معرفی زبان TesLang

در این مستند با زبان ساده TesLang آشنا می‌شویم. در گام های تمرین عملی درس طراحی کامپایلر، بخش هایی از یک مترجم برای این زبان نوشته میشوند. قواعد این زبان در ادامه ی این مستند بیان میشود.

1. این زبان دارای دو نوع داده‌ای است.
   a. Number برای اعداد صحیح
   b. List برای آرایه ها

2. برنامه‌های این زبان در یک فایل نوشته می‌شوند و هر فایل شامل تعدادی تابع است. در این زبان متغیر سراسری (Global) وجود ندارد.

3. خط اول هر تابع شامل تعریف نام (identifier) آن تابع و ورودی و نوع خروجی آن است.

4. بدنه هر تابع بین دو توکن { و } قرار گرفته و شامل تعدادی عبارت (statement) است.

5. توکن : در دو موقعیت به کار می‌رود.
   a. بعد از تعریف نام متغیر (به منظور مشخص کردن نوع داده‌ای آنها)
   b. بعد از تعریف ورودی های توابع بعد از پرانتز (به منظور مشخص کردن نوع داده‌ای خروجی تابع)

6. شباهت ساختاری زبان TesLang مشابه JavaScript است.

7. هر block از کد در بین دو توکن { و } قرار می‌گیرد.

8. در هر block می‌توان کد نویسی کرد (تعریف متغیر، انتساب و ...). حوزهٔ (scope) هر block هم مانند زبان JavaScript تعریف می‌شود.

9. در حلقه for بعد از keyword مربوطهٔ آن (for) دو شناسه که به ترتیب برای اندیس و مقدار آن خانهٔ حافظه هستند تعریف می‌شوند.

10. کامنت ها در کد با توکن // تعریف میشوند. و از مکانی که این توکن استفاده شود تا پایان خط را نادیده می‌گیریم.

11. متغیر های محلی در هر block از کد، با کلمهٔ کلیدی **let** و به شکل زیر تعریف می‌شوند.

```
let a: Number = 10;          // a is a number with some initial value
let b: List = [1, 2, 3, 4]; // b is a list of numbers
let c: Number;               // c is a number with no initial value
let d: List;                 // d is a list with no initial value
```

12. مقدار خروجی تابع با استفاده از کلمهٔ کلیدی **return** مشخص می‌شود.

```
function sum(a: Number, b: Number): Number => {
      let result: Number = 0;
      result = a + b;
      return result;
}
```

13. در صورتی که تابع چیزی برنگرداند، نوع داده‌ای باید از نوع **Null** باشد.

14. همانطور که در مثال زیر می‌بینید، یک لیست را به عنوان ورودی به تابع می‌دهیم و عناصر آن را با هم جمع می‌کنیم.

```
function sum(numList: List): Number => {
    let result: Number = 0;

    for (index, value of numList) {
        result = result + value;
    }

    return result;
}
```

15. هر برنامهٔ TesLang شامل یک تابع main است که ورودی ندارد و خروجی آن یک عدد صحیح است که همان کد برگشتی (Return Code) برنامه است.

```
function main(): Number => {
      // main code
      return 0;
}
```

16. مثال زیر یک طرز استفاده از if را در این زبان نشان میدهد.

```
function factorial(n: Number): Number => {
    if (n < 2) {
        return 1;
    }

    return n * factorial(n-1);
}
```

17. جدول توابع داخلی TesLang :

تابع	توضیح
`read()`	یک عدد را از ورودی استاندارد می‌خواند و برمی‌گرداند.
`log()`	یک عدد را در ورودی استاندارد چاپ می‌کند.
`makeList(n)`	یک لیست با n عنصر برمی‌گرداند.
`length(arr)`	یک لیست (arr) بعنوان ورودی گرفته و طول لیست را برمی‌گرداند.
`exit(n)`	برنامه را با کد برگشتی داده شد (n) به پایان می‌رساند.

معرفی زبان TesLang

قواعد تجزیهٔ زبان TesLang

در ادامه، ساختارBNF زبان TesLang نمایش داده شده است. اولویت عملگرها در زبان TesLang مشابه اکثر زبان های برنامه نویسی ... ,JavaScript ,C ,Python است. چون در گرامری که در ادامه نمایش داده میشود اولویت عملگر ها مشخص نشده است، گرامر مبهم است. **توضیحات کامل گرامر در یک مستند دیگر ارائه خواهد شد.**

قواعدی که با علامت [*] مشخص شده اند به صورت اختیاری و دارای نمرهٔ اضافه هستند.

در قاعده شمارهٔ 2 از متغیر **func** یک تابع یک منظوره نمایش داده شده است (مانند این نوع توابع را در JavaScript می‌توان دید).

برای مثال در زبان TesLang یک تابع **sum** به شکل زیر نوشته شده است:

```
function sum(a: Number, b: Number): Number => a+b;
```

به گرامر این زبان توجه کنید :

```
prog    :=
1           |
2           func prog


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


body    :=
1           |
2           stmt body
```

```
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



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



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


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



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

```
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              |
17          + expr              |
18          - expr              |
19          iden                |
20          iden = expr         |
21          iden ( clist )      |
22          number


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



number  := [0-9]+


comment := //[^\n]\n
```

موفق و پیروز باشید