



“Compiler Parser”

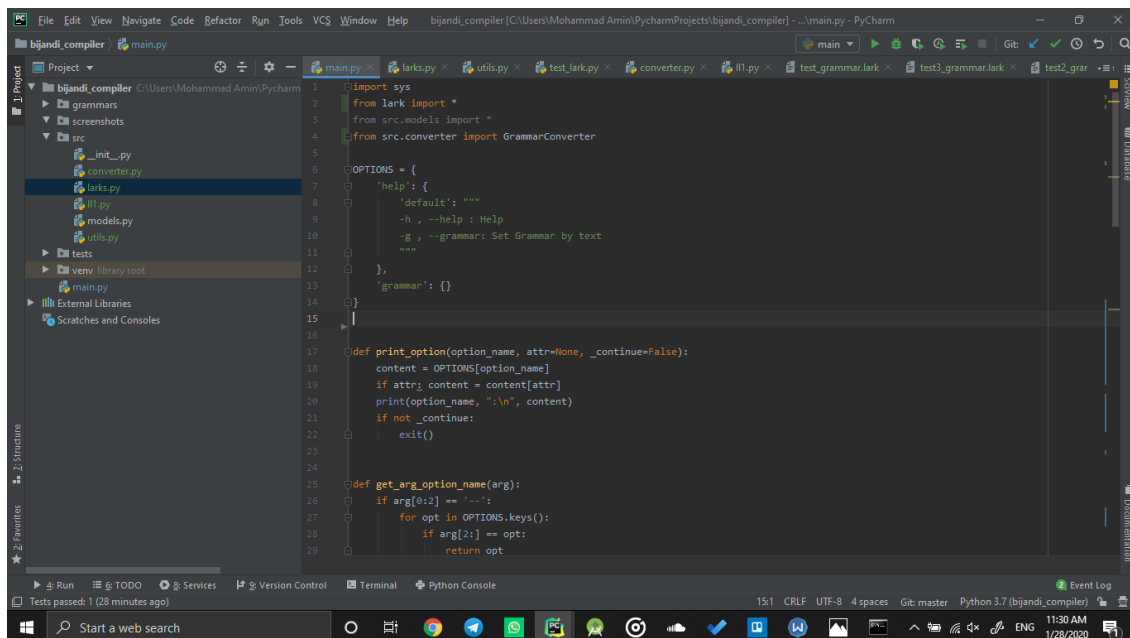
درس:	کامپایلر
استاد:	دکتر زهرا جلالیان
دانشجو:	علیرضا بیجندی
شماره دانشجویی:	۹۵۲۰۲۳۰۴۴

در ادامه مستندات پروژه پایانی درس کامپایلر خدمت شما ارائه میشود.
زبان کد پروژه پایتون (python) است.

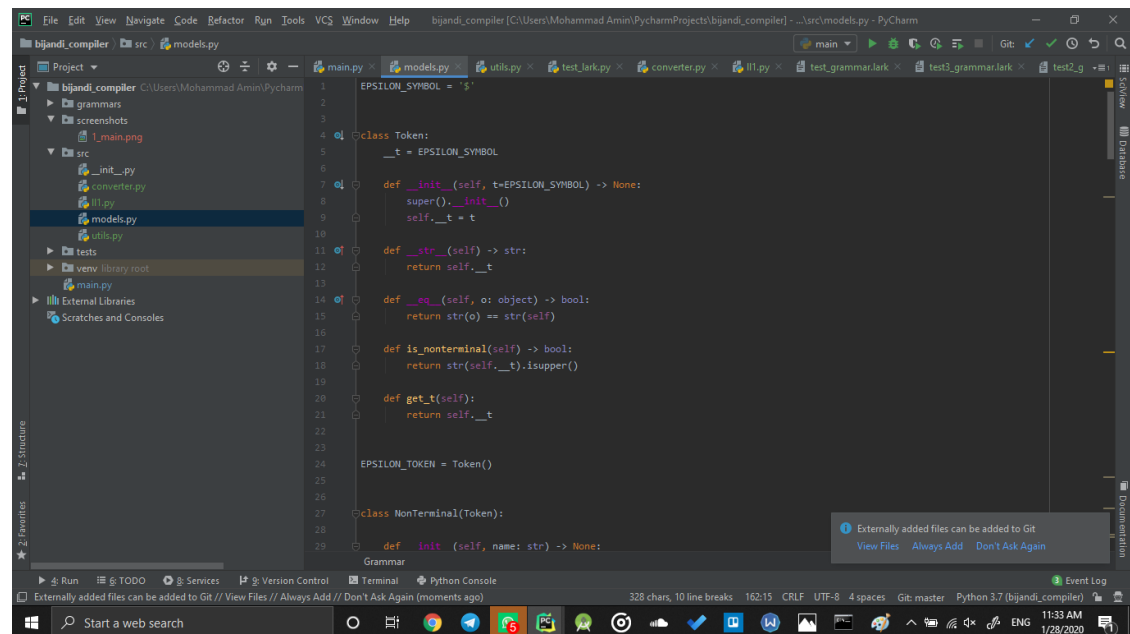
با تشکر از توجه شما
علیرضا بیجندی

در این پروژه با استفاده از زبان پایتون یک تجزیه کننده (parser) و یک تحلیلگر لغوی (lexer) در پروژه پیاده شده است که میتوان با استفاده از دستورات ترمینال و یا از طریق فایل یک گرامر برای برنامه تعریف و عملیات بر روی آن انجام داد.

تصاویری از کد برنامه:

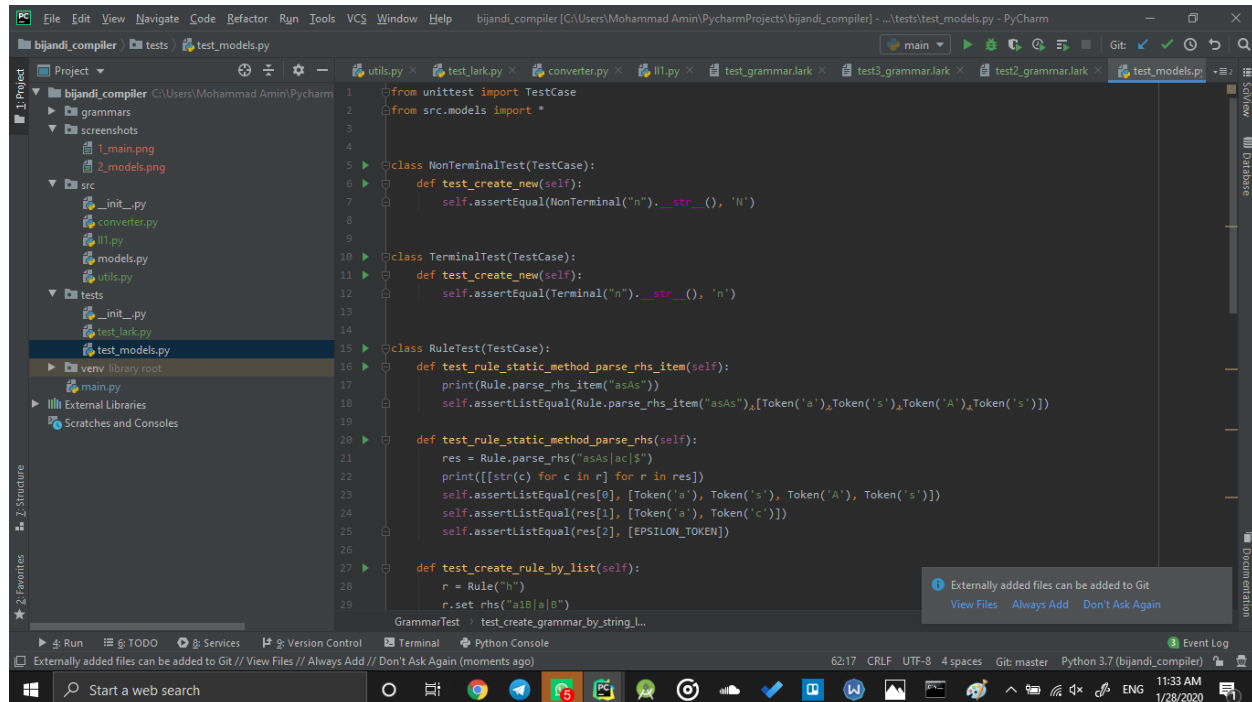


```
1 import sys
2 from lark import *
3 from src.models import *
4 from src.converter import GrammarConverter
5
6 OPTIONS = {
7     'help': {
8         'default': '---
9         -h, --help : Help
10        -g, --grammar: Set Grammar by text
11        ---
12    },
13    'grammar': {}
14 }
15
16
17 def print_option(option_name, attr=None, _continue=False):
18     content = OPTIONS[option_name]
19     if attr: content = content[attr]
20     print(option_name, "\n", content)
21     if not _continue:
22         exit()
23
24
25 def get_arg_option_name(arg):
26     if arg[0:2] == '--':
27         for opt in OPTIONS.keys():
28             if arg[2:] == opt:
29                 return opt
```



```
1 EPSILON_SYMBOL = '$'
2
3
4 class Token:
5     __t = EPSILON_SYMBOL
6
7     def __init__(self, t=EPSILON_SYMBOL) -> None:
8         super().__init__()
9         self.__t = t
10
11     def __str__(self) -> str:
12         return self.__t
13
14     def __eq__(self, o: object) -> bool:
15         return str(o) == str(self)
16
17     def is_nonterminal(self) -> bool:
18         return str(self.__t).isupper()
19
20     def get_t(self):
21         return self.__t
22
23
24 EPSILON_TOKEN = Token()
25
26
27 class NonTerminal(Token):
28
29     def __init__(self, name: str) -> None:
30         Grammar
```

در ضمن تمام قسمت های برنامه تست واحد (unit test) شده اند و تست ها در مسیر /test قرار دارند:



همانطور که در بالا گفته شد می توان در این پروژه از فایل گرامر با پسوند lark استفاده کرد.

در مسیر grammar/ از پروژه تعداد ۳ گرامر به صورت تستی تعریف شده است. گرامر های:

1. Test_grammar.lark
2. Test2_grammar.lark
3. Test3_grammar.lark

```
main.py × models.py × utils.py × test_lark.py × converter.py × ll1.py × test_grammar.lark × test3_grammar.lark × test2_g...
1 exp: sum
2   | ID OP_EQUAL sum
3 OP_EQUAL: "="
4 ID : LETTER
5   | "_" LETTER
6 sum: item
7   | sum OP_MATH item
8 OP_MATH: "+"
9   | "-"
10  | "*"
11  | "/"
12
13 item: NUMBER      -> op_number
14   | "-" item      -> op_neg
15   | "(" sum ")"
16 %import common.NUMBER
17 %import common.WS
18 %import common.LETTER
19 %ignore WS
```

```
y × models.py × utils.py × test_lark.py × converter.py × ll1.py × test_grammar.lark × test3_grammar.lark × test2_grammar.lark × ...
1 exp: sum
2   | ID OP_EQUAL sum
3 OP_EQUAL: "="
4 ID : LETTER
5   | "_" LETTER
6 sum: product
7   | sum OP_SUM product
8 OP_SUM: "+" | "-"
9 product: item
10  | product OP_PRODUCT item
11 OP_PRODUCT: "*" | "/"
12 item: NUMBER      -> op_number
13  | "-" item      -> op_neg
14  | "(" sum ")"
15
16 %import common.NUMBER
17 %import common.WS
18 %import common.LETTER
19 %ignore WS
```

```
y × models.py × utils.py × test_lark.py × converter.py × ll1.py × test_grammar.lark × test3_grammar.lark × test2_grammar.lark × ...
1 exp: sum
2   | ID "=" sum
3 ID: LETTER | "_" LETTER
4 sum: product
5   | sum "+" product -> op_add
6   | sum "-" product -> op_sub
7 product: item
8   | product "*" item -> op_mul
9   | product "/" item -> op_div
10 item: NUMBER      -> op_number
11  | "-" item      -> op_neg
12  | "(" sum ")"
13
14 %import common.NUMBER
15 %import common.WS
16 %import common.LETTER
17 %ignore WS
```

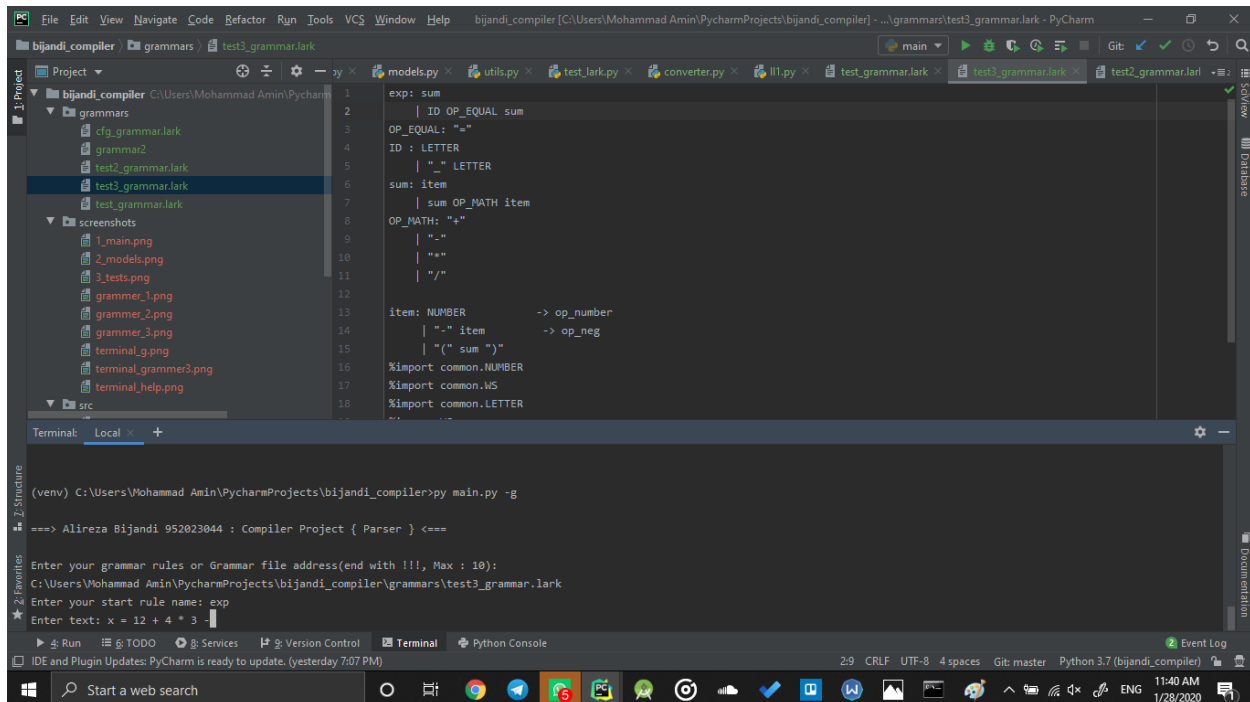
برای ورودی این گرامر ها، کافیت دستور:

`py main.py -g`

را وارد کرده تا متن زیر در ترمینال نمایش پیدا کند:

```
Terminal: Local x +
(venv) C:\Users\Mohammad Amin\PycharmProjects\bijandi_compiler>py main.py -g
==> Alireza Bijandi 952023044 : Compiler Project { Parser } <==
Enter your grammar rules or Grammar file address(end with !!!, Max : 10):
```

سپس آدرس گرامر مورد نظر را وارد میکنیم(به عنوان نمونه گرامر تست ۳)



حال کافیت قانون (Rule) شروع گرامر را تایین کنیم

در مثال بالا: exp

سپس رشته مورد نظر را وارد میکنیم، در صورت درست بودن رشته ورودی تمام Token ها و درخت تجزیه گرامر نمایش داده می شود:

```
Terminal: Local x +
Enter your grammar rules or Grammar file address(end with !!!, Max : 10):
C:\Users\Mohammad Amin\PycharmProjects\bijandi_compiler\grammars\test3_grammar.lark
Enter your start rule name: exp
Enter text: x = 12 + 4 * 3
>>> Valid input String

>>> Tokens:
[Token(ID, 'x'), Token(OP_EQUAL, '='), Token(NUMBER, '12'), Token(OP_MATH, '+'), Token(NUMBER, '4'), Token(OP_MATH, '*'), Token(NUMBER, '3')]

>>> Tree map:
Tree(exp, [Token(ID, 'x'), Token(OP_EQUAL, '='), Tree(sum, [Tree(sum, [Tree(op_number, [Token(NUMBER, '12')])]), Token(OP_MATH, '+'), Tree(op_number, [Token(NUMBER, '4')])]), Token(OP_MATH, '*'), Tree(op_number, [Token(NUMBER, '3')])])])

>>> Tree:
exp
x
=
sum
sum
sum
op_number 12
+
op_number 4
*
op_number 3
```

در صورت مطابق نبودن رشته با گرامر با خطا مواجه میشوید:

```
Terminal: Local x +
(venv) C:\Users\Mohammad Amin\PycharmProjects\bijandi_compiler>py main.py -g

===> Alireza Bijandi 952023044 : Compiler Project { Parser } <===

Enter your grammar rules or Grammar file address(end with !!!, Max : 10):
C:\Users\Mohammad Amin\PycharmProjects\bijandi_compiler\grammars\test_grammar.lark
Enter your start rule name: exp
Enter text: 2 $$ 12

>>> Invalid input String!
No terminal defined for '$' at line 1 col 3

2 $$ 12
 ^

Expecting: {'STAR', 'SLASH', 'PLUS', 'MINUS'}
```

در ضمن شما میتواند در هنگام اجرا به صورت دستی گرامر را وارد برنامه کنید:

```
Terminal: Local x +

(venv) C:\Users\Mohammad Amin\PycharmProjects\bijandi_compiler>py main.py -g

==> Alireza Bijandi 952023044 : Compiler Project { Parser } <==

Enter your grammar rules or Grammar file address(end with !!!, Max : 10):
e => s | ID "=" s
ID => LETTER | "_" LETTER
s=> i | s OP_MATH i
OP_MATH => "+"|"-"|"*"|"/"
i=>NUMBER
!!!
Enter your start rule name: 
```

(حداکثر تعداد قوانین برنامه ۱۰ و برای پایان ورودی از !!! استفاده کنید)

نتیجه:

```
Terminal: Local x +

e => s | ID "=" s
ID => LETTER | "_" LETTER
s=> i | s OP_MATH i
OP_MATH => "+"|"-"|"*"|"/"
i=>NUMBER
!!!
Enter your start rule name: e
Enter text: x = 12 - 33 * 88
>>> Valid input String

>>> Tokens:
[Token(ID, 'x'), Token(NUMBER, '12'), Token(OP_MATH, '-'), Token(NUMBER, '33'), Token(OP_MATH, '*'), Token(NUMBER, '88')]

>>> Tree map:
Tree(e, [Token(ID, 'x'), Tree(s, [Tree(s, [Tree(i, [Token(NUMBER, '12')])]), Token(OP_MATH, '-'), Tree(i, [Token(NUMBER, '33')])]), Token(OP_MATH, '*'), Tree(i, [Token(NUMBER, '88')])])])])

>>> Tree:
e
x
s
s
s
i      12
-
i 33
*
i 88
```

توکن ها:

```
>>> Tokens:
[Token(ID, 'x'), Token(NUMBER, '12'), Token(OP_MATH, '-'), Token(NUMBER, '33'), Token(OP_MATH, '*'), Token(NUMBER, '88')]
```

درخت تجزیه:

```
>>> Tree:
e
x
s
s
s
i 12
-
i 33
*
i 88
```