Cahier des charges

Compilateur RUST

Sany BAKIR, Bastian LEBRUN
Marcus RABOURDIN, Louis-Alexandre CHARNAY

Janvier 2025

Contents

1 Introduction

Ce projet vise à concevoir et développer un compilateur pour le langage Rust. À travers ce projet, nous avons pour objectif de mieux comprendre les différents processus nécessaires à la création d'un compilateur, allant de l'analyse lexicale et syntaxique à la génération de code machine. En travaillant sur ce projet, nous chercherons à explorer en profondeur les spécificités du langage Rust tout en appliquant des concepts d'algorithmie avancée. Ce projet nous permettra de renforcer notre expertise en programmation Rust et d'améliorer nos compétences en travail collaboratif, qui sont essentielles pour notre avenir.

Au-delà des aspects techniques, ce projet représente un défi en termes d'organisation et de travail d'équipe. Travailler ensemble sur un projet aussi complexe nous permettra d'apprendre à mieux communiquer et à partager nos idées pour atteindre un objectif commun. Nous aurons également l'occasion de nous familiariser avec les outils qui facilitent la gestion d'un projet collaboratif et de suivre une bonne organisation du travail. Ce projet nous permettra ainsi d'acquérir des compétences utiles pour de futurs projets en groupe et de mieux comprendre les défis de la création d'un langage de programmation.

2 Etat de l'art

2.1 Lexing

Pour le lexer, de nombreuses technologies existent, afin de répondre aux problèmes que pose l'analyse lexicale d'un langage, telles que la segmentation du texte en tokens ou la gestion des erreurs lexicales. Certains outils commme Logos, permettent de définir et d'extraire efficacement des tokens à l'aide de règles simples. Tandis que l'outil ANTLR, fournit une solution pour générer un lexer et gérer les erreurs lexicales efficacement.

2.2 Parsing

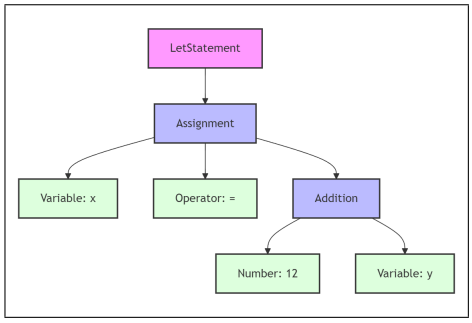
Le parser est l'outil qui permet de vérifier la syntaxe d'un code. Actuellement, il existe différentes solutions. L'analyse déterministe avec par exemple l'analyse descendante qui va regarder un à un les caractères et vérifier s'ils correspondent à une règle qu'on a défini ; ou encore l'analyse ascendante, qui permet de regarder des chaines de caractère. Cependant, dans les langages, il peut y avoir des ambiguïtés. On peut donc utiliser des analyses non déterministes, comme l'analyse CYK. Le principe est de vérifier chaque caractère s'ils respectent la syntaxe définie puis de rassemblé petit à petit les caractères et voir s'il n'y a pas de problème. Et on continue à regrouper les caractères jusqu'à ce qu'on ait le code en entier. S'il n'y a pas de problème alors on pourra passer à l'étape suivante de la compilation. On va utiliser cette solution dans notre cas pour le compilateur rust.

2.2.1 AST

Dans un compilateur, le parsing est une étape fondamentale. Il consiste à analyser le code source pour vérifier sa syntaxe et sa conformité avec les règles de la grammaire du langage. Un outil clé dans cette étape est l'AST (Abstract Syntax Tree, ou arbre de syntaxe abstraite), qui constitue la base pour les étapes suivantes du processus de compilation, telles que l'analyse sémantique et l'optimisation du code.

Avant d'effectuer le parsing, le processus repose sur une étape essentielle : le lexing. Lors du lexing, le code source est transformé en une série de tokens, représentant les différents composants du programme, tels que

les mots-clés, les identifiants et les opérateurs. Cela permet de simplifier l'analyse en réduisant le code source à une liste d'éléments fondamentaux.
Prenons un exemple de code source : `let x = 12 + y` nous pouvons obtenir comme série de tokens : LET, IDENTIFIER("x"),EQUALS— NUMBER(12), PLUS, IDENTIFIER("y") Une fois les tokens générés, un arbre de syntaxe abstraite (AST) peut être construit pour représenter la structure hiérarchique de l'expression. L'arbre montre que l'opération d'addition est l'opérateur principal, avec 12 et y comme opérandes. Les nœuds internes de l'arbre représentent des opérateurs, tandis que les feuilles sont les opérandes.



2.2.2 Gestion d'erreurs

Une étape cruciale lors du processus de parsing est la gestion d'erreur. La gestion d'erreur permet d'identifier et de signaler précisément les erreurs de syntaxe dans le code source. Lorsque le parser identifie une erreur syntaxique, il doit non seulement l'identifier, mais aussi apporter des messages clairs pour aider le développeur à la corriger.
Une bonne gestion des erreurs implique des stratégies avancées, telles que la récupération d'erreurs. (via des technique de backtracking) et la fourniture de suggestions pour corriger les erreurs syntaxiques.

2.3 Backend

Le backend d'un compilateur est une étape importante du processus de compilation.
En effet, le backend a pour rôle de générer du code machine exécutable à partir de la représentation intermédiaire (un format abstrait qui sert de pont entre le langage source et le code machine), de garantir l'exécution du code sur différentes architectures et d'appliquer des optimisations pour améliorer

les performances et réduire la taille du code.
Parmi les technologies disponibles, LLVM apparaît comme un choix pertinent. Il s'agit d'une infrastructure modulaire de compilation, c'est-à-dire un ensemble d'outils indépendants et réutilisables permettant de construire différentes étapes d'un compilateur. Grâce à son API flexible (interface de programmation), qui offre une interface adaptable pour manipuler et optimiser le code, LLVM permet de générer et d'optimiser efficacement du code bas niveau. Il est d'ailleurs utilisé par de nombreux compilateurs modernes, tels que Clang et Rust.

3 Présentation de l'équipe

3.1 LEBRUN Bastian

Je m'appelle Bastian Lebrun, je suis en deuxième année à EPITA. J'avais redoublé ma première année, mais je suis toujours autant passionné par l'informatique. Ceci sera mon troisième projet au sein de l'école, et n'ayant pas réussi à obtenir un bon résultat pour le deux premier, je suis motivé pour faire de celui-ci une réussite et je vais tout mettre en œuvre pour. De plus, le sujet que nous avons choisi me parait intéressant et amusant.

3.2 BAKIR Sany

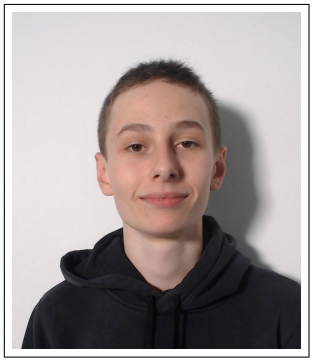
Mon intérêt pour l'informatique s'est progressivement développé, en particulier pendant mes années de lycée. En suivant les spécialités Numérique et Science de l'Informatique ainsi que la spécialité Mathématiques, j'ai pu renforcer ma passion pour le développement. En parallèle, j'ai pu entreprendre des projets

personnels dans divers langages de programmation.
J'ai donc décidé de rejoindre l'EPITA afin d'approfondir mes connaissances et compétences dans le domaine du développement et de l'informatique de manière générale. Malgré le fait que je ne possède pas de compétence dans le développement de compilateur, ce projet est une opportunité d'acquérir des compétences essentielles dans le développement, mais aussi en gestion de projet et en travail d'équipe.

3.3 CHARNAY Louis-Alexandre

Passioné par l'informatique et ses applications, je cherche à appronfondir mes notions pour aider à la résolution de problèmes complexes.
Egalement, je recherche, à travers l'enseignement et la pédagogie fournis par Epita, à participer activement à la création d'un projet avec le groupe CHAOS team. Cette expérience me permettra de développer mes compétences analytiques, ma réflexion et mon intérêt pour les principes scientifiques tout en travaillant en groupe.

3.4 RABOURDIN Marcus



Depuis mon plus jeune âge, j'ai toujours eu accès à un ordinateur. Cependant, ma véritable passion est née lorsque je me suis posé la plus innocente des questions : Comment fait-on un site web ? Je me suis donc renseigné et de fil en aiguille est apparue une volonté de comprendre ce monde de l'informatique, puisque que c'était celui que j'utilisais depuis mon enfance. Il m'a semblé naturel de me tourner vers une carrière dans ce domaine et en tant que passionné, rejoindre l'Epita était lié l'utile à l'agréable. Grâce à mes expériences personnelles et à l'enseignement reçu, je serais dans la capacité de rejoindre la CHAOS Team afin d'aider ce groupe dans la bonne réalisation de notre projet.

4 Répartition et Planning

4.1 Répartition

Tâches	Bastien	Louis-Alexandre	Marcus	Sany
Lexer (structure de données)			R	
Lexer (logique d'analyse)			R	
Parser (AST)	S			R
Parser (gestion d'erreurs)	R			S
Backend		R	S	
Merge	S	S	R	S
Site Web	R	S	S	S

R = Responsable — S = Suppléant

4.2 Planning

Soutenances	1ère Soutenance	2ème Soutenance	Soutenance finale
Lexer	40%	80%	100%
AST	20%	60%	100%
Gestion d'erreurs	30%	70%	100%
Backend	30%	60%	100%
Merge	10%	30%	100%
Site web	100%	100%	100%

5 Conclusion

En conclusion, ce projet nous offre une opportunité unique d'explorer en profondeur le langage Rust tout en consolidant nos compétences en programmation. A travers les différentes étapes de conception et d'implémentation de ce compilateur, nous allons développer une compréhension approfondie du fonctionnement des langages compilés, notamment en ce qui concerne les mécanismes de lexing, de parsing, d'analyse syntaxique, et du langage assembleur.

Ce projet nous permet, une fois de plus, d'améliorer notre gestion de projet en équipe. Le travail collaboratif va nous permettre d'apprendre à mieux coordonner nos efforts, et de manière plus globale à mieux gérer les travaux de groupe.