Introduction:

Cette partie du projet avait pour objectif principal de concevoir un analyseur syntaxique en utilisant le langage C. L'analyseur devait s'appuyer sur une méthode descendante ou ascendante pour générer les tables d'analyse syntaxique. L'approche à suivre reposait sur les transformations vues en cours, comprenant la conception et la construction des ensembles First et Follow, la génération des tables d'analyse syntaxique, ainsi que la mise en œuvre d'un parseur pour un sous-ensemble de la grammaire du langage Flamingo.

Objectifs et Démarches

1. Méthode d'analyse syntaxique :

Le choix de la méthode descendante (LL(1)) a été retenu pour ce projet en raison de sa simplicité par rapport à la méthode ascendante. La méthode descendante permet une implémentation directe et claire.

2. Grammaire ciblée :

Une grammaire non récursive à gauche, factorisée et propre a été choisie pour garantir la compatibilité avec l'analyse descendante. La grammaire choisie est la suivante :

expression -> PARENTHESEOUVRANTE expression PARENTHESEFERMANTE | terme
terme -> ID CROCHETOUVRANT expression CROCHETFERMANT | INT

3. Tâches réalisées :

A. Analyse lexicale

Une analyse lexicale spécifique a été mise en œuvre pour identifier les tokens directement utilisés dans la grammaire, comme `INT`, `ID`, `PARENTHESEOUVRANTE`, `PARENTHESEFERMANTE`, `CROCHETOUVRANT`, et `CROCHETFERMANT`.

B. Ensembles First et Follow

Calcul des ensembles First

Les ensembles First sont déterminés pour chaque non-terminal en fonction des règles suivantes :

Pour une production `A → Xα` :

- Si `X` est terminal, ajouter `X` à `First(A)`.
- Si `X` est non-terminal, ajouter les éléments de `First(X)` (sauf `εpsilon`) à `First(A)`.

Répéter jusqu'à ce qu'il n'y ait plus de changements.

Calcul des ensembles Follow

Les ensembles Follow sont calculés pour chaque non-terminal en suivant ces règles :

Pour une production `A → αBβ` :

- Ajouter `First(β)` (sauf `εpsilon`) à `Follow(B)`.
- Si `β` est vide ou contient `εpsilon`, ajouter `Follow(A)` à `Follow(B)`.

Répéter jusqu'à stabilisation.

C. Table d'analyse syntaxique

Une table LL(1) est générée en combinant les informations des ensembles First et Follow :

- Parcourir chaque production `A → α`.
- Pour chaque terminal `t` dans `First(α)`, ajouter la production dans `table[A][t]`.
- Si `εpsilon ∈ First(α)`, ajouter `Follow(A)` dans la table pour cette production.

- Gérer les conflits en vérifiant les propriétés LL(1) (non-ambiguïté).

D. Parseur

Le parseur interagit avec l'analyseur lexical via `yylex()` pour récupérer les tokens. Il utilise une pile pour gérer les transitions entre symboles terminaux et non-terminaux :

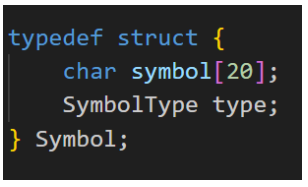
Initialisation : Pousser le symbole de départ et le marqueur # dans la pile.

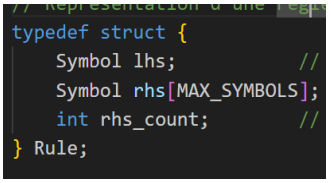
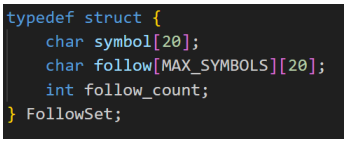
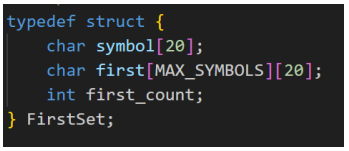
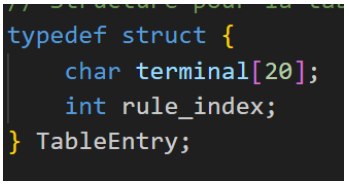
- Comparer le sommet de la pile au prochain token.
- Si le sommet est terminal et correspond au token, le dépiler et avancer dans l'entrée.
- Si le sommet est non-terminal, consulter la table pour appliquer la production correspondante.
- Signaler une erreur si aucune transition n'est trouvée.

Fin : Lorsque la pile et l'entrée sont synchronisées (toutes deux vides ou contiennent #), l'entrée est acceptée.

4. Structures utilisées:

Plusieurs structures ont été définies pour représenter les différents concepts de l'analyse syntaxique :

Structure	Explication	Capture d'écran
Symboles (Symbol)	Les symboles peuvent être des terminaux ou des non-terminaux. Ils sont représentés par une chaîne de caractères (symbol) et un type (TERMINAL ou NON_TERMINAL).	

Règles de grammaire (Rule)	Chaque règle associe un symbole à gauche (LHS) à une liste de symboles à droite (RHS).Elle contient lhs : le symbole à gauche. rhs : un tableau de symboles à droite. rhs_count : le nombre de symboles à droite.	
Ensembles First et follow (FirstSet, FollowSet)	FirstSet : ces ensembles associent un symbole à une liste de terminaux qui peuvent apparaître en première position dans ses dérivations. FollowSet: Ces ensembles associent un symbole à une liste de terminaux pouvant apparaître immédiatement après ce symbole dans une production.	 
TableEntry	La structure TableEntry est utilisée pour représenter une entrée dans la table d'analyse LL(1). Une entrée correspond à une combinaison d'un terminal (symbole attendu dans l'entrée) et d'une règle de production spécifique de la grammaire qui doit être appliquée.	
Pile pour le parseur (Stack)	La pile gère les symboles en attente d'analyse.Elle est représentée par : items : un tableau de chaînes de caractères (les symboles). top : un entier représentant l'indice du sommet de la pile.	

5. Résultats obtenus:

Pour valider le bon fonctionnement de l'analyseur syntaxique, nous avons testé différentes chaînes d'entrée. Voici un exemple illustrant les résultats obtenus avec la chaîne ‘ jihene2004{2}’:

```
C:\Users\USER\Desktop\JSP>main2.exe

Grammar Rules:
------------------------------------------------------------------------
| LHS             | RHS                                                |
------------------------------------------------------------------------
| expression      | PARENTHESEOUVRANTE expression PARENTHESEFERMANTE  |
| expression      | terme                                              |
| terme           | INT                                                |
| terme           | ID CROCHETOUVRANT expression CROCHETFERMANT       |
------------------------------------------------------------------------

First Sets:
First(expression) = { PARENTHESEOUVRANTE INT ID }
First(expression) = { }
First(terme) = { INT ID }
First(terme) = { }

Follow Sets:
Follow(expression) = { $ PARENTHESEFERMANTE CROCHETFERMANT }
Follow(expression) = { }
Follow(terme) = { $ PARENTHESEFERMANTE CROCHETFERMANT }
Follow(terme) = { }
```

```
Parsing Table:
             PARENTHESEOUVRANTE  PARENTHESEFERMANTE  INT               ID                CROCHETOUVRANT      CROCHETFERMANT      $
-------------------------------------------------------------------------------------------------------------------------------------
expression     0                 -                   1                 1                 -                   -                   -
terme          -                 -                   2                 3                 -                   -                   -
```

```
Starting LL(1) parsing for input: jihene2004{2}

+-------------------------------------------------------------+-------------------------------------------------------------------------------+----------------+
| Stack Content                                               | Input String                                                                  | Action         |
+-------------------------------------------------------------+-------------------------------------------------------------------------------+----------------+
| # terme                                                     | jihene2004{2}                                                                 | Apply Rule 1   |
+-------------------------------------------------------------+-------------------------------------------------------------------------------+----------------+
| # CROCHETFERMANT expression CROCHETOUVRANT ID               | jihene2004{2}                                                                 | Apply Rule 3   |
+-------------------------------------------------------------+-------------------------------------------------------------------------------+----------------+
| # CROCHETFERMANT expression CROCHETOUVRANT                  | {2}                                                                           | Match          |
+-------------------------------------------------------------+-------------------------------------------------------------------------------+----------------+
| # CROCHETFERMANT expression                                 | 2}                                                                            | Match          |
+-------------------------------------------------------------+-------------------------------------------------------------------------------+----------------+
| # CROCHETFERMANT terme                                      | 2}                                                                            | Apply Rule 1   |
+-------------------------------------------------------------+-------------------------------------------------------------------------------+----------------+
| # CROCHETFERMANT INT                                        | 2}                                                                            | Apply Rule 2   |
+-------------------------------------------------------------+-------------------------------------------------------------------------------+----------------+
| # CROCHETFERMANT                                            | }                                                                             | Match          |
+-------------------------------------------------------------+-------------------------------------------------------------------------------+----------------+
| #                                                           |                                                                               | Match          |
+-------------------------------------------------------------+-------------------------------------------------------------------------------+----------------+
| #                                                           |                                                                               | Accept         |
+-------------------------------------------------------------+-------------------------------------------------------------------------------+----------------+

Parsing completed successfully!
```

6. Conclusion:

Ce projet implémente un analyseur syntaxique basé sur la méthode LL(1). Il calcule les ensembles First et Follow, construit une table d'analyse, et exécute un analyseur prédictif avec une gestion des erreurs. Le code est structuré de manière à être extensible pour ajouter des règles ou des symboles supplémentaires.