Conception et Implémentation d'un Mini-Compilateur

1. Structure et Syntaxe du Langage

1.1 Structure Générale d'un Programme

Un programme est composé de deux sections distinctes et obligatoires :

- Section de déclarations des variables
- Section d'instructions du programme

1.2 Exemple de Programme

Notre langage permet d'écrire des programmes simples comme celui-ci :

```
1  Var x: byte, y: Array[10];
2  Instructions
3  mov x, 5;
4  add x, 3;
5  print(x);
6  mov y[0], 42;
7  print(y[0]);
8  halt;
```

Ce programme illustre les éléments fondamentaux du langage :

Déclarations :

- Une variable simple 'x' de type byte
- Un tableau 'y' de 10 éléments

Instructions utilisées :

- `mov` pour l'affectation de valeurs
- `add` pour l'addition
- `print` pour l'affichage
- `halt` pour terminer le programme

Exécution pas à pas :

- Initialise x à 5
- Ajoute 3 à x (résultat : 8)
- Affiche la valeur de x (8)
- Place 42 dans la première case du tableau y
- Affiche y[0] (42)6.

Termine l'exécution

Code C généré :

```c
int main(void) {
    int8_t x = 0;
    int8_t y[10] = {0};

    x = 5;
    x += 3;
    update_flags(x);
    printf("%d\n", x);
    y[0] = 42;
    printf("%d\n", y[0]);
    exit(0);

    return 0;
}
```

Resultat :

```
PS C:\Users\firas\Desktop\micro-assembleur> python compiler.py test.src -o test.c --run
Current directory: C:\Users\firas\Desktop\micro-assembleur
Files in directory: ['.git', 'codegen.py', 'compiler.py', 'compiler_error.py', 'gi.pdf', 'grammar.bnf', 'interpreter.py', 'lexer.py', 'parse
r.py', 'semantic_analyzer.py', 'test.c', 'test.src', 'test_compiler.py', '__pycache__']
Successfully compiled to test.c

Compiling and running the program...

Program output:
8
42
```

2. Composants du Compilateur

2.1 Analyseur Lexical (Lexer)

- Découpe le code source en tokens
- Utilise des expressions régulières pour identifier les différents types de tokens
- Gère les identifiants, nombres, opérateurs et mots-clés
- Produit une séquence de tokens pour l'analyseur syntaxique

2.2 Analyseur Syntaxique (Parser)

- Construit l'AST (Abstract Syntax Tree)
- Vérifie la structure grammaticale du programme
- Produit une représentation intermédiaire du code
- Détecte les erreurs de syntaxe

2.3 Analyseur Sémantique

- Vérifie la cohérence des types
- Valide les déclarations de variables

- Vérifie les accès aux tableaux
- Détecte les erreurs sémantiques comme :
- Variables non déclarées
- Accès hors limites des tableaux
- Types incompatibles

2.4 Générateur de Code
- Génère du code C à partir de l'AST
- Gère la traduction des instructions
- Implémente les fonctionnalités de la machine virtuelle
- Produit un code C exécutable

3. Caractéristiques du Langage

3.1 Types de Données
- `byte` : entier signé 8 bits (-128 à 127)
- `Array` : tableau de bytes avec taille fixe
- Toutes les variables doivent être déclarées

3.2 Instructions Supportées
- Opérations arithmétiques : `add`, `sub`, `mult`, `div`
- Opérations logiques : `and`, `or`, `not`
- Contrôle de flux : `jmp`, `jz`, `js`, `jo`
- Manipulation de pile : `push`, `pop`
- Entrée/Sortie : `input`, `print`

3.3 Gestion de la Mémoire
- Variables globales
- Pile pour les appels de sous-programmes
- Tableaux à taille fixe
- Gestion des débordements de pile

4. Limitations et Perspectives

- **Limitations structurelles**
- Pas de fonctions/procédures véritables
- Pas de variables locales (tout est global)
- Pas de paramètres pour les sous-programmes
- **Limitations des types**
- Pas de types composés
- Pas de chaînes de caractères
- Pas de nombres flottants

- Pas de types définis par l'utilisateur
- **Limitations de contrôle**
- Pas de boucles structurées (for, while)
- Pas de structures conditionnelles (if-then-else)
- Pas de récursion véritable