گرامر زبان TesLang

در این مستند به جزئیات بیشتری از گرامر زبان TesLang می‌پردازیم.

نکته: همانطور که در مستند قبل به آن اشاره کردیم، آن قواعدی از گرامر که با علامت [*] مشخص شده اند، اختیاری هستند و نمره اضافی دارند!

در BNF داده شده به نکات زیر توجه کنید :

- این رنگ ■■■ مربوط به متغیر های گرامر می‌باشد.
- این رنگ ■■■ مربوط به کلمات کلیدی زبان می‌باشد.
- این رنگ ■■■ مربوط به شماره قاعده ها می‌باشد.
- علامت های | و := برای تعریف قواعد گرامر ها استفاده میشوند و از ورودی های زبان نیستند.

عبارات منظم (Regex) مربوط به مرحله Lexical یا Tokenizer :

```
iden    := [a-zA-Z_][a-zA-Z_0-9]*

number  := [0-9]+

comment := \/\/.*$
```

iden: عبارت منظم مربوط به یک identifier را نشان میدهد، که اولین حرف می‌تواند حروف بزرگ یا کوچک یا _ باشد و در ادامه میتواند شامل اعداد نیز باشد.

number: از یک یا بیشتر رقم تشکیل می‌شود.

comment: شامل عبارت // و \n است. (که در هنگام tokenize کردن نباید در نظر گرفته شود).

گرامر زبان TesLang

قاعدهٔ آغازین برنامه :

```
prog    :=
1           |
2           func prog
```

در صورت وجود تابع وضعیت، func تجزیه خواهد شد و سپس مجدد prog اجرا می‌شود. در غیر آن صورت به وضعیت پایانی می‌رسیم.

تعریف تابع در زبان TesLang :

```
func    :=
1           function iden ( flist ) : type => { body } |
2       [*] function iden ( flist ) : type => expr
```

قاعده اول یک تابع دارای بدنه می‌باشد و قاعده دوم، یک تابع بدون بدنه است که صرفاً یک مقدار را برمی‌گرداند.

تابع با کلمهٔ کلیدی function شروع خواهد شد و یک نام آن خواهد آمد، بین دو پرانتز flist قرار می‌گیرد.

در ادامه یک توکن : قرار می‌گیرد و نوع خروجی تابع را بعد از آن مشخص میکنیم و سپس یک توکن => (arrow) قرار داده و به قانون اول رجوع میکنیم، در صورت عدم برقراری آن به قانون دوم رجوع می‌کنیم.

** پیاده سازی قانون دوم نمرهٔ اضافه دارد.

بدنهٔ زبان TesLang :

```
body    :=
1           |
2           stmt body
```

در صورت وجود یک دستور وضعیت stmt تجزیه خواهد شد و سپس مجدد body اجرا می‌شود. در غیر این صورت به وضعیت پایانی می‌رسیم.

دستورات (statements) زبان TesLang :

```
stmt    :=
1           expr;                                   |
2           defvar ;                                |
3           if ( expr ) stmt                        |
4           if ( expr ) stmt else { stmt }          |
5       [*] while ( expr ) stmt                     |
6           for ( iden , iden of expr ) stmt        |
7           return expr ;                           |
8           { body }                                |
9           func
```

بعد از توکن => بدنه تابع اجرا خواهد شد که می‌تواند تک خطی هم باشد (مانند مثالی که در مستند قبل به آن اشاره شد).

1. به اتمام یک عملیات در وضعیت expr اشاره می‌کند.
2. تعریف متغیر را نشان می‌دهد.
3. دستور شرطی if را به صورت تنها نشان می‌دهد.
4. دستور شرطی if...else را نشان می‌دهد.
5. دستور مربوط به حلقه مرسوم while است که تا زمان برقراری expr مورد نظر، stmt را اجرا می‌کند.
   پیاده سازی این بخش نمرهٔ اضافه دارد.
6. دستور مربوط به حلقه for...of را نشان می‌دهد که در زبان پایتون و جاوااسکریپت مشابه آن را داریم:

```python
# Python

    a = [1, 2, 3, 4, 5]
    for index, value in enumerate(a):
        print(f"{index}: {value}")
```

```javascript
//JavaScript

     let a = [1, 2, 3, 4, 5]
     for (let value of a) {
          console.log(value)
     }
```

مثال های بالا صرفاً برای آشنایی با ایدهٔ این نوع حلقه هاست (در زبان JavaScript مشابه ترین حالت همان حلقهٔ forEach است اما به علت تشابه بیشتر syntax، در اینجا از for...of استفاده شده). در زبان TesLang ما نیازمند یک متغیر برای ذخیره سازی اندیس ها و یک متغیر برای ذخیره سازی مقدار آن خانه ها از حافظه هستیم.

نکته: ترتیب این متغیر ها در حلقهٔ for...of در زبان TesLang مشابه مثال زده شده در زبان پایتون است!

پس iden اول همان اندیس است، و iden دوم همان مقدار است. بعد از کلمهٔ کلیدی of باید expr تجزیه شود و در عملیات مفهومی باید بررسی شود که یک لیست را نشان می‌دهد.

7. این دستور مربوط به خروجی تابع است که با کلمهٔ کلیدی return مشخص می‌شود و سپس یک مقدار به صورت expr در جلوی آن می‌آید.
8. این دستور مربوط به یک بدنه می‌باشد که مجدداً درون آن وضعیت body اجرا خواهد شد و بلوک های کد تو در تو را تشکیل می‌دهد (nested blocks).
9. این دستور باعث تشکیل توابع تو در تو (nested functions) در این زبان می‌شود.

در صورت پیاده سازی این بخش 15 نمره اضافه خواهید داشت.

برای درک بهتر این مورد، به مثال زیر که در TesLang پیاده سازی شده توجه کنید.

```
// TesLang

    function funcA(a: Number): Number => {
        function funcB(): Number => {
            return a + 10;
        }

        return funcB();
    }

    log(funcA(5)); // 15
```

تعریف متغیر (defining variables) در زبان TesLang :

```
defvar  :=
1           let iden : type |
2           let iden : type = expr
```

1. این قانون به تعریف متغیر بدون انتساب مقدار اولیه اشاره می‌کند.
2. این قانون به تعریف متغیر به همراه انتساب مقدار اولیه اشاره می‌کند.

نکته: توجه کنید که ابتدا کلمهٔ کلیدی let قرار دارد سپس نام متغیر و بعد از : نوع داده ای آن مشخص میشود و در قانون دوم هم برای انتساب مقدار اولیه کافیست یک = قرار بگیرد و بعد از آن به سراغ تجزیه expr باید رفت.

لیست آرگومان های توابع (function's argument list) در زبان TesLang :

```
flist   :=
1                                   |
2           iden : type             |
3           iden : type , flist
```

1. یک تابع میتواند صفر یا بیشتر آرگومان داشته باشد. این قانون نشان می‌دهد که هنگام تجزیه، آرگومان بعدی می‌تواند خالی باشد.
2. این قانون نشان دهندهٔ یک آرگومان است، به این نحو که ابتدا نام متغیر آمده و سپس بعد از توکن : نوع داده‌ای آن متغیر می‌آید و تجزیه پس از آن پایان می‌یابد.
3. این قانون نیز مانند قانون دوم اجرا میشود، اما با این تفاوت که پس از مشاهدهٔ توکن , مجدداً به تجزیهٔ flist می‌پردازیم.

گرامر زبان TesLang

لیست فراخوانی توابع و ورودی ها (call list) زبان TesLang :

```
clist   :=
1                       |
2           expr        |
3           expr , clist
```

1. یک تابع می‌تواند صفر یا بیشتر ورودی داشته باشد. این قانون نشان می‌دهد که می‌توان تجزیه را بدون وجود **expr** ادامه داد.
2. این قانون نشان دهندهٔ یک ورودی است که پس از آن تجزیهٔ **clist** خاتمه می‌یابد.
3. این قانون مانند قانون دوم است، اما با این تفاوت که پس از مشاهدهٔ توکن `,` مجدداً به تجزیهٔ **clist** می‌پردازیم.

انواع داده‌ای (data types) در زبان TesLang :

```
type    :=
1           Number      |
2           List        |
3           Null
```

1. نشان دهنده نوع داده‌ای از نوع اعداد صحیح است.
2. نشان دهنده نوع داده‌ای آرایه‌ای است که یک لیست از اعداد را در خودش نگه می‌دارد.
3. نوع داده‌ای تعریف نشده، بی مقدار و خالی را نشان میدهد. (برای توابع **void** نیز باید از همین نوع داده‌ای در مشخص کردن نوع خروجی تابع استفاده شود)

گرامر زبان TesLang

عبارات و اصطلاحات (expressions) زبان TesLang :

```
expr    :=
1           expr [ expr]          |
2       [*] [ clist ]             |
3       [*] expr ? expr : expr    |
4           expr + expr           |
5           expr - expr           |
6           expr * expr           |
7           expr / expr           |
8           expr % expr           |
9           expr > expr           |
10          expr < expr           |
11          expr == expr          |
12          expr >= expr          |
13          expr <= expr          |
14          expr != expr          |
15          expr || expr          |
16          expr && expr          |
17          ! expr                |
18          + expr                |
19          - expr                |
20          iden                  |
21          iden = expr           |
22          iden ( clist )        |
23          number
```

1. مربوط به خواندن یک خانه از آرایه است. expr سمت چپ باید از نوع List و expr سمت راست باید از نوع Number باشد.
2. روشی از ساخت یک آرایه می‌باشد. پیاده سازی این بخش نمرهٔ اضافه دارد.
3. نشان دهندهٔ یک ternary operator است که ابتدا صحیح بودن expr سمت چپ را بررسی کرده و در صورت درست بودن آن expr وسط را اجرا می‌کند و در غیر این صورت expr سمت راست را اجرا می‌کند. پیاده سازی این بخش نمرهٔ اضافه دارد.
4. تا قانون شمارهٔ 8. نشان دهندهٔ عملیات محاسباتی است (arithmetic operations).

نکته: برای پیاده سازی این بخش، باید رفع ابهام صورت گیرد تا از سمت چپ محاسبات دارای اولویت باشند. به ترتیب اولویت عملگرهای * / % از + - بیشتر است.

9. تا قانون شمارهٔ 14. نشان دهندهٔ عملیات مقایسه‌ای در زبان TesLang است.
15. و 16. نشان دهندهٔ عملیات short circuit هستند.

17. تا قانون شمارهٔ 19. برای عبارات یگانه (unary) است که میتواند قبل از آن از توکن های - + ! استفاده کرد.
20. یک شناسه (identifier) را برمی‌گرداند.
21. تساوی در زبان TesLang را نشان می‌دهد.
22. یک فراخوانی تابع (function call) را نشان می‌دهد.
23. یک عدد صحیح را برمی‌گرداند.