Homework 2 - General Requirements

Code submitted without complete test coverage will result in an automatic 20% penalty.

This assignment can be completed in groups of two. If you elect to do this, make a group on Canvas and submit the file as a group.

Part 1 — Maximum

Start with the interpreter with functions, and add a `max` operator that takes two numbers and returns the larger of them.

Since you must change the `Exp` datatype, and since different people may change it in different ways, you must update the `parse` function, which accepts an S-expression and produces an `Exp` value.

Some examples:

```
  (test (interp (parse `{max 1 2})
                (list))
        2)
  (test (interp (parse `{max {+ 4 5} {+ 2 3}})
                (list))
        9)
```

Part 2 — Functions that Accept Multiple Arguments

Extend the interpreter to support multiple or zero arguments to a function, and multiple or zero arguments in a function call.

For example,

```
  {define {area w h} {* w h}}
```

defines a function that takes two arguments, while

```
  {define {five} 5}
```

defines a function that takes zero arguments. Similarly,

```
  {area 3 4}
```

calls the function `area` with two arguments, while

```
{five}
```

calls the function `five` with zero arguments.

At run-time, a new error is now possible: function application with the wrong number of arguments. Your `interp` function should detect the mismatch and report an error that includes the words “wrong arity”.

To support functions with multiple arguments, you’ll have to change `fd` and `appE` and all tests that use them. When you update the `parse` function, note that `s-exp-match?` supports `...` in a pattern to indicate zero or more repetitions of the preceding pattern. *Beware of putting the multi-argument application pattern too early in `parse`, since that pattern is likely to match other forms.* In addition, you’ll need to update the `parse-fundef` function that takes one quoted `define` form and produces a `Func-Defn` value.

Just to clarify: Supporting multiple-argument functions does *not* mean changing operations like `+` or `*`. Although `+` and `*` are functions in Plait, they’re treated as non-function operator forms in Curly.

Some examples:

```
(test (interp (parse `{f 1 2})
              (list (parse-fundef `{define {f x y} {+ x y}})))
      3)
(test (interp (parse `{+ {f} {f}})
              (list (parse-fundef `{define {f} 5})))
      10)
(test/exn (interp (parse `{f 1})
                  (list (parse-fundef `{define {f x y} {+ x y}})))
          "wrong arity")
```

Remember that Plait provides `map`, which takes a function and a list, and applies the function to each element in the list, returning a list of results. For example, if `sexps` is a list of S-expressions to parse, `(map parse sexps)` produces a list of `ExprC`s by parsing each S-expression.

But also remember that `map` doesn’t work for everything. Sometimes, when you have a list to process (or maybe two lists in parallel), then you need to write a new function using the template for lists.

Part 3 — BONUS(5PTS) Function Argument Checking

A function is ill-defined if two of its argument names are the same. To prevent this problem, update your `parse-fundef` function can detect this problem and report a “bad syntax” error.

For example, `(parse-fundef `{define {f x x} x})` must report a “bad syntax” error, while `(parse-fundef `{define {f x y} x})` should produce a `Func-Defn` value.