دانشگاه اصفهان

دانشکده مهندسی کامپیوتر

تکلیف سری اول

درس زبان‌های برنامه نویسی

آشنایی با زبان RUST

سیدمحمدحسین هاشمی

استاد : دکتر آرش شفیعی

پاییز ۱۴۰۳


بسم الله الرحمن الرحيم

فهرست مطالب

۱ اسکیم در مقابل لیسپ

الف. چرا مجموعه کلمات کلیدی کوچک‌تر و مشخصات مختصر اسکیم می‌تواند مفید باشد؟

- سادگی و یادگیری آسان‌تر: تعداد کم کلمات کلیدی یادگیری زبان را برای مبتدیان ساده‌تر می‌کند.
- انعطاف‌پذیری و قابلیت بیان بالا: توسعه‌دهندگان آزادی بیشتری برای ایجاد انتزاعات دارند بدون اینکه با تعداد زیادی از ساختارهای از پیش تعریف‌شده محدود شوند.
- تمرکز بر مفاهیم اصلی: اسکیم بر سادگی تأکید دارد و برنامه‌نویسان را تشویق می‌کند تا مفاهیم اصلی مانند بازگشت (recursion) و برنامه‌نویسی تابعی را بهتر درک کنند.

ب. مثالی از تفاوت در رفتار با توجه به نوع محدوده (ایستا یا پویا):

فرض کنید کد زیر نوشته شده است:

Listing 1: Q1 B

```
(define x 10) ;; global variable x

(define (foo) x) ;; The function returns the value of x

(define (bar)
(let ((x 20)) ;; Local variable x
(foo)))    ;; calling foo

(bar)
```

- محدوده ایستا (Static Scoping): مقدار x در تابع foo بر اساس جایی که foo تعریف شده تعیین می‌شود، نه جایی که فراخوانی شده است. خروجی: ۱۰ (مقدار سراسری x).
- محدوده پویا (Dynamic Scoping): مقدار x در تابع foo بر اساس محیط فراخوانی bar تعیین می‌شود. خروجی: ۲۰ (مقدار محلی x در bar).

محدوده پویا می‌تواند رفتار غیرقابل پیش‌بینی ایجاد کند، در حالی که محدوده ایستا قابل فهم‌تر و ساده‌تر برای اشکال‌زدایی است.

۲ گام‌به‌گام ارزیابی عبارات در Lisp

الف. گام‌به‌گام ارزیابی عبارات در Lisp

۱. عبارت اول:

Listing 2: Q2 A 1

```
(eval '(+ 1 2 (eval '(+ 3 4))))
```

* گام اول
ارزیابی داخلی‌ترین عبارت ((eval ’(+ 3 4)
عبارت (+ 3 4) به دلیل نقل قول (quote) به عنوان یک داده لیست به eval داده می‌شود.
عبارت (+ 3 4) به دلیل نقل قول (quote) به عنوان یک داده لیست به eval داده می‌شود.
* گام دوم
ارزیابی عبارت (+ 1 2 7)
این عبارت یک جمع ساده است که نتیجه‌ی آن ۱۰ خواهد بود.
خروجی نهایی: ۱۰.

۲. عبارت دوم:

Listing 3: Q2 A 2

```
(cons 1 (list 2 3 (eval (cdr (cons 4 '(cdr '(5
    6)))))))
```

* گام اول
ارزیابی ((cons 4 ’(cdr ’(5 6))
cons یک لیست می‌سازد که عنصر اول ۴ و عنصر دوم ((cdr ’(5 6)’ به عنوان داده باقی می‌ماند.
نتیجه: (4 cdr ’(5 6))
* گام دوم
ارزیابی ((cdr (cons 4 ’(cdr ’(5 6))))
cdr عنصر دوم لیست ساخته‌شده در مرحله قبل را برمی‌گرداند.
بنابراین خروجی cdr برابر (cdr ’(5 6)’ می‌شود.
* گام سوم
eval ابتدا cdr را اعمال می‌کند بر روی لیست (5 6)’.
cdr عنصر دوم لیست را برمی‌گرداند که برابر با (6)’ است.
نتیجه: (6)’ (eval (cdr ’(5 6)))
* گام چهارم
list عناصر 2، 3 و (6)’ را در یک لیست قرار می‌دهد.
نتیجه: (2 3 (6))
* گام پنجم
cons عنصر 1 را به ابتدای لیست (2 3 (6)) اضافه می‌کند.
خروجی نهایی: (1 2 3 (6))

ب. پیاده‌سازی eval در Lisp با استفاده از نقل قول و پردازش لیست
eval را می‌توان در Lisp با ترکیب قابلیت‌های زیر پیاده‌سازی کرد:

۱. نقل قول (Quoting): نقل قول (’) به ما اجازه می‌دهد کد را به عنوان داده ذخیره کنیم و آن را به دلخواه پردازش کنیم.
۲. پردازش لیست‌ها (List Processing): از توابعی مانند car (برای دسترسی

به عنصر اول لیست)، cdr (برای دسترسی به بقیه عناصر لیست) و cons (برای ساخت لیست) استفاده می‌کنیم.

۳. تطبیق الگو (Pattern Matching): برای شناسایی ساختار عبارات (مثلاً توابع و عملگرها) از تطبیق الگو بهره می‌گیریم.

Listing 4: Q2 B – Pseudo code for eval

```
(defun my-eval (expr)
(cond
((atom expr) expr) ;; If expr is an atom (number,
    symbol), return itself
((equal (car expr) 'quote) (cadr expr)) ;; Quote mode
((equal (car expr) 'if) ;; simple condition
(if (my-eval (cadr expr))
(my-eval (caddr expr))
(my-eval (cadddr expr))))
(t  ;; Mode of calling functions
(apply (my-eval (car expr)) (mapcar #'my-eval (cdr
    expr))))))
```

توضیح کد:

۱. اگر expr یک اتم باشد (عدد یا نماد)، خودش برگردانده می‌شود.

۲. اگر expr یک نقل قول باشد، مقدار دوم لیست (داده‌ی نقل‌قول‌شده) برگردانده می‌شود.

۳. شرطی (if): شرط بررسی و یکی از شاخه‌ها ارزیابی می‌شود.

۴. فراخوانی توابع:

* ابتدا تابع (عنصر اول لیست) ارزیابی می‌شود.
* سپس آرگومان‌های تابع با mapcar ارزیابی می‌شوند.
* در نهایت، تابع با آرگومان‌ها اجرا می‌شود (apply).

با این منطق، می‌توان eval را به‌طور بازگشتی برای هر عبارت Lisp پیاده‌سازی کرد.

۳ کلمات کلیدی funcall در لیسپ

الف. تابع mystery شبیه به کدام تابع در ML است؟

تابع text شبیه به تابع map در زبان ML است.

- در ML، map یک تابع مرتبه بالاتر (higher-order function) است که یک تابع را به تمام عناصر لیست اعمال کرده و یک لیست جدید از نتایج ایجاد می‌کند.
- در mystery نیز همین کار انجام می‌شود:
  * هر عنصر از لیست (با استفاده از car و cdr) پردازش می‌شود.
  * نتیجه تابع اعمال‌شده به عنصر، در یک لیست جدید جمع‌آوری می‌شود (با استفاده از cons).

ب. بازنویسی mystery با استفاده از apply به‌جای funcall

برای بازنویسی تابع mystery با استفاده از apply به‌جای funcall، می‌توان به‌صورت زیر عمل کرد:

Listing 5: Q3 B

```
(defun mystery (f x)
(cond
((consp x) (cons (apply f (list (car x))) ;;    funcall  apply

)mystery f (cdr x))))
(T nil)))
```

توضیح تغییرات:

۱. در funcall f (car x)، تابع f مستقیماً با آرگومان (car x) فراخوانی می‌شد.

۲. برای استفاده از apply، نیاز است که آرگومان‌ها به‌صورت لیست ارائه شوند. بنابراین (list (car x)) ساخته می‌شود و به apply داده می‌شود.

۳. apply تابع f را به همراه لیست آرگومان‌ها اجرا می‌کند.

نکته: رفتار کد همچنان مشابه کد اصلی خواهد بود، با این تفاوت که از apply به‌جای funcall استفاده شده است.

۴ صریح‌سازی پرانتزها در عبارات λ

در حساب λ، پرانتزها برای تعیین ترتیب اعمال بسیار مهم هستند. ترتیب کلی به این شکل است:

۱. چپ‌گرایی در کاربرد توابع

۲. اولویت بالاتر برای تعریف λ در مقایسه با کاربرد تابع

با این اصول، عبارات داده‌شده به‌طور صریح بازنویسی می‌شوند.

۱. λx.xz λy.xy

گام به گام:

- ابتدا تعریف λx و کاربرد آن بر λy.xy
- در اینجا λx.xz یک تابع است که بر (λy.xy) اعمال می‌شود.

بازنویسی کامل با پرانتزهای صریح:

$$((\lambda x.(x\,z))\ (\lambda y.(x\,y)))$$

۲. (λx.xz) λy.w λw.wyzx

گام به گام:

- ابتدا λx.xz بر (λy.w λw.wyzx) اعمال می‌شود.
- درون (λy.w λw.wyzx)، ابتدا λw.wyzx تعریف‌شده و سپس λy آن را به عنوان خروجی w معرفی می‌کند.

بازنویسی کامل با پرانتزهای صریح:

$$((\lambda x.(x\,z))\ (\lambda y.\,(w\ (\lambda w.\,(((w\,y)\,z)\,x)))))$$

۳. λx.xy λx.yx

گام به گام:

- λx.xy تابعی است که بر (λx.yx) اعمال می‌شود.
- این تابع دوم (λx.yx) نیز خودش یک تعریف λ است.

بازنویسی کامل با پرانتزهای صریح:

$$((\lambda x.(xy))(\lambda x.(yx)))$$

۵ روند کاهش β

کاهش β در حساب λ به معنای جای‌گذاری آرگومان‌ها در بدنه توابع و ساده‌سازی عبارات تا حد امکان است. این فرایند به صورت بازگشتی انجام می‌شود.

۱. (λz.z)(λy.yy)(λx.xa)

- گام اول: تابع (λz.z) آرگومان اول خود را برمی‌گرداند.

$$(\lambda z.z)(\lambda y.yy) \rightarrow (\lambda y.yy)$$

- گام دوم: عبارت به شکل زیر ساده می‌شود

$$(\lambda y.yy)(\lambda x.xa)$$

- گام سوم: جای‌گذاری (λx.xa) به‌جای y

$$(\lambda y.yy)(\lambda x.xa) \rightarrow (\lambda x.xa)(\lambda x.xa)$$

- نتیجه نهایی:

$$(\lambda x.xa)(\lambda x.xa)$$

عبارت دیگر قابل کاهش نیست زیرا تابع دیگر اعمال نمی‌شود.

۲. $(\lambda x.xx)(\lambda y.yx)z$

- گام اول: ابتدا $(\lambda x.xx)$ با آرگومان $(\lambda y.yx)$ فراخوانی می‌شود. $(\lambda y.yx)$ جای‌گذاری می‌شود:

$$(\lambda x.xx)(\lambda y.yx) \rightarrow (\lambda y.yx)(\lambda y.yx)$$

- گام دوم: اکنون $(\lambda y.yx)$ روی خودش اعمال می‌شود:

$$(\lambda y.yx)(\lambda y.yx)$$

- گام سوم: تابع $(\lambda y.yx)$ با آرگومان $(\lambda y.yx)$ فراخوانی می‌شود. $(\lambda y.yx)$ جای‌گذاری می‌شود:

$$(\lambda y.yx)(\lambda y.yx) \rightarrow ((\lambda y.yx)x)$$

- نتیجه نهایی:

$$((\lambda y.yx)x)$$

۳. $(((\lambda x.\lambda y.(xy))(\lambda y.y))w)$

- گام اول: ابتدا $(\lambda x.\lambda y.(xy))$ با آرگومان $(\lambda y.y)$ فراخوانی می‌شود. $(\lambda y.y)$ جای‌گذاری می‌شود

$$(\lambda x.\lambda y.(xy))(\lambda y.y) \rightarrow \lambda y((\lambda y.y)y)$$

- گام دوم: حالا $(\lambda y.((\lambda y.y)y))$ با آرگومان w فراخوانی می‌شود. y به w جای‌گذاری می‌شود

$$(\lambda y((\lambda y.y)y))w \rightarrow ((\lambda y.y)w)$$

- گام سوم: تابع $(\lambda y.y)$ روی w اعمال می‌شود

$$(\lambda y.y)w \rightarrow w$$

- نتیجه نهایی:

$$w$$

۶ اثبات عبارات با استفاده از کدگذاری حساب λ

الف. not (not true) = true

تعاریف داده‌شده:

- not = λx.((x false)true)
- true = λx.λy.x
- false = λx.λy.y

گام ۱: محاسبه not true

* تابع not روی true اعمال می‌شود:

$$nottrue = (\lambda x.((xfalse)true))true$$

* حالا x را با true جای‌گذاری می‌کنیم:

$$(\lambda x.((xfalse)true))true \rightarrow ((truefalse)true)$$

* تعریف true=λx.λy.x را جای‌گذاری می‌کنیم:

$$((truefalse)true) = ((\lambda x.\lambda y.x)false)true$$

* حالا x را با false جای‌گذاری می‌کنیم:

$$((\lambda x.\lambda y.x)false)true \rightarrow (\lambda y.false)true$$

* تابع (λ.false) را روی true اعمال می‌کنیم:

$$(\lambda y.false)true \rightarrow false$$

* نتیجه:

$$nottrue = false$$

گام ۲: محاسبه not (not true)

* حالا not را روی not true که false است اعمال می‌کنیم:

$$not(nottrue) = (\lambda x.((xfalse)true))false$$

* x را با false جای‌گذاری می‌کنیم:

$$(\lambda x.((xfalse)true))false \rightarrow ((falsefalse)true)$$

* تعریف false=λx.λy.y را جای‌گذاری می‌کنیم:

$$((falsefalse)true) = ((\lambda x.\lambda y.y)false)true$$

* حالا x را با false جای‌گذاری می‌کنیم:

$$((\lambda x.\lambda y.y)false)true \rightarrow (\lambda y.y)true$$

* تابع (λy.y) را روی true اعمال می‌کنیم:

$$(\lambda y.y)true \rightarrow true$$

* نتیجه نهایی:

$$not(nottrue) = true$$

ب. if false then x else y = y

تعاریف داده‌شده:

- if a then b else c = a b c
- true = λx.λy.x
- false = λx.λy.y

گام ۱: جای‌گذاری در if false then x else y

تعریف if a then b else c را استفاده می‌کنیم که برابر با abc است. حال a=false، b=x و c=y:

y x if false then x else y=false

گام ۲: محاسبه false x y

* تعریف false=λx.λy.y را جای‌گذاری می‌کنیم:

$$falsexy = (\lambda x.\lambda y.y)xy$$

* ابتدا x را در λx.λy.y جای‌گذاری می‌کنیم:

$$(\lambda x.\lambda y.y)x \rightarrow \lambda y.y$$

* حالا λy.y را روی y اعمال می‌کنیم:

$$(\lambda y.y)y \rightarrow y$$

* نتیجه نهایی:

$$iffalsethenxelsey = y$$