درس طراحی زبان‌های برنامه‌سازی

پروژه

دانشکده مهندسی کامپیوتر

دانشگاه صنعتی شریف

نیم سال اول ۱۳۹۹ ـ ۱۴۰۰

استاد:

شیرین بقولی‌زاده

مهلت ارسال:

یکشنبه ۱۴ دی

ساعت ۲۳:۵۹

به موارد زیر توجه کنید:

* مهلت ارسال این فاز ساعت ۲۳:۵۹ روز یکشنبه ۱۴ دی است.
* پروژه تحویل به صورت مجازی خواهد داشت.
* 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 $P1\_StudentID$ آپلود کنید. آپلود یکی از اعضای گروه کافیست.
* لطفا هیچ کدی را از یکدیگر کپی نکنید. در صورت وقوع چنین مواردی مطابق با سیاست درس رفتار می‌شود.

۱ تعریف گرامر

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

$$command \rightarrow keyword \mid command; keyword$$
$$keyword \rightarrow if\_statement \mid assignment\_statement \mid while\_statement \mid return$$
$$while\_statement \rightarrow while\ exp\ do\ command\ end$$
$$if\_statement \rightarrow if\ exp\ then\ command\ else\ command\ end$$

$$assignment\_statement \rightarrow variable = exp$$
$$return \rightarrow return\ exp$$
$$exp \rightarrow aexp \mid aexp > aexp \mid aexp < aexp \mid aexp == aexp \mid aexp! = aexp$$
$$aexp \rightarrow bexp \mid bexp - aexp \mid bexp + aexp$$
$$bexp \rightarrow cexp \mid cexp * bexp \mid cexp/bexp$$
$$cexp \rightarrow -cexp \mid (exp) \mid posNumber \mid null \mid variable \mid true \mid false$$
$$string \mid list \mid variable\ listMember$$
$$list \rightarrow [listValues] \mid [\,]$$
$$listValues \rightarrow exp \mid exp, listValues$$
$$listMember \rightarrow [exp] \mid ob\ exp\ cb\ listMember$$

نکته: توجه کنید که number اعداد مثبت است. همچنین دقت کنید که در اعمال ریاضی + و ـ و ... همانطور که از گرامر مشخص است، این اعمال از راست به چپ انجام می‌شوند.

۲ پیاده‌سازی اسکنر و پارسر (۲۵ نمره)

اسکنر :
برای پیاده‌سازی یک مفسر ابتدا بخش اسکنر و پارسر آن باید پیاده‌سازی بشوند. همان‌طور که در درس کامپایلر هم قبلا مشاهده کردید لکسر وظیفه‌ی بخش بخش کردن رشته‌ی ورودی را بر عهده دارد. و ورودی به بخش هایی که در گرامر هم به طور دقیق‌تر نشان داده شده است تقسیم می‌شود. مثال : اگر داشته باشیم $c = a + b$ ،لکسر باید بتواند ورودی را به صورت $=,+,a, b,c$ در بیاورد.
برای این که به صورت دقیق‌تر کاری که این بخش باید انجام بدهد را مشاهده کنید فایل lexer.rkt را اجرا کنید و خروجی‌آن را مشاهده کنید.

پارسر :
پارسر وظیفه‌ی ساختن درخت را از روی خروجی لکسر خواهد داشت. برای دیدن بهتر خروجی مورد نظر parser.rkt را اجرا کنید. در نهایت باید یک ماژول پارسر داشته باشید که رشته ورودی را بگیرد و درخت پارس شده‌ي آن را خروجی بدهد.
برای پیاده‌سازی این بخش از ابزارهای آماده‌ی موجود در رکت استفاده کنید، برای اطلاعات بیشتر می‌توانید از لینک زیر استفاده کنید.
`https://docs.racket-lang.org/parser-tools/index.html`

۳ پیاده‌سازی اولیه‌ی مترجم (۶۵ 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 keyword هستند که با ؛ از هم جدا شده اند. هر keyword به یکی از شکل‌های زیر است:

- while_statement : عملکرد این دستور دقیقاً همانند دستور while زبان c است.
- if_statement : عملکرد این دستور دقیقاً همانند دستور if زبان c است.

- case_statement : عملکرد این دستور دقیقاً همانند دستور switch case زبان c است.
- assignment_statement : این دستور ابتدا مقدار مقابل = را حساب کرده و سپس آن مقدار را به متغیر قبل از = نسبت می‌دهد.
- return : با رسیدن به این دستور مقدار روبروی آن محاسبه شده و به عنوان جواب نهایی برگردانده می‌شود. این جواب باید به عنوان خروجی برنامه نمایش داده شود. پس از اجرای این دستور، اجرای برنامه متوقف می‌شود.

نوع داده‌های موجود در این زبان عبارتند از:

- number (اعداد صحیح و اعشاری)
- null
- boolean (true, false)
- string (عبارات بین "")
- list

همچنین variable کلمات متشکل از حروف بزرگ و کوچک انگلیسی هستند.
حال اعمال ریاضی روی این نوع داده‌ها به شکل زیر تعریف می‌شوند:

در عبارات بزرگتر، کوچکتر:

- number < number: مقایسه‌ي عادی
- string < string: مقایسه‌ی عادی همانند c
- number > number: مقایسه‌ی عادی
- string > string: مقایسه‌ی عادی همانند c
- list > number: مقایسه‌ی همه اعضا. درصورت عدد نبودن یکی از عضوها،خطا
- list < number: مقایسه‌ی همه اعضا. درصورت عدد نبودن یکی از عضوها،خطا
- list > string: مقایسه‌ی همه اعضا. درصورت رشته نبودن یکی از عضوها،خطا
- list < string: مقایسه‌ی همه اعضا. درصورت رشته نبودن یکی از عضوها،خطا

هر حالت دیگری به جز عبارات بالا منجر به خطا می‌شود.

در عبارات تساوی:

- number == number: مقایسه‌ی عادی
- string == string: مقایسه‌ی عادی
- null == null: همواره درست
- boolean == boolean: مقایسه‌ی عادی
- list == list: مقایسه عضو به عضو. در صورت برابر نبودن تعداد اعضا، false

هر حالت دیگری به جز عبارات بالا false خواهد بود.

در عبارات عدم تساوی:

- number =! number: مقایسه‌ی عادی
- string =! string: مقایسه‌ی عادی
- null =! null: همواره نادرست
- boolean =! boolean: مقایسه‌ی عادی
- list =! list: مقایسه عضو به عضو. در صورت برابر نبودن تعداد اعضا، true

هر حالت دیگری به جز عبارات بالا true خواهد بود.

در عبارات + و ـ و * و / (به جای f هرکدام از این اعمال را می‌توان گذاشت):

- number f number: همانند c
- number f list: اعمال عبارت روی تک‌تک اعضای لیست. در صورت عدد نبودن یکی از اعضا، خطا
- list f number: اعمال عبارت روی تک‌تک اعضای لیست. در صورت عدد نبودن یکی از اعضا، خطا
- boolean + boolean: or منطقی

- boolean * boolean: and منطقی
- boolean + list: اعمال or روی تک‌تک اعضای لیست. در صورت boolean نبودن یکی از اعضا، خطا
- list + boolean: اعمال or روی تک‌تک اعضای لیست. در صورت boolean نبودن یکی از اعضا، خطا
- boolean * list: اعمال and روی تک‌تک اعضای لیست. در صورت boolean نبودن یکی از اعضا، خطا
- list * boolean: اعمال and روی تک‌تک اعضای لیست. در صورت boolean نبودن یکی از اعضا، خطا
- string + string: append کردن دو رشته
- string + list: چسباندن رشته به ابتدای تک تک اعضای لیست. در صورت رشته نبودن یکی از اعضا، خطا
- list + string: چسباندن رشته به انتهای تک تک اعضای لیست. در صورت رشته نبودن یکی از اعضا، خطا
- list + list: چسباندن دو لیست

هر حالت دیگری به جز عبارات بالا خطا خواهد بود.

در عبارات قرینه‌کردن (-cexp):

- number-: قرینه‌ی عدد
- true :-false
- false :-true
- list-: قرینه‌کردن تک‌تک اعضا

هر حالت دیگری به جز عبارات بالا خطا خواهد بود.

در listMember به یک عضو آرایه دسترسی پیدا می‌کنیم. درصورتی که متغیر لیست نباشد، یا عدد داخل [] منفی بوده و یا از اندازه‌ی لیست بزرگتر باشد، خطا رخ می‌دهد.

۴ 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 کند. مثال:

evaluate(a.txt)

۵ هندل کردن ارور ها و پیاده سازی سوییچ کیس و پرینت (۲۰ نمره‌ی امتیازی)

در این بخش شما می‌توانید با پیاده سازی Error Handling و نمایش پیام‌های متناسب با خطای موجود، نمره‌ی اضافه دریافت کنید. همچنین نحوه‌ی پیاده‌سازی کامند های سوییچ کیس و پرینت و خطاهایی که پوشش می‌دهید، بر عهده‌ی خودتان است.