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. int برای اعداد صحیح
   b. vector برای آرایه ها
   c. str برای رشته ها (**نمره اضافه**)

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. شباهت ساختاری زبان TesLang مشابه C و Python است.

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

7. در هر block می‌توان کد نویسی کرد (تعریف متغیر، انتساب و ...).

8. در حلقه for بعد از keyword مربوطهٔ آن (**for**) یک شناسه که برای مشخص کردن نقطه شروع به آن نیاز داریم.

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

10. متغیر های محلی در هر block از کد، با کلمهٔ کلیدی **var** و به دنبال آن **نوع داده ای** آن متغیر، به شکل زیر تعریف می‌شوند.

```
var int a = 10;                 // a is a number with some initial value
var vector b = [1, 2, 3, 4];    // b is a list of numbers
var str c = "Hello, World!";    // c is a string
var int d;                      // d is a number with no initial value
var vector e;                   // e is a list with no initial value
var str f;                      // f is a string with no initial value
```

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

```
def int sum(int a, int b) {
      var int result = 0;
      result = a + b;
      return result;
}
```

12. در صورتی که تابع چیزی برنگرداند، نوع داده‌ای باید از نوع **null** باشد.

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

```
def int sum(vector numList) {
    var int result = 0;

    for (i = 0 to length(numList)) {
        result = result + numList[i];
    }

    return result;
}
```

14. 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.

```
def int main() {
      // main code
      return 0;
}
```

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

```
def int factorial(int n) {
    if (n < 2) {
        return 1;
    }

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

16. جدول توابع داخلی TesLang :

تابع	توضیح
scan()	یک عدد را از ورودی استاندارد می‌خواند و برمی‌گرداند.
print()	یک عدد را در ورودی استاندارد چاپ می‌کند.
list(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:

```
def int sum(int a, int b) return a+b;
```

به گرامر این زبان توجه کنید :

```
prog    :=
1           |
2           func prog


func    :=
1           def type iden ( flist ) { body }    |
2       [*] def type iden ( flist ) return 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 = expr to expr ) stmt        |
7           return expr ;                           |
8           { body }                                |
9           func



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



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


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



type    :=
1           int             |
2           vector          |
3       [*] str             |
4           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 && expr        |
    17          ! expr              |
    18          + expr              |
    19          - expr              |
    20          iden                |
    21          iden = expr         |
    22          iden ( clist )      |
    23          number              |
    24      [*] string


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


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


    number  := [0-9]+



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

موفق و پیروز باشید