Mini projet de compilation

While(C)

📄 **Mini-rapport – Analyseur lexical et syntaxique**

1. La grammaire choisie

La grammaire est inspirée du langage C, mais simplifiée pour se concentrer sur les **déclarations de variables**, les **boucles** while, les **expressions arithmétiques** et les **conditions logiques**. Elle est définie comme suit :

S → DECLS INSTS | TYPE main ( ) BLOCK

DECLS → DECLVAR DECLS | ε

DECLVAR → TYPE id INIT ;

INIT → = EXPR | ε

TYPE → int | float | char

INSTS → INST INSTS | ε

INST → WHILESTMT | IFSTMT | DECLVAR | SCANFSTMT | PRINTFSTMT | AFFECTATION | INCREMENT | RETURNSTMT | BREAKSTMT

WHILESTMT → while (COND_LOGIQUE) BLOCK | while (COND_LOGIQUE) INST

IFSTMT → if (COND_LOGIQUE) BLOCK ELSESTMT

ELSESTMT → else BLOCK | ε

BREAKSTMT → break ;

RETURNSTMT → return EXPR ; | return ;

SCANFSTMT → scanf (STRING , & id) ;

PRINTFSTMT → printf (STRING PRINTFARGS) ;

PRINTFARGS → , EXPR PRINTFARGS | ε

AFFECTATION → id = EXPR ; | id += EXPR ; | id -= EXPR ; | id *= EXPR ; | id /= EXPR ;

INCREMENT → id ++ ; | id -- ; | ++ id ; | -- id ;

BLOCK → { INSTS }

COND_LOGIQUE → COND_ET COND_LOGIQUE'

COND_LOGIQUE' → || COND_ET COND_LOGIQUE' | ε

COND_ET → COND_SIMPLE COND_ET'

COND_ET' → && COND_SIMPLE COND_ET' | ε

COND_SIMPLE → (COND_LOGIQUE) | EXPR RELOP EXPR | EXPR

RELOP → < | > | <= | >= | == | !=

EXPR → TERM EXPR'

EXPR' → + TERM EXPR' | - TERM EXPR' | ε

TERM → FACTOR TERM'

TERM' → * FACTOR TERM' | / FACTOR TERM' | % FACTOR TERM' | ε

FACTOR → id INCREMENT_POST | number | (EXPR) | ++ id | -- id

INCREMENT_POST → ++ | -- | ε

id → [a-zA-Z][a-zA-Z0-9_]*

number → [0-9]+ | [0-9]+.[0-9]+

STRING → ".*"

2. L'analyseur lexical

- Automate fini déterministe basé sur une **matrice de transition**.
- Reconnaît les catégories suivantes : mots-clés, identificateurs, nombres , opérateurs, délimiteurs.

Sortie : chaque lexème est affiché sous la forme lexeme : type.

3. L'analyseur syntaxique

Implémenté en Java avec une approche descente récursive.

Chaque règle de la grammaire correspond à une fonction (Z(), WhileInstr(), COND(), EXPR(), etc.).

Vérifie la validité grammaticale et affiche des erreurs explicites en cas de non-conformité.

4. Structure du projet

Arborescence des fichiers :

Code

projetcompilation/

analyselexicale.java // Analyseur lexical

analysesyntaxique.java // Analyseur syntaxique

Main (dans analyselexicale)

- saisie utilisateur

- exécution de l’analyse lexicale
- transmission des tokens à l’analyseur syntaxique

5.Cas de tests

```
run:
=== COMPILATEUR C - PROGRAMME COMPLET ===
Entrez le code (tapez 'FIN' pour terminer) :
#include <stdio.h>
int main() {
   int i = 1;
   int n = 3;
   while (i <= n) {
      int j = i;
      while (j > 0) {
         printf("%d ", i + j * 2);
         j--;
      }
      i++;
   }
   return 0;
}
FIN
=== LEXÈMES RECONNUS ===
#include <stdio.h>:Directive préprocesseur
int:Mot-clé
main:Mot-clé
(:Délimiteur
):Délimiteur
{:Délimiteur
int:Mot-clé
i:Identificateur
=:Opérateur
1:Nombre
;:Délimiteur
```

int:Mot-clé

n:Identificateur

=:Opérateur

3:Nombre

;:Délimiteur

while:Mot-clé

(:Délimiteur

i:Identificateur

<=:Opérateur

n:Identificateur

):Délimiteur

{:Délimiteur

int:Mot-clé

j:Identificateur

=:Opérateur

i:Identificateur

;:Délimiteur

while:Mot-clé

(:Délimiteur

j:Identificateur

>:Opérateur

0:Nombre

):Délimiteur

{:Délimiteur

printf:Mot-clé

(:Délimiteur

"%d ":Chaîne

,:Délimiteur

i:Identificateur

+:Opérateur

j:Identificateur

*:Opérateur

2:Nombre

):Délimiteur

;:Délimiteur

j:Identificateur

--:Opérateur

;:Délimiteur

}:Délimiteur

i:Identificateur

++:Opérateur

;:Délimiteur

}:Délimiteur

return:Mot-clé

0:Nombre

;:Délimiteur

}:Délimiteur

=== ANALYSE SYNTAXIQUE ===

--- RÉSUMÉ ---

✅ Chaine correcte - Aucune erreur detectee

BUILD SUCCESSFUL (total time: 3 seconds)

run:

=== COMPILATEUR C - PROGRAMME COMPLET ===

Entrez le code (tapez 'FIN' pour terminer) :

```
#include <stdio.h>

int main() {
   int i = 0
   while (i < 5)
      printf("%d\n", i);
      i++;

   return 0
}
```

FIN

=== LEXÈMES RECONNUS ===

#include <stdio.h>:Directive préprocesseur

int:Mot-clé

main:Mot-clé

(:Délimiteur

):Délimiteur

{:Délimiteur

int:Mot-clé

i:Identificateur

=:Opérateur

0:Nombre

while:Mot-clé

(:Délimiteur

i:Identificateur

<:Opérateur

5:Nombre

):Délimiteur

printf:Mot-clé

(:Délimiteur

"%d\n":Chaîne

,:Délimiteur

i:Identificateur

):Délimiteur

;:Délimiteur

i:Identificateur

++:Opérateur

;:Délimiteur

return:Mot-clé

0:Nombre

}:Délimiteur

=== ANALYSE SYNTAXIQUE ===

✕ Erreur : Point-virgule attendu après déclaration de '0' à 'while'

✗ Erreur : ';' attendu après return à '}'

--- RÉSUMÉ ---

✗ Chaine incorrecte - 2 erreur(s) detectee(s)

BUILD SUCCESSFUL (total time: 2 seconds)

run:

=== COMPILATEUR C - PROGRAMME COMPLET ===

Entrez le code (tapez 'FIN' pour terminer) :

```
#include <stdio.h>

int main() {
   int i = 1;
   int n = 5;

   (i <= n) {
      printf("%d\n", i);
      i++;
   }

   return 0;
}

FIN
```

=== LEXÈMES RECONNUS ===

#include <stdio.h>:Directive préprocesseur

int:Mot-clé

main:Mot-clé

(:Délimiteur

):Délimiteur

{:Délimiteur

int:Mot-clé

i:Identificateur

=:Opérateur

1:Nombre

;:Délimiteur

int:Mot-clé

n:Identificateur

=:Opérateur

5:Nombre

;:Délimiteur

(:Délimiteur

i:Identificateur

<=:Opérateur

n:Identificateur

):Délimiteur

{:Délimiteur

printf:Mot-clé

(:Délimiteur

"%d\n":Chaîne

,:Délimiteur

i:Identificateur

):Délimiteur

;:Délimiteur

i:Identificateur

++:Opérateur

;:Délimiteur

}:Délimiteur

return:Mot-clé

0:Nombre

;:Délimiteur

}:Délimiteur

=== ANALYSE SYNTAXIQUE ===

✕ Erreur : Instruction invalide à '('

✗ Erreur : Structure de contrôle manquante : 'while' attendu avant la condition à '<='

--- RÉSUMÉ ---

✗ Chaine incorrecte - 2 erreur(s) detectee(s)

BUILD SUCCESSFUL (total time: 6 seconds)