# Compiler for MiniRacket in Rust

Team Members: Abhay Patel & Mohammad Ghaemi

## Introduction

A compiler is a program that maps some source code from one programming language to another programming language (this is called the target language). Usually, the target language is a lower level programming language. The intent behind this is to use the target language to build a higher level language with more features. We will be borrowing many ideas from CMSC430, an introductory class for implementing compilers at the University of Maryland.[1]

This process will involve a few steps. We will first need to create an Abstract Syntax Tree (AST), which is a data structure allowing for representations of expressions in the language we are making (source language) a compiler for. This AST representation of our program can then be fed through either an interpreter (which would evaluate the parsed code in Rust) or through a compiler (which would convert our source language's code into our target language's instructions).[1]

In order to parse the source language, there are a few steps involved. First, a lexer must be created which can tokenize the source code. This token stream would then be passed through the parser, where recursive descent parsing (or a variety of parsing strategies) would allow for this token stream to be converted into the defined AST.[2] There are many different types of parsers, including recursive descent, non-recursive descent, LR parsers, and operator precedence parsers.[3]

## Goals

### 75%

- Define our source language to be a subset of Racket which includes the following features:
  - Support for integers, booleans, and characters
  - Simple arithmetic operations such as incrementing, decrementing, and two integer addition and subtraction
  - Conditional expressions (if-else statements)
  - Support for variable bindings and lexical scoping
- Create a tokenizer and a parser which can accept code of our source language.
- Create a compiler which can convert our target language into x86 Assembly
- Create a runtime system in C to run outputted assembly files

### 100%

- Define our source language to be a subset of Racket which includes the following features:
  - Support for integers, booleans, and characters
  - Simple arithmetic operations such as incrementing, decrementing, and two integer addition and subtraction

    - Conditional expressions (if-else statements)
    - Support for variable bindings and lexical scoping
    - Support for lists and the binary `cons` operator for list construction
    - Support for boxed values and the unary operators `box` and `unbox`
    - Error checking to provide detailed error messages and prevent program crashing without explanation
- Create a tokenizer and a parser which can accept code of our source language.
- Create a compiler which can convert our target language into x86 Assembly
- Create a runtime system in Rust to run outputted assembly files

### 125%

- Define our source language to be a subset of Racket which includes the following features:
    - Support for integers, booleans, and characters
    - Simple arithmetic operations such as incrementing, decrementing, and two integer addition and subtraction
    - Conditional expressions (if-else statements)
    - Support for variable bindings and lexical scoping
    - Support for lists and the binary `cons` operator for list construction
    - Support for boxed values and the unary operators `box` and `unbox`
    - Error checking to provide detailed error messages and prevent program crashing without explanation
    - Support for vectors and strings using the developed heap
    - Support for top-level function declarations and function calls
- Create a tokenizer and a parser which can accept code of our source language.
- Create a compiler which can convert our target language into x86 Assembly
- Create a runtime system in Rust to run outputted assembly files

## Specific Aims & Objectives

---

With this project, we aim to learn about programming language paradigms, compiler structure, parsing techniques, and Rust language features by creating a compiler for a subset of the Racket programming language. We seek to expand our knowledge of compilers which we have gained through CMSC430 and apply the skills we have learned by creating a similar compiler in a different language.

We will not be implementing very complex processes such as garbage collection in our heap (as we have not learned how to do this in CMSC430), but we will specifically try to implement each feature defined in the above stated goals and create a somewhat robust error checker for improved user experience.

## Cited References

---

1. https://www.cs.umd.edu/class/fall2021/cmsc430/
2. https://www.cs.umd.edu/class/fall2020/cmsc330/lectures/20-parsing.pdf
3. https://www.geeksforgeeks.org/types-of-parsers-in-compiler-design/