Mini Projet Compilation

Omar Jabri
Mohamed Ali ben Ltayef

1. Analyse Lexicale

- **Idée Principale:**
    - **Diviser le flux d'entrée (le code source) en une séquence de lexèmes (unités lexicales), tels que les mots-clés, les identificateurs, les opérateurs, les constantes, etc.**
    - **Construire une représentation interne de chaque lexème (par exemple, un type d'énumération et une éventuelle valeur associée).**
- **Fonction Clé:**
    - `analex()`:
        - **Cette fonction centrale gère la transition entre différents états d'une machine à états finis.**
        - **Elle lit le flux d'entrée caractère par caractère et détermine le type du lexème en cours de construction.**
        - **Elle utilise des structures de contrôle conditionnelles (if-else, switch-case) pour gérer les différentes transitions d'état.**
- **Bibliothèques Utilisées:**
    - `ctype.h`: **Fournit des fonctions pour tester les propriétés des caractères (isalpha(), isdigit(), isspace(), etc.), ce qui est essentiel pour reconnaître les différents types de lexèmes.**
    - `string.h`: **Permet de manipuler des chaînes de caractères (strcpy(), strcmp(), etc.), notamment pour stocker et comparer les identificateurs.**

2. Analyse Syntaxique

- **Idée Principale:**

    - **Vérifier si la séquence de lexèmes produite par l'analyse lexicale respecte la grammaire du langage.**
    - **Construire une représentation interne de la structure du programme (par exemple, un arbre d'analyse syntaxique).**
- **Fonction Clé:**
    - `p()`:
        - **Fonction principale de l'analyse syntaxique, elle lance le processus de parsing en appelant d'autres fonctions de parsing de manière récursive.**
        - **Chaque fonction de parsing vérifie la présence et l'ordre des lexèmes attendus selon les règles de la grammaire.**
- **Méthode:**
    - **Descente Récursive: Chaque règle de la grammaire est représentée par une fonction qui appelle d'autres fonctions pour analyser les sous-parties de la règle.**

3. Table des Symboles

- **Idée Principale:**
    - **Stocker les informations sur les entités déclarées dans le programme (variables, fonctions, etc.).**
    - **Permettre la vérification de l'utilisation correcte de ces entités (par exemple, vérifier qu'une variable est déclarée avant d'être utilisée).**
- **Fonctions Clés:**
    - `addSymbol()`**: Ajoute une nouvelle entrée (nom, type, etc.) à la table des symboles.**
    - `findSymbol()`**: Recherche une entrée dans la table des symboles à partir de son nom.**

4. Gestion des Erreurs

- **Idée Principale:**
    - **Détecter et signaler les erreurs lexicales (caractères illégaux, lexèmes non reconnus), syntaxiques (violation des règles de grammaire) et sémantiques (par exemple, utilisation d'une variable non déclarée, incompatibilité de types).**

- **Fonction Clé:**
    - `erreur()`: **Cette fonction est appelée pour signaler une erreur. Elle peut afficher un message d'erreur à l'utilisateur et éventuellement interrompre le processus de compilation.**

5. Analyse Sémantique (Partielle)

- **Idée Principale:**
    - **Effectuer des vérifications sémantiques sur le code, au-delà de la simple vérification syntaxique.**
    - **Par exemple, vérifier la compatibilité des types lors des affectations, détecter l'utilisation de variables non initialisées, etc.**
- **Fonction Clé:**
    - `checkAssignmentCompatibility()`: **Vérifie que le type de l'expression à droite d'une affectation est compatible avec le type de la variable à gauche.**

Bibliothèques Standard Utilisées:

- `stdio.h`: **Fournit des fonctions pour l'entrée/sortie standard (e.g.,** `fopen()`, `fclose()`, `fprintf()`, `fgetc()`**).**
- `stdlib.h`: **Fournit des fonctions de gestion de la mémoire (e.g.,** `malloc()`, `free()`**) et des fonctions mathématiques (e.g.,** `atoi()`**).**

Résumé

Le code fourni implémente un compilateur basique pour un langage simple. Il comprend les étapes suivantes :

1. **Analyse Lexicale: Le code est divisé en une séquence de lexèmes.**
2. **Analyse Syntaxique: La structure du code est vérifiée par rapport à la grammaire du langage.**
3. **Analyse Sémantique (Partielle): Des vérifications sémantiques de base sont effectuées.**

Ce compilateur constitue une base pour des développements futurs, tels que l'ajout de la génération de code, l'amélioration de l'analyse sémantique, la mise en place d'une meilleure gestion des erreurs et l'optimisation du code généré.