فاز دوم پروژه

کامپایلر

دکتر دلدار

نحوه تحویل:

فایل های مورد نظر را در zip که با نام، شماره دانشجویی و شماره فاز نام گذاری شده است

(مانند Ph1_LastName_40018173) در سامانه یکتا بارگذاری کنید.

در فاز اول این پروژه، شما موفق به پیاده‌سازی یک اسکنر برای زبان برنامه‌نویسی Ciut شدید. در این مرحله، هدف طراحی و پیاده‌سازی تحلیلگر نحوی (Parser) از نوع Predictive Top-Down با استفاده از روش دیاگرام‌های انتقال است. این تحلیلگر به شما کمک می‌کند تا ساختار دستورهای زبان Ciut را به‌صورت سلسله مراتبی و مبتنی بر قواعد گرامری بررسی کنید.

استفاده از کدهای ارائه‌شده در کتاب‌های درسی با ذکر منبع در این پروژه مجاز است. همچنین، برای تسهیل محاسبات مربوط به مجموعه‌های First و Follow متغیرهای غیرپایانی (non-terminal) گرامر Ciut، می‌توانید از ابزار آنلاین موجود در آدرس زیر استفاده کنید:

https://mikedevice.github.io/first-follow

این ابزار علاوه بر محاسبه مجموعه‌های First و Follow، اطلاعات مفیدی مانند مجموعه‌های Predict را نیز ارائه می‌دهد که در ساخت جدول تجزیه به شما کمک خواهد کرد.

استفاده از کدهای اینترنتی یا کدهای سایر دانشجویان در این فاز ممنوع است. همچنین، حتی اگر در فاز قبل اسکنر را پیاده‌سازی نکرده‌اید، مجاز به استفاده از اسکنرهای دیگران نیستید. در چنین شرایطی، باید هم اسکنر و هم پارسر را خودتان برای این فاز توسعه دهید.

مشخصات پارسر (Parser Specification)

پارسری که در این تکلیف پیاده‌سازی می‌کنید باید دارای ویژگی‌های زیر باشد:

۱. الگوریتم تجزیه مبتنی بر دیاگرام‌های انتقال

- تنها روش دیاگرام‌های انتقال (Transition Diagrams) برای پیاده‌سازی پارسر مجاز است.

- استفاده از هر الگوریتم تجزیه دیگری غیرقابل قبول بوده و منجر به نمره صفر می‌شود.

۲. تجزیه پیش‌بینانه (Predictive Parsing)

- پارسر باید به صورت Predictive عمل کند، یعنی هرگز نیاز به بازگشت به عقب (Backtracking) نداشته باشد.

- تصمیم‌گیری درباره تولیدگرامری مورد استفاده باید تنها بر اساس توکن فعلی و مجموعه‌های First و Follow باشد.

۳. اجرای Pipeline

- پارسر باید به صورت موازی (Pipeline)با اسکنر و ماژول‌های بعدی کار کند.

- کل فرآیند کامپایل (اسکن، تجزیه، و تحلیل) باید در یک دور (Single Pass) انجام شود.

٤. مدیریت توکن‌ها با "get_next_token"

- پارسر برای دریافت توکن‌های جدید، تابع "get_next_token" را فراخوانی می‌کند.

- اسکنر باید پس از رسیدن به انتهای فایل ورودی، توکن '$` را به عنوان نشانه پایان ورودی برگرداند.

- در هر لحظه، فقط یک توکن در اختیار پارسر است و با هر فراخوانی "get_next_token"، توکن قبلی با توکن جدید جایگزین می‌شود.

۵. بازیابی از خطا با روش Panic Mode

- پارسر باید بتواند خطاهای نحوی را با استفاده از روش Panic Mode مدیریت کند.

- برای همگام‌سازی پس از خطا، از مجموعه Followهر غیرپایانه (non-terminal) به عنوان مجموعه همگام‌سازی (Sync Setاستفاده شود.

در این فاز، مشابه فاز قبلی، برنامه شما باید یک فایل متنی ورودی با نام `input.txt` را پردازش کند. این فایل حاوی کد نوشته‌شده به زبان Ciut است که باید مراحل اسکن و تجزیه نحوی را طی کند.

خروجی‌های مورد نیاز:

۱. "parse_tree.txt" : این فایل باید درخت تجزیه (Parse Tree) برنامه ورودی را به صورت ساختارمند ذخیره کند.

۲. "syntax_errors.txt" : در صورت وجود هرگونه خطای نحوی در کد ورودی، این فایل باید پیام‌های خطای مربوطه را شامل شود. اگر برنامه ورودی از نظر نحوی صحیح باشد، این فایل باید خالی باشد یا پیامی مبنی بر عدم وجود خطا نمایش دهد.

– در این فاز، نیازی به نمایش خروجی اسکنر (لیست توکن‌ها) نیست و تنها خروجی‌های پارسر مورد ارزیابی قرار می‌گیرند.

دستور زبان Ciut

دستور زبان ارائه‌شده در زیر، یک نسخه اصلاح‌شده و بهینه‌شده از گرامر Ciut است که شرایط لازم برای پارسر پیش‌بین‌کننده (Predictive Parser) را دارد.

– نمادهای پایانه (Terminals) به صورت پررنگ مشخص شده‌اند.

– نمادهای غیرپایانه (Non-Terminals) به صورت معمولی نوشته شده‌اند.

– شما نباید هیچ‌گونه تغییر، حذف یا ساده‌سازی در قواعد این دستور زبان اعمال کنید.

– هرگونه انحراف از گرامر اصلی یا ساده‌سازی دیاگرام‌های انتقال غیرمجاز است.

1. Program -> Declaration-list
2. Declaration-list -> Declaration Declaration-list | EPSILON
3. Declaration -> Declaration-initial Declaration-prime
4. Declaration-initial -> Type-specifier **ID**
5. Declaration-prime -> Fun-declaration-prime | Var-declaration-prime
6. Var-declaration-prime -> **;** | **[NUM] ;**
7. Fun-declaration-prime -> **(** Params **)** Compound-stmt
8. Type-specifier -> **int** | **void**
9. Params -> **int ID** Param-prime Param-list | **void**
10. Param-list -> **,** Param Param-list | EPSILON
11. Param -> Declaration-initial Param-prime
12. Param-prime -> **[ ]** | EPSILON
13. Compound-stmt -> **{** Declaration-list Statement-list **}**
14. Statement-list -> Statement Statement-list | EPSILON
15. Statement -> Expression-stmt | Compound-stmt | Selection-stmt | Iteration-stmt | Return-stmt
16. Expression-stmt -> Expression **;** | **break ;** | **;**
17. Selection-stmt -> **if (** Expression **)** Statement **else** Statement
18. Iteration-stmt -> **repeat** Statement **until (** Expression **)**
19. Return-stmt -> **return** Return-stmt-prime
20. Return-stmt-prime -> **;** | Expression **;**
21. Expression -> Simple-expression-zegond | **ID** B
22. B -> **=** Expression | **[** Expression **]** H | Simple-expression-prime
23. H -> **=** Expression | G D C
24. Simple-expression-zegond -> Additive-expression-zegond C
25. Simple-expression-prime -> Additive-expression-prime C
26. C -> Relop Additive-expression | EPSILON
27. Relop -> **<** | **==**
28. Additive-expression -> Term D
29. Additive-expression-prime -> Term-prime D
30. Additive-expression-zegond -> Term-zegond D
31. D -> Addop Term D | EPSILON
32. Addop -> **+** | **-**
33. Term -> Factor G
34. Term-prime -> Factor-prime G
35. Term-zegond -> Factor-zegond G
36. G -> ***** Factor G | EPSILON
37. Factor -> **(** Expression **)** | **ID** Var-call-prime | **NUM**
38. Var-call-prime -> **(** Args **)** | Var-prime
39. Var-prime -> **[** Expression **]** | EPSILON
40. Factor-prime -> **(** Args **)** | EPSILON
41. Factor-zegond -> **(** Expression **)** | **NUM**
42. Args -> Arg-list | EPSILON
43. Arg-list -> Expression Arg-list-prime
44. Arg-list-prime -> **,** Expression Arg-list-prime | EPSILON

خروجی Parser

همانطور که پیش‌تر ذکر شد، پارسر شما یک فایل متنی به نام 'input.txt' که شامل یک برنامه Ciut است را دریافت می‌کند و درخت تجزیه برنامه ورودی را در یک فایل به نام 'parse_tree.txt' خروجی می‌دهد. همچنین پارسر یک فایل متنی به نام 'syntax_errors.txt' تولید می‌کند که شامل پیام‌های خطا در مورد خطاهای نحوی احتمالی است. اگر در برنامه ورودی خطای نحوی وجود نداشته باشد، باید عبارت 'There is no syntax error.' در فایل 'syntax_errors.txt' نوشته شود. بنابراین، این فایل خروجی باید توسط پارسر ایجاد شود، صرف نظر از اینکه خطای نحوی وجود داشته باشد یا خیر.

درخت تجزیه درون فایل 'parse_tree.txt' باید دارای فرمت زیر باشد:

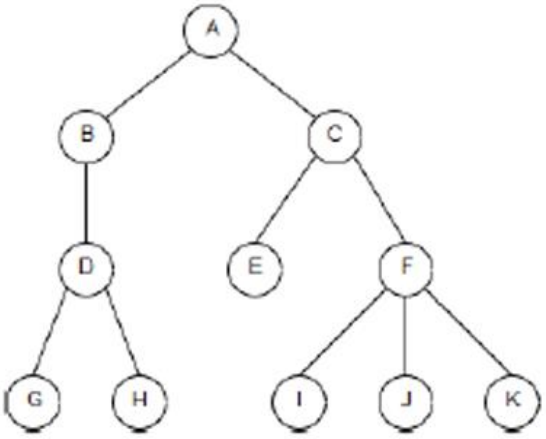
- هر خط شامل یک گره از درخت تجزیه است.

- خط اول شامل گره ریشه است که همان نماد شروع دستور زبان می‌باشد.

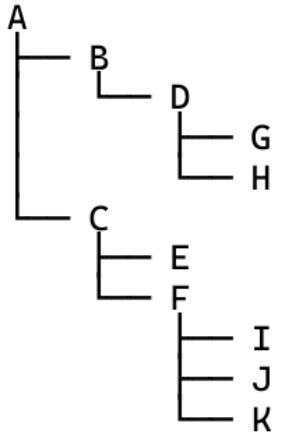
- در هر خط، عمق گره در آن خط با تعداد تب (tab) قبل از نام گره نشان داده می‌شود.

- جانشین‌های هر گره از چپ به راست به ترتیب در خطوط بعدی قرار می‌گیرند.

شکل‌های زیر یک مثال از درخت تجزیه و فرمت مطلوب خروجی را نشان می‌دهند.



Sample parse tree



Sample output

موارد تحویلی

قبل از ارسال، لطفاً اطمینان حاصل کنید که موارد زیر را انجام داده‌اید:

- مسئولیت شماست که اطمینان حاصل کنید نسخه نهایی ارسالی شما هیچ دستور چاپ برای debug ندارد و مشخصات لغوی شما کامل است.
- شما باید فایلی به نام compiler.py ارسال کنید که در این مرحله شامل کدهای اسکنر و پارسر مبتنی بر دیاگرام‌های انتقال پیش‌بینی کننده شما به زبان پایتون باشد.
- لطفاً نام کامل و شماره دانشجویی خود و هر مرجعی که ممکن است استفاده کرده باشید را به عنوان کامنت در ابتدای برنامه خود بنویسید.
- مسئولیت نشان دادن اینکه مباحث پروژه را فهمیده‌اید بر عهده شماست. کد مبهم تأثیر منفی بر نمره شما خواهد داشت، بنابراین وقت اضافی بگذارید تا کد خود را خوانا کنید.
- پارسر شما باید ماژول اصلی کامپایلر باشد، به طوری که با فراخوانی پارسر، فرآیند کامپایل آغاز شود و پارسر در صورت نیاز ماژول‌های دیگر مانند اسکنر را فراخوانی کند.
- پارسر شما با اجرای خط فرمان python compiler.py در سیستم عامل اوبونتو با استفاده از مفسر پایتون نسخه ۳.۱۲ آزمایش خواهد شد. این یک نصب پیش‌فرض از مفسر بدون هیچ کتابخانه اضافی است. لطفاً اطمینان حاصل کنید که پارسر شما به درستی در محیط ذکر شده و با دستور داده شده قبل از ارسال کد شما کامپایل شده است. این مسئولیت شماست که اطمینان حاصل کنید که کد شما به درستی با استفاده از سیستم عامل و مفسر پایتون ذکر شده کار می‌کند.
- کدهای ارسالی با استفاده از چندین مورد آزمایشی مختلف (یعنی چندین فایل input.txt) با خطا و بدون خطاهای لغوی آزمایش و نمره‌دهی می‌شوند. اسکنر شما باید input.txt را از همان پوشه کاری که پارسر شما (یعنی compiler.py) در آن قرار دارد، بخواند. لطفاً توجه داشته باشید که در صورت دریافت خطای کامپایل یا زمان اجرا برای یک مورد آزمایشی، نمره

صفر برای پارسر شما برای آن مورد آزمایشی اختصاص داده می‌شود. به طور مشابه، اگر پارسر نتواند خروجی‌های مورد انتظار (یعنی'parse_tree.txt' و 'syntax_errors.txt') را برای یک مورد آزمایشی تولید کند، نمره صفر برای آن مورد آزمایشی به آن اختصاص داده می‌شود. بنابراین، توصیه می‌شود که اسکنر خود را روی چندین مورد آزمایشی تصادفی مختلف قبل از ارسال کد خود آزمایش کنید.

هرگونه سوال یا ابهامی داشتید می توانید به دو آیدی زیر در تلگرام پیام دهید

@falakian_ali

@M_s_m1984