2	F				6															5	6
3	B	7	8	8			12	11	9		10	10									12
4	D	13																			
5	E		14	14																	
6	X				20								15	16	17	18	19	20			20
7	Y		21	22	23																
8	C								24												
9	A	26	26	26			26	26	26	25	26	26									26
10	T					28			27												
11	L										29	30									
12	Q		31	31																	

0.3.3 Exemples des erreurs de syntax

//Exemple 01

```
int main(){float a = 16.3;
```

//Output

=====Analyse report :=====

Jeton : int ,Type : Mot cle, Position: 3

Jeton : main ,Type : Identifiant, Position: 8

Jeton : (,Type : Operateur, Position: 8

Jeton :) ,Type : Operateur, Position: 9

Jeton : { ,Type : Operateur, Position: 10

Jeton : float ,Type : Mot cle, Position: 16

Jeton : a ,Type : Identifiant, Position: 18

Jeton : = ,Type : Operateur, Position: 19

Jeton : 16.3 ,Type : Litteraux, Position: 25

Jeton : ; ,Type : Operateur, Position: 25

=====Syntax analyser=====

Erreur de syntax at position 27!

//Exemple 02

```
int main()int a = 16;}
```

//Output

=====Analyse report :=====

Jeton : int ,Type : Mot cle, Position: 3

Jeton : main ,Type : Identifiant, Position: 8

Jeton : (,Type : Operateur, Position: 8

Jeton :) ,Type : Operateur, Position: 9

Jeton : int ,Type : Mot cle, Position: 13

Jeton : a ,Type : Identifiant, Position: 15

Jeton : = ,Type : Operateur, Position: 16

Jeton : 16 ,Type : Litteraux, Position: 20

Jeton : ; ,Type : Operateur, Position: 20

Jeton : } ,Type : Operateur, Position: 21

=====Syntax analyser=====

Erreur de syntax at position 13!

//Exemple 03

```
int main(){if(a>>b){}}
```

//Output

=====Analyse report :=====

Jeton : int ,Type : Mot cle, Position: 3

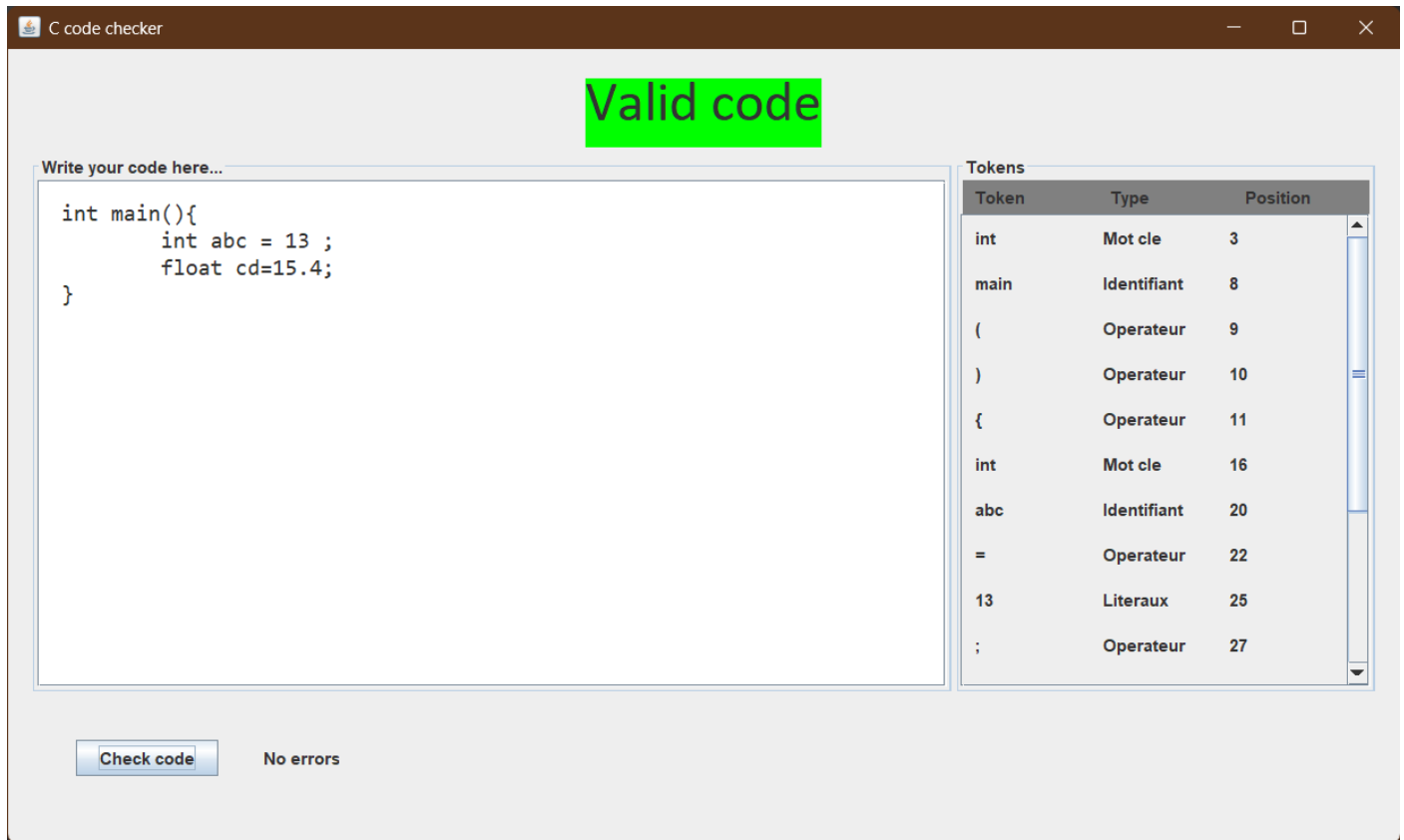
Jeton : main ,Type : Identifiant, Position: 8
Jeton : (,Type : Operateur, Position: 8
Jeton :) ,Type : Operateur, Position: 9
Jeton : { ,Type : Operateur, Position: 10
Jeton : if ,Type : Mot cle, Position: 13
Jeton : (,Type : Operateur, Position: 13
Jeton : a ,Type : Identifiant, Position: 15
Jeton : > ,Type : Operateur, Position: 15
Jeton : > ,Type : Operateur, Position: 16
Jeton : b ,Type : Identifiant, Position: 18
Jeton :) ,Type : Operateur, Position: 18
Jeton : { ,Type : Operateur, Position: 19
Jeton : } ,Type : Operateur, Position: 20
Jeton : } ,Type : Operateur, Position: 21

=====Syntax analyser=====

Erreur de syntax at position 16!

0.4 L'application de JavaSwing

C'est une simple application avec une interface graphique (GUI) avec JavaSwing qui permet d'extraire les jetons, leurs types, ainsi que leur position. Cependant, l'analyseur syntaxique de cette application n'est pas encore bien développé.



Border Layout

