
Rapport du rendu intermédiaire
PLD Compilateur

Hexanome H4212 - BOUVIER Julien, BOYER Maxime, CRISTOFORONI Stella, DELEGLISE Benoit, GUIGAL Allan, MARC Quentin, TOURNADE Aurélien

---

I. Introduction

Ce document décrit le rendu intermédiaire de notre compilateur. A l'heure actuelle, notre compilateur fonctionne de bout en bout sur un petit sous-ensemble du langage C. Notre compilateur possède les fonctionnalités exigées pour la première itération.

II. Description des fonctionnalités implémentées

a. Sous ensemble accepté

Notre compilateur est capable de comprendre un programme C composé:

- d'une seule fonction (le main)
- d'initialisation de variables (pas forcément sur une seule ligne, les variables peuvent être initialisées n'importe où)
- d'affectations (var = valeur ou var = var)
- d'opérations arithmétiques
- d'un retour

Ainsi, notre compilateur est capable d'accepter tout programme C semblable à celui présenté en page 15 du sujet.

b. Gestion des erreurs

La gestion des erreurs commence à être mise en place mais nous n'en sommes qu'au début; d'autant plus que le développement de cette fonctionnalité a été ralentie par des conflits d'architecture. Nous avons commencé par la gestion des erreurs sémantiques avec par exemple la détection de tentatives:

- de déclaration d'une variable alors qui aurait déjà été déclarée
- d'assignation de la valeur d'une variable à une autre variable alors que la variable assignante n'a été ni déclarée ni définie (ex: int a = b alors que b n'existe pas)

c. Tests

Lors du lancement des tests, il est possible que le shell (sh) de ifcc-wrapper.sh ne trouve pas certains fichiers sur la machine. Si cela est le cas, il faut remplacer !#/bin/sh par !#/bin/bash dans la 1ere ligne de ifcc-wrapper.sh.
Les deux tests qui ne fonctionnent pas sont celui où l'on doit lancer une erreur puisqu'il y a redéfinition d'une variable et celui dans lequel nous détectons une erreur lorsque que nous assignons un flottant alors que gcc fait le cast automatiquement sans erreur.

III. Structure du code

Voici un schéma décrivant l'architecture des différentes composantes de notre programme:

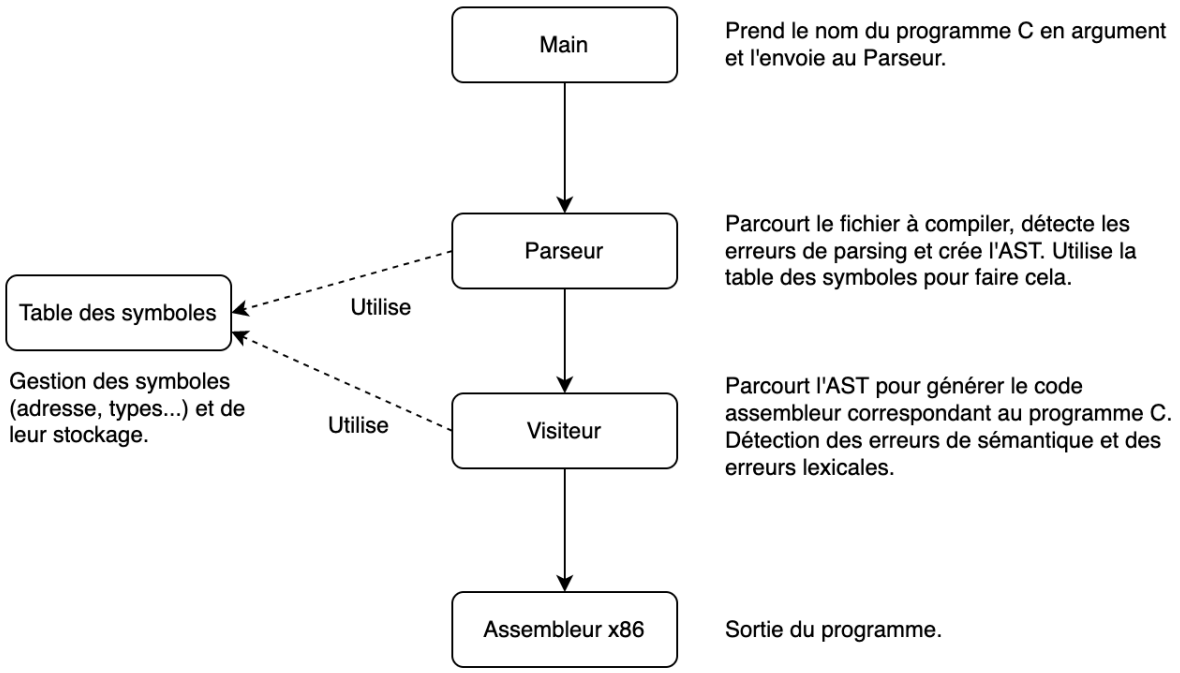

Figure 1: Architecture de la version intermédiaire de notre compilateur.

IV. Gestion du projet

Notre groupe s'est organisé de la manière suivante:

- Un groupe de 3 personnes s'est focalisé sur la création de la table des symboles. Progressivement, une personne a consacré de plus en plus de temps à la création de tests de la table des symboles pendant que les autres continuaient la création de la table.

- Un groupe de 4 personnes s'est concentré sur la création du reste et a progressivement évolué pour que chaque personne se focalise sur une fonctionnalité spécifique (gestion des exceptions, parser, génération du code assembleur).

Pour la suite du projet, le travail va être de plus en plus divisé entre les membres de l'hexanome afin que la création des différentes fonctionnalités avancent en parallèle.

V. Conclusion

Nous perdons beaucoup de temps à gérer les différences d'architecture de nos machines (mac, Windows) et les différences d'installations d'antlr. Malgré une gestion attentive du repo git, ces problèmes sapent une part importante de notre temps de travail.

Toutefois, nous sommes dans les temps. Les fonctionnalités demandées ont été créées et nous pensons pouvoir répondre aux futures exigences.