همان‌گونه که در فاز اول پروژه مشاهده کردید، هدف از قسمت اول طراحی یک تحلیل‌گر لغوی[1] برای کامپایلر بود. حال قصد داریم در فاز دوم به طراحی تحلیل‌گر نحوی بپردازیم. در ابتدا، هدف از انجام این فاز و سپس نحوه پیاده‌سازی و چندین نکته در رابطه با آن را ذکر می‌کنیم.

هدف تحلیل‌گر نحوی

زمانی که یک برنامه توسط کامپایلر خوانده می‌شود، باید علاوه بر این‌که کلمات نوشته شده درست باشند، لازم است تا برنامه از نظر نحوی نیز صحیح باشد. وظیفه کنترل درستی گرامر برنامه، بر عهده‌ی قسمت تحلیل‌گر نحوی است. این کار با تشکیل درخت مربوط انجام می‌شود. ساخت درخت همان‌گونه که در کلاس درس ذکر شده است، توسط جدولی موسوم به جدول shift-reduce می‌باشد که در مراحل مختلف ساخت آن ممکن است به مشکلاتی از قبیل conflict برخورد کند.[2] این مشکلات می‌تواند ناشی از عواملی از قبیل مبهم بودن گرامر باشد که یکی از وظایف شما در این فاز، رفع ابهام‌های گرامر داده شده است.

نحوه پیاده‌سازی تحلیل‌گر نحوی

در این قسمت، از tokenهایی که در بخش قبلی شناسایی کرده بودیم، استفاده خواهیم کرد. به این صورت که، تحلیل‌گر لغوی این tokenها را تشخیص می‌دهد و به تحلیل‌گر نحوی اعلام می‌کند و در این قسمت، برنامه با توجه به گرامر کنترل می‌شود. نحوه نوشتن دستورات، مشابه قسمت قبلی می‌باشد. یعنی، بخش‌ها در این قسمت نیز با %% از هم جدا می‌شوند و ترتیب قسمت‌ها چه در زبان C و چه در زبان جاوا همانند فاز اول است.

در ابتدا می‌بایست تمامی دستورات چاپ را که در فاز اول نوشته بودید، به صورت زیر تغییر دهید:

(در زبان جاوا)

```
{INT_KW} {System.out.println("INT_KW");} → {INT_KW} {return INT_KW;}
```

(در زبان C)

```
"program" {printf("PROGRAM_KW");} → "program" {return PROGRAM_KW;}
```

در واقع، به جای این‌که تحلیل‌گر لغوی، نوع کلمه خوانده شده را چاپ کند، آن را به تحلیل‌گر نحوی اعلام می‌کند.

حال در فاز دوم، در قسمت تعاریف[3] باید کلیه مقادیری را که تحلیل‌گر لغوی به بخش نحوی می‌دهد را تعریف کنید. به عنوان مثال:

```
%token INT_KW  PROGRAM_KW
```

---

[1] Lexical analyzer

[2] البته لازم به ذکر است که در عمل هیچ درختی ساخته نمی‌شود و تنها با استفاده از همین جدول، بخش نحوی کار خود را انجام می‌دهد.

[3] Declaration

پس از تعریف کلیه tokenها، به سراغ قسمت قوانین[1]می‌رویم. در این بخش، مشابه فاز اول عمل می‌کنید؛ با این تفاوت که باید قوانین گرامر را وارد کنید. به طور مثال، اگر جمله اول گرامر به صورت زیر باشد:

$$program \longrightarrow declaration\_list$$

تبدیل به جمله زیر در قسمت قوانین خواهد شد:

```
program : declaration_list {System.out.println("Rule 1: program -> declaration_list);}
```

و به همین ترتیب، تمامی قواعد گرامر را در این قسمت وارد می‌کنیم. بدیهی است که برای کلمات کلیدی[2]و tokenها از کلماتی که از تحلیل‌گر لغوی به بخش نحوی داده می‌شوند، استفاده می‌کنیم.

قسمت user code نیز همانند فاز قبلی می‌باشد.

پس از وارد کردن تمامی قواعد، با استفاده از ابزارهای مربوطه و فایل‌های راهنما[3]کد موردنظر برای تحلیل‌گر لغوی تولید می‌شود. تنها نکته قابل ذکر باقیمانده مربوط به رفع ابهام‌هاست که می‌توان آن‌ها را به دو روش رفع کرد:

۱) تغییر گرامر
۲) استفاده از امکانات یَک[4]

که برخی از موارد با استفاده از مورد دوم قابل حل هستند ولی در بعضی از آن‌ها لازم است تا خود گرامر را تغییر دهید. به عنوان مثال، یکی از conflictهایی که معمولا به آن برمی‌خورند، اولویت‌های مربوط به عملگرهاست که می‌توان با دستورات زیر در پایان قسمت تعاریف، به آن‌ها اولویت داد:

```
%left MINUS_KW.PLUS_KW
%left MULTIPLY_KW.DIVIDE_KW
```

دستور left به منظور left associative بودن عملگرهاست[5]و نکته دیگر این‌که هرچه عملی پایین‌تر قرار گیرد، اولویت بیش‌تری دارد. علاوه بر عنوان left، عناوین right و nonassoc نیز برای right associative و non associative بودن نیز موجود است.

برخی نکات لازم

۱) توصیه می‌شود تمامی **خطاها** و حتی **هشدارها** در این فاز رفع شوند تا در قسمت‌های بعدی به مشکل نخورید.
۲) می‌توانید برای مشاهده اخطارها و موارد ابهام از پسوند verbose -- در ابزار bison استفاده کنید.
۳) اگر تا این مرحله کامنت را در تحلیل‌گر لغوی خود وارد نکرده‌اید، آن را با توجه به نکات موجود در گرامر در تحلیل‌گر لغوی خود درج کنید.[6]

---

[1] Rules
[2] Keyword
[3] برای کامپایل و تبدیل فایل‌های نوشته شده از ابزارهایی که به پیوست خدمتتان تقدیم گردیده، استفاده کنید و با توجه به فایل‌های How to عمل نمایید.
[4] Yacc
[5] در کلاس درس به طور کامل توضیح داده شده است.
[6] کامنت با // شروع می‌شود و تا پایان خط مربوطه ادامه می‌یابد.