|  |  |  |
| --- | --- | --- |
|  | **باسمه تعالي** |  |
|  |  |  |
|  |  |  |
| **گزارش طراحی کامپایلر با استفاده از چارچوب COOL**  **پروژه پایانی درس اصول طراحی کامپایلرها** | | |

**استاد محترم:**

**سرکار خانم شرقی وند**

**ارائه دهنده**

علی فاضلی

92153110

فهرست مطالب

[1 مقدمه 1](#_Toc504226459)

[2 نحوه کارکرد کامپایلرها 1](#_Toc504226460)

[2-1 جلوبندی کامپایلرها 3](#_Toc504226461)

[2-2 عقب بندی کامپایلر 4](#_Toc504226462)

[2-3 تحلیل در فرآیند کامپایل 5](#_Toc504226463)

[2-3-1 تحلیل لغوی 5](#_Toc504226464)

[2-3-2 تحلیل نحوی 5](#_Toc504226465)

[2-3-3 تحلیل معنایی 6](#_Toc504226466)

[3 تشریح نحوه عملکرد زبان COOL 7](#_Toc504226467)

[3-1 کلاس‏ها 7](#_Toc504226468)

[3-2 انواع متغیرها 9](#_Toc504226469)

[3-3 چک کردن نوع اولیه متغیرها 10](#_Toc504226470)

[3-4 ویژگی ها 10](#_Toc504226471)

[3-5 نوع داده ای VOID 11](#_Toc504226472)

[3-6 توابع 11](#_Toc504226473)

[3-7 عبارات 12](#_Toc504226474)

[3-7-1 ثابت ها 12](#_Toc504226475)

[3-7-2 شناسه‏ها 12](#_Toc504226476)

[3-7-3 انتصاب‏ها 12](#_Toc504226477)

[3-7-4 شروط 13](#_Toc504226478)

[3-7-5 حلقه‏ها 13](#_Toc504226479)

[3-7-6 بلاک‏ها 13](#_Toc504226480)

[3-7-7 ایجاد New 14](#_Toc504226481)

[3-7-8 عبارت Isvoid 14](#_Toc504226482)

[4 تشریح ساختار پروژه 15](#_Toc504226483)

[4-1 ابزار مورد نیاز برای تحلیل لغوی و نحوی 15](#_Toc504226484)

[4-2 تحلیل لغوی 19](#_Toc504226485)

[4-3 تحلیل نحوی 24](#_Toc504226486)

[5 پیاده سازی رابط کاربری 25](#_Toc504226487)

# مقدمه

با توجه به اهمیت شناخت و نحوه کارکرد کامپایلرها در سطوح مختلف، نیاز است که نحوه کارکرد آنان بررسی شده و چگونگی عملکرد آنها در این سطوح تشریح و توجیه گردد. از این رو در این گزارش سعی بر شناخت نحوه عملکرد کامپایلرها در سطح پایین و بصورت حالتی انتزاعی با استفاده از پیاده‏سازی یک کامپایلر ساده را خواهیم داشت. در ابتدا نیاز است که اجزای مختلف گزارش در ابتدا ذکر شود:

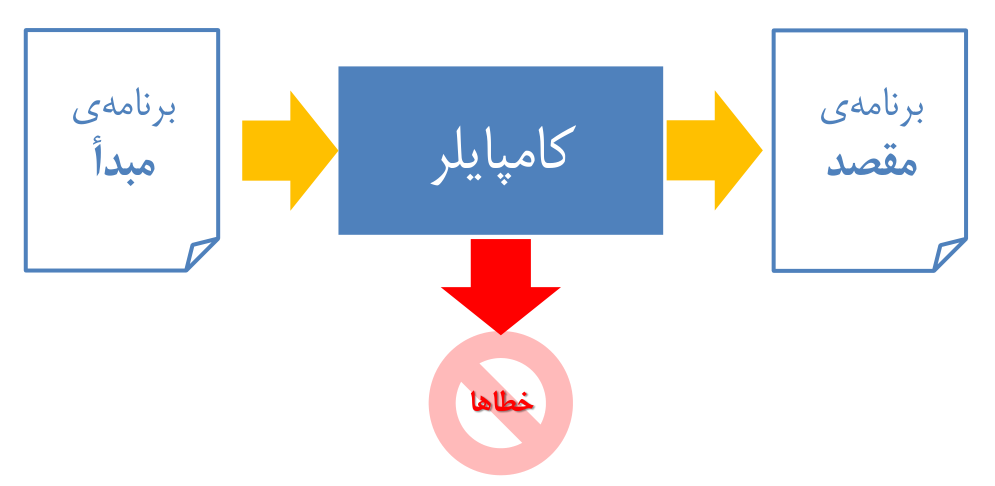
**الف)** در ابتدا شناختی از اصول اولیه کامپایلرها و ساختار آنها خواهیم داشت.

**ب)** سپس عناصر موجود در کامپایلر COOL [[1]](#footnote-1)را بررسی کرده و نحوه ارتباط قسامی مختلف این چارچوب بررسی خواهد شد.

**پ)** ساختار کلی کامپایلر ارائه شده بررسی شده و به تشریح نحوه عملکرد آن با تمام جزئیات می‏پردازیم.

# نحوه کارکرد کامپایلرها

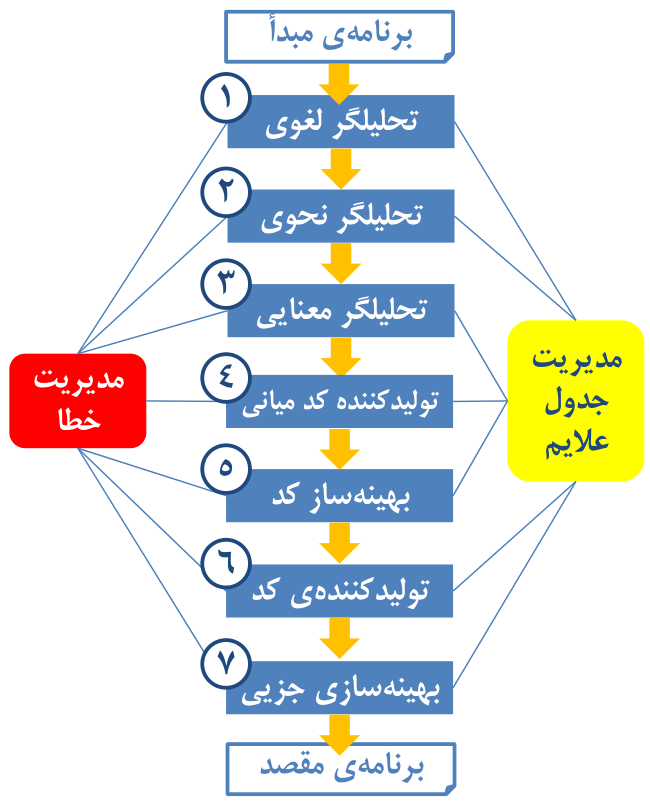
عموما به برنامه‏ای که یک برنامه (مجموعه دستور) قابل اجرا به زبان مبداء را که معمولا به زبان سطح بالا می باشد را به برنامه‏ی قابل اجرا به زبان مقصد (سطح پایین تر) تبدیل می‏کند کامپایلر گفته می‏شود. شکل (1) ساختار انتزاعی یک کامپایلر را به خوبی نشان می‏دهد.



شکل - 1: ساختار کلی کامپایلر

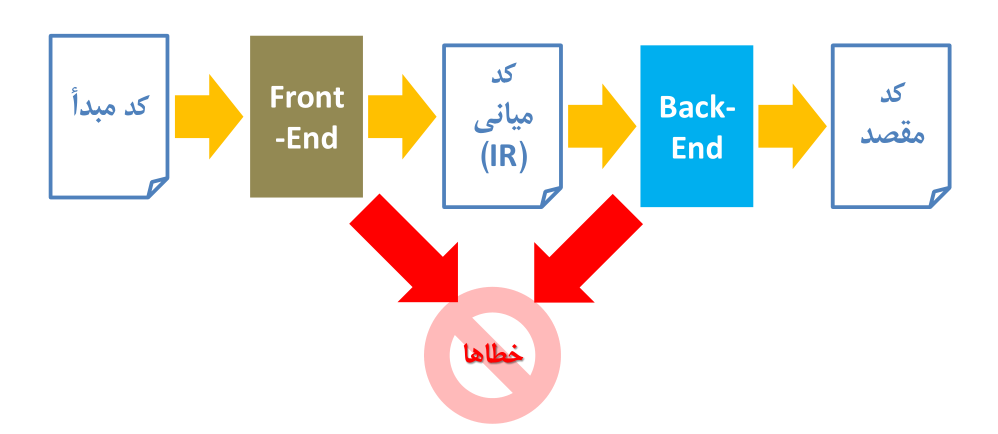
کامپایلرها از لحاظ "ساخت" به دو نوع تک‏گذره و چندگذره تقسیم بندی شده و از لحاظ نوع کد تولید شده نیز به دو نوع "مطلق" (مانند کدهای اجرایی کامپایل شده توسط سیستم عامل‏ها) و "قابل جابه جابه جایی" (مانند کد اجرایی برنامه‏های زبان برنامه نویسی جاوا که با پسوند .class موجودند.) تقسیم بندی می‏شوند.

مراحل کامپایل کردن یک کامپایلر در شکل (2) قابل مشاهده می‏باشد.



شکل - 2: مراحل کامپایل کردن برنامه‏ها

اگر فرایند کامپایل شدن یک برنامه را از مبداء تا مصد به دو قسمت جلوبندی[[2]](#footnote-2) و عقب بندی[[3]](#footnote-3) تقسیم بندی کنیم، می‏توان فرایند کامپایل شدن را به شکل زیر مصورسازی نمود.



شکل - 3: تقسیم بندی فرایند کامپایل به دو قسمت جلوبندی و عقب بندی

در قسمت اول (جلوبندی) نگاشت کد مبداء به کد میانی صورت می‏گیرد. و متعاقبا نگاشت کد میانی به کد مقصد در قسمت عقب بندی کامپایلر انجام خواهد پذیرفت.

## جلوبندی کامپایلرها

در ابتدای کار و تزریق کد مبداء به کامپایلر، اولین عنصری از کامپایلر که وارد عمل خواهد شد، تحلیل‏گر لغوی خواهد بود. وظیفه این بخش تبدیل کد منبع به نشانه[[4]](#footnote-4)های تعریف شده در زبان کامپایل مد نظر می باشد. در این مرحله اموری همچون حذف انواع فاصله‏ها در نوشتار کد منبع صورت خواهد پذیرفت.

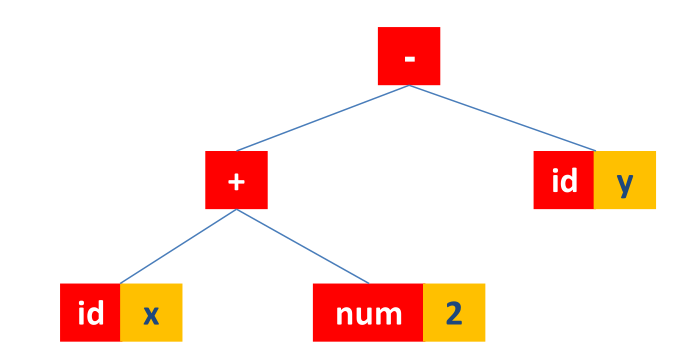
بطور مثال اگر در کد مبداء داشته باشیم: x = x + y

بخش تحلیلگر نحوی کامپایلر عناصر سازنده معتبر را استخراج کرده و به شکل زیر در داده ساختار[[5]](#footnote-5) اختصاصی زبان خود در می‏آورد:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | symbol | id | symbol | id |
| y | + | x | = | x |

در مرحله بعدی تحلیلگر نحوی وارد عمل می‏شود. در این مرحله نیاز به ساخت کد میانی قابل فهم برای کامپایلر به منظور پیشروی در امر تبدیل زبان مبداء به مقصد بدلیل انجام برخی بهینه‏سازی‏ها و رفع خطا های ممکنه خواهیم داشت. در این روند، بایستی امکانی برای تولید پیغام‏های خطای با معنا وجود داشته باشد و در عین حال تلاش برای اصلاح خطاهای احتمالی توسط کامپایلر صورت گیرد. لازم به ذکر است که در این مرحله، ابزارهایی مانند YACC[[6]](#footnote-6) می‏توانند فرایند ساخت کد میانی را تسهیل کنند.

واضح است که برای توصیف زبان‏های برنامه سازی از گرامر‏های مستقل از متن استفاده می‏شود. تحلیل گر نحوی سعی می‏کند یک برنامه را به اجزای نحوی تعریف شده در گرامر خود تقسیم کند. تحلیلگر نحوی و یا Parser را می‏توان در قالب یک درخت پارس[[7]](#footnote-7) و یا درخت نحو[[8]](#footnote-8) نمایش داد. البته درخت نحو می‏تواند به طور خلاصه تر در قالب درخت انتزاعی نحو[[9]](#footnote-9) نمایش داده شود.



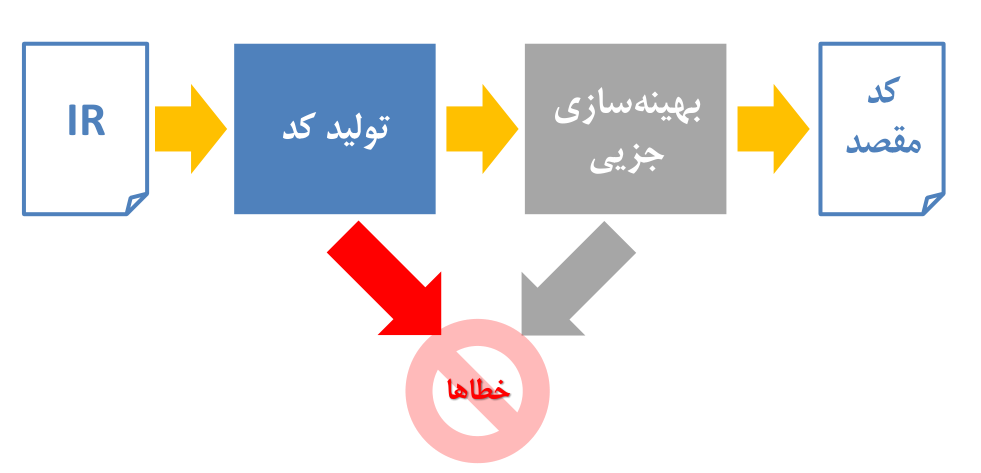
شکل - 4: درخت انتزاعی عبارت x + 2 - y

می‏توان از درخت انتزاعی نحو به عنوان یک واسطه (کد میانی) بین جلوبندی و عقب بندی کامپایلر استفاده نمود.

## عقب بندی کامپایلر

در بخش عقب بندی کامپایلر، ترجمه کد میانی به کد مقصد و تصمیم برای انتخاب دستور مناسب برای هر عمل تشخیص داده شده در کد میانی صورت خواهد پذیرفت. مطلوب در این مرحله، تولید کدی فشرده، خلاصه و مفید و عاری از هرگونه خطا خواهد بود که البته از آدرس‏دهی‏های موجود بطور بهینه استفاده کرده باشد. درشکل شماره (5) ساختار عقب بندی کامپایلر قابل مشاهده می‏باشد.

شایان ذکر است که در تمامی مراحل‏ها، وجود خطاپرداز و نظارت آن به فرایند الزامی می‏باشد.



شکل - 5: ساختار عقب بندی کامپایلر

## تحلیل در فرآیند کامپایل

بطور کلی تحلیل در فرآیند کامپایل برنامه فرضی به سه بخش کلی تقسیم بندی می‏شود که در ادامه به تفضیل به هر سه اشاره خواهیم کرد.

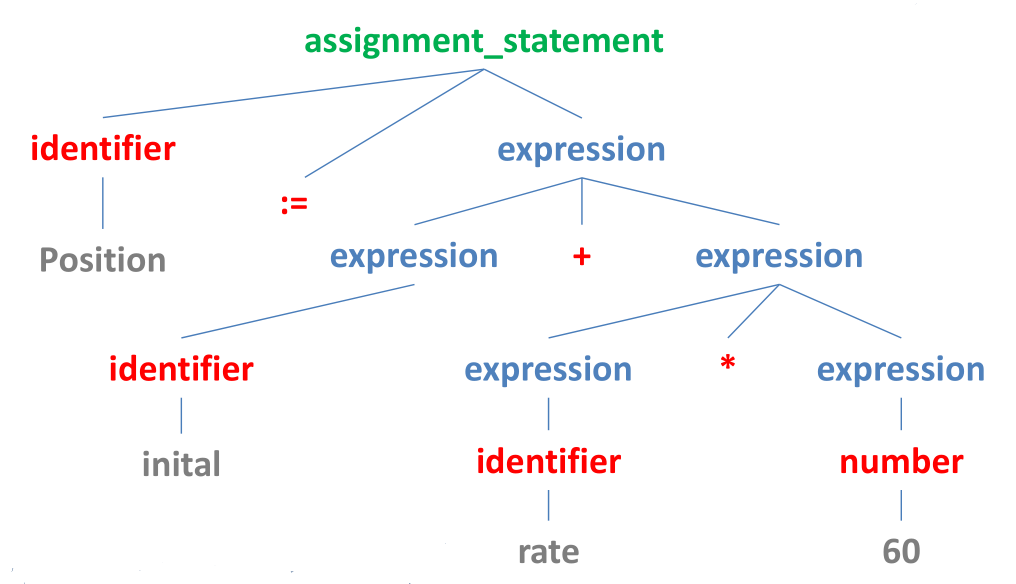
### تحلیل لغوی

در ساده‏ترین بیان، این مرحله از تحلیل به معنای پیمایش چپ به راست مجموعه کد برنامه منبع به منظور شناسایی شناسه‏ها (توالی‏ای از کاراکترها که معنی خاصی برای کامپایلر دارند) می‏باشد. واضحا مواردی همچون فاصله‏ها، سمبل‏های رفتن به ابتدای خط جدید[[10]](#footnote-10)ها و ... در این مرحله نادیده گرفته می‏شوند.

### تحلیل نحوی

تحلیل نحوی و یا پارس کردن به معنی گروه بندی نشانه‏ها در دسته‏های با معنی برای کامپایلر می‏باشد. گره های درخت پارس به کمک گرامر زبان برنامه سازی ساخته می‏شود. بطور مثال درخت پارس برای عبارت زیر بصورتی است که در شکل شماره (6) نمایش داده شده است.

Position := initial + rate \* 60



شکل - 6: درخت پارس عبارت فرضی Position := initial + rate \* 60

### تحلیل معنایی

در آخرین مرحله از فرآیند کامپایل، نوبت به تحلیل معنایی می‏رسد. وظیفه این قسمت یافتن خطاهای معنایی، وارسی نوع داده‏ای های تعریف شده در زبان و بررسی مجاز بودن عملوند‏ها و توالی و ترتیب آنها در این مرحله انجام می‏پذیرد. بدین شکل که در این مرحله جدول علائم[[11]](#footnote-11) ایجاد و نگه داری می‏شود. این جدول حاوی اطلاعاتی در مورد نشانه‏های با معنا (شناسه ها) گرامر مد نظر می‏باشد. مقادیر موجود در جدول در طی مراحل کامپایل بروزرسانی در این مرحله (تحلیل معنایی) به کامل ترین حالت خود خواهد رسید.

# تشریح نحوه عملکرد زبان COOL

زبان برنامه نویسی COOL[[12]](#footnote-12) زبانی است که می‏تواند در طی یک ترم تحصیلی و با کمک مفاهیم اصولی طراحی کامپایلر‏ها پیاده‏سازی گردد. برنامه‏های ساخته شده توسط COOL مجموعه‏ای از کلاس‏های مختلف می‏باشند که با هماهنگی هم کارایی مطلوب را ارائه خواهند داد. این زبان بر پایه عبارات بیان شده است. بدین معنی که یک کلاس معتبر از این زبان، از مجموعه‏ای از عبارات تشکیل یافته است که هرکدام از این عبارات، دارای مقدار و نوع خاصی هستند که از قبل برای آنها تعریف شده است. برای اطلاعات بیشتر می‏توانید به مستندات موجود در وب سایت COOL به نشانی زیر[[13]](#footnote-13) مراجعه کنید.

در این قسمت به تشریح اجزای مختلف زبان برنامه نویسی COOL خواهیم پرداخت:

## کلاس‏ها

تمامی دستورات (کدها) در زبان برنامه نویسی COOL در قالب کلاس[[14]](#footnote-14)ها بررسی و اجرا می‏شوند. تعریف کلاس ها در این زبان به شکل زیر انجام می پذیرد:

در قسمت بدنه کلاس متد[[15]](#footnote-15)ها قرار می گیرند. متدها ویژگی ای هستند که بیان می کنند که بطور مثال متغیر a جزء دسته خاصی از کلاس های از پیش تعریف شده قرار می گیرد. نام تمامی کلاس‌ها برای تمام کلاس‌های دیگر قابل دیدن است. نام کلاس‌ها با یک حرف بزرگ شروع می‌شود و قابلیت بازتعریف کلاس‌ها وجود ندارد.

درون یک کلاس لیستی از ویژگی‌ها قرار دارد. یک ویژگی می‌تواند یک مشخصه یا یک تابع باشد. مشخصه‌ی یک کلاس، متغیری را که بخشی از شی درست شده از کلاس A است را تعریف می‌کند. یک تابع کلاس A روالی را که طی آن کلاس A متغیرها و مشخصه‌های یک شی از آن کلاس را تغییر می‌دهد تعریف می‌کند.

یکی از موضوعات جدید زبان‌های برنامه‌نویسی امروزی، پنهان کردن اطلاعات است، ایده‌ی آن این‏ست که برخی از ویژگی‌های پیاده‌سازی یک نوع داده باید انتزاعی بوده و از نظر نوع داده‌ی کاربر پنهان باشد. Cool از پنهان‌سازی اطلاعات با استفاده از یک مکانیزم ساده پشتیبانی می‌کند: حوزه‌ی تعریف تمامی ویژگی‌ها، مختص به حوزه‌ی محلی هر کلاس است، و حوزه‌ی تعریف همه‌ی کلاس‌ها تمامی کد نوشته شده است. بنابراین تنها راه دادن دسترسی به یک شی در Cool از طریق توابع است.

نام ویژگی‌ها با یک حرف کوچک آغاز می‌شود. نام هیچ کلاسی را نمی‌توان چندین بار بر روی کلاس‌های دیگر گذاشت، اما نام یک کلاس و یک ویژگی می‌تواند یکسان باشد.

بخش کوچکی از list.cl نمونه‌ای از نحوه‏ی تعریف ویژگی‌ها و توابع است:

 در این مثال، کلاس Cons دارای دو ویژگی xcar و xcdr و دو تابع isNil و init است. توجه شود که نوع این ویژگی‌ها، و همینطور نوع پارامترها و انواع بازگشتی توابع، به صورتمستقیم توسط برنامه‌نویس مشخص شده‌اند.با توجه به شی c از کلاس Cons و شی l از کلاس List، می‌توانیم مقادیر xcar و xcdr را با استفاده از تابع init مقداردهی کنیم:

 این نمادگذاری به شیوه‌ی شی‌گرایانه است. ممکن است که تعاریف زیادی از تابع init در کلاس‌های محتلفی وجود داشته باشد. کامپایلر کلاس شی c را نگاه می‌کند تا ببیند باید از کدام تابع استفاده کند. به دلیل اینکه شی c از کلاس Cons است، تابع init از کلاس Cons فراخوانی می‌شود. با این فراخوانی، متغیرهای xcar و xcdr به ویژگی‌های شی c اشاره خواهند کرد. متغیر ویژه‌ی self به شی‌ای اشاره می‌کند که توسط آن تابع فراخوانی شده است، که در این مثال همان شی c است.

فرم ویژه‌ی new شی جدیدی از یک کلاس ایجاد می‌کند.یک شی را می‌توان به عنوان یک رکورد در نظر گرفت که دارای مکان‌های حافظه برای تمامی ویژگی‌های یک کلاس به همراه اشاره‌گرهایی که به توابع آن کلاس اشاره می‌کنند است.

یک تعریف معمولی از برای تابع init به صورت زیر است:

این مثال یک سلول جدید cons درست کرده و “car” را از سلول‌های cons با مقدار یک مقداردهی می‌کند و “cdr” را با مقدار new Nil مقداردهی می‌کند. در Cool مکانیسمی برای از بین بردن اشیا وجود ندارد. Cool دارای مدیریت حافظه‌ی خودکار است، به این معنا که اشیایی را که برنامه نمی‌تواند از آن‌ها استفاده کند به صورت خودکار توسط زباله جمع‌کن کامپایلر از بین می‌روند.

## انواع متغیرها

در زبان برنامه نویسی Cool نام هر کلاس دارای یک نوع نیز هست.

تعریف یک نوع دارای فرم x:C است که در آن x متغیر بوده و C یک نوع است. تمامی متغیرها باید در هنگام تعریف شدن دارای یک نوع باشند( همانند int، float و ...). تمامی‌ها ویژگی‌ها نیز دارای نوع هستند.

قانون پایه‌ی انواع متغیرها در Cool اینست که اگر یک متغیر یا تابع در انتظار یک متغیر از نوع P است، آنگاه هرگونه متغیر از نوع C را نیز قبول می‌کند، در صورتی که در سلسله مراتب انواع، P یک جد از C باشد. به عبار دیگر، اگر C وارث P باشد، چه مستقیم یا غیرمستقیم، آنگاه می‌توان از یک نوع C به جای یک نوع P استفاده کرد. هنگامی که یک شی از کلاس C را بتوان به جای یک شی از کلاس P استفاده کرد، می‌نویسیم C ≤ P C و می‏گوییم که C دنباله‌رو P است.

تعریف دنباله‌روی: فرض کنیم که A، C، و P نوع متغیرها هستند. آنگاه داریم:

* برای تمامی انواع A: A ≤ A
* اگر C وارث P باشد داریم: C ≤ P
* اگر A ≤ C و C ≤ P و A ≤ P

از آنجایی که Object ریشه‌ی تمامی انواع در سلسله‌ مراتب کلاس‌ها است برای A داریم: A ≤ Object

## چک کردن نوع اولیه متغیرها

سیستم تایپ Cool گارانتی می‌کند که هنگام کامپایل کردن برنامه اجرا کردن برنامه‌ی دیگری باعث به وجود آمدن خطاهای Runtime نمی‌شود. با استفاده از تعریف کردن شناسه‌ها توسط برنامه‌نویس، تایپ چکر برای هر عبارت در برنامه یک تایپ اعلام می‌کند.

این مهم است که بین نوعی که توسط تایپ چکر در زمان کامپایل به عبارت داده شده که به آن تایپ ثابت یک عبارت می‌گوییم و تایپ‌هایی که خود عبارات ممکن است هنگام اجرا ارزیابی می‌کنند که به آن‌ها تایپ پویا می‌گوییم فرق قایل شویم. این فرق‌گذاری از آنجا مهم است که در زمان کامپایل تایپ چکر نمی‌تواند اطلاعات کاملی در مورد مقدارهایی که باید پردازش کند داشته باشد. بنابراین، در کل تایپ‌های ثابت و پویا ممکن است که با هم متفاوت باشند. چیزی که ما نیاز داریم اینست که نوع اولیه[[16]](#footnote-16) متغیرهای ثابت با توجه به تایپ‌های پویا ساکن بمانند و تغییر نکنند.

## ویژگی ها

تعریف یک صفت[[17]](#footnote-17) به این شکل است:

عبارت انتهایی (expr)، عبارت دلخواهی است که هنگامی که شی جدید ایجاد می‌شود اجرا می‌شود. تایپ پویای یک عبارت باید با تایپ مشخص شده برای آن صفت مطابقت داشته باشد. اگر صفت هیچ مقداردهی اولیه‌ای نشود آنگاه از مقداردهی پیش‌فرض استفاده می‌شود.

هنگامی که شی جدیدی از یک کلاس ایجاد می‌شود تمامی صفت‌های درون کلاس و به ارث رسیده باید مقداردهی شوند. ابتدا صفت‌های به ارث رسیده به ترتیب به ارث رسیدن از بالای سلسله مراتب وراثت (از بزرگترین جد به پایین) مقداردهی می‌شوند . در یک کلاس داده شده، صفت‌ها به ترتیب جایی که در متن کد قرار دارند مقداردهی می‌شوند. صفت‌ها برای هر کلاس که در ان تعریف شده یا به ارث رسیده‌اند محلی بوده و صفت‌های به ارث رسیده را نمی‌توان دوباره تعریف کرد.

## نوع داده ای VOID

تمامی صفت‌ها در Cool حاوی داده‌ای از نوع تایپی هستند که با آن تعریف شده‌اند. مقدار ویژه خالی (Void) عضوی از تمامی تایپ‌ها بوده و به تایپ پیش‌فرض متغیرها هنگامی که برنامه‌نویس برای آن متغیر تایپی را مشخص نکند، است. (Void به جای Null در C و یا null در جاوا به کار می‌رود، Cool تایپی برابر تایپ Void سی یا جاوا ندارد). در Cool نامی به نام Void نیز وجود ندارد. تنها راه به وجود آوردن یک تایپ Void اینست که یک کلاس که از تایپ Int، String و یا Bool نباشد ایجاد کرد با نتایج یک حلقه‌ی while را ذخیره کرد.

یک عبارت ویژه isvoid وجود دارد که چک می‌کند که آیا عبارت مورد نظر Void است یا نه. علاوه بر آن می‌توان برابری عبارات Void را نیز سنجید. یک مقدار Void را می‌توان به جای یک عبارت به یک متغیر مقدار دارد و یا در هر جایی که استفاده از یک مقدار قانونی است استفاده کرد.

## توابع

تعریف یک تابع (متد) در زبان برنامه نویسی COOL به فرم زیر است:

 که می‌توان هیچ متغیر و یا متغیرهای بیشتری را در آن تعریف کرد. شناسه‌هایی که برای تعریف رسمی متغیرها استفاده می‌شوند باید مشخص باشند. تایپ کلاس نیز باید با تایپی که کلاس بر می‌گرداند نیز مطابقت داشته باشد. هنگامی که یک تایع فراخوانی می‌شود، متغیرهای رسمی‌ای که در ابتدا تعریف شده‌اند، عبارات و متغیرهای درون آن مقداردهی می‌شوند، و مقداری که در نهایت به وجود می‌آید به دلیل فراخوانی تابع بوده است. یک متغیر رسمی هرگونه متغیر همنام را می‌پوشاند.

برای اطمینان از امنیت تایپ‌ها، بر روی بازتعریفی توابعی که وارث هستند محدودیت‌هایی وجود دارد. قانون آن نیز ساده است: اگر کلاس C تابع f را از کلاس P به ارث ببرد، آنگاه C ممکن است تعریف به ارث برده f را تغیر دهد به شرطی که تعداد متغیرها، تایپ متغیرهای رسمی و تایپی که کلاس بر می‌گرداند با تعریف اولیه آن یکسان باشد.

## عبارات

عبارت[[18]](#footnote-18)‌ها بزرگترین گروه از لحاظ گستردگی syntax در زبان برنامه نویسی Cool می باشند. در ادامه به گونه های مختلف عبارات معتبر در این زبان خواهیم پرداخت

### ثابت ها

ساده‌ترین عبارات ثابت‌ها[[19]](#footnote-19) هستند. ثابت‌های بولین true و false هستند. ثابت‌های Integer رشته‌های بدون علامتی همانند 0, 123 و 007 هستند. ثابت‌های رشته‌ای ردیفی از حروف هستند که در گیومه قرار گرفته‌اند همانند "This is a string". ثابت‌های رشته‌ای می‌توانند 1024 کاراکتر طول داشته باشند.

### شناسه‏ها

نام متغیرهای محلی، متغیرهای رسمی، self ها و صفت‌های کلاس‌ها همگی عبارتند. شناسه[[20]](#footnote-20)‌ی self را می‌توان رفرنس داد، اما مقداردهی و یا بایند کردن یک self به یک let، یک case و یا یک متغیر رسمی خطاست. همچنین نام صفت‌ها را self نمی‌توان گذاشت.

متغیرهای محلی و پارامترهای رسمی دارای مقیاس واژگانی هستند. صفت‌ها در درون کلاسی که در آن تعریف شده‌اند یا به ارث رسیده شده‌اند در دسترس هستند، اما می‌توان آن‌ها را به وسیله‌ی دکلریشن‌ها و یا عبارات پدرون کلاس پنهان کرد. بایند کردن یک مرجع[[21]](#footnote-21) یک شناسه دارای مقیاس درونی است که شامل یک دکلریشن برای آن شناسه، و یا یک صفت به همان اسم در صورتی که تعریف[[22]](#footnote-22) دیگری وجود نداشته باشد، است. استثنا این قانون شناسه self است که به صورت مشخص در هر کلاس لحاظ شده است.

### انتصاب‏ها

یک انتصاب[[23]](#footnote-23) دارای فرم زیر است:

تایپ ثابت یک انتصاب باید با تایپ اعلام شده‏ی شناسه مطابقت داشته باشد. مقدار آن نیز همان مقدار عبارت است. تایپ ثابت یک انتصاب نیز همان تایپ ثابت <expr> است.

### شروط

یک شرط[[24]](#footnote-24) دارای فرم زیر است:

 در ابتدا گزاره ارزیابی می‌شود. اگر گزاره true باشد، سپس شاخه‌ی then ازریابی می‌شود. اگر گزاره false باشد، آنگاه شاخه‌ی else ارزیابی می‌شود. مقدار خروجی از شرط، مقدار شاخه‌ی ارزیابی شده است. شایان ذکر است که گزاره باید دارای تایپ ثابت Bool باشد. شاخه‌ها می‌توانند از هرگونه تایپ ثابت باشند.

### حلقه‏ها

یک حلقه[[25]](#footnote-25) دارای فرم زیر است:

 پیش از هر دور حلقه، گزاره ارزیابی می‌شود. اگر گزاره false باشد، از حلقه خارج می‌شویم و مقدار Void برگردانده می‌شود. اگر گزاره true باشد، بدنه‌ی حلقه اجرا می‌شود و دوباره این پروسه تکرار می‌شود.

گزاره باید از تایپ ثابت Bool باشد. بدنه حلقه می‌تواند از هر نوع تایپ ثابت باشد. تایپ ثابت عبارت‌های یک حلقه از نوع شئ می‏باشند.

### بلاک‏ها

یک بلاک[[26]](#footnote-26) دارای فرم زیر است:

{ <expr>; ... <expr>; }

بلاک‏ها به صورت چپ به راست ارزیابی می‌شوند. هر بلاک حداقل دارای یک عبارت است. مقدار یک بلاک نیز مقدار آخرین عبارت ارزیابی شده است. عبارات هر بلاک نیز می‌تواند هرگونه تایپ ثابت را دارا باشند. تایپ ثابت هر بلاک برابر تایپ آخرین عبارت ارزیابی شده است.

### ایجاد New

یک عبارت new دارای فرم زیر است:

 مقدار نوع عبارت مذکور برابر یک شی تازه از کلاسی است که در آن عبارت نوشته شده است.

### عبارت Isvoid

عبارت isvoid expr نیز در صورت void بودن یک عبارت مقدار true و در غیر این صورت مقدار false را برمی‌گرداند.

# تشریح ساختار پروژه

با توجه به ویژگی‏های نحوی زبان برنامه نویسی COOL که پیشتر ذکر شد و همانند مراحل مختلف تحلیل که در ساختار عمومی کامپایلرها برای پیشروی در تبدیل زبان مبداء به مقصد استفاده می‏شود، پیاده‏سازی این پروژه نیز بصورت ترتیبی و به ترتیب مراحل کامپایل کردن برنامه‏ها صورت می‏گیرد. هدف از این پروژه تحقق اهداف زیر می‏باشد:

**الف)** تشریح ابزارهای مورد نیاز برای تضمین کارکرد کارامد مراحل تحلیل لغوی و نحوی

**ب)** تحلیل لغوی مجموعه عبارات وارد شده توسط کاربر انجام شود.

**پ)** تحلیل نحوی مجموعه عبارات داده شده انجام شود.

ت) در نهایت طراحی یک رابط کاربری گرافیکی مناسب و کاربر پسند برای سهولت استفاده از پروژه لازم بنظر می‏رسد.

## پیش نیازها

این پروژه با استفاده از زبان برنامه نویسی پایتون نسخه2.7 توسعه داده شده است. برای نصب و راه اندازی کد اجرایی لازمه برای اجرای این پروژه می توان از پیوند[[27]](#footnote-27) زیر استفاده نمود. پس فراهم آوری پیش نیاز های اجرایی پروژه، کتابخانه های مورد نیاز پروژه را به لیست کتابخانه های اجرایی پایتون سیستم اضافه می کنیم. اولین کتابخانه مورد نیاز PLY نام دارد. این کتابخانه وظیفه پارس کردن گرامر فرضی را برعهده دارد. نحوه کارکرد این ماژول در قسمت قبلی توضیح داده شد. کتابخانه بعدی مورد نیاز کتابخانه Flask می باشد که یک وب سرور ازجنس زبان برنامه نویسی پایتون است. با استفاده از این کتابخانه می‏توان برنامه نوشته شده با زبان پایتون را با رابط کاربری وب ادغام کرده و تجربه استفاده و تعامل با کد را بهبود بخشید. به منظور کاربرپسندتر کردن کار با پروژه، سعی شد که رابط کاربری وب نیز برای پروژه طراحی گردد.

برای فراهم آوری و نصب این کتابخانه ها فایل requirements.txt در مسیر پروژه قرار گرفته است. برای نصب کتابخانه های مذکور در محیط CMD و یا Terminal با قرار داشتن در مسیر پروژه و با استفاده از دستور زیر می توان این کتابخانه ها را نصب نمود:



## ابزار مورد نیاز برای تحلیل لغوی و نحوی

به منظور پیاده سازی دقیق و کامل یک کامپایلر واضحا نیاز به ابزارهایی که فرآیند tokenize کردن و یا تجزیه گرامر را انجام دهند احساس می‏شود. در این پروژه از یک ابزار قدرتمند برای کمک به این فرآیند استفاده خواهیم کرد. این ابزار PLY نام دارد. PLY یک کتابخانه به زبان پایتون می‏باشد که در نسخه 2.7 زبان برنامه نویسی پایتون توسعه داده شده است. این ابزار خود به دو قسمت تقسیم می شود: PLY.lex و PLY.Yacc . این دو قسمت (و در حالت کلی ابزارPLY) برگرفته از دو ابزار Lex و Yacc که در زبان برنامه نویسی C به منظور تبدیل زبان C به زبان اسمبلی ساخته شده بود می‏باشد. به استناد پیوند زیر[[28]](#footnote-28) و پیوند زیر[[29]](#footnote-29) این دو ابزار در سال 1985 بوسیله Jeff Lee طراحی شده است. در ادامه به بررسی دقیق تر این دو ماژول از کلاس PLY می‏پردازیم.

**الف)** قسمت PLY.Lex : وظیفه جداسازی (Tokenization) واژگان موجود در گرامر را برعهده دارد. این ماژول با استفاده از توابع ازپیش تعبیه شده پردازش رشته در پایتون مانند عبارات منظم یا re[[30]](#footnote-30) وظیفه انجام فرایند جداسازی توکن‏های مختلف در گرامر ورودی و تمییز دادن فاصله‏های خالی و علائم نگارشی رزرو شده را برعهده می‏گیرد. علاوه بر این، این ماژول وظایفی همچون ایجاد هماهنگی به منظور گرفتن ورودی (گرامر) از کاربر و ایجاد خروجی مطلوب در CMD و یا Terminal را بر عهده می‏گیرد و از این جهت به منظور افزایش تمرکز برنامه نویس در طراحی قوانین گرامر و جزئیات تکنیکی کامپایلر می‏تواند بسیار مفید واقع شود. بطور مثال، نمونه برنامه‏ای بسیار ساده که با استفاده از این ابزار نوشته شده است را در زیر مشاهده می‏کنید:

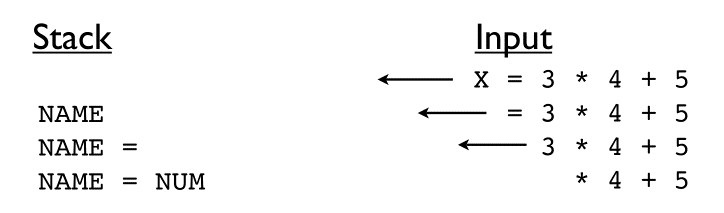


در خط اول، فراخوانی ماژول lex (که مخفف Lexical می باشد) را انجام می‏دهیم.

در خط بعدی لیست اسامی توکن‏های مد نظر که علاقه مند به جداسازی آنها می‏باشیم را ذکر می‏کنیم. در خطوط بعدی با نماد شروع t\_”token\_name” اقدام به تعریف الگوی عبارات منظم ذکر شده در لیست توکن ها (که در بالا ذکر کردیم) خواهیم نمود. بطور مثال عبارت منظم برای بازتعریف نوع توکن نام (t\_NAME) برابر است با r’[a-zA-Z\_][a-zA-Z0-9\_]که آن را به متغیرِ نوع توکنِ نام نسبت می‏دهیم. در ادامه می‏توان با استفاده از یک تابع ساده که آرگومان ورودی دلخواهی (مانند t) را دریافت می‏کند، سناریو خاصی در صورت وقوع (یافتن) توکن مد نظر را طراحی نمود. در ساده‏ترین حالت می‏توان مقدار توکن مربوطه را return کرد. در نهایت بایستی دستور lex.lex() نوشته شود تا فرایند توکنایز آغاز گردد.

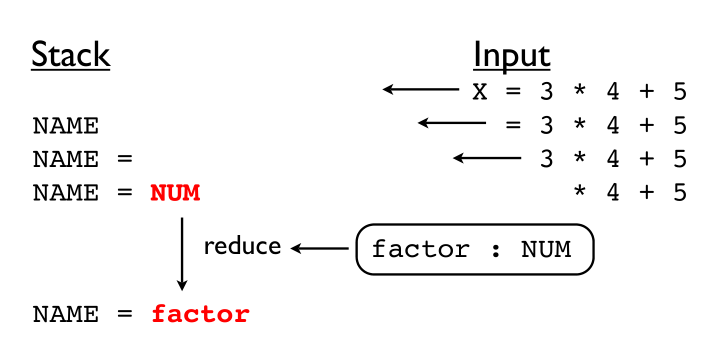
**ب)** قسمت PLY.yacc :

ماژول کاربردی دیگر کتابخانه PLY، ماژول yacc می‏باشد. این ماژول وظیفه تسهیل فرایند طراحی تجزیه کننده گرامر ورودی را بر عهده دارد. Yacc بدین نحو کار می‏کند که ابتداعا گرامر را بوسیله ماژول Lex توکنایز کرده سپس آن را پیمایش کرده و قوانین گرامر را در عناصر آن اعمال می‏کند. Yacc از جدول تجزیه LALR(1) برای تجزیه گرامر استفاده می‏کند. بدین نحو که عناصر گرامر از یک طرف به داخل پشته تجزیه عبارات گرامر Push شده و این عمل تا کامل شدن یک قانون گرامری (به عبارتی پیدا کردن یک دستگیره و اعمال عمل کاهش) ادامه پیدا می‏کند. در صورت یافتن دستگیره مناسب، عمل کاهش از بالای پشته انجام خواهد پذیرفت. این امر در شکل شماره (7) قابل مشاهده می‏باشد.



شکل - 7: نحوه اعمال دستگیره در تجزیه گر LALR(1) موجود در ماژول Yacc

در این حین اگر دستگیره‏ای یافت شود، عمل کاهش اتفاق افتاده و فاکتور کاهش استخراج می‏شود. واضح است که این عمل با استفاده از قوانینی که در قطعه کد‏ای که با استفاده از ماژول Yaccخواهیم نوشت (که اصطلاحا به آنها توابع قوانین[[31]](#footnote-31) می‏گوییم) اعمال خواهند شد. این امر در شکل شماره (8) دیده می‏شود.



شکل - 8: اعمال عمل کاهش در صورت مشاهده مطابقت یک تابع قانون با عبارت موجود در بالای پشته تجزیه ماژول Yacc

نحوه کارکرد این ماژول را با یک مثال توضیح می‏دهیم. فرض کنید گرامر BNF زیر مفروض است:

قطعه کد مناسب برای تهییه یک تجزیه گر برای این گرامر به شکل زیر می‏باشد:



در قسمت اول این ماژول فراخوانی می‏شود. و از آنجائیکه پیش‏تر از تجزیه گرامر نیاز به توکنایز کردن آن داریم ماژول از پیش نوشته خودمان را برای جداسازی توکن ها فراخوانی می‏کنیم. در قسمت بعد توابع قوانین گرامر با شناسه P\_”rule\_function\_name” تعریف می‏شوند. این توابع با گرفتن عبارات موجود بر روی پشته تجزیه به عنوان ورودی P منتظر مطابفت آن با قانون تعریف شده در خود می‏شوند و در صورت مطابقت عمل کاهش توسط ماژول Yacc انجام می‏پذیرد. این کار با عبارتی که بار رنگ سبز در بدنه تابع قرار گرفته است انجام می‏پذیرد. بدین شکل که بطور مثال برای تابع قانون assign که وظیفه انتساب مقدار متغیرها را بر عهده دارد با نوشتن عبارت '''assign : NAME EQUALS expr''' تعیین می‏کنیم که برای تابع قانون assign ، مقدار عبارت expr [[32]](#footnote-32) (که خود می تواند یک پایانی[[33]](#footnote-33) و یا غیرپایانی[[34]](#footnote-34) باشد) همیشه برابر با مقدار غیر پایانی NAME باشد. یا به عنوان مثالی دیگر تابع قانون p\_expr بیان می کند که هر متغیر می تواند عددی مثبت و یا منفی باشد.

و در آخر با فراخوانی دستور yacc.yacc() اعلام می‏کنیم که فرایند تجزیه انجام پذیرد.

## تحلیل لغوی

در این مرحله، ماژول lexical پیاده سازی شده، تشریح خواهد شد. همانطور که در بالا نیز ذکر گشت، برای فرایند توکنایز و جداسازی کلمات رزرو شده و با معنی از ابزار PLY.Lex استفاده خواهد شد. مطابق با الگوی طراحی و کارکرد Yacc (که در قسمت قبلی و با استفاده از مستندسازی[[35]](#footnote-35) موجود برای آن در پیوند زیر[[36]](#footnote-36)، و با توجه به عبارات و نشانه های تعریف شده در زبان برنامه نویسی COOL که در قسمت پیشتر آن بررسی شد) اقدام به پیاده سازی عبارات با معنی موجود در زبان مان خواهیم کرد. بدین منظور مجموعه کلمات رزرو شده موجود در زبان برنامه نویسی COOL را در ماژول ای بنام tokrules.py ذخیره خواهیم کرد. در زیر قسمتی از کلمات رزرو شده را مشاهده می‏کنید:



ساختار موجود برای این قسمت نیز همانند مثال بسیار کوچکی که پیشتر برای نحوه عملکرد Yacc.Lex زده بودیم می‏باشد. و تنها تفاوت آن اعمال کلمات رزرو شده و توکن‏های موجود برای زبان برنامه نویسی COOL می‏باشد.

در ادامه ماژول tokrules.py اقدام به طراحی سناریو‏های مختلف در رابطه با انواع توکن‏های دیده شده خواهیم کرد. برای مثال برای سناریو تشخیص متغیرهای موجود در عبارات، سناریوی مربوطه را به شکل زیر طراحی می‏کنیم:



در ابتدا حروف مجاز برای تعریف یک متغیر با استفاده از عبارت منظم r'[a-zA-Z\_][a-zA-Z\_0-9] تعریف می‏کنیم. بدین معنی که حروف یک متغیر تنها می‏توانند شامل حروف بزرگ و کوچک و اعداد باشد. در ادامه در صورت مشاهده توکن true و یا false نوع داده ای دودویی (BOOL) به آن دسته از توکن‏ها اختصاص داده می‏شود. از آنجاییکه کلمات رزرو شده در COOL همگی با حروف بزرگ نوشته می‏شوند، اگر کلمه رزرو شده ای در عبارات وارد شده با حروف کوچک نوشته شود، در این سناریو آنرا با دستور t.value.upper() به حروف تماما بزرگ تبدیل می‏کنیم و از این جهت به هوشمندی کامپایلر کمک خواهیم کرد. در غیر اینصورت آنرا رد خواهیم نمود و یا آنرا از نوع داده ای درنظر خواهیم گرفت.

پس از تکمیل ماژول tokrules.py اقدام به پیاده سازی ماژول lexical.py می‏کنیم. با توجه به الگوی طراحی یک توکنایزر با استفاده از ابزار PLY.Lex اینکار را انجام خواهیم‏داد. یک ماژول کمکی دیگر بنام commom.py نیز طراحی گردید که وظیفه یافتن موقعیت توکن در سطر خود را بر عهده دارد. به عبارت دیگر متد find\_column() این ماژول وظیفه برگردندن موقعیت یک توکن را در سطری که در آن قرار گرفته بر عهده دارد. وجود این ماژول برای مرحله تحلیل معنایی ضروری می‏باشد.

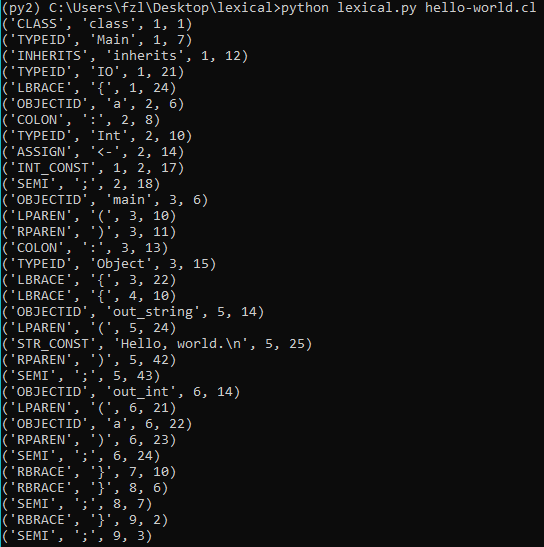
برای اجرای تحلیل لغوی یک فایل به زبان برنامه نویسی COOL و با استفاده از ماژول توکنایزر طراحی شده (lexical.py) کافیست با داشتن کد اجرایی[[37]](#footnote-37) پایتون نسخه 2.7 و با استفاده از دستور زیر اینکار را انجام داد. برای اجرای این مثال از ساده ترین برنامه نوشته شده به زبان برنامه نویسی COOL یعنی hello-world.cl استفاده می‏کنیم.



محتویات hello-world.cl به شکل زیر می باشد:



خروجی مربوط به اجرای ماژول lexical.py در شکل شماره (9) مشاهده می‏شود.



شکل - 9: خروجی ماژول lexical.py به ازای اجرای قطعه کد hello-world به زبان برنامه نویسی COOL

همانطور که در شکل مشاهده می شود خروجی توکنایزر در هر خط، نام نوع داده ای تعریف شده در tokrules.py برای هر کلمه از عبارات برنامه را همراه با سطر و ستون (شماره کاراکتر وقوع) می‏باشد.

## تحلیل نحوی

در این قسمت به تشریح ماژول syntactic.py می‏پردازیم. این ماژول وظیفه تحلیل نحوی (پارس کردن گرامر واردشده) را برعهده خواهد داشت. واضح است که برای پیشروی در امر تحلیل در این مرحله، نیاز به تحلیل‏گر لغوی (بخش lexical پروژه) و همچنین نامگذاری توکن های تعریف شده برای کامپایلر (اسامی تعریف شده در ماژول tokrules.py) خواهیم داشت. بنابراین در ابتدای ماژول syntactic.py این ماژول‏ها را فراخوانی خواهیم کرد.

سپس نوبت به تعریف توابع قوانین زبان برنامه نویسی COOL در این ماژول خواهیم نمود. این قسمت از ماژول syntactic.py برخلاف حجم بالای آن ساختار ساده ای دارد. و ساختار کلی این ماژول بر اساس ساختار عمومی طراحی ماژول Yacc که در قسمت ب بخش 4-1 این گزارش شرح داده شد می‏باشد. بنابراین برای تک تک توکن‏های تعریف شده‏ای که در زبان برنامه نویسی COOL وجود دارد تابع قانون می‏نویسیم. بطور مثال برای تابع قانون \_expression\_str با نوشتن عبارت '''expression : STR\_CONST''' در بدنه آن اعلام می‏کنیم که مقدار عبارات پارس شده در گرامر می توانند از نوع رشته با طول ثابت (STR\_CONST) خواهد بود.

به عنوان مثالی دیگر می توان به تابع قانون p\_expression\_basic\_math(p) اشاره نمود.

این تابع قانون بیانگر عملگرهای پایه ای ریاضی (جمع، تفریق، ضرب و تقسیم) می باشد. در عبارت سبز رنگ قانون بدین شکل پیاده سازی شده است که عبارات می‏توانند بصورت دو عملوند با عملگرهای اولیه تعریف شده از قبل وجود داشه باشند. نحوه تبدیل توکن‏ها به عمل ریاضی واقعی هم بطور مثال برای عمل جمع با شرط وجود "+" در توکن سوم عبارت، عملوند‏های توکن اول و چهارم را با استفاده از متد Plus باهم جمع نمود. بقیه توابع قانون نیز به همین نحو تعریف خواهند شد و روند به همین صورت خواهد بود.

# پیاده سازی رابط کاربری

این پروژه (و متعاقبا رابط کاربری آن) با استفاده از زبان برنامه نویسی پایتون نسخه2.7 انجام گرفته است. برای نصب و راه اندازی کد اجرایی لازمه برای اجرای این پروژه می توان از پیوند[[38]](#footnote-38) زیر استفاده نمود. پس فراهم آوری پیش نیاز های اجرایی پروژه، کتابخانه های مورد نیاز پروژه را به لیست کتابخانه های اجرایی پایتون سیستم اضافه می کنیم. اولین کتابخانه مورد نیاز PLY نام دارد. این کتابخانه وظیفه پارس کردن گرامر فرضی را برعهده دارد. نحوه کارکرد این ماژول در قسمت قبلی توضیح داده شد. کتابخانه بعدی مورد نیاز کتابخانه Flask می باشد که یک وب سرور ازجنس زبان برنامه نویسی پایتون است. با استفاده از این کتابخانه می‏توان برنامه نوشته شده با زبان پایتون را با رابط کاربری وب ادغام کرده و تجربه استفاده و تعامل با کد را بهبود بخشید. به منظور کاربرپسندتر کردن کار با پروژه، سعی شد که رابط کاربری وب نیز برای پروژه طراحی گردد.

برای فراهم آوری و نصب این کتابخانه ها فایل requirements.txt در مسیر پروژه قرار گرفته است. برای نصب کتابخانه های مذکور در محیط CMD و یا Terminal با قرار داشتن در مسیر پروژه و با استفاده از دستور زیر می توان این کتابخانه ها را نصب نمود:



برای اجرای پروژه می توان از رابط کاربری گرافیکی[[39]](#footnote-39) پروژه استفاده کرد. برای اجرای پروژه از طریق رابط کاربری گرافیکی، در مسیر پروژه از دستور زیر استفاده کنید:



سپس با استفاده از یک مرورگر و استفاده از آدرس زیر می توان به صفحه اصلی پروژه دسترسی پیدا کرد:

برای جلوگیری از ازدیاد صفحات گزارش، نحوه اجرا و کارکرد رابط کاربری گرافیکی بصورت ویدیویی انجام شده و از پیوند زیر[[40]](#footnote-40) و زیر[[41]](#footnote-41) قابل دسترسی می باشد.

1. Classroom Object Oriented Language [↑](#footnote-ref-1)
2. Front-End [↑](#footnote-ref-2)
3. Back-End [↑](#footnote-ref-3)
4. Token [↑](#footnote-ref-4)
5. Data Structure [↑](#footnote-ref-5)
6. yet another compiler compiler [↑](#footnote-ref-6)
7. Parse Tree [↑](#footnote-ref-7)
8. Syntax Tree [↑](#footnote-ref-8)
9. Abstract Syntax Tree [↑](#footnote-ref-9)
10. carriage return [↑](#footnote-ref-10)
11. Symbol Table [↑](#footnote-ref-11)
12. Classroom Object Oriented Language [↑](#footnote-ref-12)
13. <http://theory.stanford.edu/~aiken/software/cool/cool.html> [↑](#footnote-ref-13)
14. class [↑](#footnote-ref-14)
15. method [↑](#footnote-ref-15)
16. Primitive Type [↑](#footnote-ref-16)
17. Attribute [↑](#footnote-ref-17)
18. Expressions [↑](#footnote-ref-18)
19. Constants [↑](#footnote-ref-19)
20. Identifiers [↑](#footnote-ref-20)
21. Reference [↑](#footnote-ref-21)
22. Declaration [↑](#footnote-ref-22)
23. Assignment [↑](#footnote-ref-23)
24. Condition [↑](#footnote-ref-24)
25. Loop [↑](#footnote-ref-25)
26. Block [↑](#footnote-ref-26)
27. <https://www.python.org/download/releases/2.7/> [↑](#footnote-ref-27)
28. <https://www.lysator.liu.se/c/ANSI-C-grammar-y.html> [↑](#footnote-ref-28)
29. <https://www.lysator.liu.se/c/ANSI-C-grammar-l.html> [↑](#footnote-ref-29)
30. Regular Expression [↑](#footnote-ref-30)
31. Rule Function [↑](#footnote-ref-31)
32. کوتاه شده expression [↑](#footnote-ref-32)
33. Terminal [↑](#footnote-ref-33)
34. Non-Terminal [↑](#footnote-ref-34)
35. Documentation [↑](#footnote-ref-35)
36. <http://www.eng.utah.edu/~cs3100/lectures/l14/ply-3.4/doc/ply.html> [↑](#footnote-ref-36)
37. Binaries [↑](#footnote-ref-37)
38. <https://www.python.org/download/releases/2.7/> [↑](#footnote-ref-38)
39. Graphical User Interface [↑](#footnote-ref-39)
40. <https://www.youtube.com/watch?v=557q0fIKkIc&t=1s> [↑](#footnote-ref-40)
41. <https://www.aparat.com/v/kKze1> [↑](#footnote-ref-41)