**Language Design Proposal: Refcount**

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

**Language Name:** Refcount

**Target Language:** C

**Language Description:** A basic language with structs and a clean(ish) syntax. Has built-in reference counting, allowing for memory reclamation. Structs are immutable.

**Key Features:** Structs, reference counting.

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

**Suggested Scoring and Justification:**

* **Lexer**: 10%. Only support for reserved words, identifiers, and integers. No comments.
* **Parser**: 20%. Syntax isn't based on S-expressions.
* **Typechecker**: 15%. Very basic.
* **Code Generator**: 55%. All structs internally have components for reference counting.

**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

comma\_param ::= [param (`,` param)\*]

**Structs**

structdef ::= `struct` structname `{` (param `;`)\* `}`

**Functions**

fdef ::= `func` funcname `(` comma\_param `)` `:` type

 `{` stmt\* `}`

struct\_actual\_param ::= var `:` exp

struct\_actual\_params ::=

 [struct\_actual\_param (`,` struct\_actual\_param)\*]

comma\_exp ::= [exp (`,` exp)\*]

primary\_exp ::=

 i | `true` | `false` | var | **Integers, booleans, and variables**

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

`(` exp `)` | **Parenthesized expressions**

 **Allocate a new struct**

 `new` structname `{` struct\_actual\_params `}` |

 **Function calls**

funcname `(` comma\_exp `)`

**Accessing the field of a struct or calls**

dot\_exp ::= primary\_exp (`.` var)\*

mult\_exp ::= dot\_exp ((`\*` | `/`) dot\_exp)\*

add\_exp ::= mult\_exp ((`+` | `-`) mult\_exp)\*

less\_than\_exp ::= add\_exp [`<` add\_exp]

equals\_exp ::= less\_than\_exp ((`==` | `!=`) less\_than\_exp)\*

exp ::= equals\_exp

stmt ::= type var `=` exp `;` | **Variable declaration**

 var `=` exp `;` | **Assignment**

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

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

 `break` `;` | **break**

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

 `{` stmt\* `}` | **Block**

 `return` [exp] `;` | **Return**

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 {

 int retval = 0;

 while (list != null) {

 retval = retval + 1;

 list = list.next;

 }

 return retval;

}

Node list =

 new Node {

 value: 0,

 rest: new Node {

 value: 1,

 rest: new Node {

 value: 2,

 rest: null

 }

 }

 };

println(length(list));