به نام خدا

تعریف پروژه درس اصول طراحی کامپایلرها

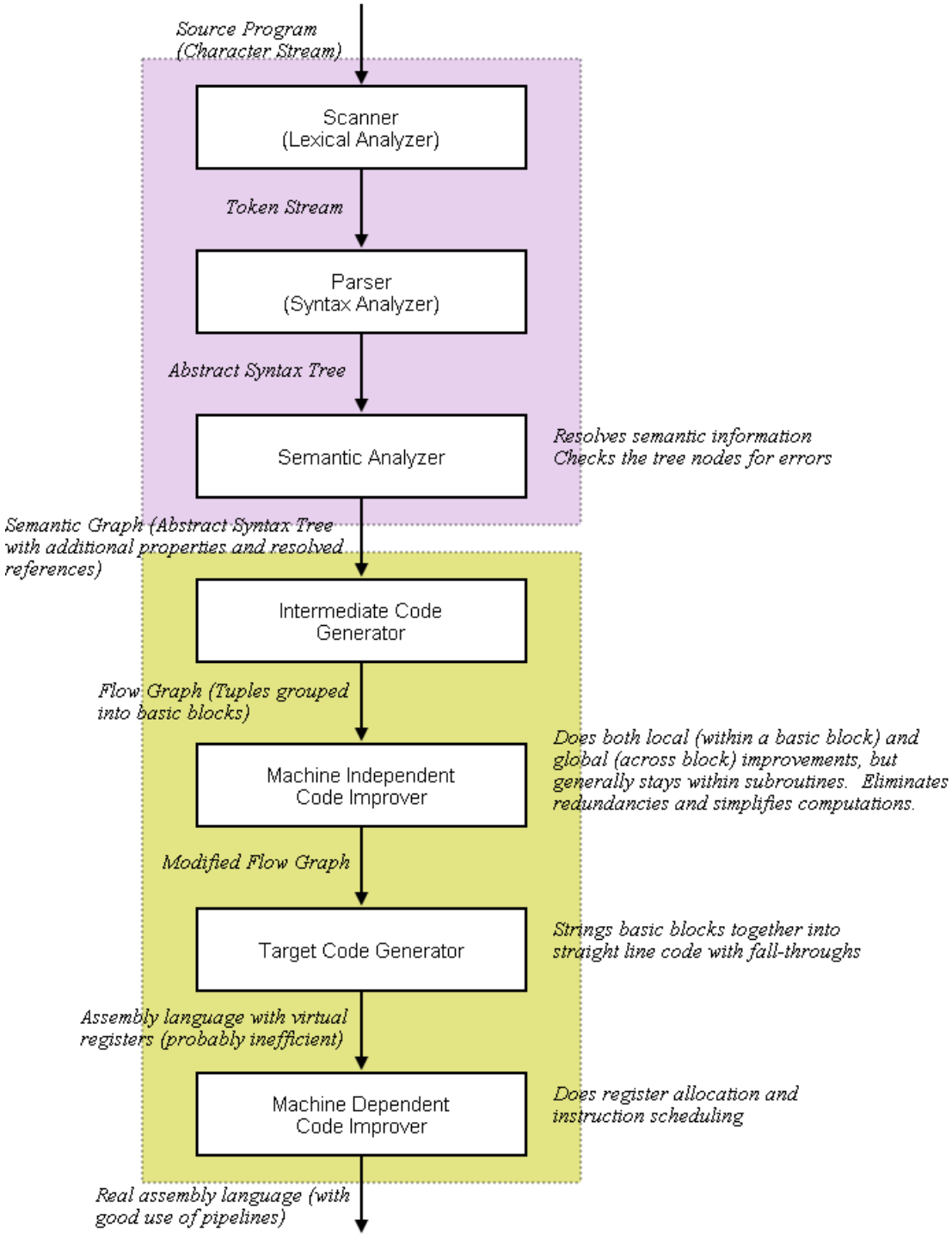
ساخت یک کامپایلر برای یک زبان برنامه‌نویسی یکی از پیچیده‌ترین فعالیت‌های طراحی و ساخت یک زبان برنامه‌نویسی محسوب می‌شد، به نحوی که برای طراحی و پیاده‌سازی زبان Fortran (سال ۱۹۵۷) -که می‌توان آن را اولین زبان عام‌منظوره نیز قلمداد کرد- حدود ۱۲ سال به طول انجامید! اما در این درس قرار است تا دانشجویان با کمک ابزارهایی، با طراحی و پیاده‌سازی یک کامپایلر front-end برای یک زبان جدید آشنا شوند که نهایتاً در طول یک ترم تحصیلی خواهد بود.

در طول این سال‌ها، ابزارهای زیادی برای کمک به فرایند طراحی زبان و تولید کامپایلر برای آن ایجاد شده‌اند. از جمله این ابزارها می‌توان به Lex &Yacc، COCKTAIL، ELI، GENTLE، PCCTS، ACCENT، AFLEX & AYACC، ALE، ANAGRAM، BISON، BTYACC، BYACC، COGENCEE، COCO، FLEX، HAPPY، HOLUB، LLGEN، PROGRAMMAR، QUEX، VISUALPARSE++، ANTLR، BYACC/JAVA، CUP، JACCIE، JAVACC، JFLEX، LOLO، PAT و... اشاره کرد. برای آشنایی با هر یک از این ابزارها می‌توانید به http://dinosaur.compilertools.net/#tools مراجعه کنید.

برای انجام این پروژه، گرامر یک زبان در اختیار شما قرار می‌گیرد که پایانه‌ها، ناپایانه‌ها و معناشناخت قواعد موجود در آن به همراه توضیحات بیشتر و مثالی از یک برنامه به آن زبان در آن وجود دارد. علاوه بر این، در جلسات توجیهی انجام پروژه، می‌توانید ابهامات پیش‌آمده را از تدریسیار بپرسید. همچنین، از بین ابزارهای اشاره شده در بالا، ابزارهای FLEX (برای پیاده‌سازی لکسر کامپایلر به زبان C) یا JFLEX (برای پیاده‌سازی لکسر کامپایلر به زبان Java) و BISON (برای پیاده‌سازی پارسر کامپایلر به زبان C یا Java) معرفی می‌شوند. امکان استفاده از دیگر ابزارها منوط به تأیید تدریسیار و استاد درس است.

همانطور که می‌دانید، کامپایلر از فازهایی تشکیل شده است که در شکل ۱ مشخص شده است. منظور از کامپایلر front-end این است که قسمت‌های Scanner، Parser، Semantic Analyzer و Intermediate Code Generator پیاده‌سازی شوند. به این ترتیب، پس از انجام کامل پروژه، یک source code به زبان تعیین‌شده به عنوان ورودی به کامپایلر داده می‌شود و در انتها کد میانی متناظر با آن برنامه به زبان C (فقط با

استفاده از ویژگی‌های ابتدایی این زبان) به عنوان خروجی خواهد بود. پس از آن، با کامپایل و اجرای کد میانی تولیدشده با استفاده از کامپایلرهای زبان C، می‌توان برنامه اولیه را اجرا کرد.



شکل ۱ - فازهای مختلف یک کامپایلر

برای سادگی کار، این پروژه به سه فاز تقسیم شده است که تحویل هر یک از فازها در موعد مقرر انجام خواهد شد و کل پروژه به صورت حضوری نیز تحویل گرفته خواهد شد. در فاز اول، Scanner به کمک ابزار Flex (یا JFLEX) ساخته می‌شود. در فاز دوم، Parser به کمک ابزار BISON تولید می‌شود و فاز سوم، تحلیل معنایی و تولید کد میانی است که هر یک از فازها در کلاس‌های تدریسیار به طور مجزا و مفصل تشریح می‌شوند. در ادامه نمای کلی از مراحل انجام این سه فاز را خواهید دید.

فرض کنید گرامر داده شده به شکل زیر باشد:

```
Exp → Exp + Exp | Exp – Exp | Exp * Exp | Exp / Exp | ID = Exp | Term

Term → ID | digit

ID → a | b | … | z | aa | …

digit → 0 | 1 | 2 | … | 9 | 10 | …
```

وظیفه Scanner، مشخص‌کردن توکن (Token)های برنامه ورودی و برچسب‌زدن به هر یک از آن‌هاست. به این ترتیب، اولین کار در این فاز این است که پایانه‌های گرامر داده‌شده را مشخص کنید. در مثال، علائم +، -، *، / ، = و اعداد و حروف به عنوان پایانه‌ها یا توکن‌ها در نظر گرفته می‌شوند. بنابراین، اگر برنامه ورودی به شکل زیر باشد:

2+3-5*6/4

انتظار می‌رود که خروجی Scanner به شکل زیر باشد:

2 : digit	+ : PLUS_KEYWORD	3 : digit	- : MINUS_KEYWORD	
5 : digit	* : MULT_KEYWORD	6 : digit	/ : DIV_KEYWORD	4 : digit

لازم به ذکر است که جدول نمادها (Symbol Table) در تمامی مراحل تولید یک کامپایلر مؤثر است. این جدول در ساده‌ترین حالت خود، جدولی مشابه زیر است که مشخص می‌کند که چه نمادی و با چه شناسه‌ای در برنامه وجود دارد. در صورتی که فرض کنیم در گرامر مثال، انتساب به یک متغیر داشته باشیم، برنامه زیر را در نظر بگیرید:

a=2 b=3 a=a*b

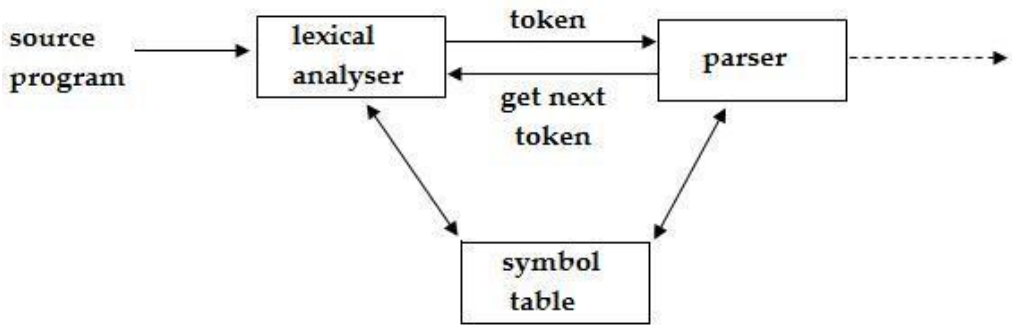
به این ترتیب، جدول نمادها (به بیان ساده) مشابه زیر خواهد بود:

ID	#
a	1
b	2

و خروجی Scanner به شکل زیر خواهد شد:

a : ID #1 = : ASSIGN_KEYWORD 2 : digit
b : ID #2 = : ASSIGN_KEYWORD 3 : digit
a : ID #1 = : ASSIGN_KEYWORD a : ID #1 * : MULT_KEYWORD b : ID #2

همانطور که در شکل ۲ قابل مشاهده است، پس از تحلیل لغوی توسط Scanner، توکن‌ها به Parser داده می‌شوند تا درخت پارس را برای برنامه داده‌شده تشکیل دهد. در این مرحله است که به قواعد داده‌شده در گرامر نیاز می‌شود؛ یعنی حتی اگر نتوان درخت پارس را تشکیل داد، اسکنر کار خود را انجام می‌دهد و وظیفه آن قسمت این است که توکن‌ها را مشخص کند. در همین مثال بند قبل نیز نمی‌توان درخت پارس را تشکیل داد. (چرا؟)



شکل ۲ - ارتباط بین Scanner، Parser و جدول نمادها

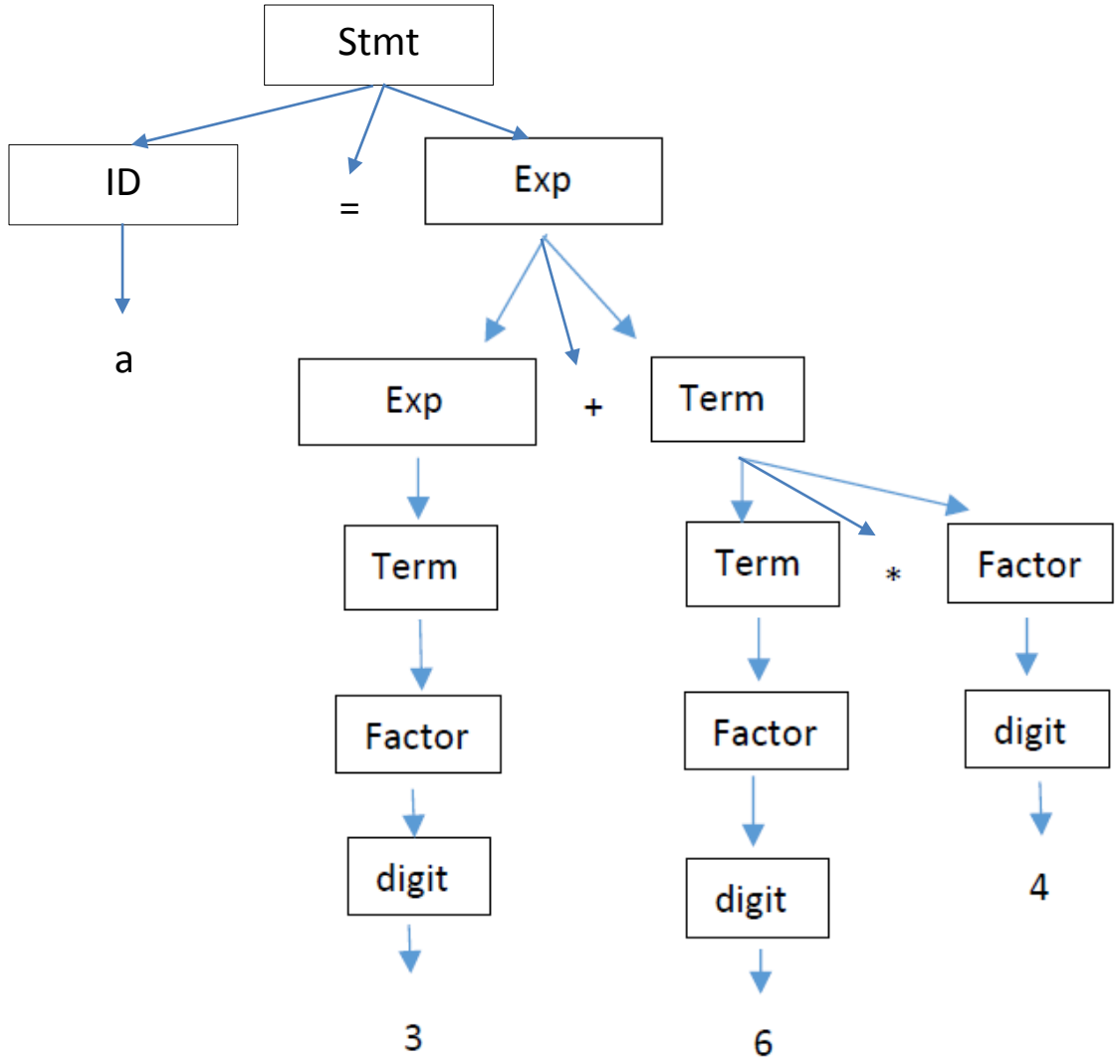
در این مرحله، باید پس از رفع ابهام از گرامر داده‌شده، قواعد را به ابزار BISON معرفی کرد. در مثال جاری، گرامر به شکل زیر تغییر خواهد کرد:

```
Stmt → ID = Exp

Exp → Exp + Term | Exp - Term | Term

Term → Term * Factor | Term / Factor | Factor

Factor → ID | digit
```

ID → a | b | … | z | aa | …

digit → 0 | 1 | 2 | … | 9 | 10 | …

حال Parser تولیدشده برنامه ورودی را با گرامر مطابقت می‌دهد و در صورتی که اشکالی وجود نداشته باشد، درخت پارس تشکیل خواهد شد. برای برنامه ورودی a = 3 + 6 * 4 درخت پارس زیر تشکیل می‌شود:



سپس برای فاز سوم باید با تحلیل معنایی برنامه، کد میانی متناظر با برنامه را تولید کرد. منظور از کد میانی برای برنامه a = 3 + 6 * 4 چنین کدی خواهد بود:

```
int t0 = 6 * 4;
int t1 = 3 + t0;
int a = t1;
```

تا اینجا یک نمای کلی از پروژه درس اصول طراحی کامپایلر را مشاهده کردید. در فایل‌های دیگر جزئیات پیاده‌سازی هر یک از فازها مطرح خواهد شد.