Homework 3 - Meta-Circular Evaluator

元循环求值器 m-eval.scm

在 `m-eval.scm` 中我们实现了一个最基本的元循环求值器，与课上 Slides 内的或者书上的基本一致。

它可以像 `racket` 一样交互式解释：

```
racket m-eval.scm
```

也可以读入一个 `scheme` 程序解释执行：

```
racket m-eval.scm primes.scm
```

也可以按顺序执行多个文件（文件之间互相独立，不共享环境）：

```
racket m-eval.scm primes.scm factorial.scm
```

不过目前的解释器有许多缺陷，包括且不限于：

- 没有 cond, let 等 special forms
- 缺乏一些 primitive procedures（当然你可以自己添加）
- 一些有关定义顺序的微妙问题，见 SICP CH4.1.6
- 运行效率低

如果发现了 bug 可以通过邮件反馈给助教。

本次的作业中你需要：

1. 阅读并理解 `m-eval.scm`，可以参考课堂 Slides 与书本 (SICP CH4.1.1 ~ CH4.1.5)，以下习题一定程度上都可以复用现有的过程
2. 添加 `cond` 语句的解释，可以参考书本 (SICP CH4.1.2)，书中给出了一种 `cond` 的实现方式
3. 添加 `let` 语句的解释
4. 添加 `let*` 语句的解释

添加 `cond` 语句的解释

就是你最熟悉的 `cond` 。

参考 racket 官方文档 中对 `cond` 的说明，我们修改了一点点：

- 语法结构为 `(cond <clause1> <clause2> ...)`
- 其中每个 `<clause>` 形如 `(<test> <expression1> <expression2> ...)`
- 最后一个 `<clause>` 可以是 else 语句，结构为 `(else <expression1> <expression2> ...)`
- 如果有 else 语句，那 else 语句必须是最后一句
- 当所有谓词都为假且没有 else 语句时，你可以让它为任意值，测试数据中不会出现这种情况

你**无须**实现为 `(<test> => <recipient>)` 形式的 `<clause>` 。

你可以修改 `m-eval` 的实现，并在合适的位置自由添加新的过程，可以通过以下命令测试实现的正确性：

```
racket m-eval.scm cond-test.scm
```

> **Tip**
>
> 可以参考书本 (SICP CH4.1.2) 把它归约到若干个 `if` 子句的嵌套

添加 `let` 语句的解释

就是你最熟悉的 `let` 。

参考 racket 官方文档 中对 `let` 的说明：

- 语法结构为 `(let <bindings> <body>)`
- 其中 `<bindings>` 形如 `(<variable1> <init1>) ...)`
- `<body>` 可以有一句或者多句表达式构成

标准 `scheme` 中如果 `<bindings>` 中同一个变量出现多次应该报错，为了简便你可以假定这种情况不会发生。

同上你需要修改 `m-eval` 的实现，可以添加需要的过程，通过以下命令测试正确性：

```
racket m-eval.scm let-test.scm
```

> **Tip**
>
> `(let ((<var1 <exp1>) ... (<varn> <expn>)) <body>)`
>
> 等价于
>
> `((lambda (<var1> ... <varn>) <body>) <exp1> ... <expn>)`
>
> 即 `let` 可以被归约到对一个匿名过程的应用

添加 `let*` 语句的解释

`let` 的强化版。

形式上和 `let` 几乎一样，语法结构为 `(let* <bindings> <body>)`，区别在于 `let*` 的求值是从左到右的，这意味着以下代码合法（而在 `let` 中是不合法的）：

```
(let* ((x 3)
       (y (+ x 2))
       (z (+ x y 5)))
  (* x z)) ; 39
```

同上你需要修改 `m-eval` 的实现，可以添加需要的过程，通过以下命令测试正确性：

```
racket m-eval.scm let-star-test.scm
```

> **Tip**
>
> `let*` 可以被归约到若干个 `let` 的嵌套

更进一步?

阅读 SICP CH4.1.6 ~ CH4.1.7 以学习更多。

~~你甚至可以让 m-eval 解释 m-eval 自己。~~

题外话

助教在 HW2 后发现可以通过以下包和设置，把所有的 `list` 都变成 `mlist`，这样就可以愉快的 `set-car!` 了：

```
(require r5rs) # use Revised^5 Report on the Algorithmic Language Scheme
(print-mpair-curly-braces #f) # print mpair like pair
```

参考资料

1. D. Friedman, M. Felleisen, *The little Schemer (4th ed.)*. MIT Press, 1996. [pdf]
2. H. Abelson, G. Sussman, *Structure and Interpretation of Computer Programs*. MIT Press, 1996. [pdf]
3. 拷打助教，Email: 23110240130[at]m.fudan.edu.cn