**Language Design Proposal: Traitor**

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

**Language Name:** Traitor

**Target Language:** JavaScript

**Language Description:** A Haskell-like language featuring typeclasses, function overloading, and higher-order functions, which compiles to JavaScript.

**Key Features:** Typeclasses, structs, higher-order functions, function overloading, syntax not based on S-expressions.

**Planned Restrictions:** No generics, no algebraic data types, or any other way to make a recursive type. This dramatically simplifies things, as we will always know what method is being referred to at compile-time.

**Suggested Scoring and Justification:**

* **Lexer**: 10%. Only support for reserved words, identifiers, and integers. No comments.
* **Parser**: 20%. Does not use S-expressions.
* **Typechecker**: 40%. Higher-order functions, typeclasses, structs, named function overloading.
* **Code Generator:** 30%. Most of the work is expected to be in handling typeclasses and function overloading, as JavaScript doesn't support these.

**Concrete Syntax:**

var is a variable

structname is the name of a struct

traitname is the name of a trait (typeclass)

i is an integer

comma\_type ::= [type (`,` type)\*]

type ::=

`Int` | `Void` | `Boolean` | **Built-in types**

`Self` | **Refers to our own type in a trait**

structname | **Structs are a valid kind of type**

`(` type `)` | **Parenthesized type**

`(` comma\_type `)` `=>` type **Higher-order function**

param ::= var `:` type

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

structdef ::= `struct` structname `{` comma\_param `}`

**Definition of an abstract method**

abs\_methoddef ::= `method` var (` comma\_param `)` `:` type `;`

**Definition of a concrete method**

conc\_methoddef ::=

`method` var (` comma\_param `)` `:` type `{` stmt\* `}`

**Definition of a trait (typeclass)**

traitdef ::= `trait` traitname `{` abs\_methoddef\* `}`

**Definition of an implementation of a typeclass**

impldef ::= `impl` traitname `for` type `{` conc\_methoddef\* `}`

**Definition of a toplevel function**

funcdef ::= `func` var `(` comma\_param `)` `:` type

`{` stmt\* `}`

stmt ::= `let` param `=` 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**

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

struct\_actual\_params ::=

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

primary\_exp ::= i | var | **Integers and variables**

`true` | `false` | **Booleans**

`self` | **Instance on which we call a method**

`(` exp `)` | **Parenthesized expression**

**Creates a new instance of a struct**

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

**Accessing a struct field or method**

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

call\_exp ::= dot\_exp (`(` comma\_exp `)`)\*

mult\_exp ::= call\_exp ((`\*` | `/`) call\_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

program\_item ::= structdef | traitdef | impldef | funcdef

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

**Example: Generic-looking add method**

trait Addable {

method add(other: Self): Self;

}

trait Printable {

method print(): Void;

}

struct IntWrapper {

value: Int

}

impl Addable for Int {

method add(other: Int): Int {

return self + other;

}

}

impl Addable for IntWrapper {

method add(other: IntWrapper): IntWrapper {

return new IntWrapper { value: self.value + other.value };

}

impl Printable for Int {

method print(): Void {

println(self);

}

}

impl Printable for IntWrapper {

method print(): Void {

println(self.value);

}

}

let a1: Int = 5;

let a2: IntWrapper = new IntWrapper { value: 7 };

let a3: Int = a1.add(2);

let a4: IntWrapper = a2.add(new IntWrapper { value: 3 });

a3.print();

a4.print();