دانشکده مهندسی کامپیوتر

اصول طراحی کامپایلر

تمرین سری اول

نصب و استفاده از انتلر

علی صداقی

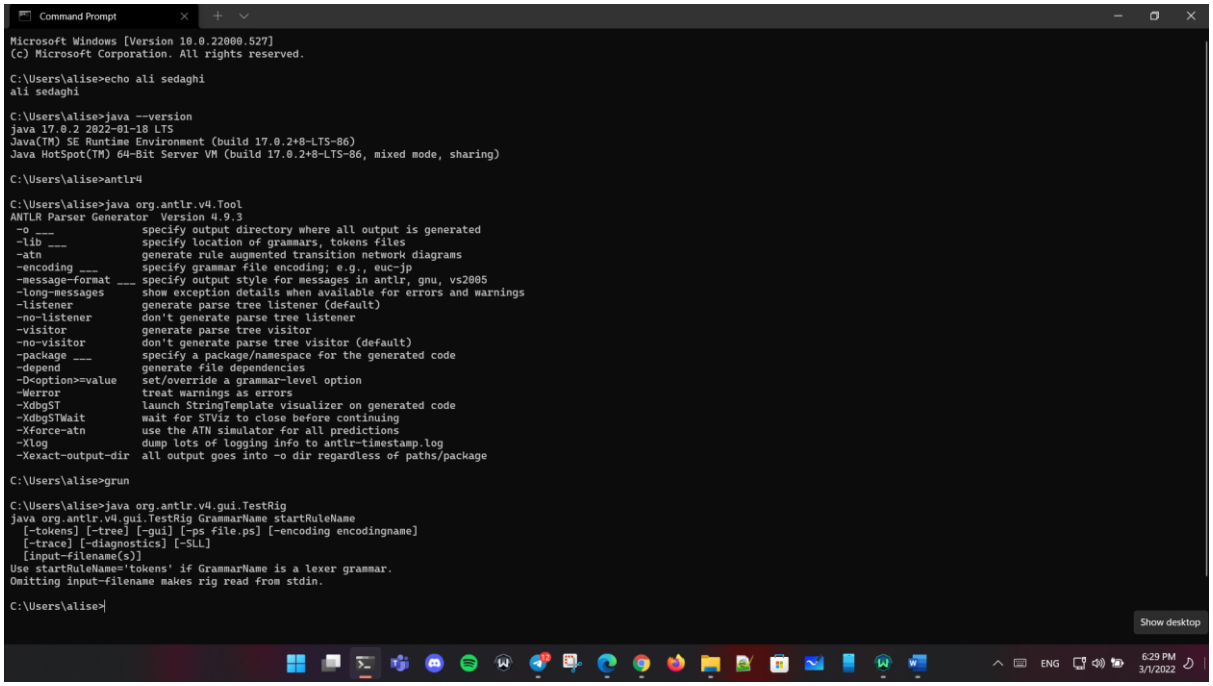
۹۷۵۲۱۳۷۸

۱ سوال اول

بنده در ترم تحصیلی ۳۹۹۲ این برنامه را نصب کردم و مراحل آن را در کلاس درس هم ارائه دادم. ارائه که شامل مراحل نصب انتلر بود در فایل زیر موجود است.

ANTLR_Installation.pdf

مراحل بالا را دوباره در ویندوز ۱۱ خود انجام دادم. خروجی دستورات به صورت زیر است:

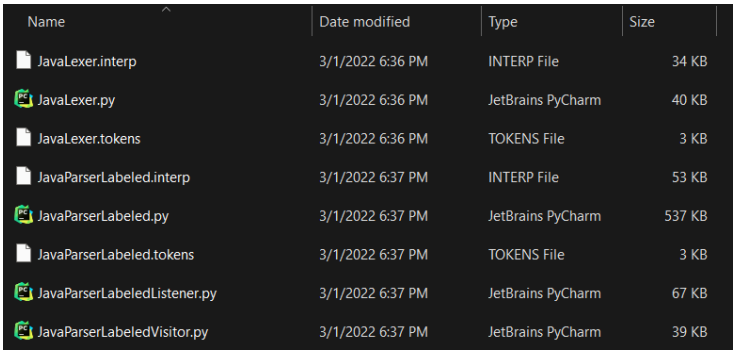


تولید فایل‌های Lexer، Parser، Listener و Visitor به دو روش ممکن است.

روش اول: استفاده از دستور:

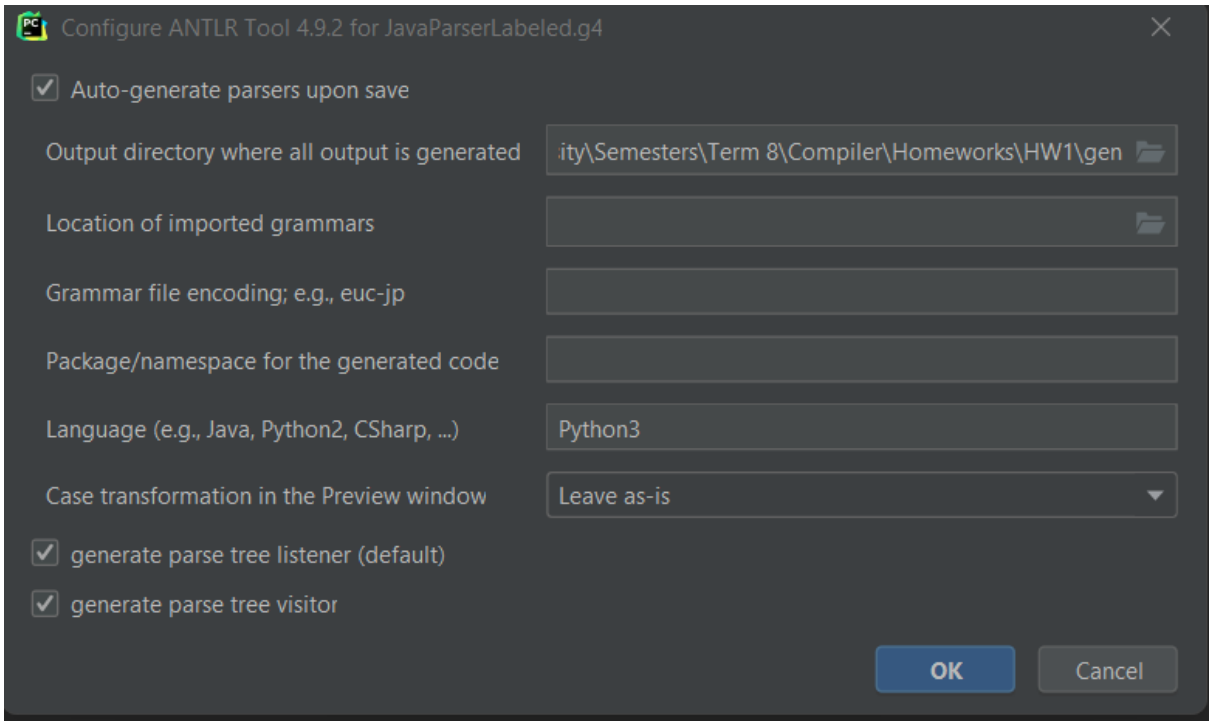
antlr4 -listener -visitor -Dlanguage=Python3 JavaLexer.g4 -o gen

antlr4 -listener -visitor -Dlanguage=Python3 JavaParserLabeled.g4 -o gen



روش دوم: استفاده از محیط PyCharm:

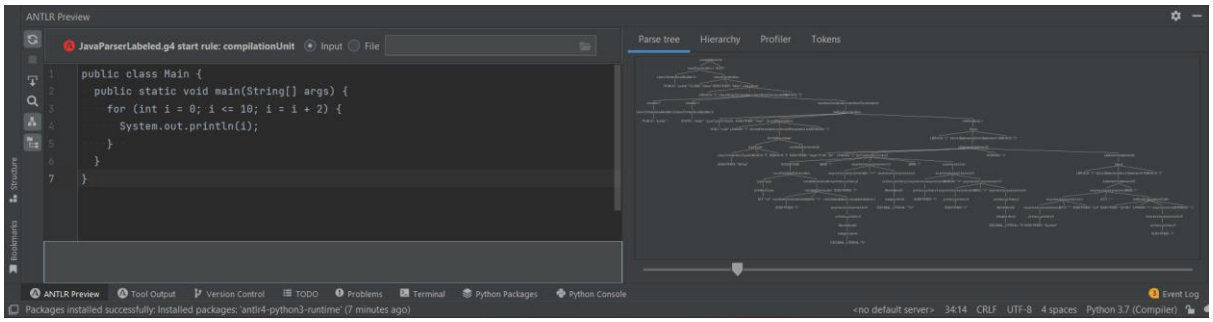
روی فایل‌های g4. کلیک راست می‌کنیم. سپس روی Configure ANTLR می‌زنیم. به صورت زیر تنظیم می‌کنیم.



روی فایل‌های g4. کلیک راست می‌کنیم و گزینه Generate ANTLR Recognizer را می‌زنیم. که ۸ فایل ایجاد می‌شود که همان فایل‌هایی است که در بالا تصویر آن داده شد.

۲ سوال دوم

فایل JavaParserLabeled.g4 را باز می‌کنیم. در خطی که گرامر شروع می‌شود (compilationUnit) کلیک راست می‌کنیم. گزینه Test Rule compilationUnit را می‌زنیم. قسمت ANTLR Preview را باز می‌کنیم. کد جاوا را وارد می‌کنیم. درخت تجزیه به صورت زیر است.



تصویر واضح درخت تجزیه درون فایل زیر موجود است.

parseTree.png

۳ سوال سوم

پیاده‌سازی این سوال در فایل Q3.py موجود است. همچنین پروژه جاوا در مسیر java_project/. موجود است.

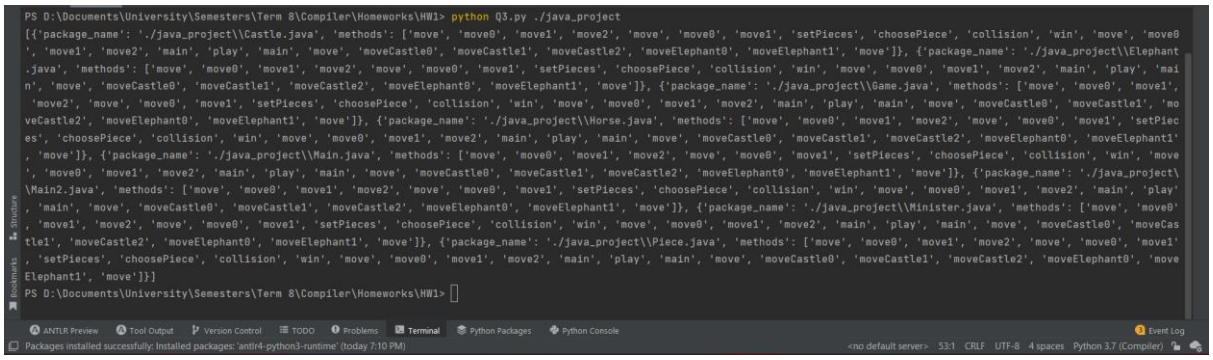
با استفاده از argparse برای اسکریپت پایتون help هم ایجاد شده است.

python Q3.py -h

```
PS D:\Documents\University\Semesters\Term 8\Compiler\Homeworks\HW1> python Q3.py -h
usage: Q3.py [-h] dir

Java methods extractor

positional arguments:
  dir         Directory of Java project

optional arguments:
  -h, --help  show this help message and exit
```

برنامه را با دستور زیر اجرا می‌کنیم.

python Q3.py ./java_project



توضیح کد:

ابتدا یک کلاس به نام FindMethods ایجاد می‌کنیم که از کلاس JavaParserLabeledListener ارث بری می‌کند. در کلاس ایجاد شده یک لیست به نام methods داریم که وظیفه نگه داری نام توابع را دارد. تابع enterMethodDeclaration را در این کلاس Override می‌کنیم. کانتکست آن را MethodDeclarationContext در نظر می‌گیریم و هر گاه که وارد حوزه تعریف یک تابع شدیم نام آن تابع (IDENTIFIER) را درون لیست methods اضافه می‌کنیم.

```
9   class FindMethods(JavaParserLabeledListener):
10      def __init__(self):
11          self.methods = []
12
13      def enterMethodDeclaration(self, ctx: JavaParserLabeled.MethodDeclarationContext):
14          self.methods.append(str(ctx.IDENTIFIER()))
```

یک تابع به نام extract_methods ایجاد می‌کنیم که با ورودی گرفتن مسیر پروژه جاوا و استخراج کردن فایل‌های Java آن و ایجاد Listener، Lexer، Parser و انجام عملیات Walk نام توابع را استخراج می‌کند. خروجی این تابع یک لیست است که به تعداد پکیج‌های جاوا عضو دارد. هر عضو این لیست یک دیکشنری شامل یک رشته برای package_name و یک لیست برای methods است.

```python
def extract_methods(project_directory):
    result = []
    for root, _, files in os.walk(project_directory):
        for file in files:
            if file.split('.')[-1] == 'java':
                file_path = os.path.join(root, file)
                stream = FileStream(file_path, encoding="utf8")

                lexer = JavaLexer(stream)
                token_stream = CommonTokenStream(lexer)

                parser = JavaParserLabeled(token_stream)
                tree = parser.compilationUnit()

                listener = FindMethods()
                walker = ParseTreeWalker()

                walker.walk(listener, tree)

                package_details = {
                    "package_name": file_path,
                    "methods": listener.methods
                }
                result.append(package_details)
    return result
```

در نهایت درون Main همه موارد را Wrap می‌کنیم و برنامه را با آرگومان‌های ترمینال اجرا می‌کنیم.

```python
if __name__ == "__main__":
    arg_parser = argparse.ArgumentParser(description='Java methods extractor')
    arg_parser.add_argument(
        'dir',
        type=str,
        help='Directory of Java project'
    )
    args = arg_parser.parse_args()
    methods = extract_methods(args.dir)
    print(methods)
```

۴ سوال چهارم

پیاده‌سازی این سوال در فایل Q4.py موجود است. همچنین پروژه جاوا در مسیر ./java_project موجود است.

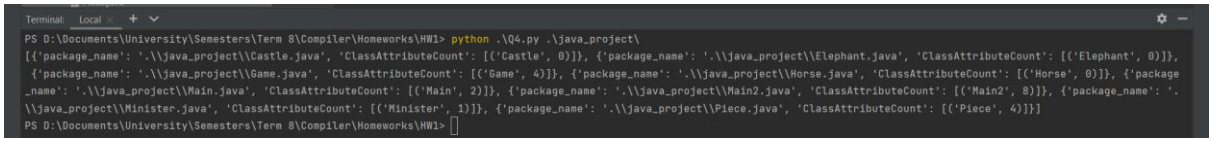
با استفاده از argparse برای اسکریپت پایتون help هم ایجاد شده است.

python Q4.py -h

```
PS D:\Documents\University\Semesters\Term 8\Compiler\Homeworks\HW1> python .\Q4.py -h
usage: Q4.py [-h] dir

Java class attributes counter

positional arguments:
  dir         Directory of Java project

optional arguments:
  -h, --help  show this help message and exit
PS D:\Documents\University\Semesters\Term 8\Compiler\Homeworks\HW1>
```

برنامه را با دستور زیر اجرا می‌کنیم.

python Q4.py ./java_project



توضیح کد:

ابتدا یک کلاس به نام FindAttributesCount ایجاد می‌کنیم که از کلاس JavaParserLabeledListener ارث بری می‌کند. در کلاس ایجاد شده یک متغیر عددی به نام class_attributes_count داریم که وظیفه نگه داری تعداد Attributeهای کلاس را دارد. همچنین از یک متغیر به نام class_name استفاده شده که وظیفه نگه داری نام کلاس را دارد. تابع enterClassDeclaration را در این کلاس Override می‌کنیم. کانتکست آن را ClassDeclarationContext در نظر می‌گیریم و هر گاه که وارد حوزه تعریف یک کلاس شدیم نام آن کلاس (IDENTIFIER) را استخراج می‌کنیم. تابع enterVariableDeclarator را نیز Override می‌کنیم. کانتکست آن را VariableDeclaratorContext در نظر می‌گیریم و هر گاه وارد حوزه تعریف یک Attribute شدیم مقدار class_attributes_count را یکی Increment می‌کنیم. تابع exitClassDeclaration هنگام خروج از حوزه تعریف متغیرات، یک تاپل می‌سازد و به لیست classes_tuple اضافه می‌کند.

```python
class FindAttributesCount(JavaParserLabeledListener):
    def __init__(self):
        self.class_attributes_count = 0
        self.class_name = ""
        self.classes_tuple = []

    def enterClassDeclaration(self, ctx: JavaParserLabeled.ClassDeclarationContext):
        self.class_name = str(ctx.IDENTIFIER())

    def enterVariableDeclarator(self, ctx: JavaParserLabeled.VariableDeclaratorContext):
        self.class_attributes_count += 1

    def exitClassDeclaration(self, ctx: JavaParserLabeled.ClassDeclarationContext):
        self.classes_tuple.append((self.class_name, self.class_attributes_count))
```

یک تابع به نام extract_attributes_count (مشابه سوال قبل) ایجاد می‌کنیم که با ورودی گرفتن مسیر پروژه جاوا و استخراج کردن فایل‌های Java آن و ایجاد Listener، Lexer، Parser و انجام عملیات Walk نام توابع را استخراج می‌کند. خروجی این تابع یک لیست است که به تعداد پکیج‌های جاوا عضو دارد. هر عضو این لیست یک دیکشنری شامل یک رشته برای package_name و یک لیست برای ClassAttributeCount (لیستی از تاپل‌ها) است.

```
25      def extract_attributes_count(project_directory):
26          result = []
27          for root, _, files in os.walk(project_directory):
28              for file in files:
29                  if file.split('.')[-1] == 'java':
30                      file_path = os.path.join(root, file)
31                      stream = FileStream(file_path, encoding="utf8")
32
33                      lexer = JavaLexer(stream)
34                      token_stream = CommonTokenStream(lexer)
35
36                      parser = JavaParserLabeled(token_stream)
37                      tree = parser.compilationUnit()
38
39                      listener = FindAttributesCount()
40                      walker = ParseTreeWalker()
41
42                      walker.walk(listener, tree)
43
44                      class_attributes = {
45                          "package_name": file_path,
46                          "ClassAttributeCount": listener.classes_tuple
47                      }
48                      result.append(class_attributes)
49          return result
```

در نهایت درون Main همه موارد را Wrap می‌کنیم و برنامه را با آرگومان‌های ترمینال اجرا می‌کنیم.

```
52  if __name__ == "__main__":
53      arg_parser = argparse.ArgumentParser(description='Java class attributes counter')
54      arg_parser.add_argument(
55          'dir',
56          type=str,
57          help='Directory of Java project'
58      )
59      args = arg_parser.parse_args()
60      attributes = extract_attributes_count(args.dir)
61      print(attributes)
```

۵ سوال پنجم

فایل گرامر در مسیر زیر موجود است:

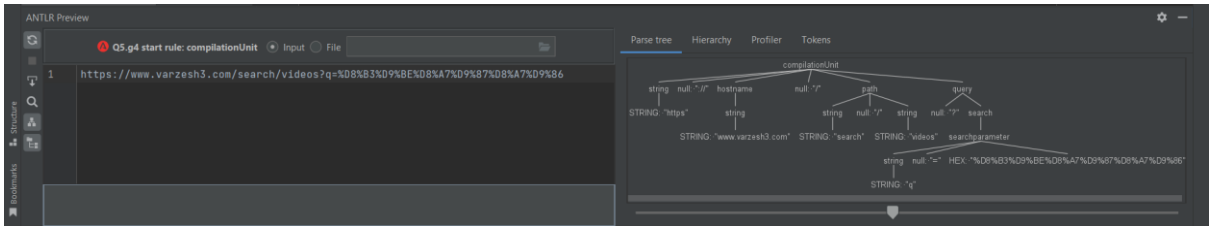
grammars/Q5.g4

برای حل این سوال از گرامرهای درون ریپازیتوری گیتهاب انتلر کمک گرفته شده است.

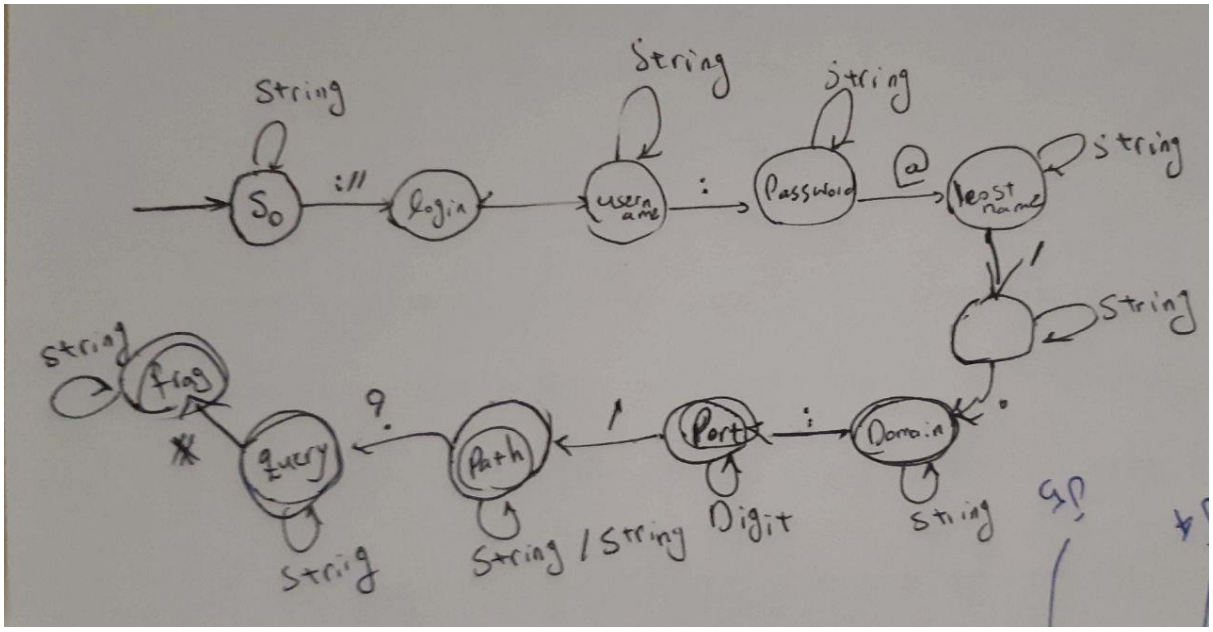
https://github.com/antlr/grammars-v4/tree/master/url

درخت تجزیه برای ورودی اول:

درخت تجزیه ورودی دوم:



دیاگرام FSA:



- ✓ در این سوال فرض کردیم که URL از نوع IPv6 نخواهد بود.

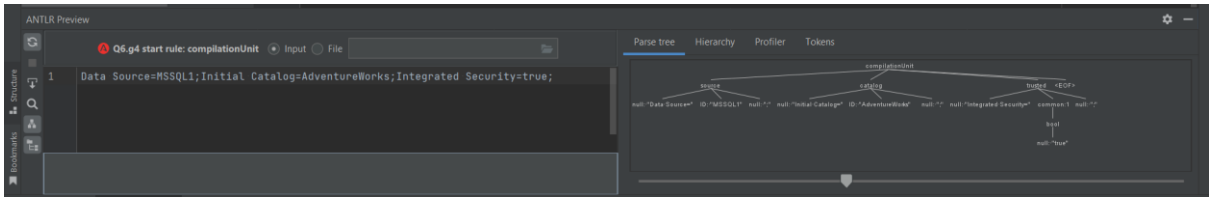
۶ سوال ششم

فایل گرامر در مسیر زیر موجود است:

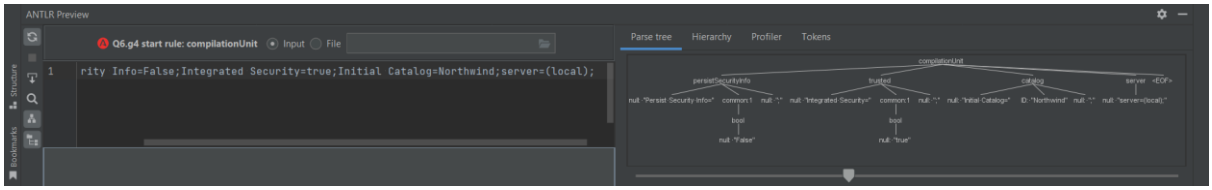
grammars/Q6.g4

گرامر نوشته شده را با ورودی‌های زیر تست می‌کنیم:

Data Source=MSSQL1;Initial Catalog=AdventureWorks;Integrated Security=true;



Persist Security Info=False;Integrated Security=true;Initial Catalog=Northwind;server=(local);



دیاگرام FSA: