

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 متغیرهای گرامر می‌باشد.
- این رنگ  مربوط به کلمات کلیدی زبان می‌باشد.
- این رنگ  مربوط به شماره قاعده‌ها می‌باشد.
- علامت‌های **|** 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]*

string  :=
1       ' ' [^"\n]* ' ' |
2       " " [^'\n]* " "

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.

string: متشکل از مجموعه ای از **حروف** است که در میان دو علامت ' و یا " قرار میگیرند. پیاده سازی این بخش نمره اضافه دارد.

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

قاعده 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          def type iden ( flist ) { body } |  
2          [*] def type iden ( flist ) return expr ;
```

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

تابع با **کلمه کلیدی** [def](#) شروع خواهد شد و سپس نوع داده‌ای آن خواهد آمد و بعد از تعریف نام تابع، بین دو پرانتز [flist](#) قرار می‌گیرد.

در ادامه به بررسی بدنه تابع می‌پردازیم، به قانون اول رجوع می‌کنیم، در صورت **عدم برقراری** آن به قانون دوم رجوع می‌کنیم.

**** پیاده سازی قانون دوم نمره اضافه دارد.**

بدنه زبان TesLang :

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

گرامر زبان TesLang

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 = expr to 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.
پایاده سازی این بخش نمره اضافه دارد.
6. دستور مربوط به حلقه for را نشان می‌دهد که به صورت زیر یک مقدار اولیه به آن داده میشود (به عنوان اندیس شروع):

```
for (int i = 0 to length(numList)) {
    print(i);
}
```

7. این دستور مربوط به خروجی تابع است که با کلمه کلیدی return مشخص می‌شود و سپس یک مقدار به صورت expr در جلوی آن می‌آید.

گرامر زبان TesLang

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

def int funcA(int a) {
    def int funcB() {
        return a + 10;
    }

    return funcB();
}

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

تعریف متغیر (**defining variables**) در زبان **TesLang** :

```
defvar :=
1      var type iden      |
2      var type iden = 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 کلمه کلیدی var قرار دارد سپس نوع داده‌ای متغیر و بعد از آن نام متغیر مشخص می‌شود و در قانون دوم هم برای انتساب مقدار اولیه کافیسست یک = قرار بگیرد و بعد از آن به سراغ تجزیه expr باید رفت.

گرامر زبان TesLang

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

```
flist :=  
1      |  
2      type iden      |  
3      type iden, 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 دهنده یک آرگومان است، به این نحو که ابتدا **نوع** متغیر آمده و سپس بعد از آن نام 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 مشاهده توکن **_** مجدداً به تجزیه **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 دهنده یک ورودی است که پس از آن تجزیه **clist** خاتمه می‌یابد.
3. این قانون مانند قانون دوم است، اما با این تفاوت که پس از مشاهده توکن **,** مجدداً به تجزیه **clist** می‌پردازیم.

انواع داده‌ای (data types) در زبان TesLang :

```
type     :=  
1         int        |  
2         vector     |  
3         str        |  
4         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. نشان دهنده نوع داده ای رشته (**string**) است. **پیاده سازی این بخش نمره اضافه دارد.**
4. نوع داده‌ای **تعریف نشده**، بی مقدار و خالی را نشان می‌دهد. (برای توابع **void** نیز باید از همین نوع داده-ای 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
24         [*] string
```

1. مربوط به خواندن یک خانه از آرایه است. expr سمت چپ باید از نوع vector و expr سمت راست باید از نوع int باشد.
 2. روشی از ساخت یک آرایه می‌باشد. پیاده سازی این بخش نمره اضافه دارد.
 3. نشان دهنده 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 نمره اضافه دارد.
 4. تا قانون شماره 8. نشان دهنده 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.

گرامر زبان TesLang

9. تا قانون شماره 14. نشان دهنده عملیات مقایسه‌ای در زبان TesLang است.
15. و 16. نشان دهنده عملیات short circuit هستند.
17. تا قانون شماره 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.
24. یک رشته از حروف را برمی‌گرداند. پیاده سازی این بخش نمره اضافه دارد.