



درس طراحی زبان‌های برنامه‌سازی

دکتر محمد ایزدی

پروژه

دانشکده مهندسی کامپیوتر

دانشگاه صنعتی شریف

نیم سال دوم ۱۴۰۰-۱۳۹۹

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 امکان پذیر است.
- * پروژه تحویل حضوری خواهد داشت. این تحویل پس از ارسال کدهایتان در کوئرا و در وی کلاس صورت می گیرد. زمان دقیق تحویل متعاقبا اعلام می شود.
- * در صورتی که بنا به هر دلیلی نیاز دارید زودتر نمره درس برایتان ثبت شود، برای تحویل زودتر از موعد با mehrdadkarrabi1997@gmail.com هماهنگ کنید.
- * پروژه را در قالب گروه های ۲ یا ۳ نفره انجام دهید.
- * در نهایت تمام فایل های خود را در یک فایل زیپ قرار داده و با نام `P_StudentID` آپلود کنید. آپلود یکی از اعضای گروه کافیت.
- * دقت کنید که در صورت عدم پیاده سازی، print باید یک ابزار نظارتی برای بررسی صحت اجرای برنامه ها داشته باشید.
- * لطفا پروژه را از یکدیگر کپی نکنید. در صورت وقوع چنین مواردی مطابق با سیاست درس رفتار می شود.



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. $Statements \rightarrow Statement \text{ ';' } | Statements Statement \text{ ';' }$
3. $Statement \rightarrow Compound_stmt | Simple_stmt$
4. $Simple_stmt \rightarrow Assignment | Return_stmt | Global_stmt$
 $| \text{'pass'} | \text{'break'} | \text{'continue'}$
5. $Compound_stmt \rightarrow Function_def | If_stmt | For_stmt$
6. $Assignment \rightarrow ID \text{ '=' } Expression$
7. $Return_stmt \rightarrow \text{'return'} | \text{'return'} Expression$
8. $Global_stmt \rightarrow \text{'global'} ID$
9. $Function_def \rightarrow \text{'def'} ID \text{ '(' } Params \text{ ')' ':' } Statements$
 $| \text{'def'} ID \text{ '(' } : \text{' } Statements$
10. $Params \rightarrow Param_with_default | Params \text{ ',' } Param_with_default$
11. $Param_with_default \rightarrow ID \text{ '=' } Expression$
12. $If_stmt \rightarrow \text{'if'} Expression \text{ ':' } Statements Else_block$
13. $Else_block \rightarrow \text{'else'} \text{ ':' } Statements$
14. $For_stmt \rightarrow \text{'for'} ID \text{ 'in'} Expression \text{ ':' } Statements$
15. $Expression \rightarrow Disjunction$
16. $Disjunction \rightarrow Conjunction | Disjunction \text{ 'or'} Conjunction$
17. $Conjunction \rightarrow Inversion | Conjunction \text{ 'and'} Inversion$
18. $Inversion \rightarrow \text{'not'} Inversion | Comparison$
19. $Comparison \rightarrow Sum Compare_op_Sum_pairs | Sum$



20. $Compare_op_Sum_pairs \rightarrow Compare_op_Sum_pair$
 $| Compare_op_Sum_pairs Compare_op_Sum_pair$
21. $Compare_op_Sum_pair \rightarrow Eq_Sum | Lt_Sum | Gt_Sum$
22. $Eq_Sum \rightarrow ' == ' Sum$
23. $Lt_Sum \rightarrow ' < ' Sum$
24. $Gt_Sum \rightarrow ' > ' Sum$
25. $Sum \rightarrow Sum ' + ' Term | Sum ' - ' Term | Term$
26. $Term \rightarrow Term ' * ' Factor | Term ' / ' Factor | Factor$
27. $Factor \rightarrow ' + ' Factor | ' - ' Factor | Power$
28. $Power \rightarrow Atom ' * * ' Factor | Primary$
29. $Primary \rightarrow Atom | Primary '[' Expression ']' | Primary '('$
 $| Primary '(' Arguments ')'$
30. $Arguments \rightarrow Expression | Arguments ', ' Expression$
31. $Atom \rightarrow ID | 'True' | 'False' | 'None' | NUMBER | List$
32. $List \rightarrow '[' Expressions ']' | []$
33. $Expressions \rightarrow Expressions ', ' Expression | Expression$

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 اسکندر 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 پیاده‌سازی 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 2 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 1 و $+$ و 2 و $+$ و 3 و $+$ و 4 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 شده‌اند. 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

```
1 def f():
2     global a;
3     a += 1;
4     ;
5
6 a = 2;
7 print(a);
8 b = f();
9 print(a);
```

مقدار ۲ و ۳ را پرینت می‌کند ولی کد زیر

```
1 def f():
2     global a;
3     a += 1;
4     ;
5
6 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 در صورتی 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 می تواند 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.

```
1 def f():
2     a = print(2);
3     a = f();
4     ;
5
6 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 صرفاً 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 environment 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.