درس: طراحی کامپایلر

عنوان: پروژه تحیل گر لغوی

عضو اول: علیرضا کریمی

شماره دانشجویی عضو اول: ۹۹۳۶۲۳۰۳۵

عضو دوم: علی پورقیصری

شماره دانشجویی عضو دوم: ۹۹۳۶۱۳۰۱۴

استاد: دکتر آرش شفیعی

فهرست مطالب

۱- مقدمه

در فاز اول پروژه، قرار بر این است که یک تحلیلگر لغوی را از صفر با استفاده از زبان پایتون پیاده سازی کنیم. تحلیلگر لغوی در این کد با استفاده از ترکیبی از دیکشنری ها برای تعریف کلمات کلیدی، عملگرها، جداکننده ها، علائم و کاراکترهای فضای خالی پیاده سازی شده است. کلاس تعریف شده در این کد ویژگی های هر توکن را در بر می گیرد و روش های درون آن بر اساس قوانین از پیش تعریف شده، نوع توکن را تعیین می کنند. تحلیلگر، خطوط ورودی را به توابع مختلفی که در ادامه توضیح می‌دهیم، پاس می‌دهد و کاراکتر های خطوط تشخیص داده شده و از هم متمایز می‌شوند سپس به لیست توکن ها اضافه می‌کند. به طور کلی تحلیلگر لغوی هر عنصر را به انواع نشانه‌های خاص دسته‌بندی می‌کند و نمایشی ساختاریافته از کد را برای پردازش یا تحلیل بیشتر ارائه می‌دهد.

۲- مجموعه متغیر های سراسری

این متغیرهای سراسری که در سرتاسر کد برای دسته‌بندی نشانه‌ها در طول تحلیل لغوی استفاده می‌شوند، از داخل فایل constants اظافه شده اند. با استفاده از آنها منطق متمرکز برای تعریف و مدیریت انواع توکن ها، افزایش خوانایی، قابلیت نگهداری و انعطاف پذیری کد را میتوان فارهم نمود. این کاراکتر ها شامل انواع علائم منطقی، نشانه گذاری، بلوک بندی و ... می باشد که معمولا در سایر زبان ها نیز وجود دارند.

۳- کلاس Token

کلاس Token در این کد به عنوان ساختاری برای نمایش توکن های فردی در کد استفاده شده است. این کلاس شامل ویژگی‌هایی برای ذخیره واژگان، نوع آن، نام نشانه یا شناسه، شماره کاراکتری که بر روی آن خطا اتفاق افتاده ، و موقعیت آن در کد (ردیف و ستون) است. در متد __str__ این کلاس، فرمت خروجی مد نظر ما هنگام چاپ کردن مشخص شده است.

۴- کلاس Error

کلاس Error در این کد به عنوان ساختاری برای نمایش خطا هایی که ممکن است در طول کامپایل اتفاق بیوفتد، پیاده سازی شده است. این کلاس شامل ویژگی هایی برای ذخیره واژگان، سطر، ستون، پیغام خطا و شماره کاراکتری که بر روی آن خطا اتفاق افتاده را از اول فایل نشان می‌دهد. در متد __str__ همانند کلاس قبل، خروجی به فرمت مد نظر چاپ می‌شود.

۵- توابع مورد استفاده در طول کد

در طول برنامه یکسری توابع استفاده شده است که پروسه تحلیل لغوی از طریق این توابع انجام می‌گیرد. این توابع به شرح زیر می‌باشند:

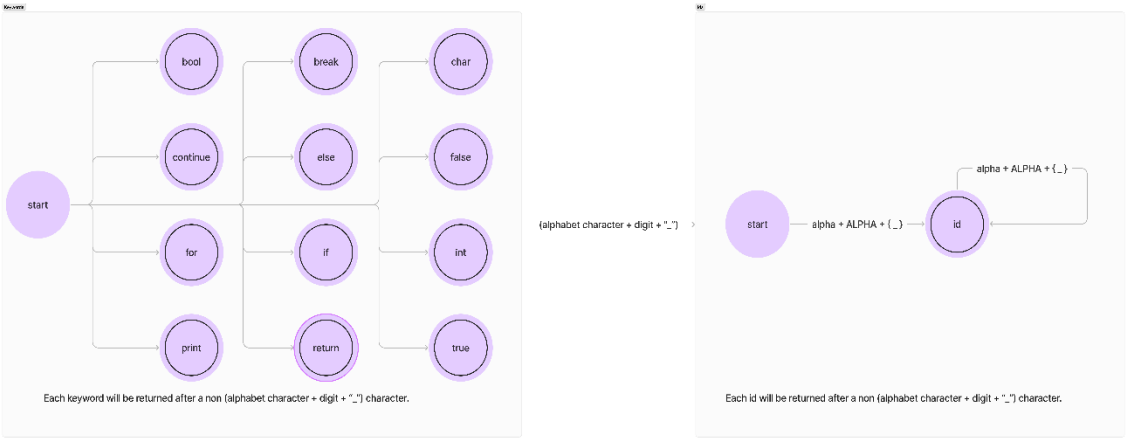
۴-۱- keyword(*line*, *col*, *row*, *file_index*): این تابع که برای هر کلمه موجود در خط صدا زده می‌شود، دنبال کلمات کلیدی می‌گردد به این صورت که ابتدا حرف اول کلمه را می‌بیند، سپس پردازش می‌کند که این حرف، ابتدای چه کلماتی هستند. سپس با پیمایش بر روی حروف بعدی، کلمه کلیدی مورد نظر پیدا می‌شود.

عبارت منظم:

keyword → $[bool|break|char|continue|else|false|for|if|int|print|return|true]$
$bool$ → bool
$break$ → break
$char$ → char
$continue$ → continue
$else$ → else
$false$ → false
for → for
if → if
int → int
$print$ → print
$return$ → return
$true$ → true

۴-۲- identifier($line$, col, row, $file\_index$): این تابع اسامی متغیر ها و توابعی که در طول برنامه نوشته شده است را مشخص و ذخیره می‌کند.

نمودار گذار:

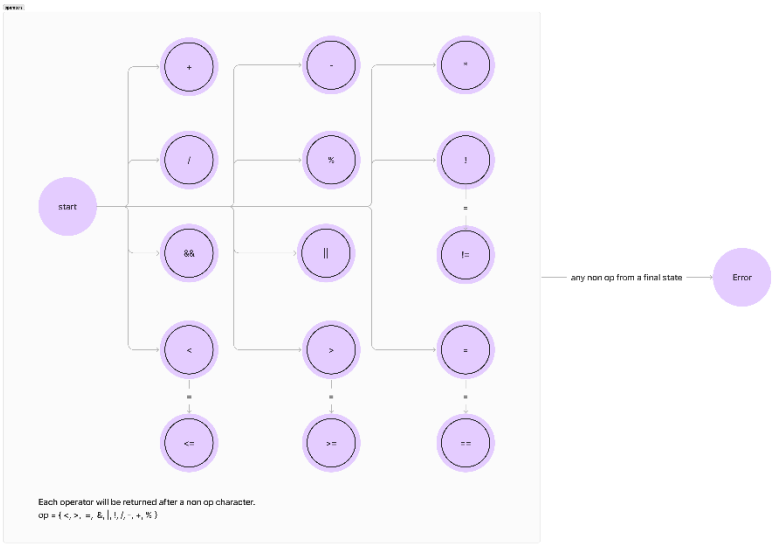


عبارت منظم:

$id \rightarrow [a-z|A-Z|underline]^{+}[0-9|a-z\ |A-z|underline]^{+}$
$underline \rightarrow$ _

۴-۳- operator(*line*, *col*, *row*, *file_index*): این تابع که برای هر عملگر موجود در خط صدا زده می‌شود، دنبال به دنبال عملگر هایی که مورد قبول است می‌گردد به این صورت که عملگر های تک کاراکتری را مشخص می‌کند، و اگر عملگر دو کاراکتری بود، در مرحله بعد تشخیص داده می‌شود.

نمودار گذار:

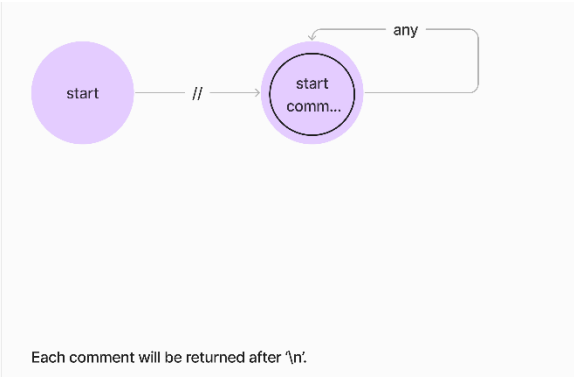


عبارت منظم:

$operator \rightarrow [\ + \mid - \mid * \mid / \mid \% \mid = \mid == \mid >= \mid <= \mid > \mid < \mid ! \mid \&\& \mid \mid\mid \mid != \ ]$

۴-۴- comment(*line*, *col*, *row*, *file_index*): این تابع، زیر رشته هایی که با // شروع یا ادامه داده شوند را به عنوان کامنت در نظر می‌گیرد.

نمودار گذار:

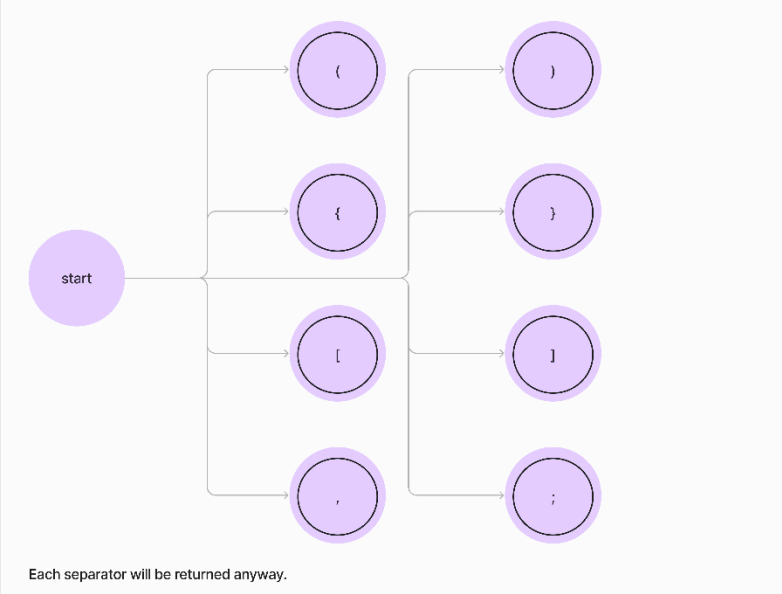


عبارت منظم:

$comment \rightarrow // \ [any - \{\backslash n\}]^*$

۴-۵- separator(*line*, *col*, *row*, *file_index*): این تابع، تمام پرانتز ها، کروشه ها، براکت ها، کاما ها و سمیکالن هارا را تحلیل و ذخیره می‌کند.

نمودار گذار:

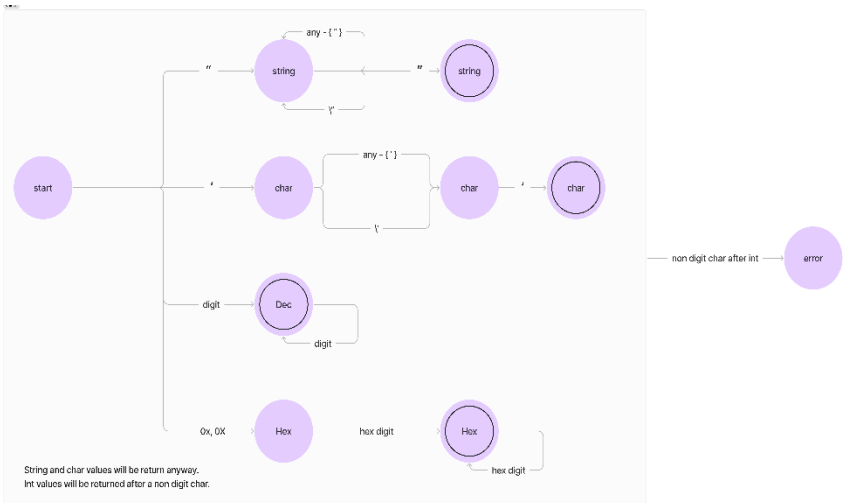


عبارت منظم:

$$separator \rightarrow [\ (\ |\ )\ |\ [\ |\ ]\ |\ \{\ |\ \}\ |\ ,|\ ;\ ]$$

۴-۶- value(*line*, *col*, *row*, *file_index*): این تابع، تمام مقادیر متعلق به متغیر ها، از جمله اعداد، کاراکتر ها و رشته ها را تحلیل و شناسایی می‌کند.

نمودار گذار:



عبارت منظم:

$value \rightarrow [string \mid char \mid dec \mid hex]$
$string \rightarrow$ " $(any - \{"\} + \{\"\})^*$"
$char \rightarrow'$ $(any - \{'\} + \{\'\})$ $'$
$any \rightarrow any\ ascii\ character$
$dec \rightarrow digit^+$
$hex \rightarrow 0xhex\_digit^+$
$digit \rightarrow [0-9]$
$hex\_digit \rightarrow [0-9|a-f|A-F]$

۴-۶- analyze_file(file_path): این تابع یک فایل را خط به خط، نویسه به نویسه می‌خواند، و اطلاعات این نویسه ها را به توابع توضیح داده شده می‌دهد تا تحلیل را بر روی آنها انجام دهند.

۴-۷- main(): این تابع جاییست که analyze_file(file_path) فراخوانی می‌شود.

مراجع

- chat.openai.com
- Introduction of Lexical Analysis - GeeksforGeeks
- https://www.youtube.com/watch?v=O4Bt_CyZWbI
- https://medium.com/@pythonmembers.club/building-a-lexer-in-python-a-tutorial-3b6de161fe84