به نام خدا

فاز اول پروژه‌ی درس کامپایلر و زبان‌های برنامه‌نویسی

بهار 1401

مهلت تحویل : 25 فروردین

در فاز اول پروژه، شما باید به کمک ابزار ANTLR4 و زبان برنامه‌نویسی جاوا، برای زبان SimpleLOOP (که سند آن در اختیار شما قرار گرفته است) تحلیلگر لغوی و نحوی بنویسید.

تحلیلگر لغوی[1] :

در این بخش از پروژه، باید به کمک ابزار ANTLR4، تمامی Tokenهای مورد نظر در زبان SimpleLOOP را مشخص و پیاده‌سازی کنید.

تحلیلگر نحوی[2] :

در این بخش از پروژه، ابتدا با نوشتن قواعد نحوی صحیح، گرامر زبان SimpleLOOP را به کمک ANTLR4 پیاده سازی می‌کنید. سپس، با اعمال ورودی‌های مورد نظر و با توجه به درخت

[1] Lexer
[2] Parser

Parse[3] گرامر خود را تست و اطلاعات خواسته شده را چاپ می‌کنید. بهتر است برای قواعد خود نام‌های مناسب انتخاب کنید تا فهم آنها راحت‌تر باشد. هم چنین توجه داشته باشید که گرامر شما نباید مشکل چپ‌گردی و ابهام داشته باشد.

در این فاز نیازی نیست که هیچ گونه قاعده معنایی[4] را بررسی و پیاده سازی کنید . برای مثال وجود دو آرگومان هم‌نام، شرطی بودن عبارت شرطی if، ناهماهنگی تایپ‌ها و . . . همگی از مواردی هستند که در فازهای بعدی بررسی می‌شوند و در این فاز نیازی به رسیدگی به آن‌ها نیست.

مواردی که در پیاده سازی گرامر خود باید رعایت کنید، به شرح زیر است:

1) ابتدا متغیرهای سراسری[5] تعریف می‌شوند. سپس کلاس‌ها تعریف می‌شوند.
2) در هر scope اگر فقط یک گزاره وجود دارد، میتوان از {} استفاده نکرد؛ در غیر این صورت حتما scope باید با این دو کاراکتر مشخص شود.
3) در هنگام تعریف متد کلاس، بین تعریف متد و اولین گزاره از متد باید حتما یک خط، فاصله وجود داشته باشد و آخرین گزاره بدنه تابع از بقیه برنامه نیز توسط یک خط جدا می شود. در صورت استفاده از } و { ،کاراکتر } میتواند در همان خط تعریف تابع یا بعد از آن بیاید؛ قبل و بعد از کاراکتر { (پایان scope فعلی) حتماً یک خط، فاصله وجود دارد. برای باقی scopeها نیز به همین صورت است. برای روشن‌تر شدن این موضوع، به مثال زیر توجه کنید:

```
class Student {
   private int method(int arg) {
       return arg * 2
   }

   public initialize() {
```

[3] درختی که ساختار نحوی یک رشته از توکن‌ها را با توجه به گرامر زبان نشان می‌دهد
[4] Semantic rules
[5] global

```
        int a
        if (true)
            print(a)
        elsif a > 4 {
            print(4)
        }
    }
}
```

4) در برنامه می‌تواند هر تعداد خط خالی از کد وجود داشته باشد و تأثیری در روند اجرا نخواهند داشت.
5) تابع print با ورودی خالی نداریم.
6) همانطور که قبلاً گفته شد لازم نیست ناهماهنگی در تایپ‌ها در این مرحله بررسی شوند.
7) اولویت عملگرها باید رعایت شود.
8) نیازی به بررسی وجود کلاس main و متد initialize در این فاز نیست.
9) تعریف متغیرهای محلی درون متدها در ابتدای متد و قبل از گزاره های دیگر صورت می گیرد و امکان تعریف متغیر در scope های درونی‌تر وجود ندارد.

خروجی:

پس از نوشتن تحلیلگرهای لغوی و نحوی، باید به کمک Action هایی که با زبان java می‌نویسید و به گرامر خود اضافه می‌کنید خروجی‌های زیر را بر اساس پیمایش Pre-order درخت parse چاپ کنید :

1) هنگام رسیدن به تعریف هر کلاس، نام کلاس را به صورت زیر (قبل از مشاهده‌ی دستورات داخل آن) چاپ کنید :

ClassDec : #name_of_class

2) در صورت وجود ارث‌بری در تعریف کلاس، عبارت زیر را چاپ کنید (قبل از مشاهده‌ی گزاره‌های داخل کلاس):

Inheritance : #name_of_class < #name_of_parent_class

*دقت کنید که کلماتی که قبلشان # نیامده باید عیناً تکرار شوند.

3) هنگام رسیدن به تعریف هر متد، نام متد را به صورت زیر (قبل از مشاهده‌ی دستورات داخل آن) چاپ کنید :

MethodDec : #name_of_method

4) هنگام رسیدن به تعریف آرگومان‌های یک متد، نام آرگومان را به صورت زیر چاپ کنید :

ArgumentDec : #name_of_argument

5) با رسیدن به تعریف هر متغیر، نام آن را به صورت زیر چاپ کنید:

VarDec: #name_of_variable

6) در صورت مشاهده دستورات if و else و elsif، به صورت زیر آنها را چاپ کنید :

Conditional : if

Conditional : elsif

Conditional : else

7) در صورت مشاهده دستور each، آن را به صورت زیر چاپ کنید:

Loop : each

8) در صورت مشاهده دستور return آن را به صورت زیر چاپ کنید :

Return

9) در صورت مشاهده دستور print آن را به صورت زیر چاپ کنید :

Built-in : print

10) در صورت مشاهده فراخوانی یک متد به عنوان یک گزاره (نه در expression)، آن را به صورت زیر چاپ کنید:

MethodCall

11) در صورت مشاهده توابع مختص به مجموعه[6]ها، آنها را به صورت زیر چاپ کنید :

NEW

ADD

MERGE

INCLUDE

DELETE

عبارت ها را در درخت عبارت به صورت Post-order پیمایش کنید. یعنی برای یک عملگر دوتایی ابتدا عملگرهای عملوند اول، سپس عملگرهای عملوند دوم و در نهایت خود آن عملگر چاپ شود .(در درخت عبارت، عملگر تک عملوندی به صورت یک عملگر دو عملوندی در نظر گرفته شود که یک فرزند دارد)

[6] Set

در صورت مشاهده یک عملگر، تنها خود عملگر را به صورت زیر چاپ کنید. عملگر های [ ] ، () و . نباید چاپ شوند.

Operator : #OperatorSymbol

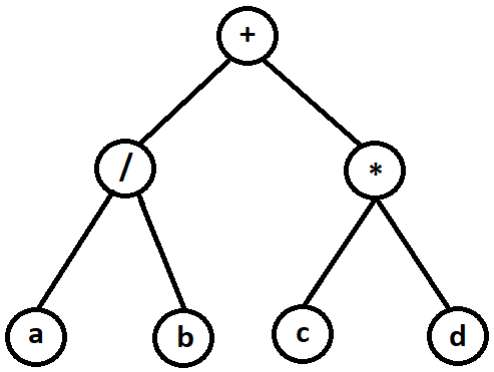
به عنوان مثال برای عبارت a / b + c * d عملگرها را به صورت زیر چاپ کنید :

Operator : /

Operator : *

Operator : +

به درخت عبارت آن توجه کنید :



همچنین دقت کنید که عبارت‌های قبل از : همگی کلیدواژه هستند و آنها را عینا چاپ کنید. تنها موارد خواسته شده را در فایل خروجی نمایش دهید و از قرار دادن خط‌های خالی و فاصله و ... نیز خودداری کنید.

نمونه‌ای از یک قطعه کد و خروجی آن در ادامه آمده است.

```
int total_grade, total_cost

class Student {
   private int grade, cost
   public initialize(int grade, int cost=100) {
       self.grade = grade
       self.cost = cost
       self.enrollments = Set.new()
       print(self.cost);
   }
   public Set<int> enrollments
   public void enroll(int cid, int cost) {
       self.enrollments.add(cid)
       self.cost = self.cost + cost
   }

}

class Course {
   private int cid, cost;
   public void set_cid(int _cid) {
       self.cid = _cid
   }
   public int get_cid() {
       return self.cid
   }
   public void set_cost(int cost) {
       self.cost = cost
   }
   Public int get_cost() {
       return self.cost
   }
}

class Main {
   public initialize() {
       Course crs
       Course crs2
       Student[5] students
       Student i
       crs = Course.new()
       crs.set_cid(1)
       crs.set_cost(120)
       students.each do |i|
               students[i].enroll(crs.get_cid(), crs.get_cost())
       if crs.get_cost() < 100 {
           print(1)
       }
       elsif crs.get_cost() == 100 {
           print(2)
       }
       elsif crs.get_cost() > 100 {
```

```
            print(3)
        }
        crs2 = Course.new()
        crs2.set_cid(crs.get_cid() == 1 ? crs.get_cost() < 100 ? 2 : 3 : 4)
        print(crs2.get_cid())
    }
}
```

```
VarDec : total_grade

VarDec : total_cost

ClassDec : Student

VarDec : grade

VarDec : cost

ArgumentDec : grade

ArgumentDec : cost

NEW

Built-in : print

VarDec : enrollments

MethodDec : enroll

ArgumentDec : cid

ArgumentDec : cost

ADD

Operator : ++
```

```
ClassDec : Course
VarDec : cid
VarDec : cost
MethodDec : set_cid
ArgumentDec : _cid
Operator : ++
MethodDec : get_cid
Return
MethodDec : set_cost
ArgumentDec : cost
Operator : +
MethodDec : get_cost
Return
ClassDec : Main
VarDec : crs
VarDec : i
VarDec : crs2
MethodCall
MethodCall
VarDec : students
```

```
Loop : each
MethodCall
Conditional : if
Operator : <
Built-in : print
Conditional : elsif
Operator : ==
Built-in : print
Conditional : elsif
Operator : >
Built-in : print
MethodCall
Operator : ==
Operator : <
Operator : ?:
Operator : ?:
Built-in : print
```

نکات مهم:

- کد خود را به صورت یک فایل SimpleLOOP.g4 آپلود کنید و نام اولین قانون زبان را simpleLoop بگذارید.
- در صورت کشف هرگونه تقلب، نمره 100- لحاظ میشود.
- دقت کنید که خروجی‌های شما به صورت خودکار تست می‌شوند؛ پس نحوه چاپ خروجی شما باید عیناً مطابق موارد ذکر شده در بالا باشد. علاوه بر آن، درخت parse شما نیز بررسی می‌شود.
- بهتر است سوالات خود را در فروم یا گروه درس مطرح نمایید تا دوستانتان نیز از آن‌ها استفاده کنند؛ در غیر این صورت به مسئولان پروژه ایمیل بزنید:

کیانوش عرشی kianoosharshi@gmail.com

سودابه محمدهاشمی s.mhashemi.221@gmail.com