پروژه شماره ۱ درس نظریه زبان‌ها و ماشین‌ها

گرامر نویسی با ANTLR

شرح پروژه:

در این پروژه هدف پیاده‌سازی ساختاری مشابه زبان برنامه‌نویسی جاوا اسکریپت و رسم درخت کدهای نوشته‌شده به این زبان است.

ساختار زبان:

1. هر برنامه شامل یک یا چند دستور import است. دستورات import در ابتدای برنامه می‌آیند.
2. هر کلاس می‌تواند شامل توابع و تعریف متغیر باشد.
3. در انتهای هر دستور باید semicolon (;) قرار داده شود.
4. بلاک‌ها با { و } تعریف می‌شوند.
5. قوانین نام‌گذاری:
   a. حداقل شامل دو کاراکتر هستند.
   b. با رقم یا '_' یا '$' شروع نمی‌شوند.
   c. متشکل از حروف کوچک و بزرگ لاتین، ارقام و کاراکترهای '**$**' و '_' هستند.
   d. کلمات مشخص شده به صورت **bold** در این فایل، کلمات کلیدی هستند و نمی‌توانند نام متغیر باشند.
6. کامنت‌گذاری به دو صورت single-line و multi-line تعریف می‌شود و کاراکتر کامنت single line // و برای multi line به صورت /*comments*/ می‌باشد.. خطوط کامنت **نباید** در درخت ترسیم شوند.

در ادامه، ساختار بعضی از دستورات و بلوک‌ها به همراه مثال آورده شده است. در موارد زیر، قسمت‌های درون [ ] دلخواه هستند و ممکن است در کد مربوط به آن آمده باشند یا نه.

- Import کردن:

```
$import “math”;
$import defaultExport from "module-name";
$import * as name from "module-name"; (* means all)
$import name1, name2  from "module-name";
```

- تعریف متغیر:

```
[$let/ var/ const] <name> [= <initial_value>];
---
xy;
$var xy = 3;
$let xy , wz = 3 , 5;
$const xy = wz + cu;
```

اعداد هنگام انتساب اولیه می‌توانند به صورت نماد علمی نیز وارد شوند (برای مثال: 1.1209e-19 یا 047.) دقت کنید که نماد علمی حداکثر یک رقم قبل از ممیز دارد.

- دستورات حلقه:

```
for ( [<type> <initialization>]; <conditions> ; <inc/dec>){
        <code>
}
```

```
// for loop examples
for (var myVar = 0; myVar < count || count > 5; myVar++) {
        sum += myVar;
}
```

```
for <variable_name> in <iterator_name>{
        <code>
}
```

```
// iterative for example
for p in myList {
        newList.add(p.name);
}
```

```
for <variable_name> of <iterator_name>{
        <code>
}
```

```
//example:
```

```
for p of users {
        console.log("hello");
}
```

```
while <conditions> {
        <code>
}
```

```
do {
        <code>
} while (<conditions>);
```

- دستورات شرط:

```
if (<conditions>) {
        <code>
}
else if (<conditions>) {
        <code>
}
else {
        <code>
}
```

```
//ternary expression
```

```
<conditions> ? <expression> : <expression>;

---

myFaveWeather = isSummer ? "Sunny" : "Snowy"; // used in assignments

...

return isSunny ? 25 : (temperature / 2);          // or in any other expression
```

- :Switch/Case

```
switch/ match <expression> {
        case <value> :
                <code>
                [break]
        [ default:
                <code>
                [break]
        ]
}

---

// example
switch month.name {
        case "Jan":
                console.log("it's January");
                break;
        case "Feb":
```

```
        case "Dec":
                console.log("close enough");
                break;
        default:
                console.log ("try again");
    }
```

- تعریف کلاس:

```
class <class-name> {
        constructore( [<parameter_list>] ){
                <code>
        }
}
```

```
//example
class Car {
        constructor(name, year) {
                this.name = name;
                 this.year = year;
          }
}
```

*کلمه **this** یک کلمه کلیدی می‌باشد.

- تعریف تابع:

```
function <function_name> ( [<parameter_list>] ) {
        <code>
        [return] <expression>;
}
```

```
//example
function divide (int num1, int num2) {
        $let result;
        if !check_zero(num2) {
                return Null;
        }
        result = num1 / num2;
        return result;
}
```

```
//Arrow functions
[$let/ var/ const] <name> = ([<parameter_list>]) => <expression>;
```

```
//example
$let myFunction = (a, b) => a * b;
```

- Exceptions:

```
try {
        <code>
}
Catch( [<err>] ) {
        <code>
}
finally {
        <code>
}
```

```
//example
try {
        res = num1 / num2;
}
catch(exeption E)  {
        console.log( "oops.");
}
finally {
        Console.log("test");
}
```

- عملگرها و اولویت: برنامه شما باید بتواند عملگرهای زیر را با اولویت‌بندی داده شده تشخیص دهد.

1. ( )
2. **
3. ~
4. - + (unary, e.g. **-a**)
5. ### (unary operator, e.g. **a++** or **++a**)
6. * / // %
7. - + (binary, e.g. **a - b**)
8. << >>
9. & ^ | (bitwise operators)
10. === == != <>
11. < > <= >=
12. ! || &&
13. = ##= (e.g. //= or +=)

توضیحات تکمیلی:

- قسمت‌های مبهم زبان را تا حد منطقی، می‌توانید خود تعریف و پیاده‌سازی کنید.
- در کنار گرامر خود، حتماً یک یا چند نمونه برنامه در زبانی که برای آن گرامر نوشته‌اید قرار دهید تا قابلیت‌های مختلف گرامرتان را نمایش دهد. در صورت عدم وجود تست‌کیس، نمره کسر خواهد شد.
- فقط فایل گرامر (.g4) و نمونه برنامه‌های خود (فایل‌های txt. یا مشابه آن) را در قالب یک فایل zip با نام **StudentID_Antlr** ارسال کنید.
- در هنگام تحویل پروژه، لازم است به گرامر خود تسلط کافی داشته‌باشید و در صورت نیاز، بتوانید تغییراتی در آن ایجاد کنید.
- انجام پروژه به صورت انفرادی است. در صورت مشاهده هرگونه تخلف، نمره پروژه فرد یا افراد، معادل ۱۰۰- خواهد بود.

- مهلت تحویل: 16 اردیبهشت، ۲۳:۵۹

موفق باشید
تیم حل‌تمرین