درس طراحی زبان‌های برنامه‌سازی

فاز اول پروژه

دانشکده مهندسی کامپیوتر

دانشگاه صنعتی شریف

نیم سال دوم ۹۹ـ۹۸

استاد:

شیرین بقولی‌زاده

مهلت ارسال:

۲۳ خرداد

ساعت ۲۳:۵۹

به موارد زیر توجه کنید:

* مهلت ارسال این فاز ساعت ۲۳:۵۹ روز جمعه ۲۳ خرداد است.
* پروژه تحویل حضوری 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.
* در مجموعِ تمامی تمارین و پروژه ۷ روز مهلت تاخیر مجاز دارید و پس از تمام شدن این تاخیرهای مجاز به ازای هر روز ۲۵ درصد از کل نمره تمرین شما کم می‌شود.
* 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:

$$command \rightarrow unitcom \mid command; unitcom$$
$$unitcom \rightarrow whilecom \mid ifcom \mid assign \mid return$$
$$whilecom \rightarrow while\ exp\ do\ command\ end$$
$$ifcom \rightarrow if\ exp\ then\ command\ else\ command\ endif$$
$$assign \rightarrow variable = exp$$
$$return \rightarrow return\ exp$$
$$exp \rightarrow aexp \mid aexp > aexp \mid aexp < aexp \mid aexp == aexp \mid aexp\ != aexp$$
$$aexp \rightarrow bexp \mid bexp - aexp \mid bexp + aexp$$
$$bexp \rightarrow cexp \mid cexp * bexp \mid cexp/bexp$$
$$cexp \rightarrow -cexp \mid (exp) \mid posNumber \mid null \mid variable \mid true \mid false$$
$$\mid string \mid list \mid variable\ listmem$$
$$list \rightarrow [listValues] \mid [\,]$$
$$listValues \rightarrow exp \mid exp, listValues$$
$$listmem \rightarrow [exp] \mid [exp]\ listmem$$

نکته: این گرامر LALR (۱) است بنابراین برای پیاده‌سازی‌های بعدی نیاز به هیچ‌گونه تغییری در آن نیست. توجه کنید که posNumber 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)

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 (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 unitcom هستند که با ; از هم جدا شده اند. هر unitcom به یکی از شکل‌های زیر است:

- whilecom : عملکرد این دستور دقیقاً همانند دستور while زبان c است.
- ifcom : عملکرد این دستور دقیقاً همانند دستور if زبان c است.
- assign : 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
- list == number: مقایسه‌ی همه اعضا. درصورت عدد نبودن یکی از عضوها، false
- list == string: مقایسه‌ی همه اعضا. درصورت رشته نبودن یکی از عضوها، false
- list == boolean: مقایسه‌ی همه اعضا. درصورت boolean نبودن یکی از عضوها، false
- list == null: مقایسه‌ی همه اعضا. درصورت null نبودن یکی از عضوها، 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
- list =! number: مقایسه‌ی همه اعضا. درصورت عدد نبودن یکی از عضوها، true

- string =! list: مقایسه‌ی همه اعضا. درصورت رشته نبودن یکی از عضوها، true
- boolean =! list: مقایسه‌ی همه اعضا. درصورت boolean نبودن یکی از عضوها، true
- null =! list: مقایسه‌ی همه اعضا. درصورت null نبودن یکی از عضوها، 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
- false-: true
- true-: false
- list-: قرینه‌کردن تک‌تک اعضا. درصورت قرینه‌پذیر نبودن یک عضو، خطا

هر حالت دیگری به جز عبارات بالا خطا خواهد بود.

در listmem 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 Error Handling (۲۰ نمره‌ی امتیازی)

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