نکاتی درباره‌ی Scheme و نوشتن پروژه‌ی PL

نسخه ۳

چون زبون Scheme و محیطش یه کم بددسته، این متنک رو نوشتم تا شما وقتتون کم‌تر از من تلف شه. هر کسی که اشتباهی دید یا نکته‌ی مفیدی فهمید که به درد بقیه می‌خوره، خوبه که به این فایل اضافه کنه و بفرسته. اگه می‌خواین تو ویندوز کار کنین، باید فایل مربوطه رو از یکی از این جاها دانلود کنین (حجمش حدود ۱۶/۵ مگابایته):

1. http://ce.sharif.edu/~mehrabian/scheme.exe
2. http://ftp.gnu.org/gnu/mit-scheme/snapshot.pkg/20070909/mit-scheme-20070909-ix86-win32.exe

این فایل رو که بگیرین، شامل همه چی هست (manual و reference داره به صورت html و editor هم داره). آدرس ای‌میلی که باید پروژه رو براش بفرستین هست: zandevakili@ce.sharif.edu.

محیط (IDE) و ارزیابی عبارات

فرضم بر اینه که تو ویندوز هستید ولی فکر نکنم خیلی تفاوتی بکنه. وقتی برنامه رو اجرا می‌کنین، یه پنجره میآد که می‌تونین توش تایپ کنین. ادیتور تحت ویندوز اسمش Edwin است و شما الآن تو Edwin هستید. تایپ کنین:

```
(DEFINE (square x) (*x x))
```

و بزنین C-x C-e.

منظور از C، کلید Control و منظور از C-x یعنی گرفتن control و زدن x.

این رو که زدین دیگه square رو می‌شناسه. زدن C-x C-e به معنای ارزیابی (evaluate) کردن چیزی است که الآن تایپ کرده‌ام. کلید Enter هیچ کاری به جز رفتن به خط جدید انجام نمی‌ده، و برای این که دستورتون رو به interpreter بفهمونین باید بنویسین و بگین ارزیابی‌اش کنه. حالا بنویسین

```
(square 7)
```

و ارزیابی‌اش کنین. می‌بینین که می‌نویسه ۴۹، یعنی درست ارزیابی‌اش کرده! به همین سادگی، شما DEFINE ها تون رو می‌نویسین و می‌گین ارزیابی‌شون کنه. و بعد هر عبارت دیگه‌ای رو می‌دین و می‌گین ارزیابی‌اش کنه (اصولاً DEFINE فرقی با عبارت‌های معمولی نداره برای interpreter). ولی باید این DEFINE ها رو دونه‌دونه بگین ارزیابی کنه. یعنی هر تعریفی که کردین باید ارزیابی‌اش کنین، نه اینکه چند تا با هم رو بنویسین بعد بگین ارزیابی کنه، چون در این صورت فقط آخریشون رو می‌بینه. مثلاً بنویسین

(DEFINE (double x) (+ x x))
(double 7)

و ارزیابی کنین. می‌گه من double رو نمی‌شناسم اصلاً! پس باید اول DEFINE رو بنویسین، ارزیابی کنین، بعد خط دوم رو نوشته و ارزیابی کنین.

وقتی یه چیزی (مثل square) رو تعریف کردین، تا وقتی برنامه رو نبستین تو حافظه هست و می‌شه ازش استفاده کرد. همچنین هر وقت خواستین می‌تونین تعریفش رو عوض کنین، گیری بهتون نمی‌ده.

یه راه دیگه برای ارزیابی اینه که دو تا escape بزنین و چیزی که می‌خواین رو بنویسین و enter بزنین. اون پائین نتیجه رو می‌نویسه.

برای این‌که آخرین دستوری که ارزیابی کردین رو بیارین باید بزنینM-p. منظور از M کلید Alt است. این خیلی پرکاربرده مثلاً برای اصلاح تعاریف وقتی اشتباه تعریفشون کردین به درد می‌خوره. M-n هم برعکسشه.

خطاها و دیباگ کردن

اگه یه عبارت خطادار بنویسین و بخواین ارزیابی‌اش کنین، interpreter یه چیزایی می‌نویسه که خط اولش مورد خطا رو نوشته و تو خطوط بعدی یه سری RESTART نوشته که نمی‌دونم یعنی چی و پرسیده می‌خواین debugger رو اجرا کنه؟ اگه از مورد خطا فهمیدین ایراد کارتون کجاست می‌تونین بزنین n و بعد M-p و چیزی که نوشتین رو اصلاح و مجدداً ارزیابی کنین. ولی اگه نمی‌دونین، بزنین y (گاهی لازمه دو بار بزنین y) و پنجره‌ی دیباگر باز می‌شه!

این دیباگرش خیلی خوبه، خیلی راحت با بالا و پائین رفتن و استفاده از ماوس می‌تونین بین زیربرنامه‌ها (که روی stack هستند) حرکت کنید و ببینید پله‌پله که می‌خواسته کدوم دستور رو اجرا کنه که به اینجاها رسیده. ضمناً تو پنجره‌ی پائینی اگه یه کم پائین برین لیست تمام متغیرهای محلی رو هم نوشته. اگه یه کم دقت کنین به احتمال زیادی می‌فهمین مشکلتون چی بوده و وقتی فهمیدین می‌تونین پنجره دیباگر رو ببندین و برمی‌گردین به محیط اولیه و بعد M-p و چیزی که نوشتین رو اصلاح و مجدداً ارزیابی می‌کنین.

خطاهای رایج

قسمت عمده‌ای از خطاها احتمالاً خطاهای بی‌خودی خواهند بود، که دو تا عمده‌شون پرانتزگذاری غلط و استفاده نکردن از quote هستند. پرانتزگذاری کاملاً چیز دقیقیه و باید رعایتش کنید و مثلاً اگه تو یه DEFINE رعایت نکنید احتمالاً بهتون می‌گه:

Ill-formed special form: …

و در ... قسمتی که غلط بوده رو می‌نویسه. به علاوه از پرانتزگذاری نمی‌شه برای محکم‌کاری استفاده کرد! یعنی مثلاً (1 x +) کاملاً با ((1 x +)) فرق داره.
درباره‌ی quote (‘ یا `) تو کتاب هم گفته. فرض کنید یه تابع f دارید که روی لیست عمل می‌کنه. اگه بخواید رو لیست (2 3 1) اعمالش کنید نباید بزنید ((2 3 1) f)، چون اگه این کار رو بکنید می‌گه 1 قابل اعمال (applicable) نیست! چون وقتی (2 3 1) رو می‌بینه، فکر می‌کنه 1 یه تابعیه که باید روی آرگومان‌های 3 و 2 اجرا بشه. برای این که این کار رو نکنه بنویسید ((2 3 1)‘ f) در این صورت کاری نداره (2 3 1) چیه، یعنی داخلش نمی‌شه اصلاً، بلکه فقط به عنوان پارامتر ردش می‌کنه برای f. دقت کنید که این وارد شدنش به پارامترها گاهی درسته. مثلاً چنین چیزی درسته:

```
(square (+ 5 1))
```

و برای این که اینو انجام بده باید اول بره تو (1 5 +)، حسابش کنه و خروجی‌اش رو بده به square. پس همیشه دقت کنید که به موقع از quote استفاده کنید.

کمک گرفتن برای دستورات

من خیلی نتونستم از tutorial یا manual یا reference برنامه کمک بگیرم. دستوراتی که تو کتاب گفته برای نوشتن پروژه عملاً کافیه. تو خود محیط هم اگه بزنید C-h C-h یه لیستی از کمک‌ها می‌آد، که اونم خیلی به کارم نیومد!

ذخیره کردن در فایل

اولاً این که با زدن c+w صفحه تمیز می شود و هرچه از آن به بعد بنویسیم save می‌شود.
وقتی که خواستین نوشته‌های خود را save کنید، c+x c+s را بزنید اگر دفعه اول باشه، پایین از شما اسم فایل را می‌خواهد اما از دفعات بعد در همان file ذخیره می‌کند.
ابن کار تمام صفحه را برای شما save می‌کند، می توانید آن file ای را که save کرده‌اید با textpad باز کنید و قسمت‌های define آن را در یک فایل دیگر با پسوند scm، save کنید و بعد آن file را load کنید. (قسمت بعدی)

بازیابی از فایل

احتمالاً شما در نهایت یه عالمه DEFINE خواهید داشت که یکی از اون‌ها تابع match خواسته شده‌است. برای این‌که هر بار نخواهید تمام این‌ها رو تایپ کنید (راستش من روشی برای copy-paste کردن متون تو Edwin بلد نیستم) می‌تونین بذارینشون تو یه فایل مثل test.scm و این فایل رو بذارین اون‌جا که نصب کردین (مثل C:\Program Files\MIT-GNU Scheme) و وقتی Scheme رو اجرا کردین بنویسین:

(load “test”)
و ارزیابی‌اش کنین. این‌طوری (اگه خطایی نداشته‌باشه فایلتون) تمام DEFINE ها وارد حافظه شدن و می‌تونین ازشون استفاده کنید.
مثلاً چنین فایلی:

test.scm:

```
(DEFINE (square x) (* x x))
(DEFINE (double x) (+ x x))
```

وقتی load ش کنین می‌تونین از دستورات square و double استفاده کنین.

cut-paste کردن

برین سر جایی که می‌خواین cut کنین و بزنین C+Space. بعد برین تهش و بزنین C+w. بعد برین جایی که می‌خواین paste کنین و بزنین C+y.

چند نکته درباره‌ی زبان Scheme

نسبت به بزرگی و کوچکی حروف حساس نیست. عبارات ریاضی prefix هستند. #t و #f کلمات تعریف‌شده‌ای هستند (true و false). = برای مقایسه‌ی اعداد، EQ? برای مقایسه‌ی دو symbolic atom و EQUAL? برای مقایسه‌ی هر دو چیزی(!) استفاده می‌شود.

درباره‌ی مسئله‌ی Stable Marriage Problem و ورودی/خروجی

عمده‌ترین گام در حل مسئله، تبدیل الگوریتم Gale-Shapley (گفته شده در Wikipedia) به یک الگوریتم بازگشتی است.
باید یه تابع match بنویسیم که این طوریه:

```
(match n men_preferences_list women_preferences_list)
        returns
stable_pairs
```

من از TA پرسیدم، گفت این signature درسته. مثلاً:

`(match 2 '((1 2) (1 2)) '((2 1) (2 1))) yields ((1 2) (2 1))`

لیست اولویت‌ها این طوریه که مثلاً اگه لیست مردی (1 3 4 2) باشه، معنی‌اش اینه که اون مرد زن ۲ را از بیش از هر زنی دوست داره، و زن ۱ رو از همه کم‌تر دوست داره. هم‌چنین در هر stable_pair، اول مرد و بعد زن می‌آید. stable_pairs لزومی ندارد که مرتب باشد.

دو مثال از اینجا:

Example 1:

```
(match
   3
   '((3 2 1) (3 1 2) (2 1 3))
   '((2 3 1) (3 2 1) (3 2 1))
)
```

returns:

```
((1 1) (2 3) (3 2))
```

Example 2:

```
(match
     5
     ‘(   (2 3 5 4 1)
          (2 4 3 1 5)
          (5 3 2 4 1)
          (1 5 4 3 2)
          (4 3 2 1 5)
     )
     ‘(
          (1 2 4 3 5)
          (3 4 1 5 2)
          (4 3 5 2 1)
          (1 5 2 4 3)
          (5 2 3 1 4)
     )
)
```

returns either of:

```
((1 2) (2 3) (3 5) (4 1) (5 4))
((1 2) (2 1) (3 5) (4 3) (5 4))
((1 4) (2 1) (3 2) (4 3) (5 5))
```

Note that the solution is not unique. Gale-Shapley algorithm finds the first solution.