**Language Design Proposal: Lowlang Function Pointers**

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

**Language Name:** Lowlang Function Pointers

**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, function pointers, 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**: 20%. Need to handle function pointers. Also need to disambiguate between calling a function directly and calling a function through a function pointer.
* **Code Generator**: 60%. Compiles expressions down to assembly. Function pointers will likely be non-trivial to handle.

**Syntax:**

var is a variable

i is an integer

type ::= `int` | **Integers are a type**

 `void` |

 `(` `func` `(` type\* `)` type `)` **Function pointer**

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

**Functions**

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

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

 `(` `assign` var 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, booleans**

var | **Variables**

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

 `(` `&` var `)` | **Getting the address of a function**

 `(` op exp exp `)` |

 **Function call. The first exp will either be a function**

 **name, or will evaluate to a function pointer. The**

 **typechecker must disambiguate between the two.**

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

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

**Example (indirectly calling a function):**

(func add ((int x) (int y)) int

 (return (+ x y)))

(func mult ((int x) (int y)) int

 (return (\* x y)))

(func choose ((int boolean)

 ((func (int int) int) f1)

 ((func (int int) int) f2))

 (if boolean

 (return f1)

 (return f2)))

(vardec (func (int int) int) f)

(assign f (call choose 0 (& add) (& mult)))

(println (call f 3 4))