**Language Design Proposal: Lowlang Structs**

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

**Language Name:** Lowlang Structs

**Target Language:** MIPS Assembly

**Language Description:** A very restricted, low-level language that compiles to MIPS assembly. Intended to explore how things can compile to assembly.

**Key Features:** Pointers, structs, expressions.

**Planned Restrictions:** Only stack allocation.

**Suggested Scoring and Justification:**

* **Lexer**: 10%. Only support for reserved words, identifiers, and integers. No comments.
* **Parser**: 10%. Uses S-expressions.
* **Typechecker**: 15%. Need to handle pointers.
* **Code Generator**: 65%. Compiles expressions down to assembly. Structs will likely be non-trivial to handle.

**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` | **Integers are a type**

`void` |

structname | **Structures are a type**

`(` `\*` type `)` **Pointer to something of type**

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

**Structs**

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

**Functions**

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

**Left-hand side of an assignment. Denotes a place where something can be assigned.**

lhs ::= var | **Assignment to a variable**

`(` `.` lhs var `)` | **Assignment to a field of a struct**

`(` `\*` lhs `)` **Assignment to something at an address**

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

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

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

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

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

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

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

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

**Arithmetic and relational operators**

op ::= `+` | `-` | `\*` | `/` | `<` | `==` | `!=`

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

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

lhs | **Accessing something in memory**

`(` `&` lhs `)` | **Address-of something in memory**

`(` `\*` exp `)` | **Dereference something**

`(` op exp exp `)` |

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

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

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

(struct Node

(int value)

((\* Node) next))

(func length (((\* Node) list)) int

(vardec int retval)

(assign retval 0)

(while (!= list null)

(assign retval (+ retval 1))

(assign list (. (\* list) next)))

(return retval))

(vardec Node first)

(vardec Node second)

(vardec Node third)

(assign (. first value) 1)

(assign (. first next) (& second))

(assign (. second value) 2)

(assign (. second next) (& third))

(assign (. third value) 3)

(assign (. third next) null)

(println (call length (& first)))