**Language Design Proposal: Reclaimer**

**Student Name(s):** Kyle Dewey

**Language Name:** Reclaimer

**Target Language:** C

**Language Description:** A basic language with structs and garbage collection. Similar to Refcount, but with a syntax based on S-expressions, and structs are now mutable.

**Key Features:** Structs, garbage collection.

**Planned Restrictions:** All structs are heap allocated.

**Suggested Scoring and Justification:**

* **Lexer**: 10%. Only support for reserved words, identifiers, and integers. No comments.
* **Parser**: 10%. Syntax based on S-expressions.
* **Typechecker**: 15%. Very basic.
* **Code Generator**: 65%. Most of the challenge is expected in writing a garbage collector. This will entail:
	+ Maintaining your own custom heap, wherein structs are allocated.
	+ Needing to keep track of struct variables on the stack, in order to gather a root set.
	+ Maintaining an additional field on each struct for a "mark" phase of a garbage collector.
	+ Reclaiming objects during a "sweep" phase of a garbage collector.
	+ Depending on the implementation, maintaining a table at runtime mapping old memory addresses of objects to new memory addresses (necessary for heap compaction).

**Syntax:**

var is a variable

structname is the name of a structure

funcname is the name of a function

i is an integer

type ::= `int` | `bool` | **Integers and booleans are types**

 `void` |

 structname | **Structures are a type**

param :: = `(` type var `)`

**Structs**

structdef ::= `struct` structname `(` param\* `)` `)`

**Functions**

fdef ::= `(` `func` funcname `(` param\* `)` type stmt\* `)`

**Left-hand side. Something we can assign to.**

lhs ::= var | `(` `.` lhs var `)`

struct\_param ::= `(` var exp `)`

exp ::= i | `true` | `false` | **Integers and booleans**

`null` | **Null; assignable to struct types**

 lhs | **Something we can access**

 **Allocate a new struct**

 `(` `new` structname `(` struct\_param\* `)` `)` |

 **Function calls**

 `(` `call` funcname exp\* `)`

stmt ::= `(` `vardec` type var exp `)` | **Variable declaration**

 `(` `=` lhs exp `)` | **Assignment**

 `(` `if` exp stmt [stmt] `)` | **if**

 `(` `while` exp stmt `)` | **while**

 `break` | **break**

 `(` `println` exp `)` | **Printing something**

 `(` `block` stmt\* `)` | **Block**

 `(` `return` [exp] `)` | **Return**

`(` `stmt`exp `)` **Expression statements**

program ::= structdef\* fdef\* stmt\* **stmt\* is the entry point**

**Example (length of a linked list):**

(struct Node

 ((int value)

 (Node rest)))

(func length ((Node list)) int

 (vardec int retval 0)

 (while (!= list null)

 (block

 (= retval (+ retval 1))

 (= list (. list next))))

 (return retval))

(vardec Node first

 (new Node

 (value 0)

 (next null)))

(vardec Node second

 (new Node

 (value 1)

 (next null)))

(vardec Node third

 (new Node

 (value 2)

 (next null)))

(= (. first next) second)

(= (. second next) third)

(println (call length first))