Members: Oscar So (ons4) & Aidan Chalnick (ac2597) & Tushar Khan (tak62)

Meeting Plan: MWF 3-5pm

Statement: Create a compiler in OCaml code that parses our own creative programming language and executes it in OCaml.

Key Features:

- Tokenizer
    - New language is mainly text? (work with NLP)
    - Breaks text file into keywords
    - Nodes for tokens for different evaluations
    - Lexer
- Parser
    - Interprets tokens and builds an abstract syntax tree
        - Will discuss about what order tree is evaluated (probably in-order)
- Function, Variables, Operators, Overload, Recursion (tail recursion -> acc?), new function types, etc...
- Could have built-in functions that do special things within a program

Narrative Description: We want to build an OCaml compiler for a custom language. We will design our custom language to support many common and expected language features such as variable assignment, functions (anonymous, recursive, etc.), operators, types, etc. Additional language features will be added as we realize their value. Our language is going to be an imperative language that gets executed “under-the-hood” in OCaml.

Roadmaps:
0-2 Weeks - MS1

SATISFACTORY:

Create GitHub repo.

Design the custom language and its functionalities. How is it different than other languages? What features does it have that make it beneficial to use?

We will define a formal language for our grammer.

GOOD:

Begin the tokenizer. Write in functions that read in text files of our language and arrange them in a format interpretable by OCaml.

EXCELLENT:

Begin writing the interpretation of the language.

- Variable Assignment
- Basic Operators
- Basic Types

2-4 Weeks - MS2

Begin writing parser

- Add support for functions
- Add support for recursion
- Add support for everything in the tokenizer already

Recursion

Utility Package

4-6 Weeks - MS3

Write the parser that reads in what the tokenizer has produced and actually evaluates what is written.

MOAR LANGUAGE FEATURES

Anything we decide to put into the language/tokenizer must be matched by functionality in the parser

Sketch:

- What are the important modules that will be implemented? What is the purpose of each module?
    - Tokenizer
        - Breaks apart the input .txt file into tokens; can implement some syntax checking system here
    - Parser
        - Parses the tokens into an AST; can implement some semantics checking system here
    - Evaluator
        - Evaluates and executes the AST to run the progra
- What data will your system maintain? What formats will be used for storage or communication? What data structures do you expect to use as part of your implementation?
    - .txt file for our input. Data structures: Trees (for the AST), list (to store data), association list (to store data/evaluate tokens).

- What third-party libraries (if any) will you use?
    - Not planning to but we will if the need arises.
- How will you test your system throughout development? What kinds of unit tests will you write? How will you, as a team, commit to following your testing plan and holding each other accountable for writing correct code?
    - At each checkpoint of language/tokenizer/parser functionality we will write code in our custom language that makes sure each piece of functionality is tested in all of its edge cases in addition to all other tests.