REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

N° d'ordre : .....

UNIVERSITE DE M'SILA

FACULTE DES MATHEMATIQUES ET DE L'INFORMATIQUE

Département d'Informatique

MEMOIRE de fin d'étude

Présenté pour l'obtention du diplôme de MASTER

Domaine : Mathématiques et Informatique
Filière : Informatique
Spécialité : Systèmes d'Informations Avancés

Par : SAOUDI Azzouz

SUJET

Un compilateur des pages HTML pour la Génération automatique des codes JavaScript basant sur ANTLR

Soutenu publiquement le : 19 / 06 /2014 devant le jury composé de :

Mr .BOUDIA Malika	**Université de M'sila**	**Président**
Mr. MOKHTARI Rabah	**Université de M'sila**	**Rapporteur**
Mr. BOUBAKIRE Mohamed	**Université de M'sila**	**Examinateur**
Mr……	**Université de M'sila**	**Examinateur**

Promotion : 2013 /2014

Remerciements :

En tout premier lieu, je remercie Allah le tout puissant, et

Je tiens à remercier très sincèrement Mr MOKHTARI Rabah pour le grand honneur qu'il m'a fait en me proposant le sujet, pour ses conseils et pour ses orientations J'ai eu l'honneur et le privilège de travailler sous son assistance et de profiter de ses qualités humaines m'ont été d'une grande utilité, Son professionnelles et de sa grande expérience. il m'a guidé tout au long de ce travail.

Je remercie tous les enseignants de l'université de M'sila, et en particulier mes enseignants de nos départements informatique.

Mes remerciements vont également aux membres de jury d'avoir accepté de juger mon travail.

Je remercie vivement toute ma famille, en particulier mes parents, ma femme..., pour m'avoir toujours soutenu au cours de mes études. Qu'ils trouvent ici le fruit de leur patience et du soutien permanent qu'ils m'ont prodigué pour affronter tous les moments difficiles.

Mes sincères remerciements s'adressent également à mes amis DJENIDI Salah Eddine, BOUKHALAT Youssef, GHADBAEN Mebarek, et tous mes collègues.

Enfin, je remercie toutes les personnes qui de près ou de loin, ont contribue à sa réalisation de ce travail.

Dédicaces :

Ça serait indigne de ma part si j'oubliais de remercier les deux êtres qui ont fait que je sois ici aujourd'hui... eh oui, ma maman et mon père, vous qui m'avez conçu, élevé, éduqué...vous qui avez toujours été là pour moi et n'avez jamais cessé de croire en moi, aucun mot ni aucune langue ne pourrait exprimer ma profonde gratitude à votre égards.

A mes chers parents

Je vous dois ce que je suis aujourd'hui grâce à votre amour, à votre patience et vos innombrables sacrifices. Que ce modeste travail, soit pour vous une petite compensation et reconnaissance envers ce que vous avez fait d'incroyable pour moi.

Que Dieu, le tout puissant, vous préserve et vous procure santé et longue vie afin que je puisse à mon tour vous combler.

A mes très chers frères

Aucune dédicace ne serait exprimer assez profondément ce que je ressens envers vous, je vous dirais tout simplement, un grand merci, je vous aime.

A ma femme

A toute la famille et surtout mes oncle Gagui Mohamad et sa femme benAzi turkia.

A mes très chers amis

En témoignage de l'amitié sincère qui nous a liées et des bons moments passés ensemble. Je vous dédie ce travail en vous souhaitant un avenir radieux et plein de bonnes promesses.

saoudi azzouz

Table des Matières

Liste des figures

Liste des tableaux

Introduction générale

Introduction :

L'informatique a occupé, ces dernières années un rôle prépondérant dans les différentes branches de l'activité humaine

L'apparition des langages évolues (fortran, pascal, HTML, XML, PHP, C...ete.) et systèmes spécialisés (recherche documentaire, ...ete.) a ouvert le monde de l'informatique a différentes classes d'utilisateurs, et a facilité la communication :
homme- machine et machine- homme

Les formulaires dans une page HTML peuvent n'accepter pas que certains types d'entrées. L'envoi de formulaires peut n'être possible que lorsque certaines exigences sont respectées. Pour résoudre la relation étroite entre les formulaires dans la page HTML et code javascript pour valider les données entrés par un utilisateur.

Nous voyons que l'ensemble de codes utilisés, dans le monde de développement web, très semblable et peut être ré exploité, pour cela il est fortement utile d'adopter un mécanisme permettant de faciliter la réutilisation de codes dans le développement web et en fournissant d'outil facilitant sa génération automatique afin de profiter des espaces de point de vue du temps et d'efforts.

Le but majeur de ce travail est faciliter le travail l'utilisateur pour générer le code javascript à partir des pages uniquement HTML.

La réalisation de notre travail nous a amené à diviser l'étude en deux phases:

- **La phase théorique:** qui est consacré a la conception d'un compilateur pour générer le code JavaScript.
- **La phase pratique:** l'implémentation du logiciel.

Notre mémoire est organisé autour des chapitres suivants : Il commence par une introduction générale introduisant le domaine de développent web en général.

Chapitre I : Dans ce chapitre, nous esquissons la structure typique d'un compilateur, et Les principes de base inhérents à la réalisation de compilateurs : analyse lexicale, analyse syntaxique,…, et les outils fondamentaux utilisés pour effectuer ces analyses.

Chapitre II : Connaitre et utiliser langages du Web (e.g HTML, JavaScript, PHP...)

Chapitre III : Dans ce chapitre on procède au compilateur des pages html pour générer le code JavaScript basé sur ANTLR qu'on a baptisé HTML2js

Enfin une conclusion générale.

Chapitre 01 :

Introduction à la compilation

I. 1. Introduction:

Tout langage informatique structuré est en fait une notation permettant de décrire des expressions et des textes à l'intention des humains et des ordinateurs. Le monde que nous connaissons s'appuie sur des langages de différents types (Java, XML, HTML, UML…) pour différents objectifs. Souvent, on aura besoin d'aller d'un langage à un autre pour transformer, translater, ou traduire un texte d'une structure à une autre. La réalisation automatique d'une telle translation est appelé une compilation. Ainsi, l'outil logiciel qui exécute cette action est le compilateur.

Dans ce chapitre, nous esquissons la structure typique d'un compilateur et Les principes de base inhérents à la réalisation de compilateurs : analyse lexicale, analyse syntaxique,…, et les outils fondamentaux utilisés pour effectuer ces analyses.

I. 1.1 Définition :

Un compilateur est un programme qui traduit un programme écrit dans un langage source vers un langage cible en indiquant les erreurs éventuelles que pourrait contenir le programme source [3].

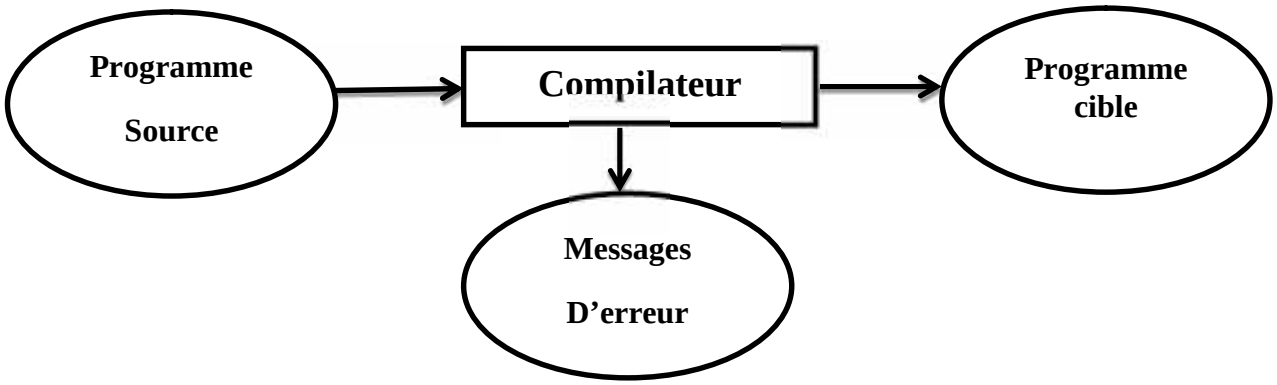


Figure I.1 : schéma d'un compilateur [1].

Les rôles essentiels de ce processus de compilation sont :

- ✓ signale au programmeur la présence d'erreurs dans le programme source.
- ✓ Rendre compréhensible par la machine un langage de haut niveau.

I. 2 Structure d'un compilateur :

I. 2.1 Structure logique:

Les phases de compilation : un compilateur est théoriquement constitué de six phases, formant un ensemble cohérent. Une décomposition typique d'un compilateur est présentée dans la figure I.2 ainsi que l'intersection entre ses phases.

❖ **Les phases d'un compilateur :**

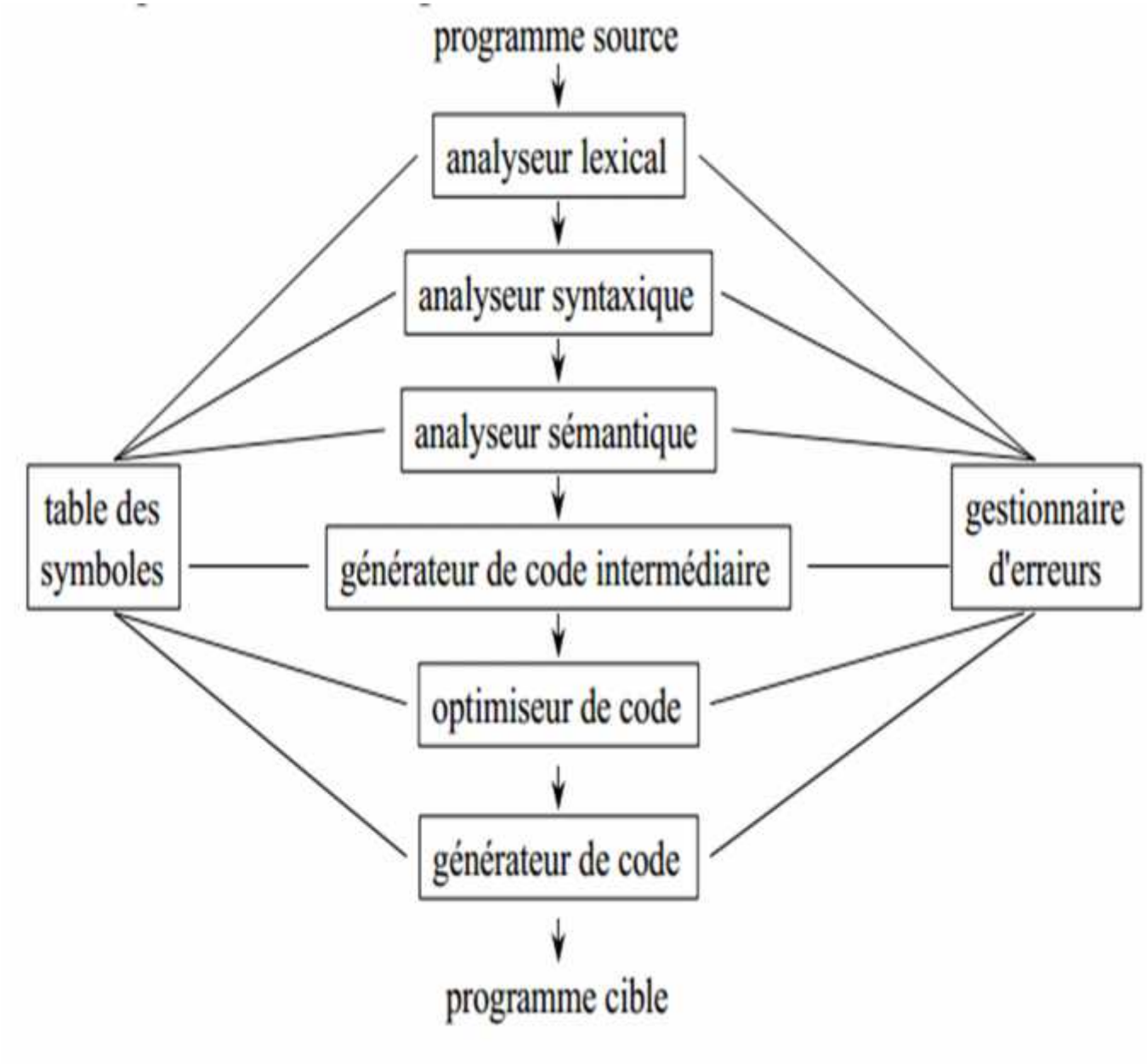


Figure I.2 : les différentes phases d'un compilateur [1].

a. analyse lexicale :

C'est phase de lecture et conversion du flot d'entrée le rôle de l'analyse lexicale :

- lire les caractères d'entrée
- réaliser un prétraitement du programme source.
- transmettre à l'analyseur syntaxique des unités lexicales.
- Initialiser la table des symboles.
- Garder un lien entre compilateur et utilisateur.

L'intérêt de l'analyse lexicale :

- Conception modulaire plus simple du compilateur.
- Simplification de l'écriture de l'analyseur syntaxique.
- Techniques spécifiques d'entrée du texte.
- Existence de techniques générales d'analyse lexicale.

Dans cette phase, les erreurs détectables sont par exemple une suite de caractères inconnue, constante grande, etc….

b. Analyse syntaxique (analyse grammaticale) :

Phase de vérification de la syntaxe du flot d'entrée.

Le rôle de l'analyse syntaxique :

- Vérifie la conformité syntaxique.
- Construit l'arbre d'analyse.
- Gère les erreurs communes de syntaxe.

Dans cette phase, les erreurs détectables sont par exemple

Une expression arithmétique mal parenthèse.

c. Analyse sémantique :

C'est la phase de vérification du sens des instructions, cette phase permet de préparer la phase ultime du compilateur qui est la génération du code.

d. génération du code intermédiaire :

Il utilise la structure produite par l'analyse syntaxique pour générer un ensemble d'instructions simples écrit dans un langage proche du langage objet.

e. **Optimisation du code :**

Cette phase est optionnelle, elle tente d'améliorer le code intermédiaire de façon que le code machine résultant s'exécute plus rapidement.

f. Génération du code objet :

Phase de transformation de chaque instruction du programme source en son équivalant en langage objet, celui-ci dépend de l'architecture de la machine sur laquelle le programme est appelé à être exécuté.

I. 2.2 Structure physique d'un compilateur :

Dans l'implémentation d'un compilateur, il arrive souvent que l'on regroupe les activités de plusieurs phases logiques en un modèle appelé passe et ce, dans le but d'accélérer le processus de compilation.

❖ **L'analyse du programme source**

Découpage en trois phases :
i) Analyse lexicale : flot de caractères regroupés en unités lexicales
ii) Analyse syntaxique : regroupement des unités lexicales en unités grammaticales
iii) Analyse sémantique : contrôle ou établissement de la cohérence sémantique

I. 3 Analyse lexicale (lexer):

I. 3.1 Le rôle de l'analyse lexicale :

L'analyseur lexical constitue la première phase d'un compilateur.la tâche principale d'un analyseur lexical est :

- Lire les caractères constituant le programme source.
- Produire en sortie (comme résultat) une suite d'unités lexicales.
- Le flot d'unités lexicales est envoyé à l'analyseur syntaxique.

Ces interactions sont représentées à la **figure I.3** [1]**.**

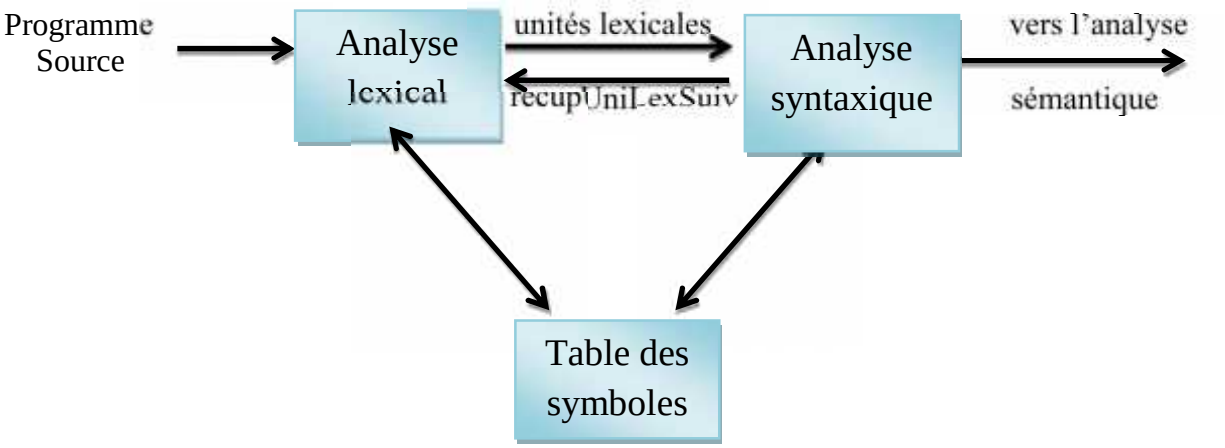


Figure I.3: Interactions entre analyseur lexical et analyseur syntaxique [1].

L'analyseur lexical réalise certaines tâches secondaires comme l'élimination de caractères superflus et des blancs (commentaires, tabulations, fin de lignes…).

Une autre de ces tâches est la corrélation des messages d'erreurs émis par le compilateur avec la source telles que :

- Caractère inconnu.
- Nombre très grand….

I. 3.2. Unités lexicales, modèle (motif) et lexèmes :

Définition :

a) **Lexèmes :** est une séquence de caractères dans le programme source qui concorde avec le modelé d'une unité lexicale [1].

Exemple : printf(X) ;

Printf : est un lexème.

'(' : est un lexème.

'X' : est un lexème.

')' : est un lexème.

b) **Unités lexicales :** est un couple constitué d'un nom d'unité lexicale et d'une valeur d'attribut optionnelle [1].

Exemple : Var : x,y,z ;

X,y,z :sont des lexèmes de l'unité lexicale(**identificateur).**

- Mots clés (if, elles, while…).
- Identificateurs (var, ind, toto…).
- Symboles des ponctuations.
- Constantes numériques.
- Constantes chaine.

c) **Modèle :** est une règle associée à une unité lexicale [1].

I. 3.3. Erreurs lexicale :

Il est difficile pour un analyseur lexical de dire, sans l'aide des autres composants, qu'il y a une erreur dans le code source. Par exemple, si la chaine **fi** est rencontrée pour la première fois dans un programme C

Fi (a==f(x))…

Un analyseur lexical ne peut pas dire si **fi** est une orthographe erronée du mot clé **if** ou un identificateur. Puisque **fi** est un lexème valide pour l'unité lexical **id**, l'analyseur lexical doit retourner L'unité lexicale **id** à l'analyseur syntaxique et laisser une autre phase du compilateur (probablement l'analyseur syntaxique, dans ce cas) gérer une erreur due à la permutation des deux lettres [1].

I. 4. Analyse syntaxique (Parser):

Tout langage de programmation possède des règles qui indiquent la structure syntaxique d'un programme bien formé [1].

I. 4.1 Rôle de l'analyseur syntaxique :

L'analyseur syntaxique reçoit une chaine d'unités lexicales, produites par l'analyseur lexical et vérifie que la chaine est conforme avec la syntaxe du langage à compiler et produit en sortie un arbre d'analyse.

Cette vérification est faite en s'appuyant sur la grammaire qui génère le langage.

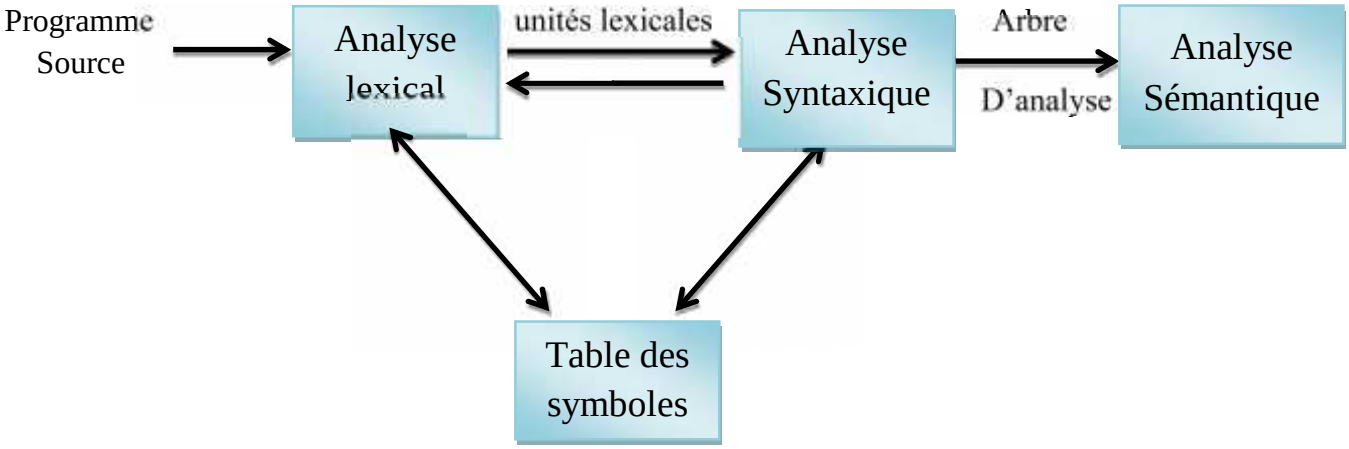
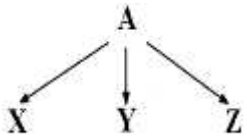


Figure I.4: place de l'analyseur syntaxique dans le modèle de compilateur [1].

❖ **Arbre d'analyse (syntaxique) :**

C'est une représentation graphique d'une dérivation. Chaque niveau de l'arbre correspondant à une dérivation et les feuilles représentent les terminaux [1].

❖ **Propriétés d'un arbre d'analyse :** A ⟶ X Y Z



1-la racine est étiquetée par l'axiome.

2-chaque feuille est étiquetée par un terminal ou par ε.

3-chaque nœud interne est étiqueté par un non-terminal [1].

I. 4.2 Méthode d'analyses (syntaxique) :

I. 4.2.1 Analyse descendant :

Ces méthodes peuvent être vues comme une tentative pour construire un arbre d'analyse de la chaine d'entrée, en partant de la racine(l'axiome)et en allant vers les feuilles en opérant par les dérivation gauche. Les méthodes les plus efficaces connus à l'heure actuelle sont [1]:

- La descente récursive.
- Analyse prédictive.

I. 4.2.2 Analyse ascendante:

Cette méthode a pour but de construire un arbre d'analyse pour une chaine source en commençant par les ficelles (le bas) et en remontant vers la racine (le haut). Ce processus peut être considéré comme la réduction d'une chaine vers l'axiome de la grammaire.

I. 5. Grammaires :

Les grammaires, ou plus précisément les grammaires non contextuelles (on dit parfois grammaire BNF (pour Backus-Naur form)[7], ou grammaires algébriques, constituent le formalisme essentiel pour décrire la structure des programmes dans un langage de programmation. En principe, la grammaire d'un langage ne décrit que la structure syntaxique, mais étant donné que la sémantique d'un langage est décrite en termes de la syntaxe, la grammaire est également pour quelque chois dans la définition de la sémantique [2].

I. 5.1 Forme d'une grammaire :

Une grammaire et formée d'un ensemble de règles de production et d'un symbole de départ. Chaque règle de production définit une construction syntaxique nommée. Une règle de production est formée de deux parties, la partie gauche et la partie droite, séparées par une flèche à droite [4].

a. **Partie gauche :** La partie gauche est le nom de la construction syntaxique.

b. **Partie droite :** La partie droite donne une forme possible de la construction Syntaxique, Voici un exemple de production :
Expression →'(' expression opérateur expression ')'

- **Symbole terminal et non-terminal :**
  La partie droite de la production contient des symboles de deux sortes, des symboles terminaux et des symboles non-terminaux [2].
- **Symbole terminal (lexèmes) :**
  Est un point final d'un processus de génération et peut faire des chaines produites par la grammaire [2].
  - La chaine vide et notée par ε.

❖ **Symbole non-terminale :**

Doit apparaître comme la partie gauche (le nom) d'au moins une production, et ne peut pas faire parties des chaines produite par la grammaire [2].

❖ **Conventions de notation :**

➢ Les symboles suivent sont des terminaux [1]:

- Les lettres minuscules du début de l'alphabet, telles que a, b et c.
- Les symboles d'opérateurs tels que +,*,…, etc.
- Les signes de ponctuation tels que les parenthèses, la virgule, etc.
- Les chiffres 0,1, …,9.

➢ Les symboles suivants sont des non-terminaux [1]:

- Les lettres majuscules du début de l'alphabet, telles que A, B et C.
- La lettre S, qui, quand elle est utilisée, est généralement le symbole de départ.
- Les mots en minuscules et en italique tels qu'expr ou instr.
- Pour parle de constructions de langages de programmation, on pourra utiliser des lettres majuscules pour représenter des non-terminaux dénotant ces constructions. Par exemple, les non-terminaux dénotant les expressions, les termes et les facteurs sont représentés par E, T et F.

I. 5.2 Processus de dérivation :

Soit la grammaire G donnée par :

E → E * E /E+E /id

On dit que e dérive la chaine « id+id*id » notons ceci par :

E → **E * E**
E→ **E + E** * E
E→ **id** + E * E
E → id + **id** * E
E → id + id * **id**

Figure I.5 : Dérivation gauche de la chaine id *id +id [4].

L'opération inverse de dérivation est appelée réduction, lors de la dérivation, on peut dériver gauche ou droite.

I. 5.3 Formes étendues de grammaires :

La notion simple pour les productions

Non-terminal→zéro, un ou plusieurs symboles grammaticaux

Que nous avons utilisée précédemment permet en principe de spécifier n'importe quelle grammaire, mais en pratique, on utilise une notation plus riche. Tout d'abord, on à l'habitude de combiner toutes les productions qui ont la même partie gauche en une seule règle [2].

Par exemple, les productions

$N \rightarrow \alpha$
$N \rightarrow \beta$
$N \rightarrow \gamma$

Sont combinées en une seule règle : $N \rightarrow \alpha \mid \beta \mid \gamma$

I. 5.4 Propriétés des grammaires :

- Un non-terminal N est **récursif à gauche** si, à partir du syntagme N, nous pouvons produire un autre syntagme commençant par N.

  Exemple : E ➔ E '+' facteur | facteur

  La récursivité à droite existe aussi, mais elle est moins importante.

- Un non-terminal N est « nullifiable » si, à partir de syntagme N, on peut dériver un syntagme vide.

  Exemple : E ➔ ε

- Un non-terminal N est **inutile** s'il ne permet pas de dériver une chaine de terminaux :

  Un exemple simple est

  E➔ '+' E | '-' E

Un logiciel de traitement de grammaire doit vérifier la présence de non-terminaux inutiles, et rejeter la grammaire s'il en trouve.

- Une grammaire est ambiguë si elle permet de produire deux arbres de dérivation différents, Avec les mêmes feuilles dans le même ordre [2].

Exemple :

Soit la chaine suivent : id *id+ id peuvent correspondre plus d'un arbre [4].

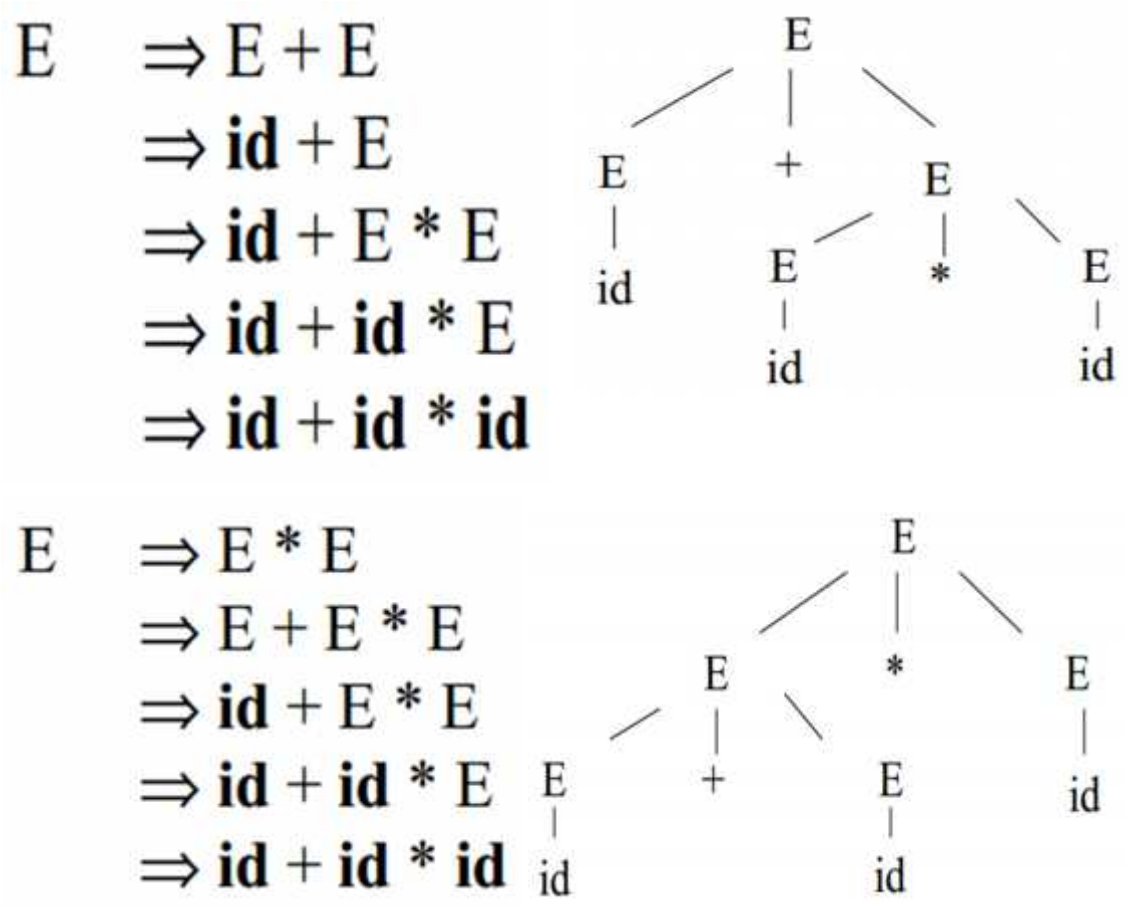


Figure I.6 : Deux arbres de dérivation pour id +id * id [4].

I. 5.5 Définition d'une grammaire :

L'unité de base des grammaires formelles est le symbole (identificateurs, lettres simples,…).

La prochaine unité de construction des grammaires formelles est la production (règle). étant donné deux ensembles de symboles V_N et V_T, une production est le couple

(N,α)tel que $N \in V_N$, $\alpha \in V_T$

La production comme le couple (N,α) mais plutôt comme N→α.

Une grammaire G est donnée par Un quadruplet G= (V_N, V_T, S, P), où:

- V_T est un ensemble des symboles terminaux, notés a, b, etc.
- V_N est un ensemble de symboles non-terminaux, notés A, B, etc.
- S ∈ V est le symbole de départ(ou **axiome**).
- P est un ensemble de productions (règle de langage) de la forme A → w, où w dénote un mot sur l'alphabet $V_N \cup V_T$. [2]

Exemple de grammaire algébrique:

- V_T= {int, (,), +, -, *, /}.
- V_N = {E}.
- S = E.
- L'ensemble P des productions est :

E → E + E

E → E / E

E → E – E

E → (E)

E → E * E

E → int

I. 6. LES OUTILS LEX & YACC :

LEX (LEXical parser) &**YACC** (Yet Another Compiler-Compiler) sont des outils qui engendrent des programmes d'analyse de texte. Les programmes générés offrent des fonctionnalités de reconnaissance, de structuration, de traduction d'un texte écrit dans un langage donné [5].

Le schéma d'utilisation de ces outils est montré à la **figure I.7**.

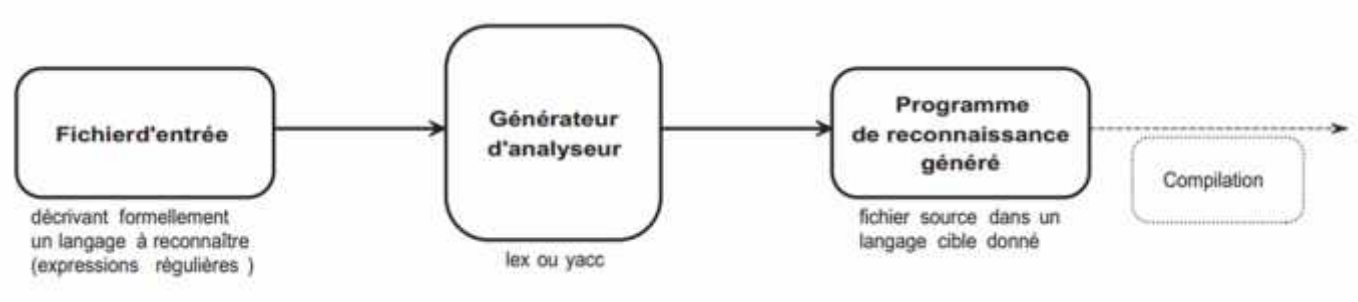


Figure I.7: principe d'utilisation des utiles Lex & yacc [5].

Les deux outils sont complémentaires

I. 6.1 Lex, un générateur d'analyseurs lexicaux :

Cet outil permet de définir un analyseur lexical en spécifiant des expirations régulières [1].

Exemple 1:

L'entrée "1.2345E-10" est une constante réelle syntaxiquement conforme à l'Expression Régulière {+|-} ? {0..9} * . {0..9} * E {+|-} ? {0..9} *

(Le ?signifie "0 ou 1 fois", le*"0 ou n fois").

- **La structure du fichier d'entrée pour Lex**

Le nom du fichier Lex est suffixé par **.l** : exemple**.l**.

On trouve trois sections dans un fichier Lex :

Section des définitions
%%
Section des règles
%%
Section définie par l'utilisateur

Exemple:

```
[a-z A-Z_]([a-z A-Z0-9_]) * {cout<< "C'est un identificateur\n";}
[0-9]+                    {cout << "C'est un entier\n";}
```

➢Dans la section des définitions peuvent apparaître des macro-définitions (on dit "macros") qui servent à nommer des expressions pour simplifier les définitions des Expressions Régulières [5].

Exemple : les expressions ci-dessus peuvent être simplifiées en utilisant les macros suivantes:

```
LETTRE [a-z A-Z_]
CHIFFRE [0-9]
%%
{LETTRE}({LETTRE}|{CHIFFRE} )* { cout<< "Identificateur\n"; }
{CHIFFRE}+                     { cout<< "Entier\n"; }
```

➢Dans la section définie par l'utilisateur, on place des variables, fonctions ... nécessaires au fonctionnement du programme [5].

- ✓ Lex fait l'analyse lexicale,
- ✓ Yacc fait l'analyse syntaxique.

Les applications principales de Lex sont :

- ➢ Analyse statique: vérifications diverses.
- ➢ Formateur: mise en page de texte.
- ➢ Traducteur /compilateur.
- ➢ Interprète.

I. 6.2 Yacc, un générateur d'analyseurs syntaxiques :

Le principe de Yacc est le même que celui de Lex, Mais l'outil est plus puissant que Lex dans la mesure où il permet de générer des analyseurs de langages descriptibles par des grammaires libres de contexte (contexte-free ou algébriques). Ces grammaires sont constituées de règles de la forme [5]

A (non terminal) → w.

Exemple 1 : l'entrée "((a+b)+c)" est un mot du langage :

S → Expr

Expr → (Expr) / Expr op Expr / Ident

Op → + / -

Ident → a / b / ... / z

- **La structure du fichier d'entrée pour yacc**

Le format d'un fichier Yacc exemple**.y** est le suivant :

Section des définitions
%%
Section des règles
%%
Section définie par l'utilisateur

➢Dans La section des définitions permet de placer des déclarations utiles au programme ou à la grammaire [5].

➢Dans La section des règles permet d'écrire les règles de production de la grammaire du langage à reconnaître [5].

La forme générale d'une règle est :

Non-Terminal : Liste_de_Symboles (Terminaux ou non) ;

Exemple : A : B C D ;

➢Dans la section définie par l'utilisateur, il faut fournir le programme principal qui appelle le parser c'est-à-dire l'analyseur engendré par Yacc; une procédure yyparse() sans argument est générée.

- **Pourquoi utiliser des générateurs ?**

Pour se concentrer sur la forme des expressions à analyser sans se soucier des détails d'implantation. Cela évite l'écriture manuelle d'un analyseur [5]:

➫ Avec les risques d'erreurs.

➫ Avec le risque d'écrire un analyseur difficile à maintenir, à faire évoluer.

➫ Avec les aspects pénibles propres aux entrées/sorties d'un langage particulier.

❖ **Les avantages de ces outils automatiques sont :**

➪ Simplicité d'utilisation: le style est déclaratif, c'est-à-dire basé uniquement sur la description formelle du langage à reconnaître.

➪ Facilité à maintenir: on peut à tout moment enrichir la description du langage.

➪ Facilité d'introduire des actions/traitements à effectuer pour engendrer un traducteur:

On sépare facilement les règles d'une grammaire et les actions sémantiques éventuelles associées à chacune des règles.

I. 6.3 ANTLR (ANother Tool for Language Recognition):

ANTLR est un outil pour la reconnaissance de langage. C'est un compilateur de compilateur, c'est-à-dire un programme informatique capable de produire certaines parties du compilateur (analyse lexicale et analyse syntaxique). ANTLR accepte un langage source et crée un langage cible composé des parties d'analyse lexicale et syntaxique [7].

I.7 Conclusion :

Un compilateur opère en une séquence de phases transformant chacune le programme source depuis une représentation intermédiaire vers une autre.

L'analyse lexicale parcourt le programme source caractère par caractère et produit en sortie une séquence d'unités lexicales qui sont en principe transmises l'une après l'autre à l'analyse syntaxique.

L'analyse syntaxique prend en entrée des unités lexicales issues d'une analyse lexicale et traite les noms de l'unité lexicale comme des symboles terminaux d'une grammaire. Une grammaire est définie par un ensemble de symboles terminaux et un autre de symboles non-terminaux.

Chapitre 02 :

Développement Web

II 1. Introduction :

HTML (HyperText Markup Langage) [8] **:** est un langage de balises (ou "tags") utilisés pour définir les différents composants d'un document, Chaque balise est encadrée par les symboles < et >. Chaque mise en forme est présentée par une balise de début **<balise>**et une balise de fin **</balise>**, suffixé par **.html** ou **.htm.**

Les documents HTML ne servent en général qu'à présenter de l'information à l'utilisateur. Rien n'était a priori prévu au départ pour que ce dernier puisse transférer de l'information dans "l'autres sens" (des utilisateurs vers le serveur de documents).

Il existe une technique permettant une communication du client (utilisateur) vers le serveur : le formulaire. Associé à des scripts CGI (Common Gateway Interface), scripts JavaScript ou PHP (Personale Home Page), le formulaire permet une certaine interaction entre l'utilisateur final et le Serveur de données grâce à des zones de saisie, boutons,…

- ✓ **La page HTML minimum :**

Une page HTML est un fichier texte commençant par la balise <HTML> et finissant par la balise </HTML>. Elle contient également un en-tête décrivant le titre de la page, puis un corps dans lequel se trouve le contenu de la page.

- L'en-tête est délimité par les balises <HEAD> et </HEAD>.
- Le corps est délimité par les balises <BODY> et </BODY>.

Ainsi la page HTML peut être représentée comme suit [10]:

```
<HTML>
        <HEAD>
                <TITEL> Le titer </TITLE>
        < /HEAD>
     <BODY>
              Contenu de la page
     < /BODY>
</HTML>
```

II 1.1 Les formulaires HTML

II 1.1.1 Définition d'un formulaire :

Avant de pouvoir utiliser les différentes sortes de formulaires (ligne de texte, liste déroulante, cases à cocher...), il faut déclarer au browser qu'il devra gérer des formulaires et ce qu'il devra en faire [9].

<FORM method="post" action="URL d'expédition" >
... les formulaires proprement dit ...</FORM>

Un formulaire est défini par les balises <form> et </form>. Deux paramètres doivent en outre être définis à l'ouverture du formulaire

Action: adresse d'envoi du formulaire

Method: la méthode de transmission des données (get ou post).

La méthode get s'est établie comme un standard mais ne convient pas au transfert de grandes quantités de données : il faut alors utiliser la méthode post [8].

II 1.1.2 Les éléments d'un formulaire :

Trois catégories :

- input : champs de saisie de texte et divers types de boutons :
  - type="text" : zone de texte
  - type="password" : zone de texte caché
  - type="checkbox" : cases à cocher
  - type="radio" : minimum 2, un seul sélectionnable
  - type="submit" : bouton de soumission du formulaire
  - type="reset" : bouton de remise à zéro des champs
  - type="hidden" : bouton caché
- select : menus déroulant, listes à ascenseurs
  - size="1" : un seul élément sélectionnable
  - size="n", n > 1 : liste à choix multiples
- textarea : zone de saisie d'un texte long.

1) L'élément INPUT :

Type	Code	Résultat
Sans	<input name= "ident" >	
	<input name="ident" value="par défaut">	Par défaut
Submit	<input type="submit" value="Envoyer" name= "Envoyer" >	Envoyer
Checkbox	<input type="checkbox" name= "pfm[]" value="linux" checked="checked" > Linux</br> <input type="checkbox" name="pfm[]" value="Dos" >Dos </br> <input type="checkbox" name="pfm[]" value="win" >Windows </br>	Linux Dos Windows
Radio	<input type="radio" name= "media" value="cd" checked="checked">CD-ROM</br> <input type="radio" name="media" value="dk" >Disquette</br>	CD-ROM Disquette
Password	<input type="password" name="pass" size="4" >	••••
Reset	<input type="reset" value="Effacer" >	Effacer

Tableau II.1: Les éléments d'un formulaire « input »

2) **L'élément SELECT :** Cet élément sert à définir des listes (menus déroulants ou ascenseurs). Il est utilisé avec l'élément **<option>.**

Code	Résultat
<select name="menu"> <option value="pomme"> Pomme</option> <option value="Banane"> Banane</option> <option value="orange"> Orange</option> <option value="citron" selected="selected"> Citron</option> <option value="pêche"> Pêche</option> <option value="poire"> Poire</option> </select>	
<select name="menu" size="4"> … </select>	

Tableau II.2: Les éléments d'un formulaire «select »

3) **L'élément TEXTAREA :** Il permet de créer une zone de texte de plusieurs lignes. Il faut spécifier sa taille avec les attributs rows et cols.

Code	Résultat
<textarea name="comm" rows="10" cols="40"> Tapez vos commentaires ici </textarea>	

Tableau II3 : Les éléments d'un formulaire « textarea»

II 1.2 Saisie de données dans le formulaire :

Il existe différents types de données que l'utilisateur est amené à saisir dans un formulaire : texte libre sur une ou plusieurs lignes, choix entre différentes options listées, etc. [8]

II 1.2.1 Zones de saisie monolignes :

Ce type de zone de saisie est créé grâce à la balise <input>. Chaque zone de saisie doit avoir un nom unique !

❖ **Définition :**

INPUT type="text" indique un champ de saisie d'une seule ligne. C'est assurément le formulaire le plus simple à mettre en œuvre [9]:

<form action="http://perso0.free.fr/cgi-bin/form2mail.pl" method=post>
Nom :< input name="nom" size=15 maxlength=20> </form>
Les attributs de la balise <input> sont :

Size: taille de la zone de saisie

Maxlength: taille maximale acceptée

Name : nom du formulaire" est quasiment obligatoire car on n'utilise que rarement un seul formulaire.

type :

- int : saisie de nombres entiers
- float : saisie de nombres décimaux acceptée
- date : saisie de dates
- url : saisie d'une adresse internet
- password : les caractères saisis ne sont pas visibles et sont remplacés par des astérisques

II 1.2.2 Zones de saisie multilignes :

Les zones de saisie multilignes se réalisent à l'aide de la balise <textarea> et se terminent par </textarea> [8].

Là aussi, chaque zone doit avoir un nom unique !

<form action="http://perso0.free.fr/cgi-bin/form2mail.pl" method=post>
Votre opinion :< textarea name="opinion" cols=15 rows=20>< /textarea> </form>
Les attributs acceptés sont :

Cols: nombre de colonnes

Rows: nombre de lignes

II 1.2.3 Liste d'options :

La balise <SELECT></SELECT> indique au browser l'usage d'une liste déroulante. . Les éléments de la liste sont introduits par la balise <OPTION> ... </OPTION> [8].

```
<form action= "http://perso0.free.fr/cgi-bin/form2mail.pl"  method=post>
<select name="Level" size="1">
<OPTION>lundi
<OPTION>mardi
<OPTION>mercredi
<OPTION>jeudi
<OPTION>vendredi
</select>
</form>
```

Si vous cliquez sur la petite flèche vers le bas, vous obtiendrez la liste déroulante où on retrouve les éléments de la liste (<OPTION>).

- size="x" : détermine le nombre d'éléments de liste affiché dans la boite d'entrée.

II 1.2.4 Boutons :

HTML fournit des boutons de formulaires que l'utilisateur peut activer en cliquant dessus. En réalité, il s'agit d'une forme dérivée de zones de saisie définie par la balise <input> [9].

a. Boutons d'option :

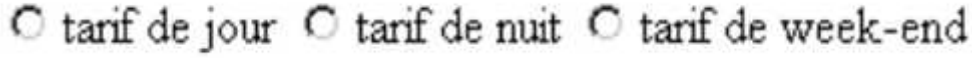
Les boutons d'option, aussi appelés boutons radio, ont comme particularité qu'une seule option à la fois peut être activée (le "ou" exclusif).

✓ **La syntaxe de base est :**

```
<FORM>
<INPUT type="radio" name="nom du groupe" value="valeur du bouton">
</FORM>
```

❖ Exemple :

```
<FORM>
<INPUT type= "radio" name="tarif" value="jour"> tarif de jour
<INPUT type= "radio" name="tarif" value="nuit"> tarif de nuit
<INPUT type= "radio" name="tarif" value="week-end"> tarif de week-end
</FORM>
```



b. Case à cocher :

La philosophie des cases à cocher [checkbox] est assez similaire aux boites d'option. Ici, cependant, plusieurs choix simultanés peuvent être réalisés [9].

✓ **La syntaxe de base est :**

```
<FORM>
<INPUT type="checkbox" name="nom" value="valeur attachée au bouton">
</FORM>
```

❖ **Example :**

```
<FORM>
<INPUT type="checkbox" name="choix1" value="1"> glace vanille
<INPUT type="checkbox" name="choix2" value="2"> Chantilly
<INPUT type="checkbox" name="choix3" value="3"> chocolat chaud
<INPUT type="checkbox" name="choix4" value="4"> biscuit
</FORM>
```

c. Boutons de commande

Pour terminer un formulaire, il faut offrir à l'utilisateur la possibilité d'envoyer le formulaire ou bien de l'annuler. Les boutons de commande sont eux aussi dérivés d'une zone de saisie et leur syntaxe est (cg. Exemple) [9]:

Bouton d'annulation: `<input type=reset name= [nom]>`

Bouton d'envoi : `<input type=submit name= [nom]>`

- **Submit et Reset :**

a. **Submit :** Le bouton Submit a la tâche spécifique de transmettre toutes les informations contenues dans le formulaire à l'URL désignée dans les attributs ACTION et METHOD du tag <FORM>.

b. **Reset :** Le bouton Reset permet d'annuler les modifications apportées aux contrôles d'un formulaire et de restaurer les valeurs par défaut.

Exemple : exemple de boutons de commandes

```
<div align="center">
<input type="submit" value="Valider" name="submit">
</div>
</td>
<td width=50%>
<div align="center">
<input type="reset" value="Effacer" name="Clear">
</div>
```

II 2. Introduction aux JavaScript :

Les formulaires HTML sont également très intéressants pour JavaScript. Leurs options sont quelque peu limitées, mais JavaScript apporte son aide.

Les données utilisateur devront être validées, les formulaires peuvent n'accepter pas que certains types d'entrées, l'envoi de formulaires peut n'être possible que lorsque certaines exigences sont respectées, etc. Tout cela, et bien d'autres choses, est rendu possible par JavaScript [13].

II 2.1 Définition :

Javascript est un langage de scripts qui incorporé aux balises Html, permet d'améliorer la présentation et l'interactivité des pages Web.
JavaScript est donc une extension du code Html des pages Web. Les scripts, qui s'ajoutent ici aux balises Html, peuvent en quelque sorte être comparés aux macros d'un traitement de texte.

Ces scripts vont être gérés et exécutés par le browser lui-même sans devoir faire appel aux ressources du serveur. Ces instructions seront donc traitées en direct et sans retard par le navigateur.

JavaScript a été initialement développé par Netscape et s'appelait alors LiveScript. Adopté à la fin de l'année 1995, par la firme Sun (qui a aussi développé Java), il prit alors son nom de JavaScript [14].

Avec du JavaScript, on peut utiliser les formulaires pour transférer des informations à l'intérieur d'une page ou même à l'intérieur d'un site. En outre, JavaScript, propose des outils particulièrement adaptés pour la vérification des données introduites par l'utilisateur dans les formulaires avant l'envoi et le traitement de celles-ci [9].

Pour faire sa en utiliser quelle que exemple pour vérifier et valider les champs d'un formulaire comme (vérification la longueur de input, le type, la forme, ect...).

II 2.2 Vérification et Validation de formulaire :

JavaScript est un très bon outil pour valider les formulaires. Il permet de communiquer des commentaires immédiats aux utilisateurs sur les données saisies. Ainsi, avant d'utiliser les données du client, validez-les également sur le serveur [15].

Exemple 1 :

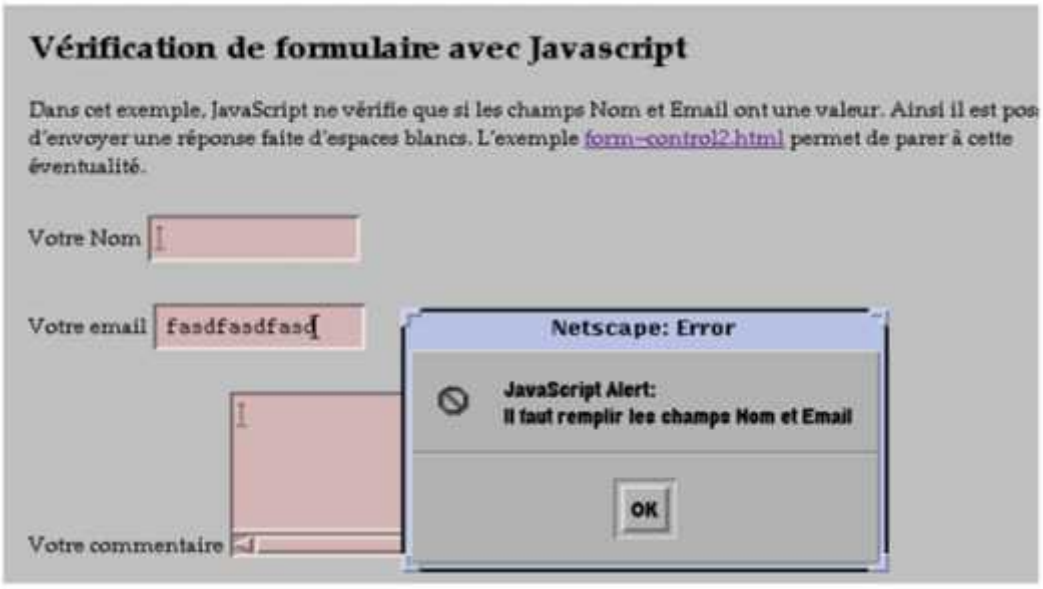


Figure II.1: Vérification de formulaire avec JavaScript. [15]

Cet exemple reprend le formulaire discuté dans le module "Les formulaires HTML" et y ajoute deux fonctions JavaScript permettant de vérifier que l'utilisateur n'a pas laissé de champs vides.

• Beaucoup de sites utilisent JS pour vérifier un formulaire avant de l'envoyer à un PHP ou CGI pour traitement [15].

Soit le code suivant qui affiche un message d'alert si l'utilisateur n'est remplié pas un champ d'un formulaire.

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function checkBlank(input,msg)
{
if (input.value == null || input.value.length == 0) {
alert ("Il faut remplir les champs Nom et Email");
return false;
} return true; }
function checkForm(form)
{
if (
!checkBlank(form.nom) ||
```

```
!checkBlank(form.email)) {
return false;
}
form.submit();
alert ("Merci pour votre reponse ...");
return true;
}</SCRIPT> </HEAD> <BODY>
<h1>Vérification de formulaire avec Javascript</h1>
<form enctype="application/x-www-form-urlencoded"
action="mailto:Patrick.Jermann@tecfa.unige.ch" method=post>
Votre Nom <input type="text" name="nom" size=15> <p>
Votre email <input type="text" name="email" size=15> <p>
Votre commentaire <textarea name="comment" rows=5 cols=30></textarea><p>
<input type="button" value="Envoi" onClick="checkForm(this.form)">
<input type="reset" value="Effacer">
</form>
</BODY></HTML>
```

- Lorsque l'utilisateur clique sur le bouton "envoi" la fonction 'checkForm' est appelée avec le contenu du formulaire en argument: checkForm(this.form)
- La fonction "checkForm" appelle la fonction "checkBlank", qui vérifie:

  - (1) est-ce que l'utilisateur a tapé quelque chose.

  - Si la question (1) reçoit une réponse positive, alors la fonction checkBlank retourne la valeur true.

Figure II.2: Vérification si l'utilisateur n'est remplier pas un champ d'un formulaire.

Figure II.3: Vérification si l'utilisateur remplier tout les champs d'un formulaire.

Exemple 2 :

Vérification la longueur : Autre exemple qui contrôler que le contenu d'un champ est d'une longueur minimale.

La fonction alert() affiche une boîte de dialogue comportant un message d'alerte. C'est la propriété length du contenu du champ qui est testée. La fonction renvoie un booléen car elle est utilisée comme valeur de l'attribut ansubmit du formulaire.

```
<SCRIPT LANGUAGE="JavaScript">
<!-- function minimum(champ, nbr)
{ if (champ.length<nbre) {
alert ("Le mot de passe doit compter "+nbre+" caracteres au minimum.");
return false;
}  return true; }
--> </SCRIPT>
Au niveau du formulaire, on aura :
<form name="form1" action="../exemples/ok.php" onsubmit="return
minimum(this.mp.value, 6)" method=post>
<p>Identificateur
    <input type="text" name="Id" id="id"> </p>
<p> Mot de passe
    <input type="password" name="mp" id="mp"> </p>
<p> <input type="submit" name="envoi" id="envoi" value="Envoyer"></p>.
```

II.3 Conclusion :

La création d'un formulaire nécessite la connaissance de quelques balises HTML indispensables (Structure, Champ de saisie de text, les Boutons….).Les formulaires sont le moyen le plus pratique pour le visiteur de transmettre des informations à votre site.

Avant de transmettre les données d'un formulaire d'un client il est très important de les vérifier au niveau du navigateur avant les envoyer au serveur pour des raisons de diminuer la charge sur le serveur. Pour cet objectif s'introduit le langage JavaScript.

JavaScript est plus simple à mettre en œuvre car c'est du code que vous ajouterez à votre page écrite en Html.

Chapitre 03 :

Génération des codes JavaScript

III.1 Introduction :

Durant ce projet, nous avons travaillé sur la conception d'un compilateur, générant code javascript. Nous avons utilisé ANTLR pour générer (**Parser** et **lexer)** en java, à partir d'une grammaire proposée permettant d'utiliser le langage HTML en y ajoutant d'autres types de paramètres guidant la génération de code javascript.

Après l'élaboration de quelques généralités sur la notion d'analyseur, nous aborderons les règles d'écriture de grammaires sous ANTLR. Ensuite, nous verrons comment définir et générer des analyseurs à partir d'une grammaire. Nous terminerons par un exemple qui illustrera l'utilisabilité de notre démarche.

III.2 ANTLR (ANother Tool for Language Recognition):

ANTLR (un autre outil pour la reconnaissance des langues). Il utilise les grammaires de type LL (k) [17] pour générer des analyseurs syntaxique et lexicale de langage et qui fournit un cadre pour la construction des compilateurs, et des traducteurs de descriptions grammaticales contenant des actions dans une variété des langages cibles (c#, java ...).

ANTLR (Un générateur d'analyseurs) lit une grammaire et génère une *Recognizer* de la langage définie par la grammaire (c'est à dire un programme qui lit un flux d'entrée et génère une erreur si le flux d'entrée n'est pas conforme à la syntaxe spécifiée par les règles de cette grammaire) [18].

En plus du générateur d'analyseur, ANTLR fournit d'autres fonctionnalités y afférentes telles que la construction d'arbres, l'insertion des actions dans les règles de grammaire, la gestion d'erreurs et le débogage. Il existe un environnement d'utilisation graphique appelé AntlrWorks. ANTLR et AntlrWorks sont libres et open source.

ANTLR fournit le compilateur avec le soutien de lexer, parser, et les étapes de l'analyseur d'arbres (tree parser). Ces trois étapes sont réalisées grâce à la surcharge de trois modèles de classe: Lexer, Parser, et TreeParser.

Le cadre général d'ANTLR est représenté sur la figure III.1

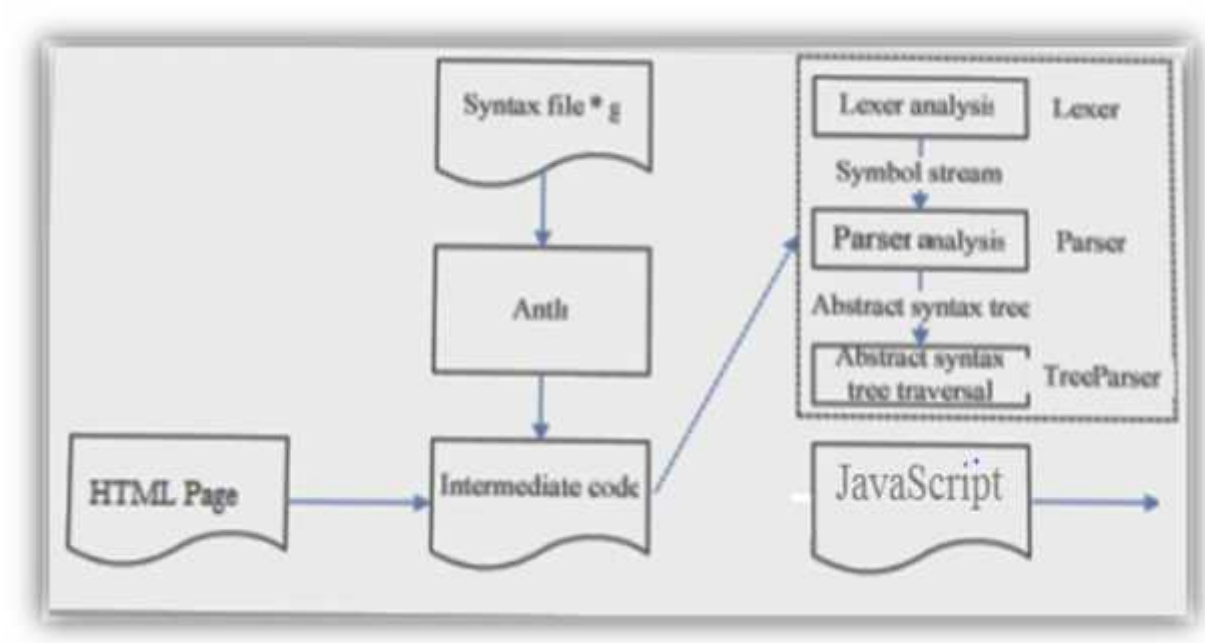


Figure III.1 : Le cadre global d'ANTLR.

❖ Que peut-on réaliser avec ANTLR ?

Antlr a trois cas d'utilisation principaux. Il peut être utilisé pour implémenter :

- **un validateur ou *recognizer*.** Antlr génère le code qui permet de valider si le texte d'entrée respecte les règles de grammaire
- **un processeur.** Antlr génère du code qui valide et traite le texte d'entrée. La grammaire intègre des actions qui peuvent être des instructions de calcul, des accès à une base de données, etc. Cependant, il n'y a pas de règle de réécriture ;
- **un traducteur.** Antlr génère le code qui valide et traduit le texte d'entrée dans un autre format. Par exemple, il peut servir à traduire un texte écrit dans un langage en un texte écrit dans un autre langage. Ici, on retrouvera des actions, notamment les instructions d'affichage (println).

III.2.1 Les fichiers générés :

Antlr génère un ensemble de fichiers. Il s'agit notamment :

- d'une classe pour le parser.
- d'une classe pour le lexer.
- d'une ou plusieurs classes pour les Tokens générés XXXTokensType.java.

III.2.2 Les avantage d'Antlr :

- ✓ Avec ANTLR vous pouvez spécifier votre compilateur et laisser ANTLR faire le travail difficile de générer le compilateur.
- ✓ ANTLR peut générer de reconnaissance pour de nombreux langages de programmation (par exemple Java, C #, Python, etc).
- ✓ ANTLR est bien pris en charge et a une communauté active d'utilisateurs.
- ✓ Fichier unique pour lexer et parser.

III.3 Ecriture de la grammaire :

L'écriture d'une grammaire sous Antlr obéit à un formalisme qui doit être respecté aussi bien en ce qui concerne la structure du fichier en ce qui concerne l'écriture des règles. Pour un débutant, il est conseillé de s'inspirer d'une grammaire existante.

III.3.1 Structure d'un fichier grammaire ANTLR :

❖ **La syntaxe d'une grammaire Antlr :**

La grammaire Antlr content quatre partie (champs) **:**

- ➢ En-tête :(nom de la grammar, option)
- ➢ Tokens: (spécifie les mots-clés)
- ➢ Parser Rules (définition des règles de la grammaire)
- ➢ Lexer Rules

La structure générale est la suivante :

Grammar *name* (nom de la grammaire);

options { *name1* = *value*; *name2* = *value2*; . . . }

tokens { *token-name1*; *token-name2* = *value*; . . . }

Parser Rules
<nom_regle**1**>: corps de la règle**1**;
...
...
<Nom_regleN>: corps de la règle **N**;

Lexer Rules
<nom_regle**1**>: corps de la règle**1**;
...
...
<Nom_regleN>: corps de la règle **N**;

Voici un exemple qui respecte la syntaxe d'une grammaire:

```
Grammar HTML;

Options {
   Language=Java;
}
/*------------------------------------------------------
 * PARSER RULES
 *------------------------------------------------------*/
Exemple:
Page_HTML : Element*
 Element:
            (LTAG | ENDTAG | name_elements+ parameters* (RTAG | SPTAG)) comment*
            ;
 /*------------------------------------------------------
 * LEXER RULES |
 *------------------------------------------------------*/
Exemple:
INTEGER: ('0'..'9') +;
STRING: "" (ID |~ ('\\' | ""))* ""
            ;
ID: ('a'..'z' | 'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'0'..'9'|'_')*;
WS: (' '|'\t'|'\n'|'\r') + ->skip;
```

III.3.2 Les multiplicités:

Dans le corps d'une règle, des formalismes sont utilisés pour caractériser la multiplicité d'une expression dans une construction. Ces formalismes sont :

- **(expression) *** : **zéro ou plus.** Ceci signifie que l'expression entre parenthèses peut être rencontrée zéro ou plusieurs fois dans la construction à reconnaître ;
- **(expression) +** : **un ou plus**. Ceci signifie que l'expression entre parenthèses peut être rencontrée une ou plusieurs fois dans la construction à reconnaître ;
- **(expression) ?** : **zéro ou un**. Ceci signifie que l'expression entre parenthèses peut être rencontrée une fois ou ne pas du tout être rencontrée dans la construction à reconnaître.

Exemple de règle avec multiplicité :

```
Nombre_decimal:
        ('0'..'9')+ (',' ('0'..'9')*)?
        ;
```

III.3.3 Les alternatives:

Lorsque plusieurs constructions se rapportent au même lexème, au lieu de définir plusieurs règles, il est judicieux de les regrouper dans une seule règle à plusieurs alternatives. Les alternatives sont séparées les unes des autres par un "|" dans le corps de la règle.

Les alternatives d'une règle :

```
nom-regle // les options liées uniquement à cette règle }
 : alternative 1
 | alternative 2
 | ...
 | alternative n
 ;
```

Exemple :(reconnaissance des espaces dans un flux de caractères**)** définir une seule règle à plusieurs alternatives comme suit :

```
White_Space
  : ' '
  | '\t'
  | '\r' '\n'
  | '\n'   ;
```

Lorsque plusieurs constructions se rapportent au même lexème, au lieu de définir plusieurs règles, il est judicieux de les regrouper dans une seule règle à plusieurs alternatives. Les alternatives sont séparées les unes des autres par un "|" dans le corps de la règle.

III.3.4 Les options:

Dans la section Options d'une grammaire ANTLR, vous pouvez spécifier une série de clé/valeur qui modifient la façon dont ANTLR génère du code.

La section des options doit venir après l'en-tête de la grammaire et doit avoir la forme suivante :

```
options {
        nom1 = valeur1;
        nom2 = valeur2;
        ...
       Nom N = valeur N; }
```

Les noms des options sont toujours des identificateurs, mais les valeurs peuvent être des identificateurs, des chaînes de littéraux entre des guillemets simples, des nombres entiers. Les valeurs sont toutes des littéraux et, par conséquent, ne peuvent pas être des noms d'option. Pour les chaînes littérales formées d'un seul mot comme « Java », vous pouvez l'écrire tout simplement, comme indiqué ci-après :

```
options {
Language=Java;
 }
```

La liste qui suit résume les options ANTLR au niveau de la grammaire.

- **Language :** Précisez le langage cible dans lequel ANTLR devra générer les analyseurs.

- **Output :** Générer des modèles de sortie, un modèle ou des arbres AST. Cette option est disponible uniquement pour les grammaires qui contiennent des analyseurs syntaxiques et un analyseur d'arbres. La valeur par défaut est de ne rien générer.
- **tokenVocab :** Indiquez comment ANTLR devrait nommer l'ensemble de Tokens prédéfinis ou générés. Cela sera nécessaire pour une grammaire qui veut utiliser les Tokens d'une autre. Typiquement, une grammaire d'arbre utilisera les Tokens de la grammaire de l'analyseur qui crée ses arbres. La valeur par défaut aucun.

- **K :** Limite l'analyseur généré de cette grammaire à utiliser une profondeur maximale d'anticipation de la valeur de k. Ceci transforme l'analyse LL(*) classique en une analyse LL(k). La valeur par défaut est * pour une grammaire LL(*).

Voici un petit exemple d'une grammaire Antlr pour les expressions arithmétique :

```
grammar Exp;
options {
  language = Java;
}
stat: expr '=' expr ';'                    // e.g., x=y; or x=f(x);
        | expr ';'                         // e.g., f(x); or f(g(x));
        ;
expr: expr '*' expr
        | expr '+' expr
        | expr '(' expr ')'                // f(x)
        | ID
        |INT
        ;
ID : ('a'..'z' | 'A'..'Z' | '_')+;
INT : [0-9]+ ;
WS : : (' '|'\t'|'\n'|'\r') + ->skip;      // ignore whitespace
```

De la grammaire Expr, ANTLR généré Expr Parser et Expr Lexer…

III.4 Grammaire pour les formulaires HTML :

La première partie consiste à l'écriture d'une grammaire, qui respecte l'ensemble des contraintes.

ANTLR utilise la grammaire proprement parlée pour générer un analyser des pages HTML composées essentiellement d'un ensemble de balises. En effet, l'outil ANTLR ne nécessite que l'écriture d'une grammaire spécifique au langage HTML (html.g4)

La grammaire obtenue est la suivante :

```
grammar HTML;
options {
  language = Java;           // La langue cible est Java.
}
LTAG:          '<';
RTAG:          '>';
ENDTAG:        '</';
QUOTE:         '"';
EQUALS:        '=';
SPTAG:         '/>';

page_HTML: element*;
element:
      ((LTAG|ENDTAG) name_elements+ parameters* (RTAG|SPTAG))
(comment|INTEGER)*
     ;
parameters:    name_parameter EQUALS attribute    ;
Attribute:     (QUOTE? ID QUOTE? | INTEGER |STRING)    ;
name_parameter: ID;
name_element
     : ('html' | 'HTML')
     | ('head '| 'HEAD')
     | ('title' | 'TITLE')
     | ('body' | 'BODY')
     | ('input' | 'INPUT')
     | ('textarea' | 'TEXTAREA')
     | ('select' | 'SELECT')
     | ('option' | 'OPTION')
     |comment
     ;
comment: ID;
INTEGER :('0'..'9') +;
STRING:'"' (ID | ~ ('\\'|'"'))* '"'     ;
ID: ('a'..'z' | 'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'0'..'9'|'_')*;
WS: (' '|'\t'|'\n'|'\r') + ->skip;
```

III.5 L'outil ANTLR, exécution, et le code généré

ANTLR 4 est constitué d'un seul fichier JAR; en plus de contenir des classes qui peuvent être incorporés dans votre application à l'exécution, ce JAR peut être appelé à la ligne de

commande pour générer à partir d'un fichier « grammaire » un ensemble de fichiers Java constituant à la fois l'analyseur syntaxique et lexical.

Après l'exécution de cette commande, Qu'est-ce que ANTLR Générer?

- **Les fichiers générés par ANTLR**

De la grammaire HTML.g4, ANTLR génère 6 fichiers démontrés dans la figure III.2.

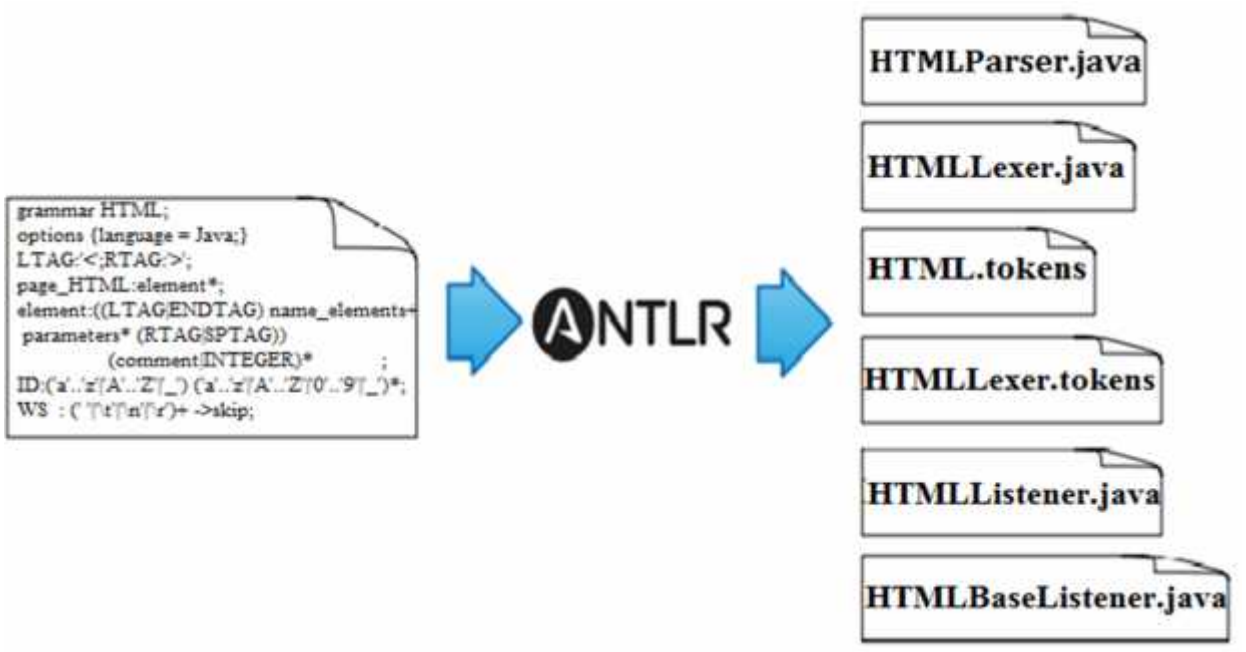


Figure III.2: les classes générées par ANTLR à partir de la grammaire HTML.

III.5.1 Description de code générer :

Fichier généré	Description
HTMLParser.java	Ce fichier contient le parseur (Parser) des pages HTML
HTMLLexer.java	Cette classe implémente l'analyseur lexical (Lexer), elle a utilisée par le (Parser)
HTML.tokens	Ce fichier contient l'ensemble de Tokens (les mots clés) utilisé dans une page HTML.
HTMLBaseListener	Un Listener est une interface qui pourra être utilisée pour implémenter les règles de translation.
HTMLLexer:	traduit un flux de caractères en flux de Tokens.
HTMLParser:	construit l'arbre syntaxique.

Tableau III.4 : description déférant code générer par ANTLR.

III.6 Les règle de translation :

Pour écrire un programme qui réagit à l'entrée, tout ce que nous avons à faire est de mettre en œuvre quelques méthodes dans une classe étendue de ***HTMLBaseListener***. La stratégie de base est que chaque méthode sert à écouteur l'apparence des nœuds utilisés dans les méthodes ***Override*** dans la classe étendue pour les exécuter afin de générer les différents codes javascript.

La figure III.3 décrit l'ensemble d'événement pourra être exploités durant le parcoure de l'arbre syntaxique établi par le Parser.

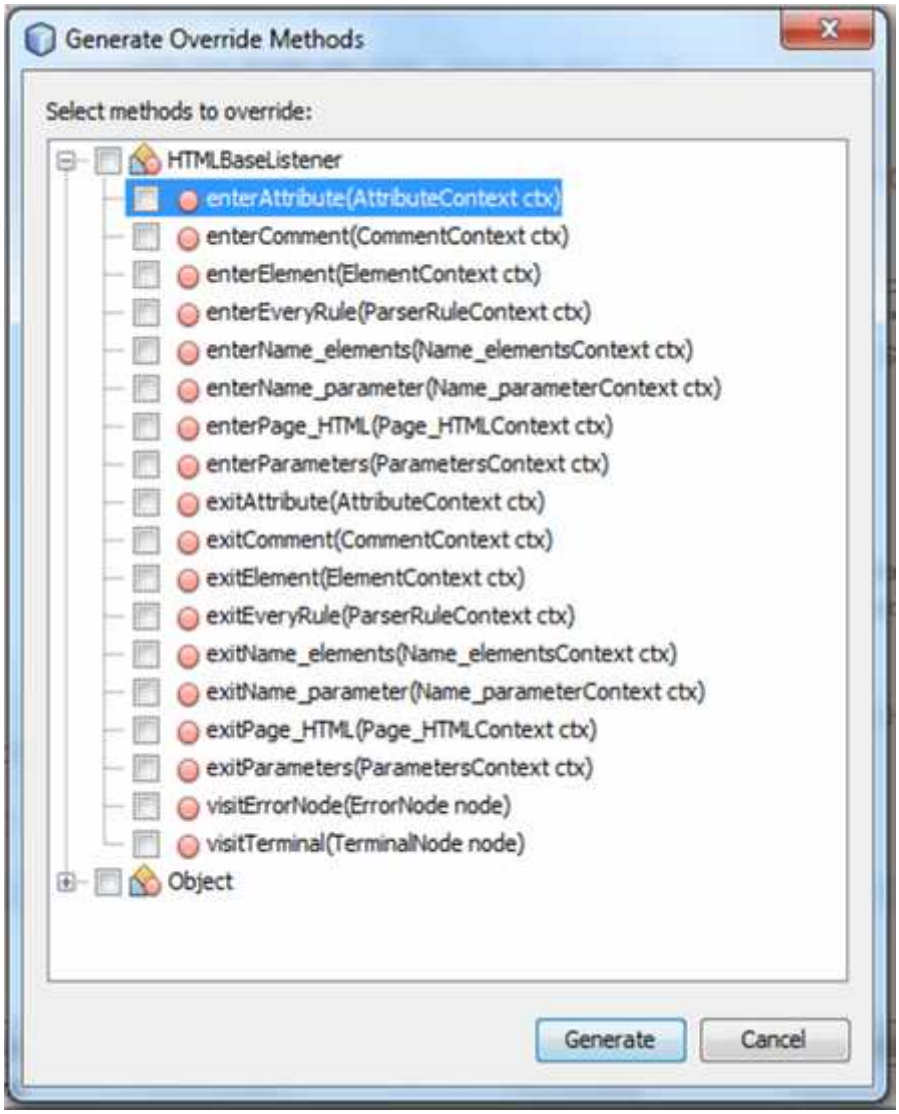
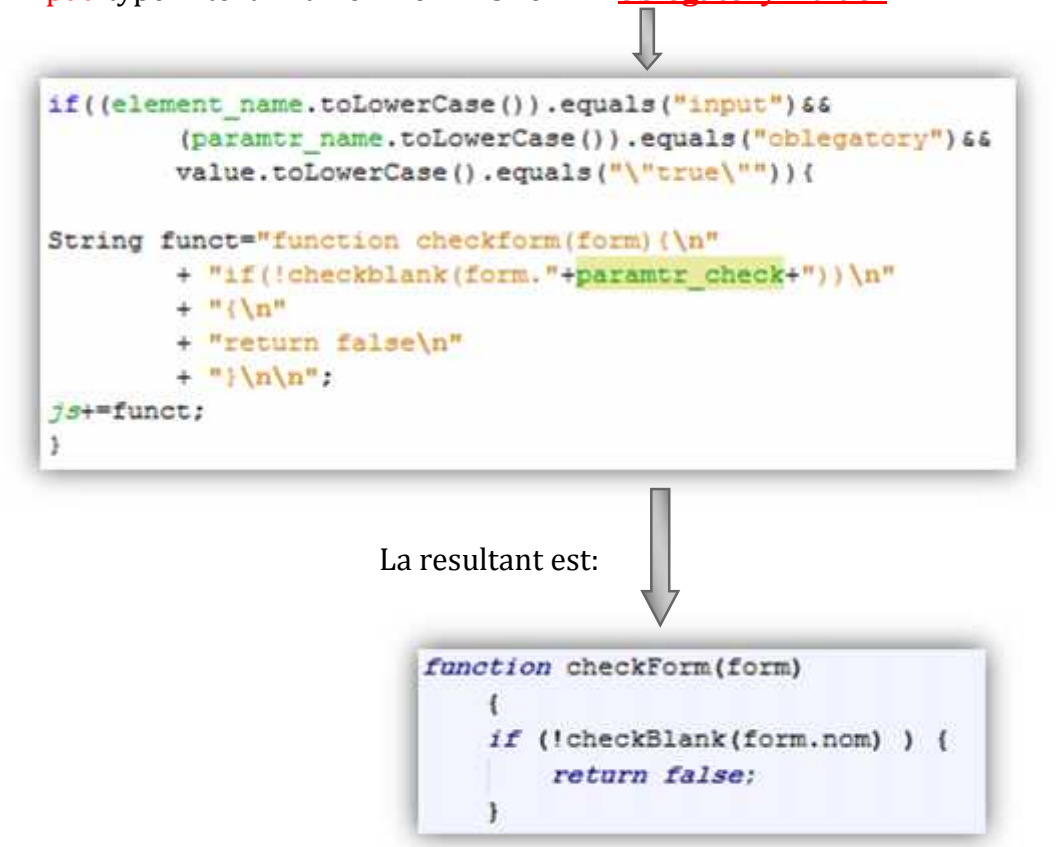


Figure III.3 : Déférant méthode pour parcoure l'arbre syntaxique.

Exemple : Translation du paramètre oblegatory= ''true'' ver le code JS suivent.

<input type= ''text'' name=''nom'' size=12 **oblegatory="true"** >



L'exemple précédent d'écrit comment générer en js la fonction permettant de vérifier le nom d'un formulaire.

➢ Les règles :

✓ **La méthode enterPage_HTML :** Pour accéder la page_HTML complet.

```
@Override
public void enterPage_HTML(Page_HTMLContext ctx) {
    super.enterPage_HTML(ctx);
   // System.out.println(ctx.getText());
}
```

✓ **La méthode enterElement :** Pour accéder les éléments de formulaire.

```
@Override
public void enterElement(ElementContext ctx) {
    super.enterElement(ctx);
    c = ctx;
   // element_name=ctx.getChild(1).getText();
   // System.out.println(ctx.getChild(1).getText());
}
```

- ✓ **La méthode enterName_elements :** Pour accéder les name_elements.

```
@Override
public void enterName_elements(Name_elementsContext ctx) {
    super.enterName_elements(ctx);
    element_name=ctx.getText();

   // System.out.println(element_name);
}
```

- ✓ **La méthode enterParamaters :** Pour accéder les paramètres de formulaire.

```
@Override
public void enterParameters(ParametersContext ctx) {
    super.enterParameters(ctx);
    if(ctx.getChild(0).getText().equals("name"))
        paramtr_check=ctx.getChild(2).getText().replaceAll("\"", "");
    else
    paramtr_name=ctx.getChild(0).getText();
}
```

- ✓ **La méthode enterName_parameter :** Pour accéder les name_parameter.

```
@Override
public void enterName_parameter(Name_parameterContext ctx) {
    super.enterName_parameter(ctx);

}
```

- ✓ **La méthode enterAttribut :** Pour accéder les attributs de formulaire.

```
@Override
public void enterAttribute(AttributeContext ctx) {
    super.enterAttribute(ctx);
    String value="";
    value=ctx.getText();
```

✓ **La méthode exitPage_HTML :** Pour accéder la fin de la page_HTML.

```
@Override
public void exitPage_HTML(Page_HTMLContext ctx) {
    super.exitPage_HTML(ctx);
    js+="\n alert(\"Merci pour votre réponse ...\");\n"
            + "return true;\n"
            + "}\n"
            + "\n</script>";
   System.out.println(js);
}
```

✓ **La méthode enterAttribut :**
Translation du paramètre oblegatory= ‘’true’’ ver le code JS :

```
@Override
public void enterAttribute(AttributeContext ctx) {
    super.enterAttribute(ctx);
    String value="";
    value=ctx.getText();

    if((element_name.toLowerCase()).equals("input")&&
            (paramtr_name.toLowerCase()).equals("oblegatory")&&
            value.toLowerCase().equals("\"true\"")){

    String funct="function checkform(form){\n"
            + "if(!checkblank(form."+paramtr_check+"))\n"
            + "{\n"
            + "return false\n"
            + "}\n\n";
    js+=funct;
    }
```

Nous n'avons pas besoin d'*Override* toutes les méthodes enter /exit; nous faisons seulement ceux qui nous intéressent. **Exemple :** enterPage_HTML et exitPage_HTML.
EnterAttribute et exitAttribute

III.7 Présentation du logiciel :

Environnement de travail :

Pour atteindre notre but qui est la réalisation d'un compilateur des pages HTML pour générer le code JavaScript. Nous avons utilisé le langage de programmation Java.

Commençons par l'interface générale du logiciel :

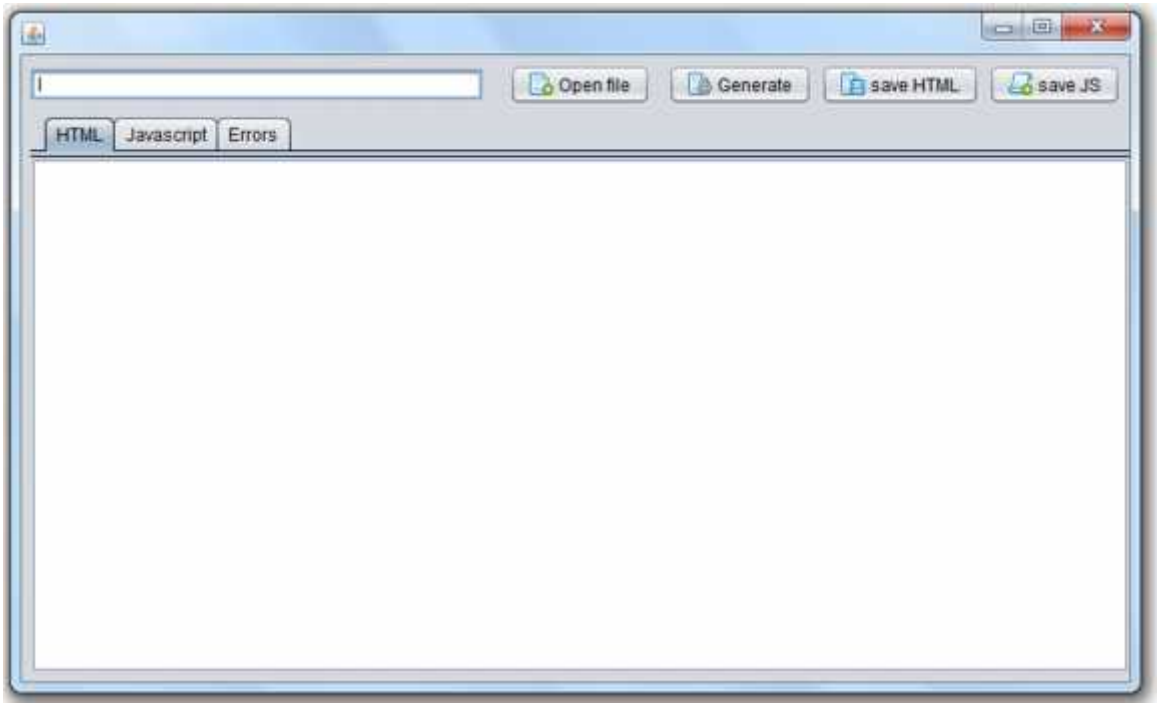


Figure III .4 : l'interface générale du logiciel.

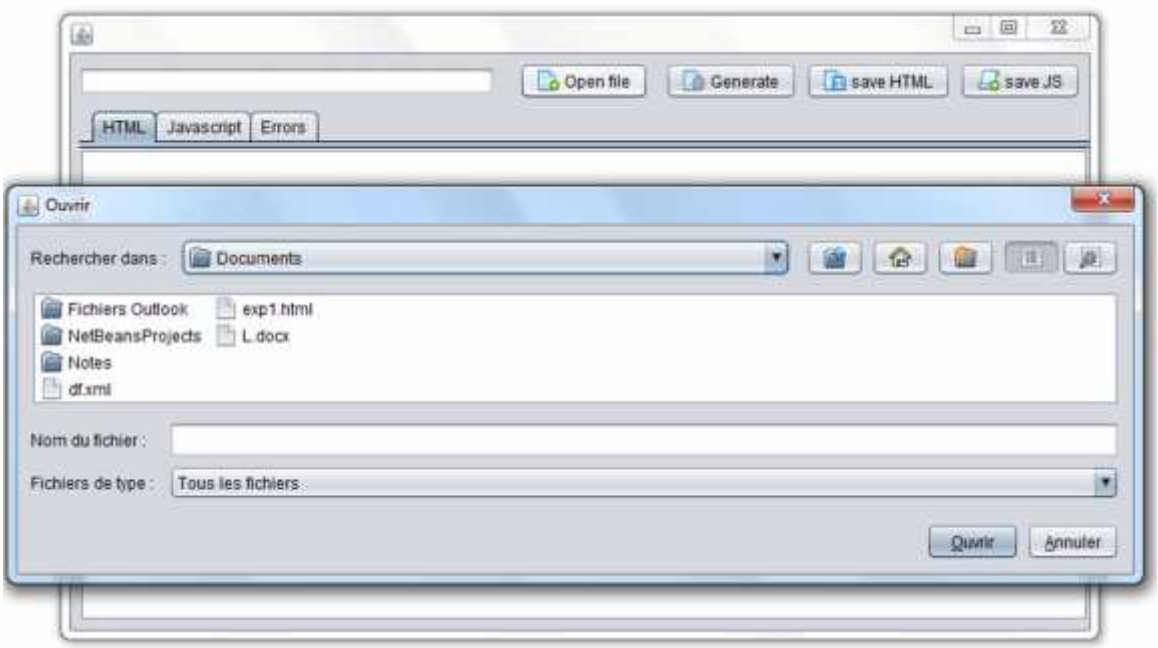


Figure III.5 : interface pour ouvrir un ficher html à générer.

Figure III.6 : Message de dialogue concernant le type de fichier ouvert

- **Exemple d'une page HTML correcte**

La page HTML ouverte et montrée dans l'interface de la figure xx contient 4 formulaires contenant chacune un ensemble varié de paramètres de différents types.

Dans un premier temps, notre compilateur vérifie la syntaxe de cette page afin de générer les codes illustrés dans la figure III.7.

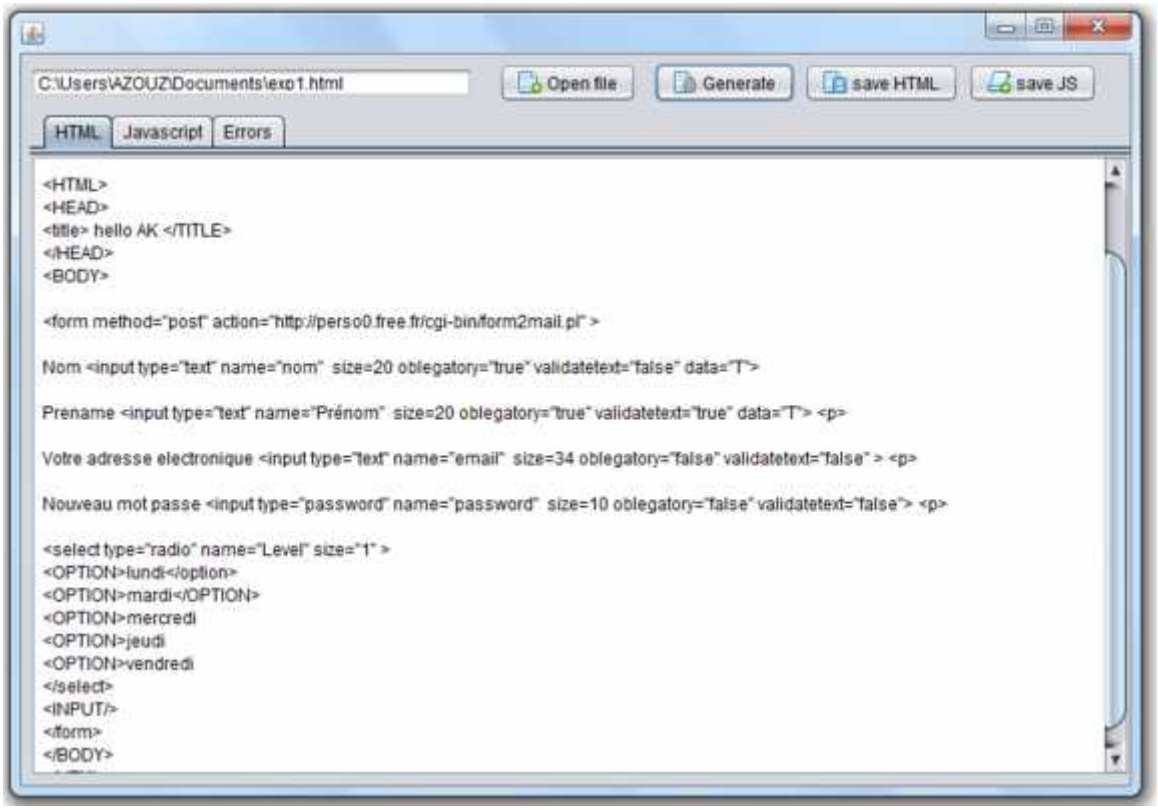


Figure III.7 : Une page HTML à compiler.

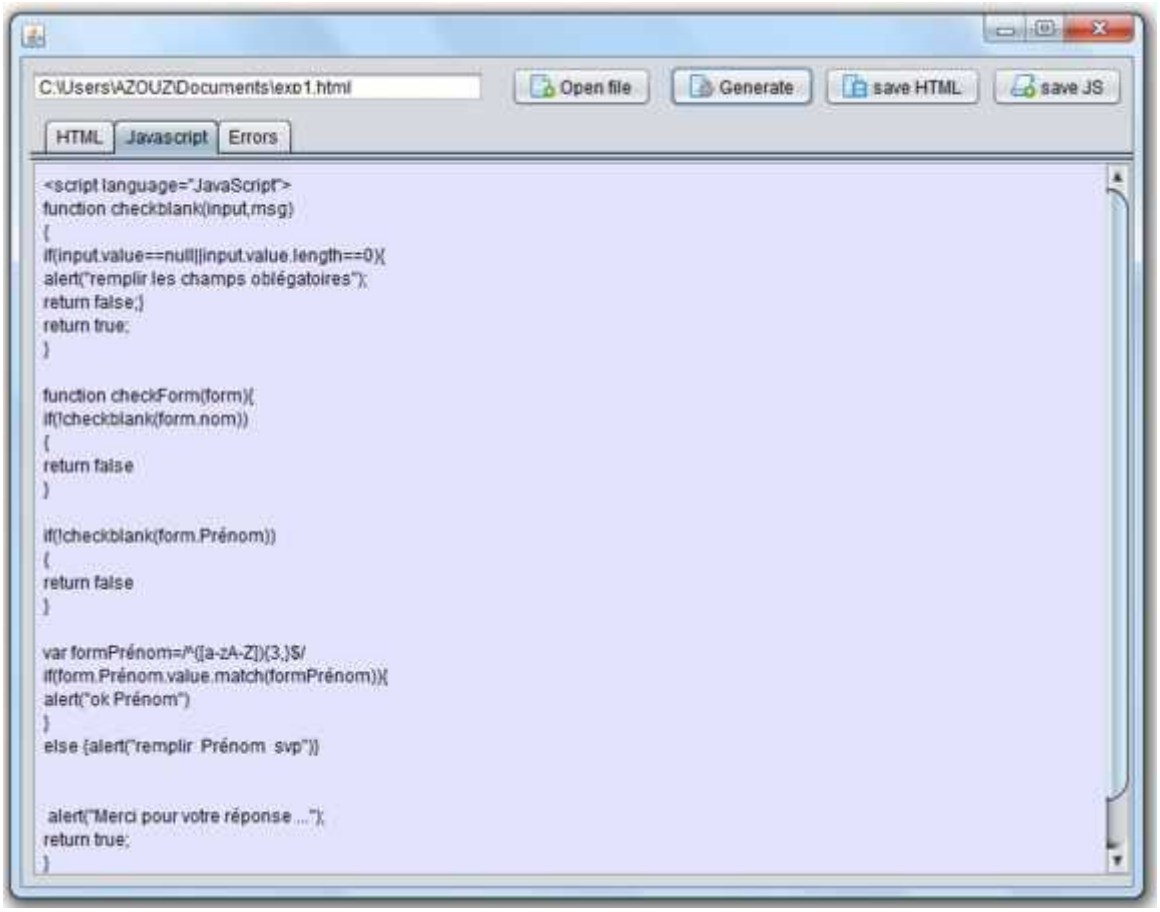


Figure III.8 : interface affiche le code JavaScript généré.

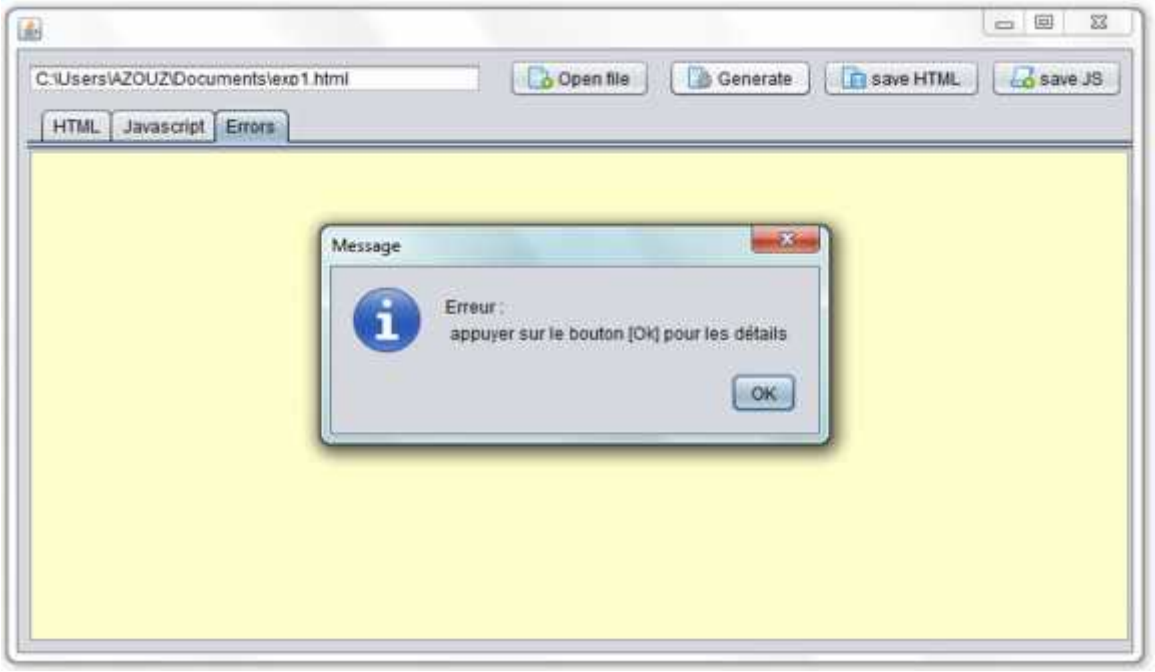


Figure III.9 : Message de dialogue concernant les erreurs de la page HTML à compiler.

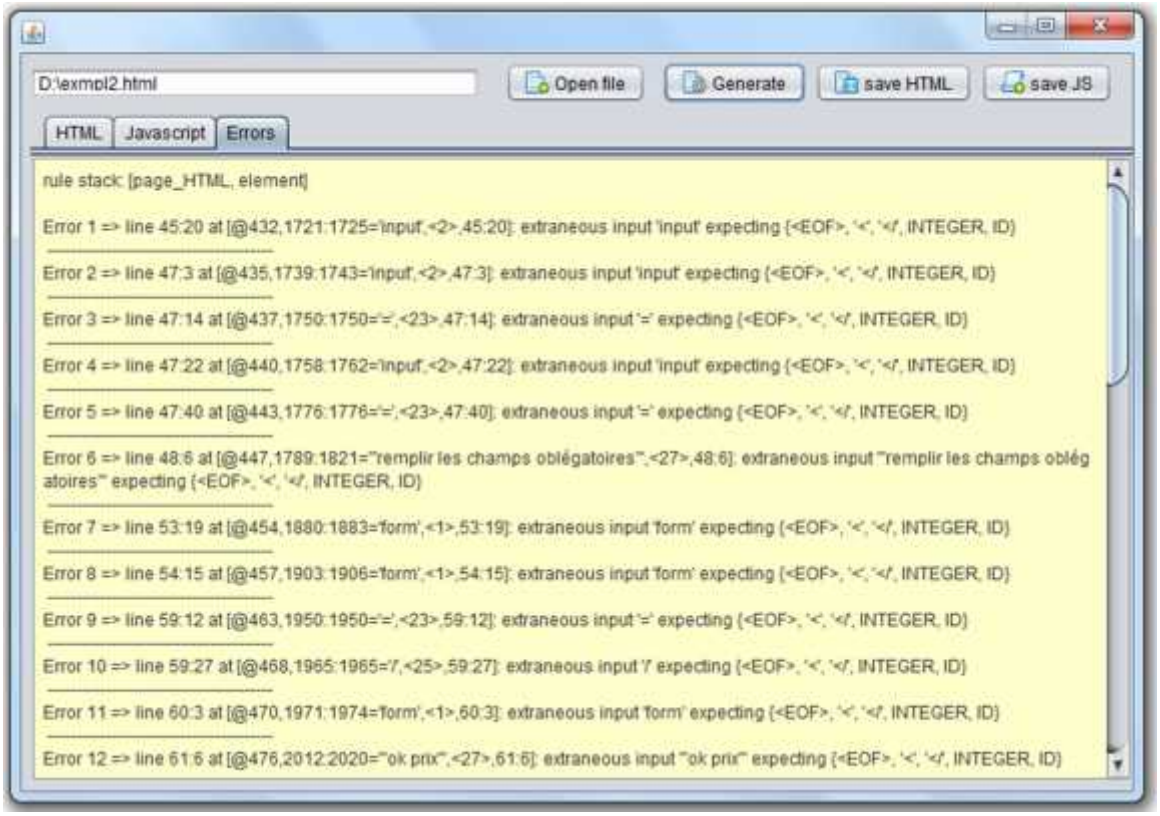


Figure III.10: interface affiche les erreurs de la page HTML détaillé.

III.8 CONCLUSION :

ANTLR regroupe l'analyse lexicale et syntaxique dans un fichier unique, ce qui implique le besoin de maîtriser un seul outil.

ANTLR génère beaucoup de fichiers que regroupant à la fois l'analyseur syntaxique et lexical.

Nous avons défini un ensemble de règles de grammaire HTML, et met en œuvre un analyseur HTML basé sur l'outil ANTLR. Donc, les programmeurs peuvent utiliser ces analyseurs générés auparavant en tout fiabilité dans un logiciel indépendant.

Conclusion Générale

Tout au début de notre projet, l'objectif fixé était de réaliser un compilateur permettant aux programmeurs qui générer code JavaScript à partir des pages HTML.

Nous définissons des règles syntaxiques d'une grammaire HTML, et mettons en œuvre un analyseur HTML basé sur la technologie ANTLR, utilisé pour faire des analyses lexicales et syntaxiques.

Nous sommes arrivés à développer une application Java indépendante en exploitant l'ensemble de classes déjà générées par ANTLR. Notre application sert à compiler des pages HTML, contenant des formulaires, et générer code JavaScripts basant sur un ensemble de nouveaux paramètres proposés pour maitriser le contenu des codes générés.

Enfin, nous disons que ce projet nous à permis d'approfondir nos connaissances dans les domaines compilation et Web.

Nous cherchons dans le futur d'étendre notre démarche pour aller plus profondément jusqu'à la manipulation des bases de données dans un projet web dynamique.

Bibliographie

Bibliographie

[1] Alfred Aho, Monica Lam, R sethi ET J ullman. " COMPILATEURS principes, Techniques et outils" Achevé d'imprimer en France le 7 novembre 2007, p 1-129.

[2] Dick Grune, Hebri E. Bal, Ceriel J.H. Jacobs, Coen G. Langendoen "COMPILATEURS cours et exercices corrigés" imprimer en Belgique le aout 2002.

[3] H-DRIAS, "compilation coure et exercice", 1992, p 5.

[4] Nicolas Delestre "Introd à la Compilation", 2001, p 8-27.

[5] Éric Hervet, "LES OUTILS LEX & YACC PRINCIPES ET APPLICATIONS", 2009, p 1-10.

[6] Henri Garreta, "Techniques et outils pour la compilation ", Faculté des Sciences de Luminy - Université de la Méditerranée Janvier 2001, p 19.

[7] William Levy et Rémy Rysman, "Dessin Automatique de Graphes", 2012, p 7.

[8] Ph. Truillet – UPS, "Les formulaires HTML", 27 Septembre 2001, p 1-4.

[9] Van Lancker Luc, "Les formulaires", 1998, p 1-8.

[10] Damien Brémont, "Mémo PHP/HTM", novembre 2006, p 4-6.

[11] Olivier Hondermarck, "le guide complet JavaScript", juin 2009, p 276.

[12] Jean Engels, "PHP5 Cours et exercices", 2eme édition, en 2009, p 155-176.

[13] Christian Wenz, "JavaScript", 2009, p131-154.

[14] Y. Mine, "Cours d'Initiation au langage JavaScript", 2002, p 5.

[15] Daniel K. Schneider, "Patrick Jermann et all, introduction au javascript", 2010, p 39-41.

[16] Etienne vandeput, "Développer une application avec PHP et MySQL", 2005, p31-32.

[17] Donghui Bai "Design and implementation for SQL parser based on ANTLR" Los Angeles: IEEE Press, 2010.

Abstract :

We look in this work to present a compiler for HTML pages to parse and translate them into javascript codes based on some new introduced parameters. Our compiler is based on some java classes generated by ANTLR tool.

Used an LL(*) grammar proposed for HTML pages, ANTLR tool has generated a set of java classes which we have used in a separated application showing the importance of our approach.

Keywords: Compiler, lexical analysis, parsing, grammar, ANTLR, web development, HTML, JavaScript.

Résumé :

On cherche à présenter un compilateur des pages HTML pour les manipuler (faire des analyses lexicales et syntaxiques) et les translater basant sur des paramètres prédéfinis vers un code JavaScript. Ce compilateur est implémenté utilisant l'outil ANTLR.

A partir d'une grammaire de type LL(*) ANTLR nous a généré un ensemble de classes java permettant de parser (analyser le lexique et la syntaxe) d'une page HTML. En exploitant ces classes java nous avons développé un simple outil visualisant l'utilité de notre approche.

Mots clés : Compilateur, analyse lexicale, analyse syntaxique, grammaire, ANTLR, Développent web, HTML, JavaScript.

ملخص:

نأمل من خلال هذه المذكرة, في تقديم مترجم للتعامل مع صفحات خاصة **بلغة** HTMLو بالخصوص السمات المكونة لها بغية ترجمتها الى جافا سكريبت واليوم اصبح اكثر شيوعا استخدام جافا سكريبت للتأكد من صحة البيانات المدخلة من طرف المستخدم قبل ارسالها الى **الخادم**.

تعتمد هذه الترجمة على **أداة** ANTLR لإنشاء التعليمات البرمجية جافا سكريبت.

انطلاقا من القواعد النحوية من النوع LL(*) **وباستعمال أداة** ANTLR **نحصل على مجموعة من** classes java **التي** تقوم بتحليل المفردات و بناء الجملة من صفحة HTML من خلال استغلال هذه الفئات(classes java) **من اجل ترجمة رمز** جافا سكريبت انطلاقا من سمة متفق عليها مسبقا, ولإتمام العملية لا بد من تحديد مجموعة قواعد للتحويل بمقدورها انجاز الإجراء بصفة آلية.

كلمات مفتاحية: **مترجم**, جفا سكريبت, **قواعد النحو**, HTML, التحليل النحوي, ANTLR, تحليل المفردات,

تطوير مواقع **الويب**