CS 321: Programming Languages — Homework 4

1 Functions with Multiple Arguments (with Deferred Substitutions)

Start with the F1WAE interpreter *for deferred substitution*, and extend the implementation to support any number of arguments to a function (including zero), and any number of arguments (including zero) in a function application:

```
<FunDef> = {deffun {<id> <id>*} <FnWAE>}
<FnWAE> = <num>
        | {+ <FnWAE> <FnWAE>}
        | {- <FnWAE> <FnWAE>}
        | {with {<id> <FnWAE>} <FnWAE>}
        | <id>
        | {<id> <FnWAE>*}
```

As with homework 3, you must change the `F1WAE` datatype, and you must thus provide a `parse` function that produces values of your modified `F1WAE` datatype. It must accept a quoted expression and produce an `FnWAE` value. Similarly, you must provide a `parse-defn` function to parse definitions.

See homework 3 for details on these functions.

You must also provide an `interp-expr` function with signature:

```
FnWAE (listof FunDef) -> Number
```

This function should be a very simple wrapper for your `interp` function, which our tests will call on the results of your parsers. If yours is more than a few lines (one line is possible), you may be overthinking it.

2 Errors

Your interpreter and parser must obey the formats and precedence rules described in homework 3.

Otherwise, assume that the input to your parser is a well-formed program.

3 Conditionals

Add `if0`, a conditional expression. It has three subexpressions:

```
<FnWAE> = ...
        | {if0 <FnWAE> <FnWAE> <FnWAE>}
```

Evaluating an `if0` expression evaluates the first subexpression; if it produces `0`, then the result of the entire expression is the result of the second subexpression. Otherwise, the result is the result of the third subexpression.

Examples:

```
(test (interp-expr (parse '{if0 0 1 2}) '()) 1)
(test (interp-expr (parse '{if0 1 2 3}) '()) 3)
```

4 Negative predicate

Implement, in the `FnWAE` language (without any extensions, i.e., you cannot add new kinds of expressions to the language or to your interpreter), a predicate `neg?` that determines if an integer is negative.

```
{deffun {neg? x} ...}
```

It must return either `0` (if the input is negative), or `1` (if not). The number `0` is not itself considered negative.

5 Multiplication on integers

Implement, in the FnWAE language (without any extensions), a function `mult` that computes the product of two integers.

```
{deffun {mult x y} ...}
```

6 Handin instructions

Provide definitions for `parse`, `parse-defn`, and `interp-expr`, as above.

Provide a PLAI-level definition of `mult-and-neg-deffuns` that is bound to a list of (unparsed) deffuns that contains both `neg?` and `mult` as well as any helper functions you need:

```
(define mult-and-neg-deffuns
  (list `{deffun {neg? x} ...}
        `{deffun {mult x y} ...}
        ; other deffuns okay, too, for your helpers
        ))
```

Do not leave in any unused code (i.e., no `subst`). Tests for the above functions (or helpers you may use along the way) are fine, though.

Have the 8 rules from the Provost's website (see the homework 1 for more details).

Submit your code via Canvas.

Your submission must include your test cases; submissions without test cases will get a grade of 0.