


پروژه انتلر درس نظریه زبان‌ها و ماشین‌ها

شرح پروژه:

در این پروژه هدف پیاده سازی ساختار کلی یک زبان ساده برنامه نویسی و رسم درخت کدهای نوشته شده به این زبان است.

ساختار زبان:


1. هر برنامه شامل یک یا چند کلاس و دستوراتی برای اضافه کردن کتابخانه هاست. این دستورات پیش از تعریف کلاس ها می آیند.
2. هر کلاس میتواند شامل توابع و تعریف متغیر باشد.
3. اولین تابع هر کلاس constructor آن است.
4. کامنت گذاری به دو صورت single-line و multi-line تعریف میشود و کاراکتر مشخص کننده این کامنت ها را میتوانید به دلخواه مشخص کنید. خطوط کامنت نباید در درخت ترسیم شوند
5. کلمات کلیدی case-sensitive نیستند.
6. قوانین نامگذاری:
 - با رقم شروع نمیشوند.
 - متشکل از حروف کوچک و بزرگ لاتین، ارقام و کاراکترهای '\$' و '_' هستند.
 - کلمات مشخص شده به صورت bold و با رنگ قرمز در این فایل، کلمات کلیدی هستند و نمیتوانند نام متغیر باشند. بخش هایی که با رنگ زرد نمایش داده شده اند وجود آنها اختیاری است.
 - حرف اول نام کلاس باید capital باشد.

دستورات اضافه کردن کتابخانه‌ها: 

```
Import : : from ( <library_name> . <class_name> , <library_name> . <class_name> )  
require ( <library_name> , <library_name > )
```

Example:


```
import : : from ( math. BigInteger , random. Float )  
require ( math , random )
```

تعريف كلاس: 

```
Public | private CLASS <class_name> Inherited from <parent_class_name> ,<parent_class_name>  
    constructor <class_name> <- function ( <parameter_list> ) :  
        <statements>  
    end  
end_Class
```

Example:

```
public class Dot Inherited from Plottable, Movable  
    int x, y  
    CoNstructor Dot <- function ( int px , int py ) :  
        x= px  
        y=py  
    end  
end_class
```

تعريف توابع: 

```
<function_name> <- Public | private function ( <parameter_1_type> : <parameter_1_name> ,  
<parameter_2_type> : <parameter_2_name> ) :  
    <statements>  
end
```

Example:

```
myfunction <- function() :  
    @print(" hi ")  
end
```

فراخوانی توابع:

```
@ <function_name> ( <variable_name> , <value> )
```

Example:

```
@print ( " hi " )
```

دستورات حلقه :

```
while [ <conditions> ] do <statements> done  
do <statements> as_long_as [ <conditions> ]  
for (( <parameter_1_type> <initialization> ; <conditions> ; <inc | dec> )) do <statement> done
```

Example:

```
while [ true ] do  
    i++  
done  
  
do k -- as_long_as [ k > 0 ]  
  
for (( int index =0 ; index != 20 ; index ++ ))  
do  
    <statements>  
done
```

دستورات switch_case :

```
Case <expression> :  
when <value>  
    <statements>  
when <value>  
    <statements>  
else
```

<statements>

End

Example:

case fruit

when "apple"

 @print("This is an apple")


when "banana"

 @print("This is a banana")

else

 @print("I don't know what this fruit is")

end

دستورات شرط: 

if [<expression>] :

then

 <statements>

else

 <statements>

Fi

Example:

If [index == 0]


Then

 @print("equal")

Else

 @print(" not equal")

Fi

تعریف متغیر: 


Public | private const < datatype > < name > <= < initial value >

Example:

```
const string myConst <= "Lorem Ipsum" // defining constants
```

```
private int myVar <= 25 // defining integer
```

```
private bool flag
```

تعریف آرایه: 


Public | private const < datatype > < name > **new []** <= < datatype > [<length>]

Public | private const < datatype > < name > **[]** <= [<value> , <value> , <value> , ...]

Example:

```
int myArray new [] <= int[4] // array definition
```


```
float arr [] <= [0.05, 42, 42.25, 43] // my Initiated Array
```

تعریف ارجاعات: 

Public | private < class_name > < reference_name > <- **new** < class_name > [<parameters>]

Example:

```
Circle circle <- new Circle [ x, y ]
```

انواع عملگرهای زبان: 

انواع عملگرهای مورد استفاده در زبان که باید تعریف شده و به ترتیب ذکر شده زیر شناسایی و مورد

استفاده قرار گیرند، عبارتند از :

- ()
- ^ (توان)
- عملگرهای یگانه :

این عملگرها فقط روی یک عملوند، عمل میکنند و دقت داشته باشید که نباید میان عملوند و این عملگرها ،

فاصله ای وجود داشته باشد.

#++	○
#--	○
++#	○
--#	○
-	○
(شيفت) >>	<< ●
*	/ // % ●
-	+
&&	●
and	& ●
	●
or	●
!=	== ●
=<	=> ●
<	> ●
=+	=- ●
=*	=/ ●

:Exceptions 

try:

<statements>

except <variable_name> , <variable_name> :

<statements>

Example:

try:

except ValueError, Argument :

@print ("Error ",err)

توضیحات تکمیلی:

- ❖ پیاده سازی موارد ذکر شده در این فایل برای تحویل پروژه کافی است. اما در صورتی که موارد دیگر و بیشتری به طور منطقی و منطبق و مرتبط با موارد ذکر شده به ساختار این زبان اضافه کنید، تا حد امکان نمره اضافه برای شما لحاظ خواهد شد و همینطور تمیزی کد شما مشمول نمره اضافه خواهد شد.
- ❖ در کنار گرامر خود، حتماً یک یا چند نمونه برنامه در زبانی که برای آن گرامر نوشته اید قرار دهید تا قابلیت های مختلف گرامرتان را نمایش دهد. در صورت عدم وجود تست کیس، نمره کسر خواهد شد.
- ❖ فقط فایل گرامر (.g4) و نمونه برنامه های خود (فایل های .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 داشته باشید و در صورت نیاز، بتوانید تغییراتی در آن ایجاد کنید.
- ❖ هم چنین لازم به ذکر است پروژه باید به صورت انفرادی انجام شود و در صورت دیده شدن هر گونه تخلف و تقلب، برای فرد یا افراد مربوطه نمره 100- ثبت خواهد شد.

موفق باشید