Plan de la demo

Auteurs :
Mehdi El Idrissi El Fatmi
Khadija El Adnani
Alaa Jennine
Badr El Khoundafi
Yassine El Kouri

22/01/2025

January 24, 2025

Contents

1 Introduction

Ce document a été conçu pour servir de plan détaillé à la démonstration de notre compilateur Deca. Il décrit les étapes clés et les fonctionnalités principales à présenter, en mettant en évidence les aspects techniques et méthodologiques du projet.

L'objectif de ce plan est de structurer la présentation afin de montrer, de manière claire et progressive, le fonctionnement et les capacités du compilateur, tout en répondant aux attentes définies pour cette démonstration. Chaque étape de la démonstration sera accompagnée d'explications détaillées.

2 Présentation du compilateur

Cette partie servira à introduire brièvement le compilateur Deca . Nous commencerons par une description simple de ses fonctionnalités principales, suivie d'une présentation synthétique de ses caractéristiques importantes.

Ensuite, nous détaillerons les grandes étapes du fonctionnement du compilateur, notamment l'analyse lexicale, l'analyse syntaxique, la vérification contextuelle, et la génération de code assembleur. Chaque étape sera abordée de manière concise pour donner une vue d'ensemble avant de passer aux démonstrations détaillées.

3 Analyse lexico-syntaxique

Dans cette partie de la démonstration, nous présenterons les fonctionnalités principales de l'analyse lexicale et syntaxique du compilateur Deca.

Nous expliquerons d'abord comment l'analyse lexicale identifie les unités lexicales (tokens) telles que les mots-clés, les identifiants, et les opérateurs, en s'assurant qu'ils respectent les règles définies par la grammaire du langage Deca. Ensuite, nous décrirons le rôle de l'analyse syntaxique, qui vérifie que la structure du programme suit les règles syntaxiques spécifiées, en construisant un arbre syntaxique abstrait .

La démonstration inclura des exemples concrets de programmes sources avec ou sans objets, mettant en évidence la manière dont l'analyse lexico-syntaxique détecte des erreurs ou produit un arbre syntaxique valide en sortie.

4 Analyse contextuelle

Dans cette partie de la démonstration, nous mettrons en avant les fonctionnalités principales de l'analyse contextuelle du compilateur Deca.

Nous expliquerons d'abord le rôle de cette étape, qui consiste à vérifier que les règles sémantiques du langage sont respectées. Cela inclut, par exemple, la vérification des déclarations et des types des variables, la compatibilité des types dans les expressions, ainsi que la gestion des portées et des méthodes des classes.

La démonstration illustrera ces points à l'aide de programmes contenant des erreurs contextuelles, comme l'utilisation de variables non déclarées, des appels à des méthodes inexistantes ou des incompatibilités de types. Ces exemples permettront de montrer comment le compilateur détecte et signale ces erreurs. Nous conclurons avec un exemple valide pour démontrer le bon fonctionnement de cette étape dans un cas sans anomalies.

5 Génération de code assembleur

Dans cette partie de la démonstration, nous aborderons le processus de génération de code assembleur par le compilateur Deca.

Nous commencerons par expliquer les principes de cette étape, qui consiste à traduire le code source Deca en instructions assembleur compréhensibles par la machine. Cette étape inclut la gestion des registres, la mise en place de la pile pour les appels de fonctions, ainsi que la génération des instructions nécessaires pour manipuler les variables, les expressions et les structures de contrôle.

La démonstration comprendra des exemples concrets montrant le passage du code source Deca à son équivalent assembleur. Nous mettrons en avant des cas particuliers, comme la gestion des boucles, des conditions, et des appels de méthode. Ces exemples permettront de démontrer la validité du code généré et sa capacité à être exécuté efficacement sur une architecture cible. Enfin, nous présenterons un programme complet compilé et exécuté pour illustrer le résultat final.

6 Améliorations possibles

Bien que notre projet soit bien avancé et atteigne la majorité des objectifs fixés, certaines améliorations restent envisageables. Par exemple, des fonctionnalités avancées du langage Deca, notamment en lien avec la programmation orientée objet, pourraient être approfondies pour enrichir encore davantage le compilateur.

7 Extension ARM

Nous avons également progressé sur l'extension ARM du projet. Une démonstration spécifique à cette extension sera réalisée, mettant en avant les fonctionnalités implémentées et les résultats obtenus.

8 Impact énergétique

L'impact énergétique a été pris en compte dans le développement du compilateur.

9 Gestion et bilan de l'équipe

Dans cette partie, nous détaillerons notre méthodologie de travail, la répartition des tâches entre les membres de l'équipe, ainsi que les outils utilisés tout au long du projet. Nous expliquerons comment la coordination a été assurée, les défis rencontrés dans la gestion collaborative, et les solutions apportées pour garantir un avancement harmonieux du projet.