



Ecole Nationale Supérieure d'Informatique et d'Analyse des Systèmes - RABAT

Rapport de Projet de compilation : Compilateur bl meghribi

Réalisé par :

EL JAZOULY Anass
EL HAFIANE Asmaa
ABAID Lamyaa

Encadré par :

Pr. TABII Youness
Pr. OULAD HAJ THAMI Rachid

Année académique : 2021-20222



Remerciements :

Nous voudrions tout d'abord adresser toute notre gratitude à notre professeur du module Monsieur TABII Youness pour sa disponibilité et cette opportunité qui nous a permise de bien maîtriser les concepts fondamentaux de la compilation.

Nous désirons aussi remercier Monsieur OULAD HAJ THAMI Rachid qui a contribué à la réussite de ce projet avec ses connaissances et ses conseils.



RÉSUMÉ

Ce rapport présente le projet que nous avons réalisé en groupe pour le module de compilation de la deuxième année à l'ENSIAS. Le but de ce projet est de faire la conception d'un langage de programmation, qu'on a nommé BL MEGHRIBI, et la réalisation d'un simple compilateur (analyse lexical, syntaxique et sémantique) pour ce langage. Dans ce rapport, nous allons présenter dans un premier temps le langage de programmation et les cas d'utilisation potentiels. Ensuite, nous allons nous intéresser à la présentation et la validation de notre grammaire LL (1). A la fin, un ensemble de tests sera effectué dans le but de vérifier que le compilateur arrive effectivement à faire l'analyse lexicale, syntaxique et sémantique d'un code source introduit.

ABSTRACT

This report presents the project that we carried out as a group for the compilation module of the second year at ENSIAS. The goal of this project is to design a programming language, which we named BL MEGHRIBI, and to create a simple compiler (for lexical, syntactic and semantic analysis) specific to this language. In this report, we will at first present the programming language and its potential use cases. Next, we will focus on the presentation and validation of our LL (1) grammar. At the end, a set of tests will be carried out in order to verify that the compiler actually manages to do the lexical, syntax and semantic analysis of an introduced source code.

Table des matières

| | | |
|----------|-------------------------------------------------------------------|-----------|
| 1 | Contexte général du projet | 1 |
| 1.1 | Présentation du projet..... | 1 |
| 1.1.1 | Projet de compilation | 1 |
| 1.1.2 | Présentation du sujet..... | 1 |
| 1.1.3 | Problématique..... | 1 |
| 1.1.4 | Objectif | 1 |
| 2 | Analyseur lexical | 2 |
| 2.1 | Définition :..... | 2 |
| 2.2 | Les mots clés du langage :..... | 2 |
| 2.2.1 | Les blocs du programme :..... | 2 |
| 2.2.2 | La partie déclaration :..... | 2 |
| 2.2.3 | Les instructions :..... | 3 |
| 2.2.4 | Les fonctions déclarées par l'utilisateur..... | 3 |
| 2.2.5 | Les fonctions définies par le langage :..... | 4 |
| 2.2.6 | Les symboles utilisés au niveau du langage | 5 |
| 2.3 | Le code lexical..... | 5 |
| 2.3.1 | La structure des tokens | 5 |
| 2.3.2 | Les fonctions utilisées dans le code lexical | 5 |
| 2.4 | Test de l'analyseur lexical..... | 6 |
| 3 | Analyseur syntaxique | 6 |
| 3.1 | Définition | 6 |
| 3.2 | La structure du programme..... | 6 |
| 3.3 | La grammaire | 7 |
| 3.4 | Les fonctions utilisées au niveau de l'analyseur syntaxique | 10 |
| 3.5 | Test de l'analyseur syntaxique..... | 12 |
| 4 | Analyseur sémantique | 11 |
| 4.1 | Définition :..... | 11 |
| 4.2 | Les règles sémantiques..... | 11 |
| 4.3 | Les fonctions utilisées au niveau de l'analyseur sémantique..... | 11 |

Table des figures

| | | |
|-----|---------------------------------------------------------|----|
| 2.1 | La structure des tokens..... | 5 |
| 3.1 | Grammaire..... | 7 |
| 3.2 | Grammaire..... | 8 |
| 3.3 | Grammaire..... | 9 |
| 4.1 | La structure des éléments de la table des symboles..... | 11 |

Liste des tableaux

| | | |
|-----|----------------------------------------------------------|----|
| 2.1 | Tableau des blocs..... | 2 |
| 2.2 | Tableau des déclarations..... | 2 |
| 2.3 | Tableau des instructions | 3 |
| 2.4 | Tableau des fonctions déclarées par l'utilisateur..... | 3 |
| 2.5 | Tableau des fonctions définies par le langage..... | 4 |
| 2.6 | Tableau des symboles..... | 5 |
| 2.7 | Tableau des fonctions utilisées dans le lexical..... | 5 |
| 3.1 | Tableau des fonctions utilisées dans le syntaxique | 10 |
| 4.1 | Tableau des fonctions utilisées dans le sémantique..... | 12 |

Introduction générale

La compilation informatique désigne le procédé de traduction d'un programme, écrit et lisible par un humain, en un programme exécutable par un ordinateur.

De façon plus globale, il s'agit de la transformation d'un programme écrit en code source, en un programme transcrit en code cible, ou binaire. Habituellement, le code source est rédigé dans un langage de programmation (langage source), il est de haut niveau de conception et facilement accessible à un utilisateur. Le code cible, quant à lui, est transcrit en langage de plus bas niveau (langage cible), afin de générer un programme exécutable par une machine.

Dans ce cadre intervient notre projet visant à mettre en œuvre un compilateur.

La première mission consiste à mettre en œuvre un nouveau langage en spécifiant son intérêt et en définissant sa grammaire.

Ensuite nous allons travailler sur son compilateur en clarifiant toutes les étapes utiles afin de le créer.

Le présent document rapporte nos missions accomplies durant ce projet et il comporte quatre chapitres :

- **Chapitre 1 :**

Se consacrera à une présentation générale du cadre de projet, dans lequel on va définir notre problématique et le but de notre projet.

- **Chapitre 2 :**

Dans ce chapitre nous allons présenter la démarche de construction de notre analyseur lexical.

- **Chapitre 3 :**

Ce chapitre est dédié à la réalisation de l'analyseur syntaxique.

- **Chapitre 4 :**

Ce chapitre présente la partie sémantique du compilateur.

Chapitre 1

Contexte général du projet

1.1 Présentation du projet

1.1.1 Projet de compilation

Dans le cadre de ce projet, nous étions demandés à penser à un nouveau langage et de réaliser le compilateur associé en C afin de valider les connaissances acquises durant les séances de cours ainsi que celles du TP.

1.1.2 Présentation du sujet

Dans le domaine de l'informatique, connaître un langage de programmation est un atout de plus en plus important sur le marché du travail.

Les développeurs marocains sont alors confrontés à un grand défi qui s'incarne dans la nécessité de comprendre la langue anglaise avec laquelle tous les langages de programmation ont été développés.

D'où vient le besoin de créer un langage adapté à notre dialecte pour faciliter la tâche aux programmeurs.

1.1.3 Problématique

Aujourd'hui, disposer de compétences dans les langages de programmation et le codage constitue un atout majeur pour la carrière professionnelle d'un développeur, or la langue anglaise avec laquelle les langages de programmation sont développés constitue un handicap surtout pour les développeurs marocains.

1.1.4 Objectif

La réalisation de ce projet a pour but de mettre en oeuvre un compilateur développé en C d'un langage dont les mots clés sont écrits dans notre dialecte afin de faciliter la tâche de programmation aux développeurs marocains.

Chapitre 2

Analyseur lexical

2.1 Définition :

L'analyse lexicale se trouve tout au début de la chaîne de compilation, elle collabore avec l'analyse grammaticale pour passer de la syntaxe concrète à la syntaxe abstraite. La mission de l'analyse lexicale est de transformer une suite de caractères en une suite de mots, dit aussi lexèmes (tokens).

2.2 Les mots clés du langage :

Dans cette partie, nous allons spécifier les mots clés de notre langage BL MEGHRIBI.

2.2.1 Les blocs du programme :

Notre programme développé BL MEGHRIBI est composé de trois parties :

| | |
|---------|-------------------------------------------------------------------------|
| L3AYBAT | Afin de commencer le bloc de déclaration. |
| FCT | Afin de spécifier le bloc dans lequel les fonctions vont être définies. |
| DEREJ | Afin de commencer le bloc des instructions. |

TABLE 2.1 – Tableau des blocs

2.2.2 La partie déclaration :

Notre langage est un langage non typé, la déclaration se fait de deux manières :

| | |
|---------|----------------------------------|
| TABT | Afin de déclarer les constantes. |
| MT7AREK | Afin de déclarer les variables. |

TABLE 2.2 – Tableau des déclarations

2.2.3 Les instructions :

| | |
|----------------------------------|--------------------------------------------------------------------|
| KTEB | Afin d’afficher du texte et/ou des variables et/ou des constantes. |
| 9RA | Afin de lire des variables. |
| ILAKAN... ILAMAKAN | Afin d’exprimer IF...ELSE. |
| MA7ED | Afin d’exprimer la boucle while. |
| 3LA7ASSAB | Afin d’exprimer SWITCH. |
| WACH | Afin d’exprimer CASE. |
| BARAKA | Afin d’exprimer BREAK. |
| TA7AJA | Afin d’exprimer DEFAULT. |
| 3EYET | Afin d’appeler une fonction déclarée par l’utilisateur. |
| MENHADI() LHADI() NEMCHIWBHADI() | Afin d’exprimer la boucle for(). |

TABLE 2.3 – Tableau des instructions

2.2.4 Les fonctions déclarées par l'utilisateur

La déclaration d’une fonction définie par l’utilisateur se fait au sein du bloc FCT en utilisant les mots clés suivants :

| | |
|--------|-------------------------------------------------------|
| KHAWYA | Afin d’exprimer que la fonction ne retourne rien . |
| 3AMRA | Afin d’exprimer que la fonction retourne un résultat. |
| REJE3 | Afin d’exprimer RETURN. |

TABLE 2.4 – Tableau des fonctions déclarées par l'utilisateur

2.2.5 Les fonctions définies par le langage :

| | |
|------------------|-------------------------------------------------------------------|
| wach3adadwla7arf | Afin de vérifier si l'argument donné est un chiffre ou caractère. |
| wach7arf | Afin de vérifier si l'argument donné est un caractère ou non. |
| wach3adad | Afin de vérifier si l'argument donné est un chiffre ou non. |
| wach3alama | Afin de vérifier si l'argument donné est un point de ponctuation. |
| Vabsolue | Afin d'exprimer la fonction valeur absolue. |
| lba9i | Afin d'exprimer la fonction modulo. |
| sin | Afin d'exprimer la fonction sin. |
| cos | Afin de d'exprimer la fonction cos. |
| tan | Afin d'exprimer la fonction tan . |
| sqrt | Afin d'exprimer la fonction racine carré. |
| log | Afin d'exprimer la fonction logarithme. |
| exp | Afin d'exprimer la fonction exponentielle. |
| puissance | Afin d'exprimer la focntion puissance. |
| 9ssem | Afin d'exprimer la fonction division. |
| asin | Afin d'exprimer la fonction asin. |
| acos | Afin d'exprimer la fonction acos. |
| atan | Afin d'exprimer la fonction atan. |
| l3addad | Afin d'exprimer la fonction atoi. |
| kifmakan | Afin d'exprimer la fonction rand. |
| hez7et | Afin d'exprimer la fonction strepy. |
| 9aren | Afin d'exprimer la fonction strcmp. |
| jme3 | Afin d'exprimer la fonction strcat. |
| toul | Afin d'exprimer la fonction strlen. |

TABLE 2.5 – Tableau des fonctions définies par le langage

2.2.6 Les symboles utilisés au niveau du langage

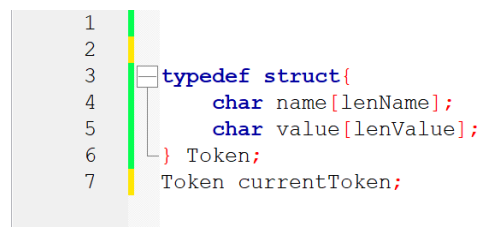
| | |
|-------|--------------------------------------------------------------------|
| = | Afin d'exprimer l'égalité. |
| {...} | Afin d'exprimer le début et la fin d'un bloc. |
| [...] | Afin de contenir un ensemble de valeurs affectées à un tableau. |
| "..." | Afin d'exprimer une chaîne de caractères. |
| , | Afin de séparer un ensemble de valeurs ou variables ou constantes. |
| ; | Afin d'exprimer la fin d'une instruction. |

TABLE 2.6 – Tableau des symboles

2.3 Le code lexical

2.3.1 La structure des tokens

Avant de décomposer le code entré par l'utilisateur en des tokens, nous avons travaillé sur la déclaration d'une structure dans laquelle nous allons sauvegarder les informations de chaque token trouvé dans le code.



```
1
2
3 typedef struct{
4     char name[lenName];
5     char value[lenValue];
6 } Token;
7 Token currentToken;
```

FIGURE 2.1 – La structure des tokens

2.3.2 Les fonctions utilisées dans le code lexical

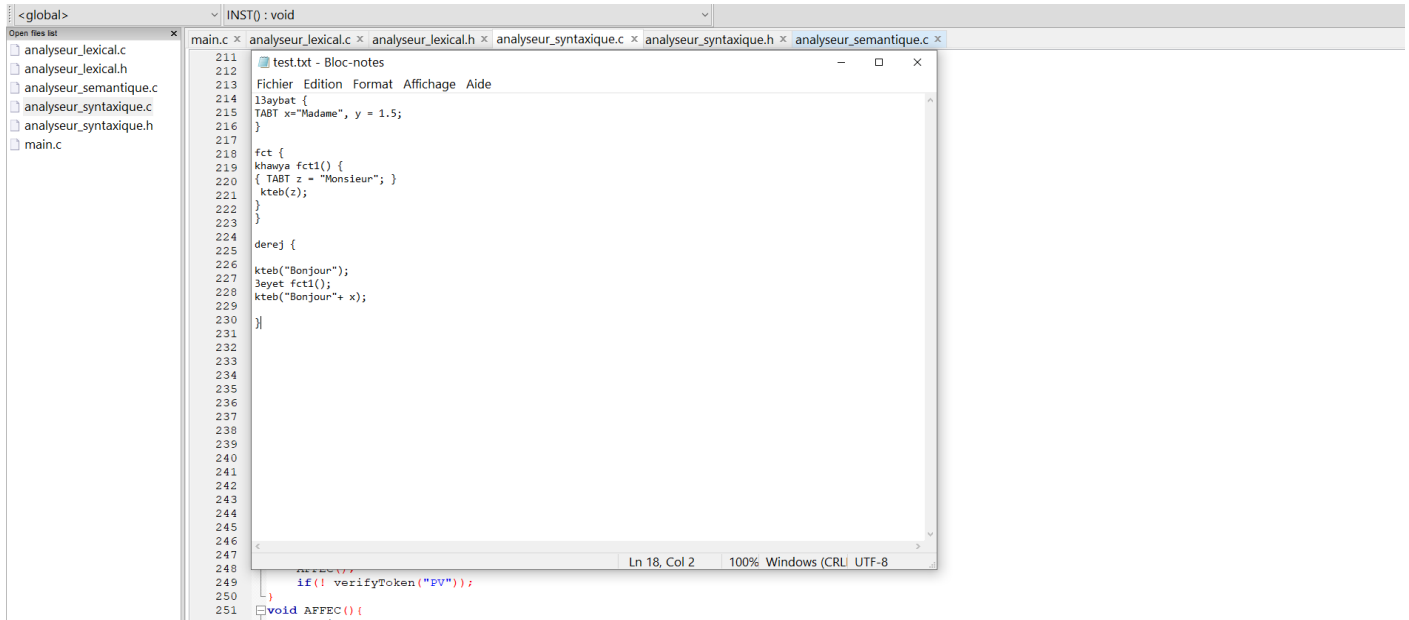
Lors de la réalisation de cet analyseur nous avons utilisé un ensemble de fonctions à savoir :

| | |
|---------------------|------------------------------------------------------------------------------|
| Nextchar() | Afin de récupérer le prochain caractère dans le code créé par l'utilisateur. |
| SyntaxError() | Afin de générer les messages d'erreurs et sortir du programme. |
| ignoreWhiteSpaces() | Afin d'ignorer les espaces blancs. |
| getToken | Afin de récupérer un token. |
| isNumber() | Afin de vérifier si le token courant est un nombre. |
| isWord() | Afin de vérifier si le token courant est un mot. |
| isSpecial() | Afin de vérifier si le token est un symbole spécial. |

TABLE 2.7 – Tableau des fonctions utilisées dans le lexical

2.4 Test de l'analyseur lexical

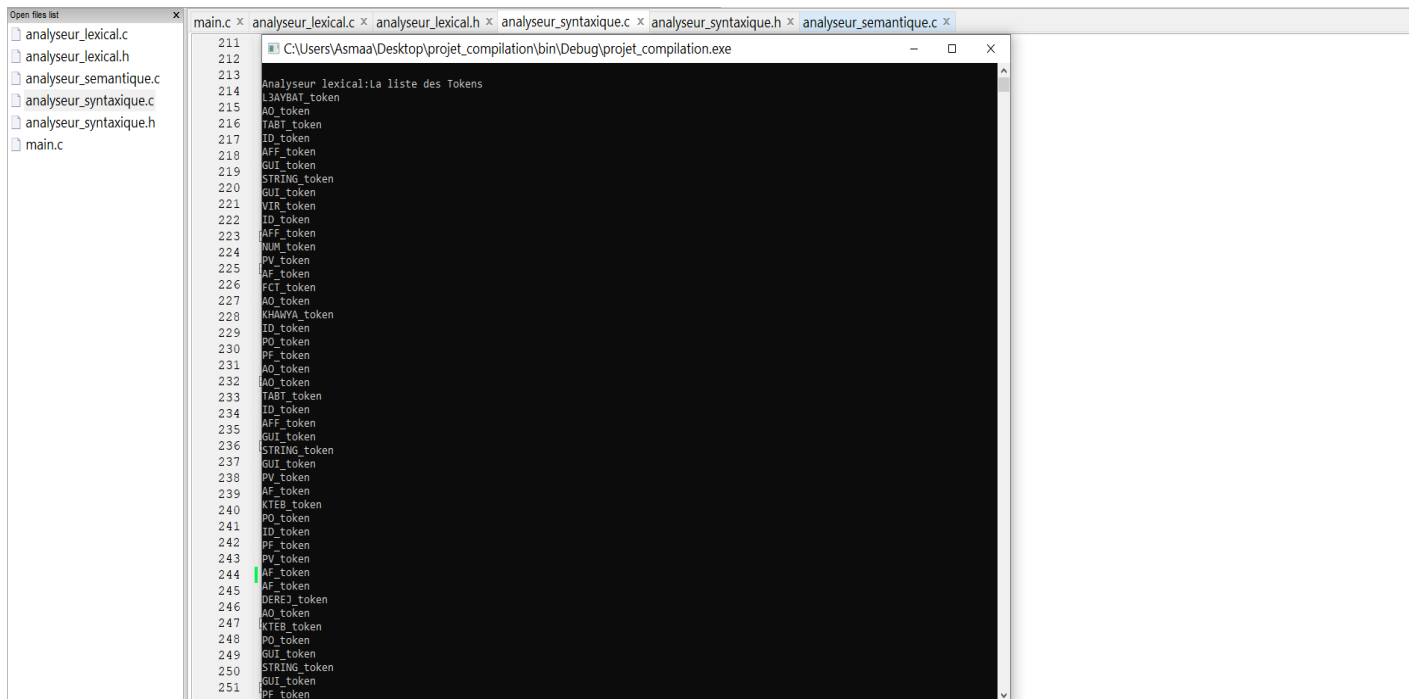
Après l'exécution du code suivant :



The screenshot shows a code editor with a file explorer on the left and a code window on the right. The file explorer lists the following files: `analyseur_lexical.c`, `analyseur_lexical.h`, `analyseur_semantique.c`, `analyseur_syntactique.c`, `analyseur_syntactique.h`, and `main.c`. The code window displays the content of `main.c`, which includes a test program for the lexical analyzer. The code defines a `test` function that calls `analyseur_lexical` and prints the results. The code is as follows:

```
211 // test
212 //
213 // Fichier Edition Format Affichage Aide
214 //
215 // TABT x="Madame", y = 1.5;
216 //
217 //
218 // fct {
219 //   khawya fct1() {
220 //     TABT z = "Monsieur"; }
221 //   kteb(z);
222 // }
223 //
224 // derej {
225 //
226 //   kteb("Bonjour");
227 //   beyet fct1();
228 //   kteb("Bonjour"+ x);
229 // }
230 //
231 //
232 //
233 //
234 //
235 //
236 //
237 //
238 //
239 //
240 //
241 //
242 //
243 //
244 //
245 //
246 //
247 //
248 //
249 //
250 //
251 //
```

On a obtenu le résultat suivant :



The screenshot shows a terminal window with the output of the lexical analyzer. The output is a list of tokens extracted from the input text. The tokens are as follows:

```
Analyseur lexical: La liste des Tokens
L3AYBAT token
AO_token
TABT_token
ID_token
AFF_token
GUI_token
STRING_token
GUI_token
VIR_token
ID_token
AFF_token
NUM_token
PV_token
AF_token
FCT_token
AO_token
KHAWYA_token
ID_token
PO_token
PF_token
AO_token
ID_token
TABT_token
ID_token
AFF_token
GUI_token
STRING_token
GUI_token
PV_token
AF_token
KTEB_token
PO_token
ID_token
PF_token
PV_token
AF_token
DEREJ_token
AO_token
KTEB_token
PO_token
GUI_token
STRING_token
GUI_token
PF_token
```

Chapitre 3

Analyseur syntaxique

3.1 Définition

L'analyse syntaxique est le processus d'analyse du langage naturel avec les règles d'une grammaire formelle. Les règles grammaticales s'appliquent aux catégories et aux groupes de mots, et non aux mots individuels. L'analyse syntaxique attribue essentiellement une structure sémantique au texte.

3.2 La structure du programme

Notre programme est décomposé en principe en 3 blocs dont la structure est la suivante :

```
L3AYBAT{  
...  
}  
FCT{  
...  
}  
DEREJ{  
...  
}
```


3.3 La grammaire

Afin de réaliser un bon analyseur syntaxique nous avons mis en place la grammaire LL(1) suivante :

```
PROGRAM ::= l3aybat DEF fct FONCTION derej BLOCK
DEF ::= { CONSTS VARS }
CONSTS ::= Tabt CONSTID [, CONSTID]; | epsilon
CONSTID ::= ID ALPHA
ALPHA ::= (Num) = \[TABCONST\] | =ALPHANUM
TABCONST ::= NUM[, NUM] | "STRING"[, "STRING"]
VARS ::= MT7AREK ID VAR [, ID VAR ]; | epsilon
VAR ::= epsilon | (NUM)
ALPHANUM ::= NUM | "STRING"
FONCTION ::= { khawya|3amra ID(PARAM) {
    DEF
    INSTS
    [ reje3 (NUM|ID); ]
    [khawya|3amra ID(PARAM) {
        DEF
        INSTS
        [ reje3 (NUM|ID); ] ] } | epsilon }
BLOCK ::= { INSTS }
INSTS ::= INST
INST ::= AFFECT INST | SI INST | ECRIRE INST | LIRE INST | FOR INST | WHILE INST | SWITCH INST | APPELFCT INST | epsilon
AFFECT ::= ID AFFEC;
AFFEC ::= = AFFEC2 | AFFECTAB
AFFECTAB ::= (NUM)=EXPR
AFFEC2 ::= EXPR | \[TABCONST\]
SI ::= ilakan COND { INSTS } [ilamakan { INSTS }]
ECRIRE ::= kteb( ECRIREA );
ECRIREA ::= ID ECRIREB | "STRING" ECRIREB
ECRIREB ::= + ECRIREA | epsilon
LIRE ::= 9ra(ID[, ID]);
FOR ::= menhadi (ID=NUM) lhadi (ID=NUM) nemchiwbhadi(+|- NUM){ INSTS }
```

Figure 3.1 – Grammaire

```

WHILE ::= ma7ed(COND){ INSTS }
SWITCH ::= 3la7assab (ID) {
    wach ALPHANUM: INSTS
        [baraka;]
    [wach ALPHANUM : INSTS
        [baraka;]]
    ta7aja : INSTS [baraka;] }
PARAM ::= ID[,ID]
EXPR ::= TERM [+|- TERM]
TERM ::= FACT [*|/ FACT]
FACT ::= ID | NUM | (EXPR)
ID ::= LETTRE [LETTRE | CHIFFRE]
NUM ::= CHIFFRE [CHIFFRE]
CHIFFRE ::= 0|..|9
LETTRE ::= a|b|..|z|A|..|Z
COND ::= ( EXPR RELOP EXPR )
RELOP ::= == |<|<|>|<|=
APPELCT ::= 3eyet APPEL.
APPEL ::= FCTDEFC1|FCTDEFC2|FCTDEFC3|FCTDEFC4|FCTDEFC5 |FCTDEC
FCTDEC ::= ID(VAL[,VAL]);
VAL ::= ID | NUM |"STRING"

```

Figure 3.2 – Grammaire

```

FCTDEFC1 ::= wach3adadwla7arf(ID) |
    wach7arf(ID) |
    wach3adad(ID) |
    wach3alama(ID) |
    Vabsolue(ID) |
    lba9i(ID) |
    sin(ID) |
    cos(ID) |
    tan(ID) |
    sqrt(ID) |
    log(ID) |
    exp(ID) |
    puissance(ID) |
    9ssem(ID) |
    asin(ID) |
    acos(ID) |
    atan(ID) |
    l3addad(ID) |
    kifmakan(ID)

FCTDEFC2 ::= khroj(NUM)
FCTDEFC3 ::= hez7et(ID,ARG1) | jme3(ID,ARG1)
FCTDEFC4 ::= toul(ARG1)
FCTDEFC5 ::= 9aren(ARG1,ARG1)
ARG1 ::= ID | "STRING"

```

Figure 3.3 – Grammaire

3.4 Les fonctions utilisées au niveau de l'analyseur syntaxique

A la fin de l'analyse lexicale, nous avons sauvegardé les tokens dans une liste chaînée, afin de les analyser syntaxiquement.

Lors de la réalisation de cet analyseur nous avons utilisé un ensemble de fonctions à savoir :

| | |
|---------------------------------|------------------------------------------------------------------------------|
| insertion_syntax(char* ,char*) | Afin de sauvegarder le nom et la valeur des tokens dans une liste chaînée. |
| SyntaxError() | Afin de générer les messages d'erreurs et sortir du programme. |
| verifyToken(char *) | Afin de vérifier si le token de la liste chaînée est dans sa bonne position. |

TABLE 3.1 – Tableau des fonctions utilisées dans le syntaxique

En plus des fonctions précédentes, nous avons ajouter celles qui définissent la grammaire.

```
.void PROGRAM();
void DEF();
void CONSTS();
void CONSTID();
void ALPHA();
void TABCONST();
void VARS();
void VAR();
void ALPHANUM();
void FONCTION();
void BLOCK();
void INSTS();
void INST();
void AFFECT();
void AFFEC();
void AFFECTAB();
void AFFEC2();
void SI();
void ECRIRE();
void ECRIRE1();
void ECRIREB();
void LIRE();
void FOR();
void WHILE();
void SWITCH();
void PARAM();
void EXPR();
void TERM();
void FACT();
void COND();
void APPELFCT();
void APPEL();
void FCTDECO();
void VAL();
void FCTDEFC1();
void FCTDEFC2();
void FCTDEFC3();
void FCTDEFC4();
void FCTDEFC5();
void ARGS();
```

3.5 Test de l'analyseur syntaxique :

Soit le code suivant :

The screenshot shows a C++ IDE with a project named 'global'. The 'main.c' file is open, displaying a C++ program. The code includes 'analyseur_lexical.h' and 'analyseur_syntaxique.h'. The 'main' function is defined, and a 'fct' function is called. The IDE interface includes a file explorer on the left, a top toolbar with 'INST0 : void', and a status bar at the bottom showing 'Ln 18, Col 2' and '100% Windows (CRLF) UTF-8'.

Le résultat obtenu après l'analyse syntaxique est :

Open files list

- analyseur_lexical.c
- analyseur_lexical.h
- analyseur_semantique.c
- analyseur_syntaxique.c
- analyseur_syntaxique.h
- main.c

C:\Users\Asmaa\Desktop\projet_compilation\bin\Debug\projet_compilation.exe

```

BONJOUR
GUI
"
(PF
)
(PV
;
BEVET
BEVET
ID
PCT1
PO
(
(PF
)
(PV
;
KTER
KTER
PO
(
(
GUI
"
STRING
BONJOUR
GUI
"
PLUS
+
ID
X
PF
)
(PV
;
AF
)
BRAVO!!!

Process returned -1073741819 (0xC0000005)   execution time : 5.095 s
Press any key to continue.

```

Chapitre 4

Analyseur sémantique

4.1 Définition :

L'analyse sémantique d'un code est la phase de son analyse qui en établit la signification en utilisant le sens des éléments (tokens), par opposition aux analyses lexicales et syntaxique qui décomposent le code à l'aide d'un lexique ou d'une grammaire.

4.2 Les règles sémantiques

Pour que le code saisi par l'utilisateur soit correct et ait une signification, un certain nombre de règles et de contrôles doivent être respectés, ci-dessous l'ensemble de ces règles :

1. Les identificateurs utilisés dans le bloc DEREJ doivent être déjà déclaré.
2. Un identificateur ne doit pas être déclaré plus d'une fois.
3. Les constantes ne doivent pas être lues.
4. Nous ne pouvons pas changer les valeurs des constantes.
5. Dans les appels des fonctions, nous devons respecter le nombre des arguments fourni dans la déclaration.
6. Dans la déclaration d'un tableau, la taille fournie doit correspondre au nombre des valeurs insérées.
7. En affectant un élément à une case du tableau en utilisant la méthode de l'indice, ce dernier doit être inférieur à la taille du tableau.

4.3 Les fonctions utilisées au niveau de l'analyseur sémantique

La première tâche dans cette analyse consiste à créer la table des symboles, afin de rassembler l'ensemble des identificateurs.

Pour réaliser cette tâche, nous avons créer une structure qui comporte les informations des identificateurs. L'image ci-dessous montre la structure utilisée pour les éléments de la table des symboles :

```
struct Ident{
    char name[20];
    char value[20];
    char tidf[20];
    int taille_tab_fct;
    Ident *suivant;
};
Ident* chaine_tab_symb;
-
```

Figure 4.1 – La structure des éléments de la table des symboles

Dans le but d'assurer le bon fonctionnement de cet analyseur nous avons utilisé un ensemble des fonctions ci-après :

| | |
|-----------------------------------------------|------------------------------------------------------------------------|
| table_symb(Element*) | afin de créer la table de symbole. |
| insertion_tab_sym(char* ,char* ,char * ,int) | Afin de sauvegarder un identificateur trouvé dans la table de symbole. |
| SyntaxError() | Afin de générer les messages d'erreurs et sortir du programme. |
| check(Element*) | Afin de vérifier les règles sémantiques présentées ci - dessus. |

TABLE 4.1 – Tableau des fonctions utilisées dans le sémantique

Conclusion générale

L'objectif de ce projet était de concevoir un langage de programmation et de développer un analyseur lexical, syntaxique et sémantique de ce dernier.

D'abord, nous avons présenté notre langage BL MGHRIBI et sa grammaire, ensuite nous avons présenté l'analyseur lexical, syntaxique et sémantique et finalement nous avons montré quelques exemples d'exécution.

Ce projet nous a permis d'appliquer et de bien maîtriser ce qu'on a appris tout au long des séances des cours et des Tps.

L'amélioration que nous puissions appliquer sur ce projet c'est de réaliser la partie concernant la génération du code.

Bibliographie