طراحی و پیاده سازی تحلیلگر لغوی

زمستان 1402

دانشجو : حمیدرضا بازیار

استاد : دکتر فاطمه یوسفی نژاد

درس : طراحی کامپایلرها

طراحی

برای طراحی تحلیلگر لغوی, ابتدا باید انواع کلاس های توکن مورد استفاده در زبان مان را تعیین کنیم. لیست کلاس های توکن عبارتند از :

ID

INT

FLOAT

IF

ELSE

WHILE

STRING

COMMENT

SEMICOLON

PLUS

MINUS

DIVIDE

MULTIPLICATION

EQUAL

NOT_EQUAL

INITIALIZE

LESS_THAN

GREATHER_THAN

LESS_THEN_INITIALIZE

GREATHER_THAN_INITIALIZE

PAEANTES_OPEN

PARANTES_CLOSE

BRACE_OPEN

BRACE_CLOSE

WHITE_SPACE

اکنون باید به ازای هر کلاس, علارت منظم متناسبی بنویسیم. عبارت های منظم هر کلاس توکن و هر ورودی عبارت است از:

Σ = {a, b , …, z, A, … , Z, \, /, * , - , + , 0, 1, …, 9, _ , . , (, { ,) , }, “ , null, ;}

Letter = ‘a’ + ‘b’ + … + ‘z’ + ‘A’ + ‘B’ + … + ‘Z’

Number = ‘0’ + ‘1’ + … + ‘9’

Under_line = ‘_’

Point = ‘ . ‘

Sign = ‘+’ + ‘-‘ + λ

PLUS = ‘ + ‘

MINUS = ‘ – ‘

DIVIDE = ‘ / ‘

MULTIPLE = ‘ * ‘

INITIALIZE = ‘ = ‘

EQUAL = ‘ == ‘

NOT_EQUAL = ‘ != ‘

LESS_THAN = ‘ < ‘

GREATHER_THAN = ‘ > ‘

LESS_THEN_INITIALIZE = ‘ <= ‘

GREATHER_THAN_INITIALIZE = ‘ >= ‘

Double_qutation = ‘ “ ‘

ID = (Under_line + Letter) . (Letter + Number) *

INT = Sign . Number $^{+}$

FlOAT = Sign . Number $^{+}$. Point . Number $^{+}$

KEYWORD = ‘int’ + ‘float’ + ‘string’ + ‘scan’ + ‘print’ + ‘if’ + ‘else’ + ‘while’

STRING = Double_qutation . Σ $^{+}$. Double_qutation

WHITE_SPACE =(‘\n’ + ‘\t’ + null) $^{+}$

SEMICOLON = ‘ ; ‘

PARANTES_OPEN = ‘ (‘

PARANTES_CLOSE = ‘) ‘

BRACE_OPEN = ‘ { ‘

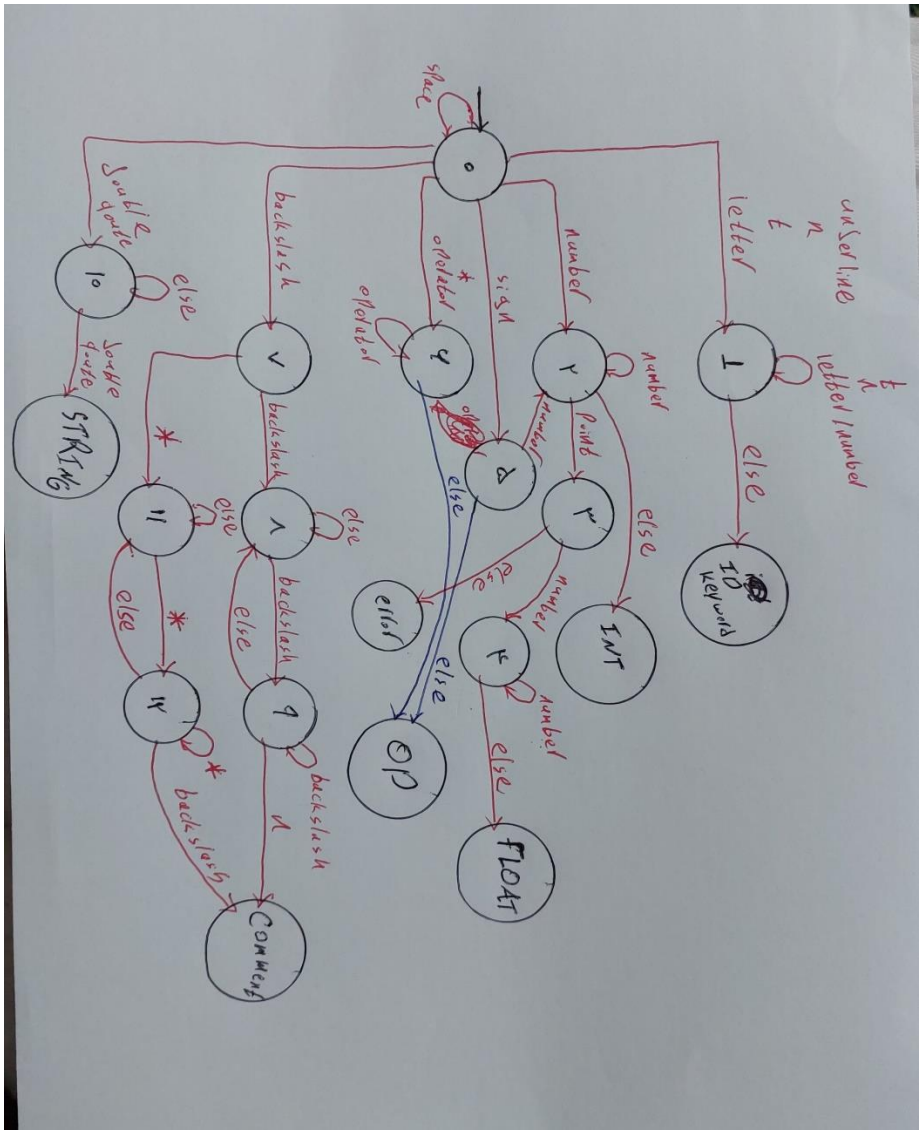
BRACE_CLOSE = ‘ } ‘

SEPERATORS = PARANTES_OPEN + PARANTES_CLOSE + BRACE_OPEN + BRACE_CLOSE

COMMENT = (' \\ ' . Σ^+) + (' * ' . Σ^+ . *\)

SIMPLE = KEYWORD + ID + INT + FLOAT + STRING + SEMICOLON + SEPERATORS + COMMENT

اکنون به ازای عبارت منظم توصیف کننده ی زبان, دیاگرام مناسبی طراحی می کنیم.



Lexical-analyzer DFA

پس از رسم دیاگرام مناسب, جدول مناسبی برای تعیین اینکه از چه حالتی با چه ورودی به کجا باید برویم, طراحی می کنیم.

	0	1	2	3	4	5	6	7	8	9	10	11	12
letter	1	1	INT	ERROR	FLOAT	OP	OP	ERROR	8	8	10	11	11
number	2	1	2	4	4	2	OP	ERROR	8	8	10	11	11
point	ERROR	ID/KW	3	ERROR	FLOAT	OP	OP	ERROR	8	8	10	11	11
operator	6	ID/KW	INT	ERROR	FLOAT	OP	6	ERROR	8	8	10	11	11
sign	5	ID/KW	INT	EROOR	FLOAT	OP	OP	ERROR	8	8	10	11	11
underline	1	ID/KW	INT	ERROR	FLOAT	OP	OP	ERROR	8	8	10	11	11
“	10	ID/KW	INT	ERROR	FLOAT	OP	OP	ERROR	8	8	STRING	11	11
\	7	ID/KW	INT	ERROR	FLOAT	OP	OP	8	9	8	10	11	COMMENT
‘n’	1	1	INT	ERROR	FLOAT	OP	OP	SPACE	8	COMMENT	10	11	11
‘t’	1	1	INT	ERROR	FLOAT	OP	OP	SPACE	8	8	10	11	11
‘*’	6	ID/KW	INT	ERROR	FLOAT	ERROR	6	11	8	8	10	12	12

پیاده سازی

برای پیاده سازی از زبان پایتون و برنامه نویسی شی گرا استفاده شده است. برای این منظور چند کلاس نیاز داریم. کلاس اول, کلاس inputs می باشد که تنها نوع هر ورودی را تعیین می کند. کلاس دوم, token می باشد که پس از پیدا کردن lexim و نوع آن, از این کلاس یک شی ساخته و آن را ذخیره می کنیم. کلاس دیگر که کلاس اصلی ما می باشد, lexical_analyser می باشد. در این کلاس قواعد و اصول مربوط به تحلیلگر لغوی پیاده سازی شده است. در متد سازنده این کلاس, تنها کد برنامه را از ورودی دریافت می کنیم. همچنین متغیر های مورد نیاز همانند next برای نگهداری کاراکتر بعدی, state برای نگهداری حالت فعلی و ... نیز مقدار دهی می شوند. در این کلاس متد هایی وجود دارد که در زیر به آن ها اشاره می کنیم.

متد get_next_char

این متد وظیفه دارد که کاراکتر بعدی را از متن کد بخواند و در متغیر next قرار دهد.

متد input_type

این متد مشخص می کند که کاراکتر ورودی از چه نوعی می باشد. همانطور که در بخش طراحی تعیین کردیم, ورودی ها می توانند از انواع letter,number,... باشند.

متد reset

پس از اجرای عملیات های خاص مانند ذخیره یک توکن, برخی متغیر ها مانند state به حالت اولیه باز گردند. متد reset این کار را انجام می دهد.

متد save_token

پس از پیدا شدن lexim و نوع آن, باید آن را ذخیره کنیم. در این متد, متن lexim را از کد جدا می کنیم, یک نمونه مناسب از کلاس توکن می سازیم و سپس آن را ذخیره می کنیم.

متد error

به هر دلیلی ممکن است در روند برنامه خطایی رخ دهد. هر جا که خطا رخ دهد و نیاز به گزارش خطا باشد, این تابع متن مناسب را در آرگومان خود دریافت می کند و به کاربر گزارش می دهد.

متد get_type

این متد, پیاده سازی پیمایش روی دیاگرام مربوطه می باشد. این متد روی متن کد تا پیدا کردن lexim مناسب پیمایش می کند و سپس نوع آن را به دست آورده و تابع ذخیره را صدا می کند.

متد run

این متد وظیفه ی اجرای مراحل تحلیلگر لغوی را روی کد تا رسیدن به انتهای آن یا یک مشکل در کد را دارا می باشد. این متد اصلی ترین متد کلاس ما می باشد. برای استفاده از این تحلیلگر باید شی مناسبی از کلاس ایجاد و این تابع را برای آن صدا بزنیم.

نتایج

یک رشته که کد برنامه مورد نظر می باشد را به برنامه می دهیم و نتایج را برسی می کنیم.

ورودی 1 :

A = 12; if(a) scan(int);

```
myfile = 'A = 12; if(a) scan(int);'
```

از آن رو که یک کد معتبر می باشد, باید یک لیست از توکن ها بدون خطا باز گردد.

خروجی 1 :

```
('ID', 'A')
('OP', ' =')
('INT', ' 12')
('SEMICOLON', ';')
('ID', ' if')
('PARANTES OPEN', '(')
('ID', 'a')
('PARANTES CLOSE', ')')
('ID', ' scan')
('PARANTES OPEN', '(')
('INT', 'int')
('PARANTES CLOSE', ')')
('SEMICOLON', ';')
```

ورودی 2 :

A = 12; b = .5;

```
myfile = 'A = 12; b = .5'
```

از آن رو که یک کد نامعتبر می باشد, باید خطا باز گرداند.

خروجی 2 :

```
('ID', 'A')
('OP', ' =')
('INT', ' 12')
('SEMICOLON', ';')
('ID', ' b')
('OP', ' =')
Error: Invalid Input!
```