

به نام خدا

پروژه پایانی درس کامپایلر  
**LEXYN**

زهرا صادقی عدل  
هومن مهرآفرین

تابستان ۹۷

## LEXYN:

برای انجام پروژه هر یک از قسمت‌های برنامه را در یک فایل جدا نوشته و به کمک توابع handler در فایل main آنها را فراخوانی و مدیریت میکنیم.

```
*** LEXYN ***  
Hello, Please select what you want:  
1. Lexical Analysis  
2. Syntax Analysis
```

پس از انتخاب هر یک از موارد بالا وارد توابع handler شان میشویم که در ادامه به توضیح آن میپردازیم.

## تحلیلگر لغوی:

-اجرا:

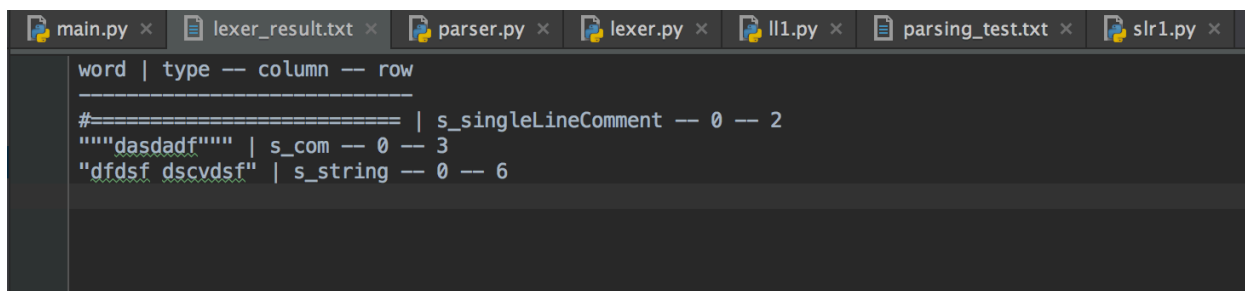
با انتخاب عدد ۱ وارد قسمت تحلیلگر لغوی میشویم. حال برنامه از شما نام فایل مورد نظر را میخواهد:

```
*** LEXYN ***
Hello, Please select what you want:|
1. Lexical Analysis
2. Syntax Analysis
1
Please Enter File name :
```

با وارد کردن نام فایل، خروجی تحلیلگر لغوی در فایلی به نام lexer\_result.txt ذخیره میشود.

```
*** LEXYN ***
Hello, Please select what you want:
1. Lexical Analysis
2. Syntax Analysis
1
Please Enter File name :lexer_test.txt
done!
check lexer_result.txt!

Process finished with exit code 0
```



```
main.py x  lexer_result.txt x  parser.py x  lexer.py x  ll1.py x  parsing_test.txt x  slr1.py x
word | type -- column -- row
=====
#===== | s_singleLineComment -- 0 -- 2
""dasdadf"" | s_com -- 0 -- 3
"dfdsf dscvdsf" | s_string -- 0 -- 6
```

- ساختار کد:

این بخش مربوط به lexer است که کد python را parse می کند، اما فقط به lexer کد python محدود نمی شود و می توان با تغییر دادن rules آن را به lexer زبان دیگری تبدیل کنیم.

نحوه‌ی کارکرد آن بدین صورت است که ابتدا کل ورودی را از فایل گرفته و سپس با استفاده از regex، \f و \r را حذف می‌کنیم. این ورودی گرفته شده سپس در متغیری به نام باف ذخیره می‌کنیم و شماره ستون را 0 و شماره سطر را 1 قرار می‌دهیم. سپس با استفاده از شماره ستون هر کدام از token ها رو match می‌کنیم و شماره ستون را به آخر token تغییر می‌دهیم و هر بار که \n مشاهده شد line را به اندازه ۱ واحد اضافه می‌کنیم و چون شماره ستون باید از صفر شروع شود شماره ستون آخرین \n را نگه می‌داریم و از شماره ستون آخرین token بعدی کم می‌کنیم و در انتها همه token ها را چاپ می‌کنیم.

```
if __name__ == '__main__':  
    rules = [  
        ('if|else|elif|while|for|import|class|def|in|range|print|len', 'key_word'),  
        ('[+-]?(\d+(\.\d*)?)|\.\d+', 's_num'),  
        ('[a-zA-Z_]\w*', 's_id'),  
        ('\\n', 's_newLine'),  
        ('\: ', 's_colon'),  
        ('\; ', 's_semicolon'),  
        ('\+ ', 's_plus'),  
        ('\| ', 's_or'),  
        ('<= ', 's_le'),  
        ('>= ', 's_ge'),  
        ('< ', 's_lt'),  
        ('> ', 's_gt'),  
        ('<>', 's_ne'),  
        ('(', 's_openpar'),  
        (')', 's_closepar'),  
        ('[', 's_openbr'),  
        (']', 's_closebr'),  
        ('\\*\\*', 's_pow'),  
        ('-', 's_minus'),  
        ('\\*', 's_mul'),  
        ('\\/ ', 's_div'),  
        ('\\( ', 's_openpar'),  
        ('\\) ', 's_closepar'),  
        ('==', 's_eq'),  
        ('!=', 's_noteq'),  
        ('=', 's_assign'),  
        ('#([^\n])*', 's_singleLineComment'),  
        ('\"([^\"])*\"|\'([^\']*)*\'', 's_string'),  
        ('\\\\\\\\\\\\\\\\'([\\\\\\\\]|\\\\\\\\'|\\\\\\\\'([\\\\\\\\]|\\\\\\\\.)|\\\\\\\\\"([\\\\\\\\]|\\\\\\\\\")*)*\\\\\\\\\\\\\\\\'|  
        '\\\\\\\\\\\\\\\\'([\\\\\\\\]|\\\\\\\\\\\\\\\\'|\\\\\\\\'([\\\\\\\\]|\\\\\\\\\\\\\\\\')|\\\\\\\\\\\\\\\\'([\\\\\\\\]|\\\\\\\\\\\\\\\\'))*\\\\\\\\\\\\\\\\\\\\\\'',  
        's_com')
```

## تحلیلگر نحوی:

با انتخاب عدد ۲ در منوی اصلی وارد منوی تحلیلگرهای نحوی میشیم:

```
*** LEXYN ***
Hello, Please select what you want:
1. Lexical Analysis
2. Syntax Analysis
2
Type of syntsx analysis
1. LL(1)
2. LR(1)
3. LALR(1)
4. SLR(1)
```

حال با انتخاب هر یک تابع handler مربوطه فراخوانی میشود.  
قابل توجه است که فرمت ورودی برای هر ۴ تحلیلگر یکسان و به صورت زیر میباشد:  
پایانه ها  
یک خط خالی  
قواعد همراه با قاعده  $S' \rightarrow S\$$

توجه داشته باشید که این قواعد باید با فاصله نوشته شوند مثل:

a d b

```
$ S' -> S
S -> a A B
S -> S D b
A -> a d B
A -> A b
B -> B D a
B -> a b D
D -> D a
D ->
```

توجه داشته باشید بجای قاعده لاندا مانند خط اخر مثال بالا عمل میکنیم.  
لاندا در همه تحلیلگرها قابل اجراست به جز (1)ll.

## LL(1):

با انتخاب عدد ۱ وارد تحلیلگر لغوی میشویم، تابع ll1\_handler به عنوان ورودی نام فایل ورودی و یک رشته برای تشخیص درستی دریافت میکند:

```
*** LEXYN ***
Hello, Please select what you want:
1. Lexical Analysis
2. Syntax Analysis
2
Type of syntsx analysis
1. LL(1)
2. LR(1)
3. LALR(1)
4. SLR(1)
1
Please Enter File name :parsing_test.txt
String : aaa
```

پس از اجرا نتایج به صورت زیر نمایان میشود:

```
***** LL(1) *****

--- parsing table ---
STACK   STRING   ACTION
$AA | aaa$ | S -> AA
$AAa | aaa$ | A -> aA
$AA | aa$ | 
$AAa | aa$ | A -> aA
$AA | a$ | 
$AAa | a$ | A -> aA
$AA | $ | 
aaa$ : REJECT

--- LL(1) table ---
  a | b | $
-----
S | A A | A A |
A | a A | b |
```

ساختار:

برای پیاده سازی این قسمت از این [لینک](#) استفاده کردیم، منتهی مشکلاتی از جمله فرمت ورودی و خروجی و فرمت جدول که به صورت دیکشنری بود جود داشت.

برای حل مشکل ورودی به جای ورودی دادن به تابع به صورت قاعده به قاعده از ورودی به شکل فایل استفاده کردیم همچنین برای خروجی جدول ها با تابع `print_table` ان را به صورت ماتریسی در آوردیم و جدول تجزیه که وجود نداشت را نیز به کمک خود استک و اکشن های موجود نوشتیم.

تابع `makeParser` با گرفتن فایل ورودی جدول نحوی را میسازد تابع `match` با گرفتن یک رشته جدول تجزیه را ساخته و اعلام میکند برای این زبان هست یا خیر.

و در نهایت با تابع `print_table` جدول `ll(1)` را نمایش میدهد.

## LR(1):

با انتخاب عدد ۲ در لیست تحلیلگرها وارد بخش `LR(1)` میشویم. در این قسمت هم ورودی یه فایل و یک رشته است:

```
*** LEXYN ***
Hello, Please select what you want:
1. Lexical Analysis
2. Syntax Analysis
2
Type of syntsx analysis
1. LL(1)
2. LR(1)
3. LALR(1)
4. SLR(1)
2
Please Enter File name :parsing_test.txt
String : aaa
```

سپس خروجی به شکل زیر نمایش داده میشود:

```

FIRST AND FOLLOW OF NON-TERMINALS

S
  First:  {'a', 'b'}
  Follow: {'$'}

A
  First:  {'a', 'b'}
  Follow: {'b', 'a', '$'}

['S', 'A']
['a', 'b', '$']
I0:
  Z->.S, $
  S->.AA, $
  A->.aA, a|b
  A->.b, a|b
I1:
  Z->S., $
I2:
  S->A.A, $
  A->.aA, $
  A->.b, $
I3:
  A->a.A, a|b
  A->.aA, a|b
  A->.b, a|b
I4:
  A->b., a|b
I5:
  S->AA., $
I6:
  A->a.A, $
  A->.aA, $
  A->.b, $
I7:
  A->b., $
I8:
  A->aA., a|b
I9:
  A->aA., $

```

#### ---- LR(1) TABLE----

```

0  OrderedDict([('S', '1'), ('A', '2'), ('a', {'s3'}), ('b', {'s4'})])
1  OrderedDict([('$', 'accept')])
2  OrderedDict([('A', '5'), ('a', {'s6'}), ('b', {'s7'})])
3  OrderedDict([('A', '8'), ('a', {'s3'}), ('b', {'s4'})])
4  OrderedDict([('a', {'r3'}), ('b', {'r3'})])
5  OrderedDict([('$', {'r1'})])
6  OrderedDict([('A', '9'), ('a', {'s6'}), ('b', {'s7'})])
7  OrderedDict([('$', {'r3'})])
8  OrderedDict([('a', {'r2'}), ('b', {'r2'})])
9  OrderedDict([('$', {'r2'})])

```

0 s/r conflicts | 0 r/r conflicts

Enter String:: **aaaa**

Test String: aaaa\$

STACK	STRING	ACTION
0	aaaa\$	Shift S3
0a3	aaa\$	Shift S3
0a3a3	aa\$	Shift S3
0a3a3a3	a\$	Shift S3
0a3a3a3a3	\$	

---NOT ACCEPTED---

Process finished with exit code 0



-ساختار:

برای این قسمت از این لینک استفاده شده است منتهی دو چالش وجود داشت:

۱. ورودی آن خط به خط از کاربر گرفته میشد.

۲. خروجی فقط شامل جدول LR(1) بود.

که با کمک فایل و تغییرات در تابع main در فایل firstfollow مشکل اول و به کمک جدول LR(1) مشکل دوم یعنی نداشتن جدول تجزیه برای یک ورودی و تعیین ریجکت یا اکسپت را حل کردیم.

## LALR(1):

با انتخاب عدد 3 در لیست تحلیلگرها وارد بخش LALR(1) میشویم. در این قسمت هم ورودی فقط یک فایل است: ( البته بعد از دادن گرامر رشته نیز به عنوان ورودی می‌پذیرد )

```
*** LEXYN ***
Hello, Please select what you want:
1. Lexical Analysis
2. Syntax Analysis
2
Type of syntsx analysis
1. LL(1)
2. LR(1)
3. LALR(1)
4. SLR(1)
3
Please Enter File name :|
```

بخشی از خروجی به شکل زیر است:

```
Please Enter File name :parsing_test.txt
I0
S -> . A A , $
A -> . a A , a | b
A -> . b , a | b

on A
I1
S -> A . A , $
A -> . a A , Eps | $
A -> . b , Eps | $

on A
I2
S -> A A . , $

on a
I3
A -> a . A , Eps | $
A -> . a A , Eps | $
A -> . b , Eps | $

on A
I4
A -> a A . , Eps | $

on a
on b
I5
A -> b . , Eps | $
```

## ساختار کد:

برای پیاده سازی از این لینک استفاده شد اما ورودی را از فایل نمی گرفت که این مشکل را با تغییر دادن ساختار کد درست کردیم.

## SLR(1):

بار دیگر با وارد کردن یک عدد ( ۴ ) از لیست تحلیلگر ها می توان SLR(1) را از میان آن ها انتخاب کرد:

```
*** LEXYN ***
Hello, Please select what you want:
1. Lexical Analysis
2. Syntax Analysis
2
Type of syntsx analysis
1. LL(1)
2. LR(1)
3. LALR(1)
4. SLR(1)
4
Enter Grammar File Name::
```

```
-----RULES-----
(1, "S'->S$")
(2, 'S->aAB')
(3, 'S->SDb')
(4, 'A->adB')
(5, 'A->a')

-----AUGMENTED RULES-----
S'->S$
S->aAB
S->SDb
A->adB
A->a

-----STATES-----
I0 :
S'->.S$
S->.aAB
S->.SDb

I1 :
S'->S.$
S->S.Db

I2 :
S->a.AB
A->.adB
A->.a

I3 :
S'->S$.

I4 :
S->SD.b
```

```
I7 :
S->SDb.

I8 :
S->aAB.

I9 :
A->ad.B

I10 :
A->adB.

-----GOTO OPERATIONS-----
[0, 'S', 1]
[0, 'a', 2]
[1, '$', 3]
[1, 'D', 4]
[2, 'A', 5]
[2, 'a', 6]
[4, 'b', 7]
[5, 'B', 8]
[6, 'd', 9]
[9, 'B', 10]

-----REDUCTION-----
[1, '$', 'Accept']
SR conflict
[[1, '$', 3], [1, '$', 'Accept']]
```