

# Project Milestone 1

---

Team Members: Abhay Patel & Mohammad Ghaemi

## What we have accomplished

---

- Defined the Abstract Syntax Tree (AST) for a small subset of our source language MiniRacket (based on the Racket programming language)
- Created a lexer which takes raw text (string) and converts it into a list of tokens
- Created a parser which takes a list of tokens and converts it into an expression defined by our AST
- Began defining enums & structures to represent x86 assembly instructions
- Created boilerplate code for compiler, which converts an expression defined by our AST into a list of x86 assembly instructions
- Created a language with the single feature of Integer literal values

## What we will do next

---

We will update a86.rs to include enum representations for the necessary x86 assembly instructions that we do not already have. We will also be adding structures to represent x86 programs and sequences of x86 instructions. This is our immediate focus so that our compiler can be filled in to produce the a86 representations of x86 instructions. The wrappers exist for abstraction and we add functionality to the enums/structures to produce strings of instructions.

Now that we have created most of the baseline for our compiler (i.e. created boilerplate code for lexer, parser, and compiler), we want to add features to our language. We will follow the goals set in our proposal as we add sets of features to our language. This will require updating each component of our system, including the AST, token types, lexer, parser, and compiler. Here is a list of features that we plan to add:

- Boolean literal values
- Character literal values
- Unary operations including `add1` and `sub1`
- Binary operations including `+` and `-`
- Conditional expressions (i.e. if-else expressions)
- Variable binding with lexical scope through `let` expressions

These are the minimum set of features we will implement, however we plan on adding additional features including:

- List values including the empty list literal
- The binary `cons` operator for list construction
- Boxed values
- Unary operations `box` and `unbox`
- Expand on existing error checking/messages
- Vector and string values using a developed heap
- Top level function declarations and function calls

We also do not yet have a runtime system for our programming language. We will begin by using the runtime system from CMSC430, which is written in C. We will simply copy over the runtime system and make any minor changes to accept the output of our compiler. However, one of our advanced goals is to create a runtime system using Rust, so if time permits, we will replicate the C code in Rust.

## Unexpected obstacles

---

We have not run into many obstacles yet. Much of the work we have completed required us to revisit some concepts we learned in CMSC330 (i.e. tokenizing raw text and creating a recursive descent parser). We were able to reference the lecture notes from CMSC388Z along with the extensive Rust documentation to implement our base compiler system, and have not yet run into any major issues. The most difficult aspect of this project seems to be interfacing with assembly, and we are in the process of resolving this by creating enums and structures to interface with x86 assembly.

## Which goal we expect to meet

---

We definitely expect to meet our 75% goal, and likely will be able to reach our 100% goal without much issue. Once we get our base language (with only integer literals) to both compile and run (using CMSC430's runtime system), we simply need to add features to the language and will not have to make major structural changes to our code. We hope to achieve our 125% goal, however will primarily focus on achieving our 100% goal. We anticipate the most challenging aspects to be the x86 interfacing as well as implementing the runtime system using Rust.

## Link to project repository

---

Our project repository can be found here: <https://github.com/apatelcs/CMSC388Z-Final-Project>