درس طراحی زبان‌های برنامه‌سازی

دکتر محمد ایزدی

پروژه

دانشکده مهندسی کامپیوتر

دانشگاه صنعتی شریف

نیم سال دوم ۱۴۰۰ـ۱۳۹۹

مهلت ارسال:
۲۴ تیر ۱۴۰۰
ساعت ۲۳:۵۹

به موارد زیر توجه کنید:

* مهلت ارسال تمرین ساعت ۲۳:۵۹ روز ۲۴ تیر ۱۴۰۰ است.
* امکان ارسال با تاخیر تنها تا ۱ روز و با کسر ۳۰ درصد نمره امکان‌پذیر است.
* پروژه تحویل حضوری خواهد داشت. این تحویل پس از ارسال کدهایتان در کوئرا و در وی کلاس صورت می‌گیرد. زمان دقیق تحویل متعاقبا اعلام می‌شود.
* در صورتی که بنا به هر دلیلی نیاز دارید زودتر نمره درس برایتان ثبت شود، برای تحویل زودتر از موعد با mehrdadkarrabi۱۹۹۷@gmail.com هماهنگ کنید.
* پروژه را در قالب گروه‌های ۲ یا ۳ نفره انجام دهید.
* در نهایت تمام فایل‌های خود را در یک فایل زیپ قرار داده و با نام $P\_StudentID$ آپلود کنید. آپلود یکی از اعضای گروه کافیست.
* دقت کنید که در صورت عدم پیاده‌سازی ،print باید یک ابزار نظارتی برای بررسی صحت اجرای برنامه‌ها داشته باشید.
* لطفا پروژه را از یکدیگر کپی نکنید. در صورت وقوع چنین مواردی مطابق با سیاست درس رفتار می‌شود.

۱ تعریف گرامر

در این پروژه قصد داریم یک مفسر برای یک زبان ساده طراحی کنیم. گرامر این زبان به شکل زیر است:

1. $Program \rightarrow Statements\ EOF$

2. $Statements \rightarrow Statement\ \text{‘;‘} \mid Statements\ Statement\ \text{‘;‘}$

3. $Statement \rightarrow Compound\_stmt \mid Simple\_stmt$

4. $Simple\_stmt \rightarrow Assignment \mid Return\_stmt \mid Global\_stmt \mid \text{‘}pass\text{‘} \mid \text{‘}break\text{‘} \mid \text{‘}continue\text{‘}$

5. $Compound\_stmt \rightarrow Function\_def \mid If\_stmt \mid For\_stmt$

6. $Assignment \rightarrow ID\ \text{‘} = \text{‘}\ Expression$

7. $Return\_stmt \rightarrow \text{‘}return\text{‘} \mid \text{‘}return\text{‘}\ Expression$

8. $Global\_stmt \rightarrow \text{‘}global\text{‘}\ ID$

9. $Function\_def \rightarrow \text{‘}def\text{‘}\ ID\ \text{‘(‘}\ Params\ \text{‘)‘}\ \text{‘ : ‘}\ Statements \mid \text{‘}def\text{‘}\ ID\ \text{‘() : ‘}\ Statements$

10. $Params \rightarrow Param\_with\_default \mid Params\ \text{‘,‘}\ Param\_with\_default$

11. $Param\_with\_default \rightarrow ID\ \text{‘} = \text{‘}\ Expression$

12. $If\_stmt \rightarrow \text{‘}if\text{‘}\ Expression\ \text{‘ : ‘}\ Statements\ Else\_block$

13. $Else\_block \rightarrow \text{‘}else\text{‘}\ \text{‘ : ‘}\ Statements$

14. $For\_stmt \rightarrow \text{‘}for\text{‘}\ ID\ \text{‘}in\text{‘}\ Expression\ \text{‘ : ‘}\ Statements$

15. $Expression \rightarrow Disjunction$

16. $Disjunction \rightarrow Conjunction \mid Disjunction\ \text{‘}or\text{‘}\ Conjunction$

17. $Conjunction \rightarrow Inversion \mid Conjunction\ \text{‘}and\text{‘}\ Inversion$

18. $Inversion \rightarrow \text{‘}not\text{‘}\ Inversion \mid Comparison$

19. $Comparison \rightarrow Sum\ Compare\_op\_Sum\_pairs \mid Sum$

20. $Compare\_op\_Sum\_pairs \rightarrow Compare\_op\_Sum\_pair$
$| \; Compare\_op\_Sum\_pairs \; Compare\_op\_Sum\_pair$

21. $Compare\_op\_Sum\_pair \rightarrow Eq\_Sum \mid Lt\_Sum \mid Gt\_Sum$

22. $Eq\_Sum \rightarrow \text{`} == \text{`} \; Sum$

23. $Lt\_Sum \rightarrow \text{`} < \text{`} \; Sum$

24. $Gt\_Sum \rightarrow \text{`} > \text{`} \; Sum$

25. $Sum \rightarrow Sum \; \text{`} + \text{`} \; Term \mid Sum \; \text{`} - \text{`} \; Term \mid Term$

26. $Term \rightarrow Term \; \text{`} * \text{`} \; Factor \mid Term \; \text{`/`} \; Factor \mid Factor$

27. $Factor \rightarrow \text{`} + \text{`} \; Factor \mid \text{`} - \text{`} \; Factor \mid Power$

28. $Power \rightarrow Atom \; \text{`} * * \text{`} \; Factor \mid Primary$

29. $Primary \rightarrow Atom \mid Primary \; \text{`[`} \; Expression \; \text{`]`} \mid Primary \; \text{`()`}$
$\mid Primary \; \text{`(`} \; Arguments \; \text{`)`}$

30. $Arguments \rightarrow Expression \mid Arguments \; \text{`,`} \; Expression$

31. $Atom \rightarrow ID \mid \text{`}True\text{`} \mid \text{`}False\text{`} \mid \text{`}None\text{`} \mid NUMBER \mid List$

32. $List \rightarrow \text{`[`} \; Expressions \; \text{`]`} \mid \text{`[]`}$

33. $Expressions \rightarrow Expressions \; \text{`,`} \; Expression \mid Expression$

نکات گرامر:

- این گرامر (۱) LALR است بنابراین برای پیاده‌سازی‌های بعدی نیاز به هیچ‌گونه تغییری در آن نیست.
- توجه کنید که NUMBER اعداد مثبت صحیح و یا اعشاری است.
- موارد درون " terminal هایی هستند که به همین شکل در زبان ظاهر خواهند شد. مواردی که همه حروف آن‌ها بزرگ است مانند $NUMBER$ نیز terminal هستند با این تفاوت که مقدار مورد نظر برنامه‌نویس به جای آن‌ها می‌آید. کلماتی هم که با حروف بزرگ شروع می‌شوند nonterminal هستند.

۲ پیاده‌سازی اسکنر و پارسر (۱۰ نمره)

برای پیاده‌سازی این بخش باید از ابزارهای موجود در زبان رکت استفاده کنید. برای منبع می‌توانید از https://docs.racket-lang.org/parser-tools/index.html استفاده کنید.

مرحله‌ی اول پیاده سازی هر مفسری، پیاده‌سازی لکسر و پارسر مربوط به آن است. این دو ماژول با گرفتن رشته‌ی برنامه‌ی ورودی، درخت مربوط به آن برنامه را برمی‌گردانند. به طور مثال در صفحه‌ی ۶۴ کتاب مشاهده می‌کنید که برنامه‌ی ورودی به شکل «(-(-(x,3),-(v,i))» نوشته شده‌است. اما همان‌طور که می‌دانید برنامه‌ی ورودی مفسر، یک رشته است. حال به اختصار وظیفه‌ی هرکدام از این دو ماژول را بیان می‌کنیم:

* لکسر: این ماژول رشته‌ی ورودی را تکه‌تکه کرده و به کلمات اصلی برنامه می‌شکند. به طور مثال برنامه‌ی "a=2" به کلمات a و = و ۲ شکسته می‌شود. برای مشاهده‌ی بهتر کارکرد این ماژول مثال آورده‌شده به زبان رکت lexer.rkt را اجرا کرده و خروجی را مشاهده کنید.

* پارسر: این ماژول وظیفه‌ی ساخت درخت را از روی کلمات خروجی لکسر و از روی گرامر دارد. به طور مثال گرامر ساده‌ی exp -> exp + number | number را در نظر بگیرید. حال فرض کنید رشته‌ی ورودیمان "1 + 2 + 3 + 4" است. می‌دانیم لکسر این رشته را تبدیل به کلمات ۱ و + و ۲ و + و ۳ و + و ۴ می‌کند. حال انتظار ما از پارسر این است که با این کلمات درخت (+ (+ (+ (1) 2) 3) 4) را بسازد. برای نمونه کد parser.rkt را اجرا کرده و خروجی آن را ببینید.

پس از انجام این بخش باید یک تابع parser داشته‌باشید که یک رشته را به عنوان ورودی بگیرد و درخت پارس‌شده‌ی آن را خروجی دهد.

۳ پیاده‌سازی اولیه‌ی مترجم (۶۰ نمره)

در این بخش شما باید به کمک آموخته‌های خود در درس، یک مفسر ساده برای این زبان پیاده‌سازی کنید.

دقت کنید که در تست‌های نهایی، برنامه‌ی با خطا داده نخواهد شد. بنابراین نیازی به پیاده‌سازی Error Handler نیست.

همانطور که در گرامر این زبان مشخص است برنامه‌های این زبان شامل تعدادی statement هستند که با ; از هم جدا شده اند. این گرامر ساده‌شده‌ی گرامر زبان python است و نحوه‌ی کلی برنامه مشابه پایتون خواهد بود. یکی از تفاوت‌های موجود آن است که بین هر دو statement ; خواهد آمد و در مقابل نیازی به رعایت INDENT نیست. حال

به توضیح خط‌های مورد نیاز گرامر می‌پردازیم:

- $pass$ دستوری است که هیچ کاری انجام نمی‌دهد.
- خط ۸: عبارت $global$ برای آن استفاده می‌شود که یک متغیر موجود در بلوک بیرونی یک تابع را درون تابع استفاده کنیم. دقت کنید اگر متغیری را بعد از تعریف تابع نیز تعریف کنید، می‌توانید آن را استفاده کنید. به طور مثال کد زیر

```
def f():
    global a;
    a += 1;
    ;

a = 2;
print(a);
b = f();
print(a);
```

مقدار ۲ و ۳ را پرینت می‌کند ولی کد زیر

```
def f():
    global a;
    a += 1;
    ;

b = f();
```

مشکل خواهد داشت و شما نیازی به در نظر گرفتن آن را ندارید. در صورتی که یک متغیر را global کنیم، تغییرات آن درون تابع به بیرون تابع نیز اعمال خواهد شد. همچنین در صورتی که درون یک تابع یک متغیر را گلوبال تعریف نکنیم، به هیچ وجه نمی‌توان از آن استفاده کرد. (این مورد با پایتون تفاوت دارد.)

- خط ۹: در هنگام تعریف تابع باید به تمام متغیرهای آن مقدار اولیه بدهید. در فراخوانی تابع در صورتی که تعداد بیشتر از پارامترهای تابع ورودی داده شود خطا است و در صورتی که تعداد کمتری ورودی داده شود، به ترتیب ورودی‌ها به پارامترها نسبت داده شده و پارامترهای باقی‌مانده مقدار پیشفرض خود را می‌گیرند.
- اعداد در زبان ما به دو شکل صحیح و اعشاری هستند. هنگام تعریف درصورتی که عدد تعریف شده . نداشت، صحیح و در غیر این صورت اعشاری است. عملیات / دو عدد (از هر نوعی) گرفته و مقدار دقیق حاصل تقسیم آن (حتما عددی اعشاری) را برمی‌گرداند.

- خط ۱۴ : جلوی عبارت in تنها یک لیست می‌تواند بیاید. (این مورد بر خلاف پایتون است)
- عبارات شرطی مورد نیاز if و for و تنها باید از نوع boolean باشند. (این مورد بر خلاف پایتون است.)
- خط ۲۵ و ۲۶ و ۲۸ گرامر: عملگرهای ریاضی تنها روی داده‌هایی از یک نوع قابل اجرا هستند.
- خط ۲۵ و ۲۶ و ۲۸ گرامر: برای نوع داده‌ی boolean (یعنی $True$ و $False$) تنها عملگرهای $+$ به معنای or منطقی و $*$ به معنای and منطقی قابل اعمال هستند.
- خط ۲۵ و ۲۶ و ۲۸ گرامر: برای نوع داده‌ی list تنها عملگر $+$ به معنای $concat$ (به هم چسباندن دو لیست) قابل اعمال است.
- خط ۲۷ گرامر: این دو عملگر $+$ و $-$ تنها بر روی اعداد قابل استفاده هستند.
- خط ۲۹ : فراخوانی تابع به شکل call by value است.
- در نهایت در هر قسمتی از پیاده‌سازی نیازی به فرض خاصی داشتید، آن را در یک داک نوشته و به همراه پروژه ارسال کنید. (نیاز به تهیه داک در حالت کلی نیست.)

۴ خاصیت بازگشتی توابع (۱۵ نمره)

در این بخش شما باید ذخیره‌ی تابع را به نحوی انجام دهید که بتوان درون یک تابع خودش را صدا زد. به مثال زیر توجه کنید.

```
def f():
    a = print(2);
    a = f();
    ;

a = f();
```

۵ خواندن کد از فایل (۱۰ نمره)

در این بخش باید یک تابع evaluate بنویسید که به عنوان ورودی آدرس کد موردنظر را گرفته و آن را اجرا کرده و خروجی را نمایش دهد. مثال:

evaluate(”a.txt”)

۶ تابع print (۵ نمره)

در این بخش باید یک تابع print را به زبان اضافه کنید به نحوی که با اجرای این تابع ورودی آن (که می‌تواند لیست و یا عدد باشد) نمایش داده شود. نحوه‌ی نمایش اهمیتی ندارد و این بخش صرفا جهت تست نهایی است. همچنین جهت راحتی فرض کنید ورودی این تابع نیازی به evaluate شدن ندارد و ورودی Atom یا لیستی از Atom هاست. این تابع را می‌توانید به گرامر اضافه کرده و یا به شکل یک تابع از پیش آماده در envinroment بگذارید.

۷ Lazy Evaluation (۲۰ نمره امتیازی)

در این بخش شما باید Lazy Evaluation را پیاده‌سازی کنید که در زبان ما شامل موارد زیر می‌شود:

- در هنگام * کردن، در صورتی که سمت چپ ضرب ۰ بود، سمت راست محاسبه نشده و مقدار ۰ برگردانده می‌شود. همچنین اگر سمت چپ ضرب false بود، بدون محاسبه‌ی سمت راست، مقدار false برگردانده شود.
- یک متغیر که در Assignment مقدار دهی می‌شود، تا وقتی که از آن استفاده نشود، محاسبه نمی‌شود.
- تا وقتی از یک ورودی تابع استفاده نشده است، آن مقدار محاسبه نمی‌شود.