**طراحی و پیاده سازی تحلیلگر نحوی**



**زمستان 1402**

**دانشجو : حمیدرضا بازیار**

**استاد : دکتر فاطمه یوسفی نژاد**

**درس : طراحی کامپایلرها**

**طراحی**

برای ساخت کامپایلر ابتدا باید دیاگرام مناسبی رسم, سپس جدولی متناسب با آن طراحی و در گام آخر شروع به پیاده سازی آن جدول کنیم. در ادامه به توضیح شیوه ساخت آن می پردازیم.

در طراحی دیاگرام, ابتدا قاعده ی افزایشی S’🡪S$ را به گرامر اضافه می کنیم. پس داریم:

S’🡪S$

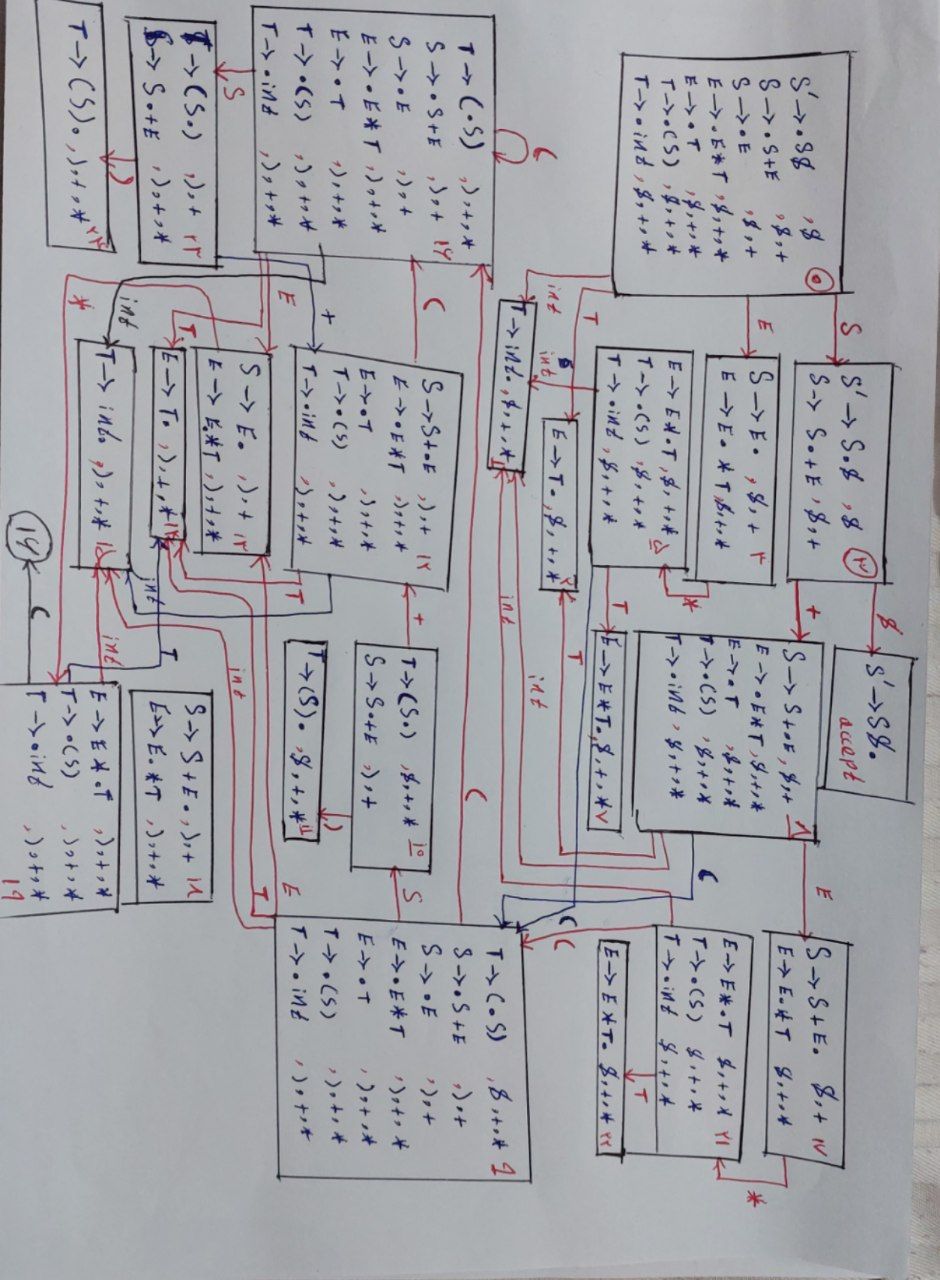
S 🡪S+E | E

E 🡪E\*T | T

T 🡪 (S) | int

اکنون با توجه به گسترش های هر آیتم, دیاگرام حالت مربوطه را رسم می کنیم.

دیاگرام رسم شده در زیر آمده است. این دیاگرام دارای 23 حالت می باشد.

LR(1) DFA of grammar

برای رسم جدول مربوط به دیاگرام, دو بخش برای این جدول در نظر گرفته می شود؛ بخش action و بخش go to . در بخش action یکی از عملیات های کاهش یا شیفت انجام می شود. در بخش go to تعیین می شود که پس از انجام عملیات کاهش, به ازای متغیر های ورودی باید به کدام حالت برویم. در بخش action خانه های خالی به معنای بروز خطا می باشد. در بخش go to خانه های خالی هیچ گاه اتفاق نمی افتد.

برای سادگی تعیین عملیات شیفت از sx و برای تعیین عملیات کاهش از rx استفاده می باشد. در هنگام عملیات شیفت x عدد مربوط به حالت بعدی است که باید پس از انجام عملیات شیفت به آنجا برویم. در هنگام عملیات کاهش به x عدد مربوط به قاعده ای است که به وسیله آن باید کاهش انجام شود. پس برای این منظور قاعده های گرامر را شماره گزاری می کنیم.

1 : S🡪 S+E

2 : S🡪 E

3 : E🡪 E+T

4 : E🡪 T

5 : T🡪 (S)

6: T🡪 int

اکنون به کمک جدول و قواعد شماره گزاری شده, جدول زیر به دست می آید.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Go To** | | |  | **Action** | | | | | **states** |
| **T** | **E** | **S** | **$** | **)** | **Int** | **(** | **\*** | **+** |
| 2 | 4 | 3 | Error | error | S1 | S9 |  |  | 0 |
|  |  |  | R6 |  |  |  | R6 | R6 | 1 |
|  |  |  | R4 |  |  |  | R4 | R4 | 2 |
|  |  |  | Accept |  |  |  |  | S8 | 3 |
|  |  |  | R2 |  |  |  | S5 | R2 | 4 |
| 7 |  |  |  |  | S1 | S9 |  |  | 5 |
|  |  |  | R3 |  |  |  | R3 | R3 | 7 |
| 2 | 17 |  |  |  | S1 | S9 |  |  | 8 |
| 14 | 13 | 10 |  |  | S15 | S16 |  |  | 9 |
|  |  |  |  | S11 |  |  |  | S12 | 10 |
|  |  |  | R5 |  |  |  | R5 | R5 | 11 |
| 14 | 18 |  |  |  | S15 | S16 |  |  | 12 |
|  |  |  |  | R2 |  |  | S19 | R2 | 13 |
|  |  |  |  | R4 |  |  | R4 | R4 | 14 |
|  |  |  |  | R6 |  |  | R6 | R6 | 15 |
| 14 | 13 | 23 |  |  | S15 | S16 |  |  | 16 |
|  |  |  | R1 |  |  |  | S21 | R1 | 17 |
|  |  |  |  | R1 |  |  | S19 | R1 | 18 |
| 14 |  |  |  |  | S15 | S16 |  |  | 19 |
| 22 |  |  |  |  | S1 | S9 |  |  | 21 |
|  |  |  | R3 |  |  |  | R3 | R3 | 22 |
|  |  |  |  | S24 |  |  |  | S12 | 23 |
|  |  |  |  | R5 |  |  | R5 | R5 | 24 |

**پیاده سازی**

برای پیاده سازی از زبان برنامه نویسی پایتون استفاده شده است. همچنین از تکنیک برنامه نویسی شی گرا استفاده شده است. برای این منظور کلاسی تحت عنوان parser در نظر می گیریم. در متد سازنده این کلاس, یک لیست از توکن ها را دریافت می کنیم. همچنین یک دیکشنری تحت عنوان table که پیاده سازی شده ی همان جدول LR(1) است را دریافت می کنیم. برای پیاده سازی این جدول یک دیکشنری در نظر گرفته ایم. هر کلید از این دیکشنری بیانگر یک حالت است. همچنین مقدار هر کلید نیز یک دیکشنری است. در این دیکشنری کلید ها ورودی های مجاز برای آن حالت و مقادیر عملیات است. به طور خلاصه ساختار استفاده شده به ازای هر حالت به صورت زیر است:

{عملیات:ورودی}: حالت

ورودی دیگری که در متد سازنده این کلاس تعیین می شود, قوانین هستند. قوانین نیز به صورت دیکشنری دریافت می شوند. کلید های این دیکشنری شماره قوانین هستند. هر آیتم از این دیکشنری شامل یک لیست که حاوی دو مقدار است, می باشد. مقدار اول متغیر سمت چپ گرامر می باشد, مقدار دوم طول سمت راست آن قاعده است. همچنین در داخل این متد, پشته و head آن را نیز مقدار دهی می کنیم. متغیر هایی برای نگهداری شماره حالت فعلی و ورودی بعدی را نیز در نظر می گیریم.

متد های دیگر این کلاس به شرح زیر می باشند:

**متد pop**

این متد عملیات برداشتن یک مقدار از پشته را برای ما انجام می دهد و سپس مقدار خارج شده را باز می گرداند.

**متد push**

این متد عملیات افزودن یک مقدار را به پشته انجام می دهد. مقدار مورد نظر, در پارامتر ورودی دریافت می شود و به پشته اضافه می شود.

**متد action**

این متد برسی می کند که عملیات کاهش یا شیفت باید انجام شود, پس از آن متد متناسب با عملیات را صدا می زند. این عمل بر اساس بخش action از جدول تجزیه انجام می شود. در این متد در هنگام برخورد با خانه های خالی جدول, syntax error گزارش می شود.

**متد go\_to**

این متد, متناظر با بخش go to در جدول عمل می کند. این متد برسی می کند که به ازای متغیر دریافت شده در پارامتر ورودی خود باید به کدام حالت برود. در انتها این حالت را باز می گرداند.

**متد shift**

این متد متناظر با عملیات شیفت طراحی شده است. طبق قرار داد این متد وظیفه دارد که توکن ورودی و حالت فعلی را در پشته ذخیره کند و توکن بعدی را بخواند.

**متد reducee**

این متد متناظر با عملیات کاهش می باشد. این متد, دو برابر اندازه قاعده ی انتخاب شده, از پشته نماد بر می دارد, سپس با توجه به حالت خارج شده از پشته و متغیر سمت چپ قاعده, تعیین می کند که باید به کدام حالت برویم. در نهایت متغیر سمت چپ قاعده و حالت به دست آمده را به پشته باز می گرداند.

**متد get\_next\_token**

این متد وظیفه دارد که لیست توکن ها را پیمایش کند و نظیر به نظیر توکن را در متغیر next قرار دهد.

**متد error**

از این متد برای نمایش خطا به کاربر استفاده می شود.

**متد run**

این متد, اصلی ترین متد این کلاس می باشد. در این متد, تا زمانی که به انتهای لیست توکن ها برسیم, تابع انتخاب action را صدا می زند. بنابراین برای استفاده از این پارسر کافی است که پس از ایجاد یک نمونه از این کلاس, متد run را صدا بزنیم.

**نتایج**

یک دنباله از توکن ها را در یک لیست قرار می دهیم و آن را به برنامه می دهیم و نتایج را برسی می کنیم.

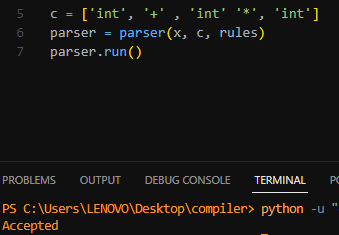
ورودی 1 :

Int + int \* int



از آن رو که یک رشته معتبر می باشد, باید توسط برنامه پذیرش شود.

خروجی 1 :



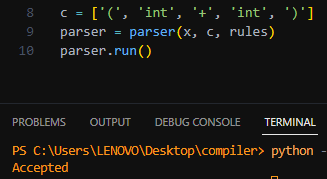
ورودی 2 :

(int + int)



از آن رو که یک رشته معتبر می باشد, باید توسط برنامه پذیرش شود.

خروجی 2 :



ورودی 3 :

Int + \* int



از آن رو که یک رشته معتبر نمی باشد, نباید توسط برنامه پذیرش شود.

خروجی 3 :

