République Algérienne Démocratique et Populaire

Ecole Polytechnique

Rapport de Mini-Compilateur C

réalisé par

Badis Marshall

Table des matières

1 Introduction

L'ntérêt de ce mini-projet est de créer une version simplifié d'un compilateur du langage C. Pour cela, on est amené à créer un analyseur lexicale : pour reconnaître les différents lexèmes (tokens) du code source et un analyseur syntaxique : afin de vérifier l'ordre des lexèmes reconnues dans le code. Les outils utilisés sont : visual stdio pour compiler le code source de projet et Qt creator pour faire l'interface graphique de ce mini compilateur.

2 Analyse Lexicale

Dans cette phase le flot de carctéres séparé par des blancs formant le programme source est transformé en entités (mot clé, identificateur, chiffres, opérateurs et séparateur).

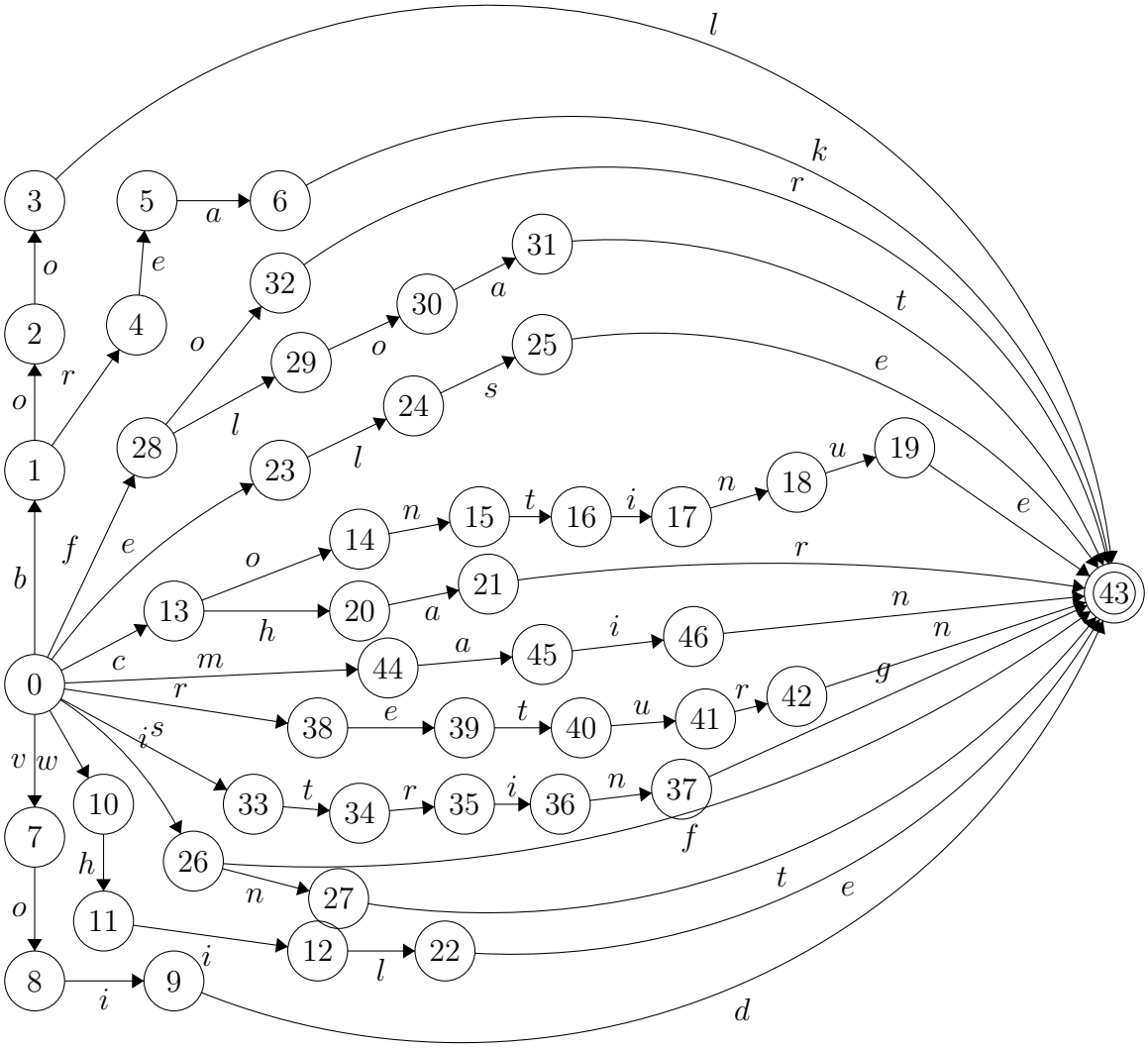
2.1 Définition des entités lexicales

2.1.1 Les Mots Clés

Les mots clés sont des mots réservés et des données du programme à compiler,Les mot clé qui seront charger par ce compilateur ce sont :

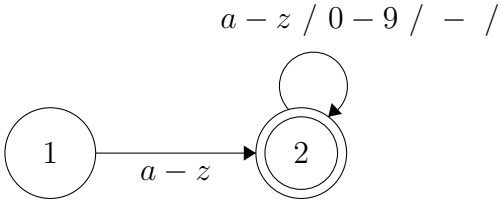
- $i$nt / float / char / bool / void / string
- $f$or / while
- if / else
- $m$ain
- $r$eturn / break / continue

Le automate des mots clés est comme suit :



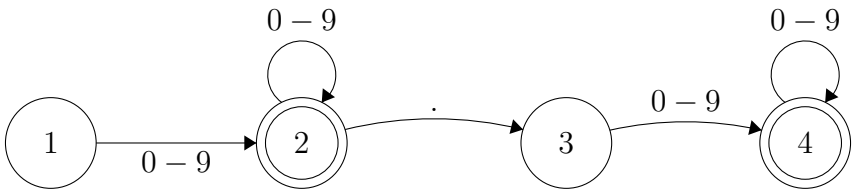
2.1.2 Les Identificateurs

Ce sont les mots qui doivent commencer par une lettre et qui ne sont pas des mots réservés, ils suivent l'automate :



2.1.3 Les Constants

Ce sont les chiffres et les numéros, ils suivent l'automate :



2.1.4 Les Séparateurs

Les séparateurs utilisé ce sont :

— (,) , , ,; , , [,]

2.1.5 Les Opérateures

Les Opérateures utilisé ce sont :

— > , < , = , == ,!= , <= , >= , += , -= , /= , *= , + , - , * , / , ||

2.2 Le but d'analyse lexicale

L'analyse lexicale consiste à partir d'un programme qui est une suite de caractères séparés par des blancs à :

— *S*pécifier les différentes entités (les mots) du langage, on parlera d'entités lexicales parmi ces entités, on reconnaît : Les identificateurs, Les mots clés (Les mots réservés), Les constantes, Les séparateurs.

— *E*liminer les blancs, ainsi que les commentaires.

— *C*onstruire la table des informations.

2.3 Exmple d'un résultats d'analyse lexicale

3 Analyse Syntaxique

L'analyse syntaxique a pour rôle la vérification de la forme ou encore de l'écriture de la suite des entités lexicales, on parlera alors de la syntaxe du langage . La vérification est faite à l'aide de la grammaire spécifique au langage appelée grammaire syntaxique .

3.1 La grammaire syntaxique

<S> -> <program>

program -> <vardefinition> main ( ) <body> <funcdefinition>

<vardefinition> -> <vardef> <vardefinition> | ϵ

$< funcdefinition > - >< funcdef >< funcdefinition > |\epsilon$

$< vardef > - >< type >< var >;$

$< type > - > int|float|string|bool|char$

$< var > - >< idnum >< varextra >$

$< varextra > - >, < var > |\epsilon|[< idnum >] < varextra >$

$< idnum > - > id|num$

$< funcdef > - > id(< paramlist >)< body >$

$< paramlist > - >< type >< param > |\epsilon$

$< param > - >< idnum >< paramextra >$

$< paramextra > - >, < paramlist > |\epsilon$

$< body > - >< vardefinition >< statlist >$

$< statlist > - >< ifelsestat >< statlist > | < forstat >< statlist > | < whilestat >< statlist > | < aorf >< statlist > |return < expression >< statlist > |continue; < statlist > |break; < statlist > |\epsilon$

$< aorf > - > id < assignorfunc >$

$< assignorfunc > - >< funccall > | < assignstat >$

$< funccall > - > (< inparamlist >);$

$< inparamlist > - >< factor >< inparams > |\epsilon$

$< inparams > - >, < inparamlist > |\epsilon$

$< assignstat > - >=< expression >;$

$< expression > - >< exp >< exp1 >$

$< exp1 > - > + < exp >< exp1 > |- < exp >< exp1 > |\epsilon$

$< exp > - >< factor >< exp2 >$

$< exp2 > - > * < factor >< exp2 > |/ < factor >< exp2 > |\epsilon$

$< factor > - > (< expression >)| < idnum >$

$< judgement > - >< factor >< relop >< factor >$

$< relop > - >! = | >= | <= | == | < | >$

$< ifelsestat > - > if(< judgement >)< body > < elsestat >$

$< elsestat > - > else< body > |\epsilon$

$< forstat > - > for(id < assignstatfor >; < judgement >; id < assignstatfor >)< body >$

$< assignstatfor > - >=< expression >$

$< whilestat > - > while(< judgement >)< body >$

La méthode d'analyse utilisée est la decente récursive, cette méthode est l'une des méthodes de l'analyse desendante, Elle consiste à associer à chaque non-terminal une procédure qui traite tous les MDP du non-terminal . La grammaire utilisée est LL(1) donc on peut faire

cette analyse.

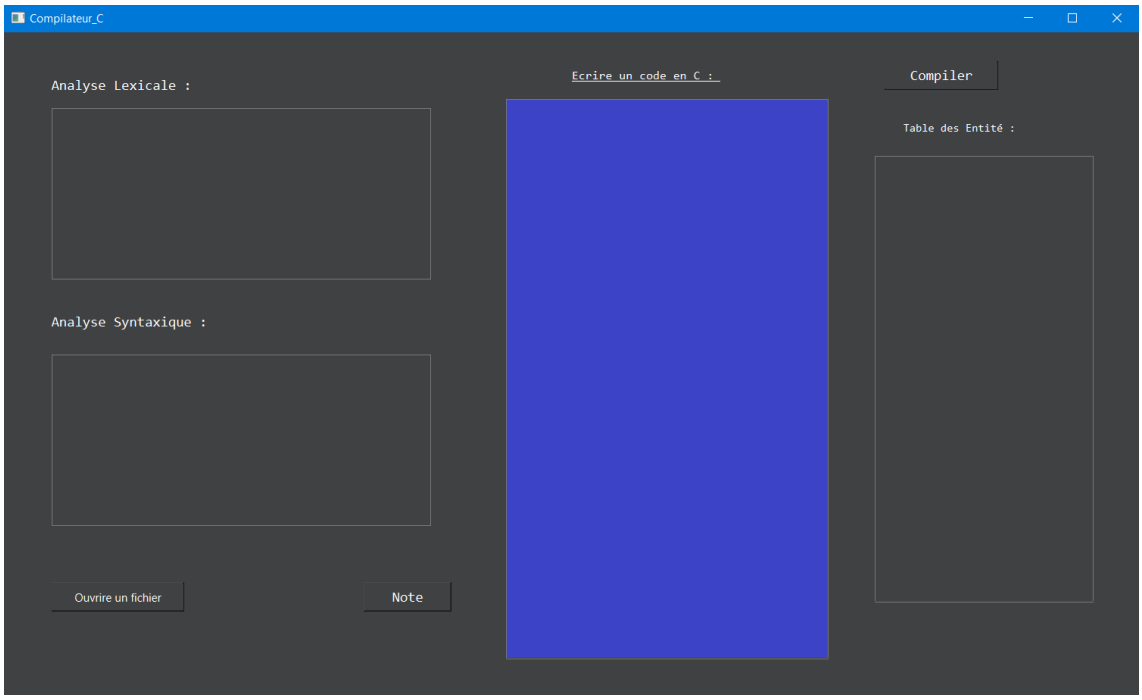
le mode de l'analyseur syntaxique est le mode panic.

3.2 Table de debut et suivant

FIRST	FOLLOW	Nonterminal
{'',void,int,float,char,string}	{$}	S
{'',void,int,float,char,string}	{$}	program
{'',int,float,char,string}	{void,},return,break,continue,if,for,while,id}	vardefinition
{'',id}	{$}	funcdefinition
{int,float,char,string}	{void,int,float,char,string,},return,break,continue,if,for,while,id}	vardef
{int,float,char,string}	{id,num}	type
{bool}	{undefined}	typr
{id,num}	{;}	var
{=,,,'',[}	{;}	A
{,,'',[}	{;}	varextra
{id,num}	{=,,,;,[,],),*,/,+,-,},return,break,continue,if,for,while,id,!=,>=,<=,==,<,>}	idnum
{id}	{$,id}	funcdef
{int,float,char,string,''}	{)}	paramlist
{id,num}	{)}	param
{'',,}	{)}	paramextra
{'',return,break,continue,int,float,char,string,if,for,while,id}	{}}	body
{'',return,break,continue,if,for,while,id}	{}}	statlist
{id}	{},return,break,continue,if,for,while,id}	aorf
{(,=}	{},return,break,continue,if,for,while,id}	assignorfunc
{(}	{},return,break,continue,if,for,while,id}	funccall
{'',(,id,num}	{)}	inparamlist
{,,''}	{)}	inparams
{=}	{},return,break,continue,if,for,while,id}	assignstat
{(,id,num}	{},return,break,continue,if,for,while,id,;,)}	expression
{+,-,''}	{},return,break,continue,if,for,while,id,;,)}	exp1
{(,id,num}	{+,-,},return,break,continue,if,for,while,id,;,)}	exp
{*,/,''}	{+,-,},return,break,continue,if,for,while,id,;,)}	exp2
{(,id,num}	{,,),*,/,+,-,},return,break,continue,if,for,while,id,;,!=,>=,<=,==,<,>}	factor
{(,id,num}	{),;}	judgement
{!=,>=,<=,==,<,>}	{(,id,num}	relop
{if}	{},return,break,continue,if,for,while,id}	ifelsestat
{'',else}	{},return,break,continue,if,for,while,id}	elsestat
{for}	{},return,break,continue,if,for,while,id}	forstat
{=}	{;,)}	assignstatfor
{while}	{},return,break,continue,if,for,while,id}	whilestat

4 Interface Graphique

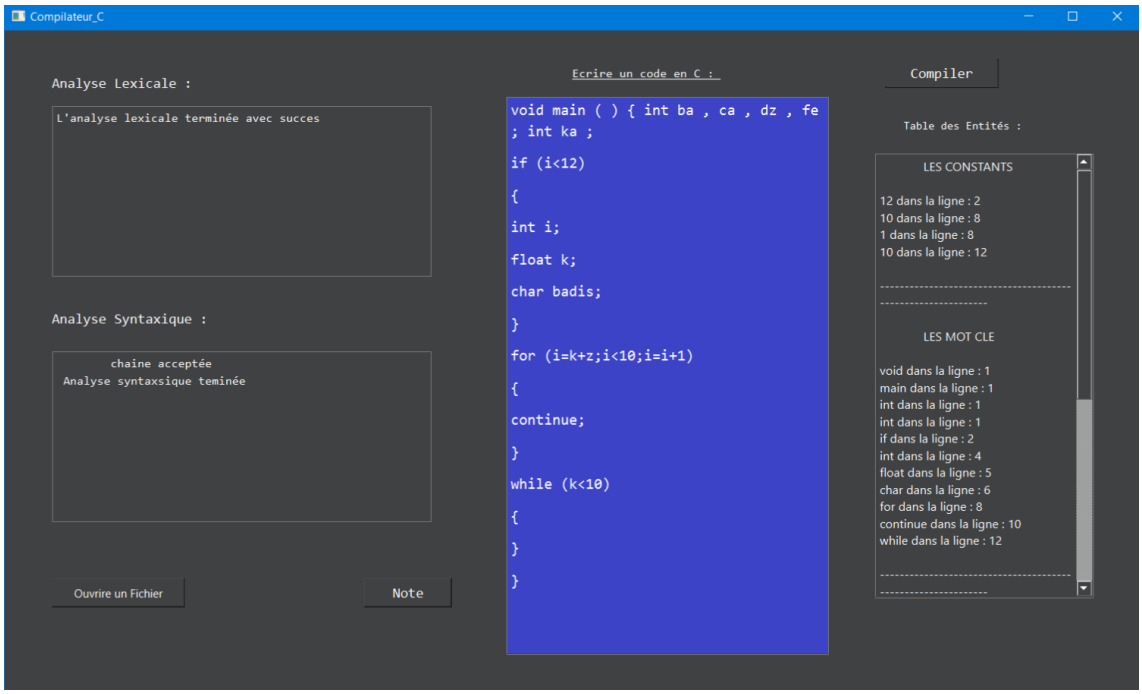
L'interface graphique est crée à l'aide de la bibliothèque Qt.



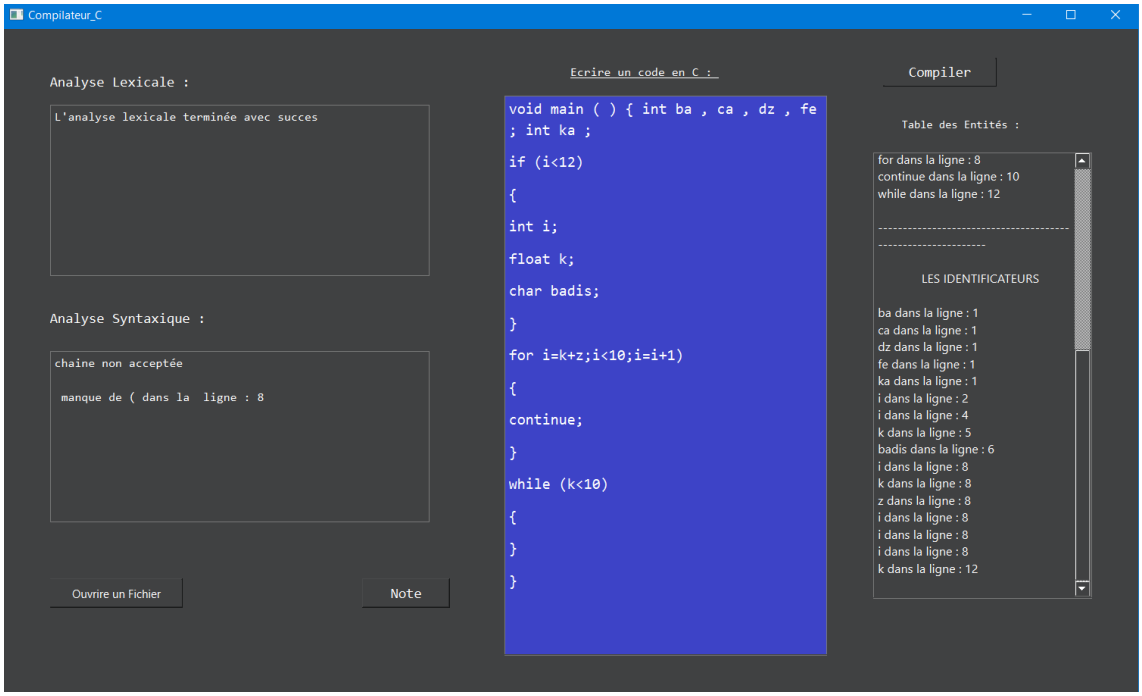
le champ de Analyse Lexicale indique le résultat d'analyse lexicale si il y' a aucun erreur lexicale il afiche un message "analyse lexicale terminée avec succes", et aussi le champ Analyse Syntaxique affiche "chaine accptée" si il y' a pas une erreur syntaxique, dans le cas contraire le champ Analyse Syntaxique affiche un erreur avec la ligne corespandante. le compilateur analyse le text ecrit dans le champ coloré en bleu, aprés avoir cliqué sur le button "Compiler" le résultat s'affiche, les déffirents entités de code son affiché dans le champ "Table des Entités".

Il est possible aussi de séléctioner un fichier text depuis l'ordinateur du utilisateur en appuyant sur le button "Ovrire un Fichier".

voici un exmple d'un bout de code en C :



voici un exmple de code avec erreur :



le compilateur affiche bien l'erreur avec le numéro de la ligne.

Note : le code de Compilateur est uploder avec le rapport.

5 Conclusion

L'étape de compilation à un rôle primordiale avant l'exécution des programmes car elle nous permet de détecter les erreurs de programmation ; pour passer au différentes analyses de programme source. Mais durant la conception de ce compilateur il faut avant tout passer par l'étape de modélisation qui nous permet de définir le langage, choisir la manière dont on représente le langage en machine et elle permet aussi de définir la grammaire avec laquelle on va aborder l'étape de l'analyse syntaxique.