بسمه تعالی

دانشگاه صنعتی شریف

زبان‌های برنامه‌سازی

پروژه‌های امتیازی

استاد

دکتر ایزدی

تهیه و تدوین

تیم دستیاران درس زبان‌های برنامه‌سازی

پاییز ۱۴۰۲

توجه داشته باشید!

- پروژه‌های امتیازی برای علاقه‌مندان به زبان‌های برنامه‌نویسی و برنامه‌نویسی تابعی و برای عمیق کردن و متنوع‌سازیِ دانش‌تان در نظر گرفته شده‌اند. **لطفا اگر صرفا پروژه را به چشم نمره امتیازی می‌بینید، از پرداختن به آن صرف نظر کنید.**
- تمامی پروژه‌ها در **تیم‌های دونفره** باید انجام شوند و **دو نمره امتیازی** خواهند داشت. هر تیم حداکثر یک پروژه را می‌تواند انتخاب کند.
- هر پروژه را **حداکثر چهار تیم** می‌تواند انتخاب کند. برای انتخاب پروژه، شماره دانشجویی اعضای تیم دونفره خود را در **این لینک** کامنت کنید.
- برای هر یک از پروژه‌ها باید **یک گزارش بنویسید**، توضیحاتی کلی درباره نحوه انجام و پیاده‌سازی پروژه بدهید و چند نمونه تست و خروجی را در گزارش‌تان بیاورید.
- کد و گزارش خود را زیپ کرده و در سامانه کوئرا آپلود کنید.
- ددلاین تمامی پروژه‌ها **13 بهمن** می‌باشد. این **ددلاین غیرِ قابلِ تمدید** است!

I. ساخت یک فهرست کارهای روزانه ساده با Elixir

طراح: اشکان خادمیان

فصل ۰: توضیحات اولیه

مقدمه

الکسیر یک زبان برنامه‌نویسی تابعی[1] و پویا[2] برای ساخت برنامه‌های کاربردی با محوریت بر روی ویژگی‌های **مقیاس‌پذیری**[3] و **قابلیت نگهداری**[4] می‌باشد.

در سال ۲۰۱۲ تیم ایجاد زبان روبی[5] تصمیم گرفتند یک زبان تابعی ایجاد کنند که ویژگی‌هایی مشابه زبان روبی داشته باشد. بستر مورد استفاده آن‌ها برای ایجاد این زبان جدید BEAM VM (ماشین مجازی اجرا کننده‌ی زبان تابعی اِرلنگ[6]) بود. در ادامه استفاده از این ماشین مجازی، یکی از نقاط قوت اصلی الکسیر شد چرا که بسیاری از پکیج‌ها و کتابخانه‌هایی که پیش‌تر برای اِرلنگ نوشته شده بودند حال برای الکسیر قابل استفاده می‌شدند. همچنین این تصمیم به الکسیر ویژگی‌های محبوب اِرلنگ - شامل **تاخیر کم**[7]، **توزیع‌پذیری**[8] و **مقاومت نسبت به خطا**[9] - را هدیه داد.

رهنمود کلی

اهداف

- یادگیری مبانی استفاده از زبان صنعتی و کاملا تابعی الکسیر
- یادگیری مبانی از استفاده از ابزار ساخت خودکارِ Mix
- استفاده از رویه کاری[10] شدیدا توصیه شده‌ی مستندسازی درون کد
- استفاده از رویه کاری توصیه شده‌ی آزمون خودکار در الکسیر

---

[1] Functional
[2] Dynamic
[3] Scalability
[4] Maintainability
[5] Ruby
[6] Erlang
[7] Low-Latency
[8] Distribution
[9] Fault Tolerance
[10] Practice

نمره‌دهی

★ در ادامه‌ی مستندات، نمره‌دهی بخش‌های مختلف پروژه مطرح شده و در بسیاری از آن‌ها، نمرات بر حسب درصد ذکر شده‌اند.

★ **برای از دست ندادن نمره، حتما خطوط قرمز رنگِ این مستند را مطالعه کرده باشید.**

منابع یادگیری

یادگیری زبان الکسیر اگر فرد با زبان‌های سطح بالایی چون جاوااسکریپت یا پایتون و همچنین یک زبان تماما تابعی مانند رَکِت آشنایی داشته باشد زحمت چندانی نخواهد داشت. ما به عنوان پیشنهاد منابع زیر را توصیه می‌کنیم:

- اگر به خواندن کتاب عادت دارید این کتاب را در اینترنت جستجو کنید: Elixir in Action[11]
- اگر به خواندن مستندات عادت دارید این سایت یکی از بهترین منابع یادگیری الکسیر است.
- اگر به دیدن ویدئوهای آموزشی عادت دارید، این ویدئوی کوتاه توضیحات نسبتا جامعی ارائه می‌کند.

در صورتی که منابع فوق به نظرتان کافی نبودند یا به هر نحوی پیچیده بودند، از تیم دستیاران برای پیدا کردن منحنی یادگیری خود کمک بگیرید.

فصل ۱: نیازمندی‌ها (+ بارم‌بندی)

چشم‌انداز[12]

در این پروژه از شما یک سیستم کوچک مدیریت کارهای روزانه خواهیم خواست تا بتوانیم وظایف روزمره خود را توسط آن مشاهده، نگهداری و اداره کنیم. نیازی به رابط کاربری خاصی در این پروژه نخواهد بود و عملیات‌ها درون CLI انجام خواهد شد. از سوی دیگر، به علت آن که کیفیت مصنوعات تحویل دادنی شما برای ما حائز اهمیت است، فاکتورهای کیفی‌ای چون مستندسازی و آزمون خودکار نویسی باید در طول پروژه رعایت شده باشد.

[11] در صورتی که کتاب مذبور را پیدا نکردید از تیم دستیاران درخواست کنید.

[12] Vision

ملاحظات کیفی: نیازمندی‌های غیروظیفه‌ای (۰.۸ نمره)

۱. لیست وظایف باید به صورت سخت ذخیره شوند از این رو ذخیره رو یک فایل CSV توصیه می‌شود. دقت کنید برای هندل کردن خواندن‌ها و نوشتن‌ها روی CSV **نمی‌توانید** از پکیج‌های موجود برای این موضوع استفاده کنید و باید خودتان توابع مورد نیاز را پیاده سازی کنید. (۰.۱ نمره)

۳. مستندسازی شهروند درجه اول زبان الکسیر است. با استفاده از منابعی که در اختیارتان گذاشته شده و دیگر منابعی که میابید نحوه صحیح مستندسازی ماژول‌ها و توابع را آموخته، تمام ماژول‌ها و توابع پروژه خود را دارای مستندات docstring کنید. (۰.۲ نمره)

۴. آزمون[13] عضو جداناپذیر پروژه‌های این زبان است. اطمینان حاصل کنید تمام منطق اصلی برنامه‌تان به انضمام رابط کاربری فراهم آورده شده دارای آزمون خودکار باشد و پیش از تحویل تمام این آزمون‌ها پاس شوند. (۰.۵ نمره)

ملاحظات کیفی: نیازمندی‌های برجسته معماری (۰.۲ نمره)

۱. پروژه خود را باید استفاده از ابزار Mix راه‌اندازی کنید. این ابزار چارچوب اولیه بسیاری از خواسته‌های این پروژه را در اختیارتان قرار خواهد داد. (۰.۱ نمره)

۲. از تحویل دادن کد تک-ماژولی خودداری کنید. انتظار می‌رود در حداقل حالت پروژه شما دارای دو ماژول باشد که یکی وظایف مربوط به CLI (ورودی - خروجی) را بر عهده دارد و دیگری منطق اصلی برنامه را شامل می‌شود. ماژول‌های مختلف خود را در فایل‌های مختلف قرار دهید که خوانایی کد تهیه شده بالا باشد. (۰.۱ نمره)

نیازمندی‌های وظیفه‌ای (۱ نمره)

انتظار می‌رود پروژه شما چنین دستوراتی قبول کند و به ازای هر دستور عملیات مناسبی انجام دهد و به خروجی مطلوبی برسد. دستورات مورد انتظار به شرح زیر هستند.

دستورات اولیه (۰.۱ نمره)

- راه اندازی: برنامه باید بتواند اجرا شود و بلافاصله پس از اجرا، فایل حاوی وظایف را خوانده و راه اندازی کند و لیست کارهای روزمره از روی آن بسازد تا آماده دریافت دستورات بعدی شود.
- پایان: برنامه با یک دستور باید بتواند خاتمه یابد.

[13] Test

دستور ذخیره لیست وظایف (۰.۱ نمره)

با این دستور، وضعیت فعلی لیست وظایف به روی فایل ذخیره می‌شوند.

دستور افزودن وظیفه جدید (۰.۲ نمره)

با این دستور، وظیفه جدید از کاربر دریافت شده و به لیست وظایف اضافه می‌شود. هر وظیفه یک تیتر، اولویت، تاریخ اضافه شدن (که به صورت پیش‌فرض لحظه اضافه شدن وظیفه وگرنه برابر تاریخ وارد شده توسط کاربر می‌باشد)، و یادداشت خواهد داشت. وظیفه‌ی جدید به آخر لیست وظایف افزوده خواهد شد.

دستور حذف وظیفه (۰.۱ نمره)

با این دستور یک وظیفه را می‌توان حذف کرد.

دستور نمایش وظایف (۰.۲ نمره)

با این دستور، لیست وظایف به یک قالب قابل فهم (ولی نه الزاما خیلی پر زحمت) نمایش داده می‌شود.

دستور تغییر ترتیب نمایش (۰.۳ نمره)

با این دستور، می‌توان لیست وظایف را بر اساس تاریخ افزودن وظیفه یا اولویتشان مرتب نمود. شایان توجه است تغییر ترتیب به هنگام ذخیره لیست وظایف نیز تاثیر خود را خواهد گذاشت (کارها با آخرین ترتیب مطرح شده روی فایل ذخیره خواهند شد). از این رو انتظار می‌رود بابت سطح اهمیت این نیازمندی، **یک آزمون که آن را مورد توجه قرار دهد نوشته باشید.**

ملاحظه مهم

پنجاه درصد نمره این پروژه مختص عملیات‌ها و وظایف سیستم تحویل داده شده توسط شما بوده و پنجاه درصد باقی مربوط به ملاحظات کیفی (۲۰ نمره مربوط به معماری پیشنهادی و ۳۰ نمره مربوط به نیازمندی‌های غیروظیفه‌ای) می‌باشد. **شایان توجه است اگر از بخش «نیازمندی‌های وظیفه‌ای» نصف نمره را دریافت نکنید، نمرات مربوط به ملاحظات کیفی - ۵۰٪ نمره‌ی کل - برای شما در نظر گرفته نمی‌شود** چرا که در وهله اول کد شما باید کار کند تا باقی ملاحظات کیفی در مورد آن موضوعیت داشته باشند.

فصل ۲: تحویل دادنی‌ها

انتظار می‌رود موارد زیر در هنگام تحویل شما وجود داشته باشد؛ لطفا به ملاحظات موجود در هر موردِ تحویل توجه اکید داشته باشید.

پروژه گیتهاب/گیتلب

انتظار می‌رود با هم‌تیمی خود برای اجرای این پروژه یک **مخزنِ خصوصی**[14] در گیتهاب یا گیتلب ایجاد کرده باشید. بعد از ضرب‌العجل[15] پروژه تعدادی از اعضای تیم تدریس به مخزن شما اضافه خواهند شد و کد شما را بررسی خواهند کرد. **شایان توجه است commit هایی که بعد از موعد ضرب‌العجل روی مخزن وارد شوند در نظر گرفته نخواهند شد**؛ پروژه ابتدا بر روی آخرین commit پیش از ضرب‌العجل بازگردانی شده سپس نمره‌دهی می‌شود.

گزارشات متنی

همراه با کد یک حداقلی از توضیحات متنی به قالب PDF باید در اختیار تیم تدریس تا پیش از ضرب‌العجل قرار دهید. این گزارشات باید شامل موارد زیر باشد:

- نحوه همکاری هر عضو از گروه و تاثیرگذاری او روی خروجی‌های پروژه یا بخش‌های مختلف سیستم.
- یک توضیحی از CLI که در واقع مانند راهنمای کاربری[16] عمل کند و تیم دستیاران کمک کند چگونه برنامه شما را اجرا کنند.
- دستورات لازم و نحوه‌ی اجرای آزمون‌های خودکار سطح کدتان - در صورت وجود.
- ملاحظه یا مشکلاتی که در سیستم خودتان پیدا کردید و لازم می‌دانید ما از آن مطلع باشیم.

لازم به ذکر است **نیازی نیست معماری و ساختار کد را توضیح دهید**؛ از این جهت که علی‌الخصوص اگر نیازمندیِ مربوط به مستندات درون کد را محقق کنید، از طریق همان docstring سطح کد می‌توان کد شما را درک نمود.

با توضیحات مطرح شده انتظار می‌رود گزارشاتتان بیش از دو یا سه صفحه متن نشود. **اما همین حداقل متن اگر فراهم نیاید به هر حدی که درک پروژه و کار با سیستم خروجی برای دستیاران درس دشوار شود، ممکن است تا ۵۰ درصد نمره دریافتی خود را از دست بدهید.**

[14] Private
[15] Deadline
[16] User Manual

II. پیاده‌سازی دیتابیس SimpleDB با استفاده از Clojure

طراحان: محمد بروغنی - آرمان محمدی

توضیح پروژه

موضوع این پروژه، پیاده‌سازی دیتابیس میباشد. ویژگی های مدنظر SimpleDB در ادامه توضیح داده شده اند. باید که برای پیاده‌سازی، از زبان clojure استفاده شود.

برای راحت تر شدن بررسی‌کردن کد و توضیح بهتر، از قالب پروژه ای که در این مخزن قرار داده شده، استفاده کنید. میتوانید از fork برای در اختیار داشتن مخزن استفاده کنید.

یک سری سناریو برای تست کردن کد ها هم در این مخزن در اختیار قرار داده شده که می‌توانید برای بررسی درست‌بودن پیاده‌سازی از آنها استفاده کنید. توجه کنید که لازم است پروژه را به صورتی پیاده سازی کنید که مشکلی در اجرای تست‌ها وجود نداشته باشد. (برای ارزیابی هم از همین تست‌ها هنگام تحویل پروژه استفاده خواهد شد.)

توجه کنید که استفاده کنندگان این دیتابیس، مثل یک library از این دیتابیس استفاده خواهند کرد و نیازی به پیاده‌سازی این دیتابیس به صورت یک سرویس مجزا از برنامه استفاده کننده نیست.

کارکردهای اصلی

اضافه‌کردن entity به دیتابیس (۰.۱ نمره)

به هر داده‌ای که در SimpleDB ذخیره می‌کنیم، یک entity می‌گوییم. entity ها هیچ شکل خاصی ندارند و تنها ویژگی‌ای که باید داشته‌باشند این است که به شکل map باشند.

در فایل insert_test.clj عملکرد مورد نیاز تعریف شده است.

کوئری‌زدن (۰.۵ نمره)

برای زدن query از فرمتی مشابه datalog استفاده می‌کنیم. در این لینک توضیح در مورد این فرمت query زدن داده شده است.

در فایل query_test.clj عملکرد مورد نیاز تعریف شده است.

ذخیره کردن وضعیت فعلی db در فایل و استفاده کردن از این فایل (۰.۱۵ نمره)

در فایل flush_and_retrieve_test.clj عملکرد مورد نیاز تعریف شده است.

ذخیره در دیسک و batching ی (۰.۵ نمره)

موقع ساختن db پارامتر هایی به نام های batch-size و in-memory? به تابع سازنده db داده میشود.

وقتی که in-memory? مقدار false دارد، باید در هر لحظه اطلاعات db در فایلی ذخیره سازی شود. در غیر اینصورت اطلاعات در memory قرار می‌گیرد.

موقعی که batch-size مقدار n دارد، دیتابیس بعد هر عملیات insert فایل شامل اطلاعات دیتابیس را تغییر نمی‌دهد و این تغییرات را زمانی روی فایل اجرا می‌کند تعداد این تغییرات اعمال نشده به اندازه n شود. همچنین اگر که query زده شود، کل این تغییرات اعمال‌نشده را باید اجرا کنیم.

در فایل storing_test.clj عملکرد مورد نیاز تعریف شده است.

پیاده‌سازی نسخه unblocking برای توابع insert و query از آن (۰.۲۵ نمره)

در حالت عادی، (sync) بعد از صدا زدن insert ، entity به دیتابیس اضافه میشود، و بعد از صدا زدن query ، entity های مدنظر برگردانده میشوند. در مدل async ، به این شکل عمل می‌کنیم که این دو تابع، یک callback function به عنوان ورودی می‌گیرند. بعد از صدا زدن این توابع، در یک ترد (به غیر از ترد اصلی) به اجرای عملیات مورد نظر می‌پردازد و بعد از تمام شدن، callback function صدا زده می‌شود. (در نتیجه تردی که این دو تابع را صدا میزند، block نخواهد شد و بعد از صدا زدن این دو تابع، بالافاصله اجرای باقی کد ادامه پیدا می‌کند.)

در یکی از تست ها، یکی از کاربرد های نسخه async را مشاهده می‌کنیم. در این تست از ساختاری به نام coroutine استفاده شده. در این روش، فرض می‌کنیم که یک مجموعه محدودی از ترد ها در اختیار داریم که coroutine ها در داخل از ترد ها اجرا می‌شوند.

نحوه اجرای coroutine ها به این شکل است که بعد از ساخته شدن هر coroutine مثل c، این coroutine منتظر میماند که یکی از ترد ها امکان اجرای c را داشته باشد و مشغول اجرای coroutine دیگری نباشد. زمانی که c در یک ترد اجرا میشود، تا زمانی که از تابع !< و یا !> را روی a thread safe channel (queue) استفاده نکنیم، c در ترد اجرا می‌شود. در صورت صدا زدن این دو تابع، اگر که در coroutine در حال گرفتن داده از یک channel خالی و یا در حال قرار دادن یک مقدار در یک channel

پر باشیم، اجرا شدن c در ترد متوقف می‌شود و ترد دوباره آزاد می‌شود. هر زمانی که channel به حالت مطلوب برگردد، اجرای c در یکی از thread ها ادامه پیدا خواهد کرد.

در فایل aync_test.clj عملکرد مورد نیاز تعریف شده است.

Aggregation Pipeline (نمره ۰.۵)

یکی از کارکرد هایی که برخی اوقات از یک سرویس دیتابیس می‌خواهیم، این است که بعد از گرفتن داده مدنظر بعد از اجرای یک query، یک سری تغییرات روی این داده انجام بدهد و نتیجه این تغییرات به ما برگردانده شود. برای این بخش از پروژه، توابع group-by ، sort ، sum، filter و count را پیاده سازی کنید. این توابع را باید به صورتی پیاده سازی کنید که خروجی آنها یک transducer باشد.

در فایل pipeline_test.clj عملکرد مورد نیاز تعریف شده است.

منابع مطالعه

نصب کردن leiningen (یک build tool برای clojure)

برای نصب کردن از این لینک استفاده کنید.

IDE

برای راه‌اندازی در vscode میتوانید از اکستنشن calva استفاده کنید. از این لینک برای آشنایی می‌توانید استفاده کنید.

برای راه‌اندازی در emacs از این لینک استفاده کنید.

منابع برای آشنایی با clojure

کتاب Getting Clojure برای آشنایی با زبان منبع مناسبی است. مطالعه بخش اول کتاب برای آشنایی با syntax اولیه، و همچنین فصل های ۱۳، ۱۶، ۱۷ و ۱۸ برای آشنایی با بعضی امکانات زبان (که در پروژه هم کاربرد دارند) توصیه می‌شود.

می‌توانید همین فصل های پیشنهادی را در منابع دیگری مثل ۱ یا ۲ هم مطالعه کنید. برای آشنایی بیشتر با موضوع state management هم میتوانید فصل ۳ و ۴ این کتاب را مطالعه کنید.

برای آشنایی با transducer لینک های ۱، ۲ و ۳ پیشنهاد میشوند.

III. پروژه COBOL

طراح: مهدی صابر

نکات اولیه پروژه

- برای پروژه خود **حتما** گزارش تهیه کنید. دقت کنید که داکیومنت‌نویسی یک مهارت بسیار مهم در ارائه یک پروژه می‌باشد و نصف نمره این پروژه به گزارش شما اختصاص دارد و بدون ارسال گزارش، تمامی این بخش از نمره را **از دست خواهید داد!**
- شما تنها مجاز به تغییر در فایل‌های interpreter.hs و environment.hs می‌باشید و هرگونه تغییری روی فایل‌های lexer.hs، lexer.x، parser.hs، parser.y نادیده گرفته خواهد شد.
- مستندات مورد نیاز به‌طور کامل در این لینک موجود است.
- پیشنهاد می‌شود که حتما حداقل یک بار توضیحات موجود در ریپازیتوری را به طور کامل بخوانید تا با زبان نوشته شده آشنایی بیشتر کسب کنید. همچنین یک قطعه کد آزمایشی با نام code.txt در ریپازیتوری موجود است تا نحوه کلی ساختار زبان و یک برنامه قابل اجرا را مشاهده کنید.

آشنایی با گرامر COBOL

در این پروژه زبانی مشابه زبان COBOL به شما داده می‌شود و انتظار می‌رود تا تغییرات خواسته شده را در زبان اعمال کنید. دقت کنید که گرامر داده شده تنها برای تسلط بیشتر شما به زبان می‌باشد و کد مفسر آن به طور کامل در زبان Haskell پیاده شده است. گرامر این زبان به شکل زیر می‌باشد:

1. Program → Identification_division Data_division Procedure_division End_statement
   | Identification_division Procedure_division End_statement
2. Identification_division → `IDENTIFICATION` `DIVISION` `.`
   `PROGRAM` `-` `ID` `.` ID
3. Data_division → `DATA` `DIVISION` `.` Data_section
4. Procedure_division → `PROCEDURE` `DIVISION` `.` Procedure_section
5. End_statement → `END` `PROGRAM` `.` ID | ϵ
6. Data_section → `WORKING` `-` `STORAGE` `SECTION` `.` Data_declarations
7. Procedure_section → Paragraph | Procedure_section Paragraph
8. Data_declarations → Data_declarations Variable_declaration | ϵ
9. Paragraph → ID `.` Statements
10. Variable_declaration → INTEGER ID `PIC` Datatype Default_value `.`

11. Datatype → `A` | `S9`
12. Deault_value → `VALUE` Expression | ∈
13. Value → STRING | NUMBER
14. Statements → Statements Statement `.` | Statement `.`
15. Statement → Move_stmt | Arithmetic_stmt | Compute_stmt | If_stmt | Perform_stmt
    | Function_stmt | Display_stmt | `EXIT`
16. Move_stmt → `MOVE` Expression `TO` Variables
17. Variable → ID | ID `[` Expression `]`
18. Arithmetic_stmt → Add_stmt | Sub_stmt | Mult_stmt | Div_stmt
19. Add_stmt → `ADD` Expressions `TO` Variable
    | `ADD` Expressions `GIVING` Variable
20. Sub_stmt → `SUBTRACT` Expressions `FROM` Variable
    | `SUBTRACT` Expressions `,` Expression `GIVING` Variable
21. Mult_stmt → `MULTIPLY` Expressions `BY` Variable
    | `MULTIPLY` Expressions `GIVING` Variable
22. Div_stmt → `DIVIDE` Expression `INTO` Variable
    | `DIVIDE` Expression `INTO` Expression `Remainder` Variable
    | `DIVIDE` Expression `,` Expression `GIVING` Variable
23. Compute_stmt → `COMPUTE` Variable `=` Expression
24. If_stmt → `IF` Disjunction`THEN` Statements `ELSE` Statements `END` `-` `IF`
    | `IF` Disjunction `THEN` Statements
25. Perform_stmt → `PERFORM` Statements `UNTIL` Disjunction | `PERFORM` ID
    | `PERFORM` `UNTIL` Disjunction Statements `END` `-` `PERFORM`
    | `PERFORM` `VARYING` ID `FROM` Expression `TO` Expression Step
    `UNTIL` Disjunction Statements `END` `-` `PERFORM`
26. Step → `BY` Expression | ∈
27. Function_stmt → `FUNCTION` ID `=` Expressions `=>` Expression
28. Display_stmt → `DISPLAY` Expression
29. Variables → Variables `,` Variable | Variable
30. Expressions → Expressions `,` Expression | Expression
31. Disjunction → Conjunction | Disjunction `OR` Conjunction
32. Conjunction → Inversion | Conjunction `AND` Inversion
33. Inversion → `NOT` Inversion | Comparison
34. Comparison → Eq | Not_eq | Lt | Lte | Gt | Gte
35. Eq → Expression `EQUALS` Expression
36. Not_eq → Expression `NOT` `EQUALS` Expression
37. Lt → Expression `<` Expression
38. Lte → Expression `<=` Expression

39. Gt → Expression `>` Expression
40. Gte → Expression `>=` Expression
41. Expression → `(` Disjunction `)` `?` Expression `:` Expression | Sum
42. Sum → Sum `+` Term | Sum `-` Term | Term
43. Term → Term `*` Factor | Term `/` Factor | Factor
44. Factor → `+` Factor | `-` Factor | `(` Expression `)` | Element
45. Element → Variable | ID `(` Expressions `)` | Value

نکات گرامر:

- گرامر بالا تنها برای آشنایی بیشتر با COBOL و سادگی بیشتر برای انجام پروژه در اختیار شما قرار داده شده است.
- توجه کنید که لکسر و پارسر و همچنین مفسر این گرامر به طور کامل نوشته شده و در ریپازیتوری در اختیار شما قرار داده شده است و هیچ تغییری در گرامر از جانب شما نیاز نیست.

آشنایی با زبان Haskell

این پروژه در زبان Haskell صورت می‌گیرد. بنابراین نیاز است تا تسلط کافی بر آن داشته باشید. برای یادگیری این زبان می‌توانید از لینک‌های زیر استفاده کنید:

- 194 CIS
- Haskell World Real
- Programmers Imperative for Haskell

اهداف پروژه

از اهداف این پروژه می‌توان به موارد زیر اشاره کرد:

- یادگیری زبان Haskell
- آشنایی ابتدایی با زبان COBOL
- یادگیری پیاده‌سازی توابع بازگشتی
- یادگیری پیاده‌سازی type inference و type checking
- یادگیری تغییر و کار با environment ها

پیاده‌سازی توابع بازگشتی (۰.۵ نمره)

در این بخش شما باید کد را به شکلی تغییر دهید که بتواند توابعی به شکل بازگشتی تعریف کند. در حال حاضر این کد این قابلیت را نداشته و با صدا زدن یک تابع در خودش اجرای برنامه متوقف می‌شود. برای این کار می‌توانید فایل‌های **interpreter** و **environment** را تغییر دهید.

پیاده‌سازی قابلیت صدا زدن تمامی پاراگراف‌های برنامه (۰.۲ نمره)

در کد مفسری که در اختیار شما قرار گرفته است، پاراگراف‌های یک برنامه به شکل توابع در c تعریف می‌شوند. به این معنی که درون یک پاراگراف، نمی‌توانیم پاراگرافی که در پایین این پاراگراف تعریف شده را صدا بزنیم و تنها می‌توانیم پاراگراف‌هایی که بالای این پاراگراف تعریف شده‌اند را اجرا کنیم. شما در این قسمت از پروژه باید کد **interpreter** و **environment** را به گونه‌ای تغییر دهید که یک پاراگراف قابلیت صدا زدن همه پاراگراف‌ها (حتی پایین‌تر از خود) را داشته باشد.

پیاده‌سازی type checking مفسر (۱ نمره)

در قسمت‌های مختلف کد مفسر، کامنت‌هایی مبنی بر چک کردن دیتاتایپ در عملیات مختلف قرار گرفته است. این کد هیچگونه type checking خاصی انجام نمی‌دهد و هنگام برخورد به یک مشکل تایپی مانند جمع کردن یک عدد با یک رشته، با اورورهای مربوط به خود زبان Haskell از برنامه بیرون می‌رود. مثال دیگر حالت‌هایی است که تلاش می‌کنیم در برنامه مقداری را در متغیری بریزیم که از آن جنس نیست، برای مثال می‌خواهیم یک مقدار عددی را درون یک متغیر از جنس رشته قرار دهیم. در این حالات باید اورورهای مربوطه پرینت شوند و جزئیات آن بیان شود. پیاده‌سازی جزئیات آن بر عهده شماست و از فرمت خاصی پیروی نمی‌کند اما باید به گونه‌ای باشد که با مشاهده ارور، علت خطا از لحاظ منطبق نبودن تایپ‌ها مشخص باشد و متوجه اشتباه کد شده باشیم و صرفا اورورهای خلاصه و نامفهومی مانند `type error!` و مشابه آنها پرینت نشود.

پیاده‌سازی type inference مفسر (۰.۳ نمره)

در این قسمت باید type inference را پیاده کنید. برای پیاده‌سازی این قسمت تنها کافیست که برنامه بدون DATA DIVISION نیز اجرا شود. زمانی که بخش DATA DIVISION تعریف نمی‌شود، مفسر باید خود به خود بر اساس مقداردهی‌های هنگام برنامه تشخیص دهد که تایپ هر متغیر چیست. دقت کنید که هنگامی که یک متغیر برای اولین بار مقداردهی می‌شود، تایپ آن مشخص

می‌شود و نمی‌توان پس از آن تایپ آن را تغییر داد. برای مثال نمی‌توان یک متغیر را ابتدا با یک عدد مقداردهی کرد و در اواسط برنامه یک رشته درون آن قرار دهیم.

IV. طراحی مفسر زبان Let و صحت‌سنجی آن در Coq

طراح: حمیدرضا کلباسی

صحت‌سنجی نرم‌افزارها یکی از مسائل مهم است. روش‌های زیادی برای صحت‌سنجی یک نرم‌افزار وجود دارد، مثل نوشتن تست، استفاده از فازرها، و ...اما کامل‌ترین روش صحت‌سنجی نرم‌افزار روش‌های صوری هستند که تضمین می‌کنند برنامه همواره و به ازای هر ورودی درست و مطابق با اسپسیفیکیشن خود عمل می‌کند. ابزارهای زیادی برای صحت‌سنجی صوری نرم‌افزار وجود دارد که ما در این پروژه از Coq استفاده می‌کنیم. Coq یک زبان برنامه‌نویسی است که به کمک تایپ سیستم غنی آن می‌توان اثبات‌های ریاضی را نیز در آن انجام داد. برای آشنایی با Coq می‌توانید ارائه دانشجویان در ترم پیش را ببینید و برای یادگیری آن می‌توانید از منابع قرار داده شده در ضمیمه استفاده کنید.

در این پروژه ما ابتدا درون Coq یک مفسر برای زبان Let که در طول درس معرفی شد خواهیم ساخت و سپس درون Coq به صورت صوری و چک شده با کامپیوتر ثابت می‌کنیم که مفسر ساخته شده خواص لازم برای اجرای یک برنامه Let را دارد.

فاز اول: ساخت مفسر(۰.۵ نمره)

یک برنامه Let را در Coq به کمک ساختار زیر تعریف می‌کنیم:

```
Inductive Expression :=
| Num (n: nat)
| Var (v: string)
| Plus (l: Expression) (r: Expression)
| Let (var: string) (initializer: Expression) (result: Expression).
```

شما باید یک تابع

```
evaluate: Expression -> EvalResult
```

بنویسید که برنامه داده شده را اجرا کند و نتیجه را در قالب داده ساختار زیر برگرداند:

```
Inductive EvalResult :=
| Ok (n: nat)
| Error.
```

که در صورتی که از یک متغیر تعریف نشده استفاده شده باشد خروجی خطا است و در غیر این صورت عدد باید محاسبه شود.

فاز دوم: اثبات درستی مفسر (۱ نمره)

درباره تابعی که در قسمت قبل نوشتید، گزاره‌های زیر را ثابت کنید:

```
Lemma evaluate_plus: forall n1 n2 e1 e2, evaluate e1 = Ok n1 ->
evaluate e2 = Ok n2
  -> evaluate (Plus e1 e2) = Ok (n1 + n2).
Admitted.

Lemma evaluate_plus_comm: forall e1 e2, evaluate (Plus e1 e2) =
evaluate (Plus e2 e1).
Admitted.

Lemma evaluate_error: forall e, evaluate e = Error <-> exists v,
IsFree v e.
Admitt
```

در قسمت بالا IsFree به صورت زیر تعریف شده است:

```
Inductive IsFree (var: string) : Expression -> Prop :=
| IsFreeVar: IsFree var (Var var)
| IsFreePlusLeft (e1: Expression) (e2: Expression): IsFree var e1 ->
IsFree var (Plus e1 e2)
| IsFreePlusRight (e1: Expression) (e2: Expression): IsFree var e2
-> IsFree var (Plus e1 e2)
| IsFreeLetInit (other_var: string) (e1: Expression) (e2:
Expression):
      IsFree var e1 -> IsFree var (Let other_var e1 e2)
| IsFreeLetBody (other_var: string) (e1: Expression) (e2:
Expression):
      IsFree var e2 -> ~ var = other_var -> IsFree var (Let
other_var e1 e2).
```

بخش های جانبی

این بخش ها هر کدام بیست و پنج صدم نمره دارند که از چهارتا می‌توانید حداکثر دو تا را انجام دهید:

- تبدیل کد به Ocaml یا Haskell: Coq قابلیت تولید خروجی به زبان OCaml و Haskell را دارد. از این قابلیت استفاده کنید و به کمک آن یک برنامه بنویسید که یک برنامه به زبان Let از ورودی استاندارد ورودی بگیرد، آن را parse کند، و نتیجه پارس شده را به مفسر اثبات شده Coq بدهد تا آن را اجرا کند.
- بررسی کامل بودن اسپسیفیکیشن:
- گزاره‌هایی که در بالا مطرح شد، برای اثبات درستی مفسر کافی نیستند؛ یعنی یک تابع evaluate می‌تواند وجود داشته باشد که در یک برنامه درست کار نکند ولی هم چنان همه گزاره‌ها برای آن درست باشند و ثابت شوند. یک چنین مثال نقضی را ارائه دهید. سپس مجموعه گزاره‌های لازم و کافی برای درستی یک مفسر در زبان Let را پیدا کنید، و در Coq ثابت کنید که هر تابعی که این شرایط شما را داشته باشد، شرایط بالا را نیز خواهد داشت.
- افزودن شرط و تابع بازگشتی به زبان: به ساختار Expression شرط و تابع بازگشتی را اضافه کنید و در تابع evaluate از آن‌ها پشتیبانی کنید. با توجه به این که زبان جدید ساخته شده تورینگ کامل است و Coq تورینگ کامل نیست، نمی‌توان یک مفسر کامل برای این زبان را در Coq نوشت. بنابراین شما باید با اضافه کردن یک محدودیت خلاقانه، بدون خدشه‌دار کردن ماهیت زبان، آن را غیر تورینگ کامل کنید تا بتوانید آن را در Coq اجرا کنید.
- مشارکت در پروژه CompCert: پروژه CompCert یک کامپایلر C است که در Coq نوشته شده است و درستی عملکرد آن اثبات شده است. از این کامپایلر در کاربردهای بحرانی ایمن مثل خودرو، هواپیما، وسایل پزشکی و ... استفاده می‌شود. یکی از ایشوهای باز در این کامپایلر را پس از تایید TA حل کنید و برای آن pull request دهید. نیازی نیست که pull request شما در مخزن اصلی ادغام شود اما اگر ادغام شود می‌توانید تا بیست و پنج صدم نمره اضافی دریافت کنید.

منابع برای یادگیری

- آشنایی با Coq ارائه ترم گذشته دانشجویان
- یادگیری به عنوان یک زبان برنامه‌نویسی
- یادگیری به عنوان یک ابزار اثبات