

## بسمه تعالی

گزارش فاز اول پروژه ی کامپایلر

محمدیاسر حاجیان-مرتضی نوروزی

98243017 - 98243066

در فاز اول پروژه ، به کمک jflex یک Scanner ساختیم که بتواند Token های این زبان که ما تعریف کرده ایم را شناسایی کند و آن ها را طبق رنگ و استایل مشخص شده Highlight کند.

ابتدا option های مورد نیاز را تعریف کرده ایم. اسم کلاس موجود در فایل اسکنر ما My\_Scanner میباشد که public است. همچنین اسم تابع اصلی برنامه nextToken میباشد که خروجی آن یک object از کلاس Token که خودمان تعریف کرده ایم میباشد.

در قسمت بعدی ، ابتدا کلاس توکن را تعریف کرده ایم که شامل سه ویژگی type و value و lineNumber است که به ترتیب type نوع آن Token است که یکی از انواع مشخص شده در صورت پروژه است و value مقدار آن توکن است که توسط yytext خوانده میشود و lineNumber شماره ی خط آن توکن میباشد.

همچنین دو متغیر ICV و RCV برای نگه داری اعداد صحیح و اعشاری تعریف می کنیم.

سپس رجکس ها را تعرف می کنیم که بعدا در استیت ها از آنها استفاده کنیم.

➤ رجکس ها :

ابتدا دو رجکس digit و letter را تعریف کرده ایم که بترتیب شامل ارقام و حروف است .

سپس به رجکس اعداد میرسیم.

رجکس decimal integer باید شامل یک یا بیشتر رقم باشد.

رجکس hexadecimal با صفر شروع میشود سپس یک x یا X بیاید و بعد از آن باید از بین ۰-۹ یا a-f یا A-F یک یا بیشتر بیاید.

رجکس Real Number با یک یا بیشتر رقم شروع می شود و سپس یک نقطه می آید و در آخر می تواند صفر یا بیشتر رقم بیاید.

رجکس scientific Notation با یک decimal integer یا real number شروع میشود سپس e یا E می آید سپس علامت مثبت و منفی میتواند بیاید هم میتواند نیاید که اگر نیاید توان مثبت در نظر گرفته میشود و سپس یک یا بیشتر رقم در آنها باید بیاید.

سپس رجکس special char که ابتدا با یک بک اسلش شروع میشود و بعد از آن باید یکی از حروف n یا t یا r یا \ یا ' یا " بیاید و این شش حالت کاراکترهای خاص ما هستند که داخل استرینگ خوانده می شوند.

اپراتور ها هم به صورت بسیار ساده به چند دسته بندی تقسیم کردیم

دسته اول آنهایی که با مساوی تمام میشوند. >= , == , /= , -= , +=

دسته دوم دسته دوتایی ها : && , || , -- , ++

دسته سوم دسته تک حرفی ها : ^ , & , | , % , .....

رجکس identifier هم بسیار ساده با یک حرف شروع می شود و در ادامه میتواند ۰ تا ۳۰ حرف یا رقم یا underline بیاید.

رجکس line terminator هم واضح است.

رجکس input characters هم برابر هرچیزی به جز line terminator است.

رجکس white space هم علاوه بر line terminator و space و tab نیز دارد.

برای کامنت هم دو دسته تعریف می کنیم

دسته اول چند خطی که با /\* شروع و با \*/ تمام میشود.

دسته دوم تک خطی که با // شروع و تا زمانی که line terminator خوانده نشود به اتمام نمی رسد.

و اجتماع آنها را در یک رجکس به عنوان کامنت ذخیره میکنیم.

سپس reserved word ها را به ترتیب نوشته و اجتماع شان را در رجکس مربوط به آن ذخیره میکنیم.

بعد از رجکس ها به سراغ دستورات و استیت ها می رویم.

به این صورت که برای هر دسته خاص از رجکس ها که تعریف کردیم یک حالت در نظر میگیریم و برای هر دسته توکن مربوطه را return می کنیم و همینطور ادامه میدهیم و برای بعضی حالات مقادیر خاصی مانند ICV و RCV را نیز محاسبه می کنیم.

اما برای استرینگ از استیت اصلی YYINITIAL خارج می شویم و به یک استیت دیگر می رویم و پس از اتمام کار به استیت اصلی برمیگردیم.

مثلا استرینگ که با خواندن استرینگ لیترال به استیت استرینگ حرکت می کند و بقیه توکن ها را تا زمانی که استرینگ لیترال بعدی را بخواند داخل استیت استرینگ میخواند که می تواند white space, special char, string باشد و با خواندن استرینگ لیترال دوم به استیت اصلی برمیگردد. و مانند زبان C استرینگ اگر در یک خط بسته نشود خط های بعدی نیز به عنوان استرینگ شناخته می شوند تا زمانی که در یک خط به استرینگ لیترال دوم برسیم.

در نهایت هم اگر Token ای داشته باشیم که در هیچ کدام از استیت ها و حالت ها پذیرش نشود ، به عنوان ارور شناسایی شده و شماره خط و شماره ی کاراکتر آن چاپ میشود و در فایل output به رنگ قرمز خواهد بود.

در فایل Main.java ابتدا یک ArrayList برای ذخیره ی توکن ها تعریف کرده ایم. سپس از روی فایل اسکنر که خودمان طراحی کردیم ، یک اسکنر میسازیم و فایل input.cool را به آن پاس میدهیم. سپس به کمک حلقه ی while تا وقتی که به پایان فایل ورودی نرسیده ایم ، تابع nextToken را فراخوانی میکنیم و توکن خوانده شده را به ArrayList اضافه میکنیم. بعد از پایان حلقه ، متد highlight را فراخوانی میکنیم و ArrayList توکن ها و FileWriter ساخته شده که قرار است در فایل Output.html بنویسد را به آن پاس میدهیم.

درون این متد ، ابتدا یک HashMap ساخته ایم که key آن یک عدد صحیح است که در واقع شماره ی خطوط فایل ورودی خواهد بود و value آن یک ArrayList از توکن هاست که در واقع توکن هایی هستند که در آن خط هستند. به این ترتیب ما توانستیم توکن هایی که در فایل ورودی در یک خط هستند را در فایل خروجی هم در یک خط چاپ کنیم. در حلقه ی اول ، ما روی ArrayList توکن های پیمایش میکنیم و با توجه به شماره ی خط هر توکن ، آن را در ArrayList مربوط به آن شماره ی خط اضافه میکنیم.

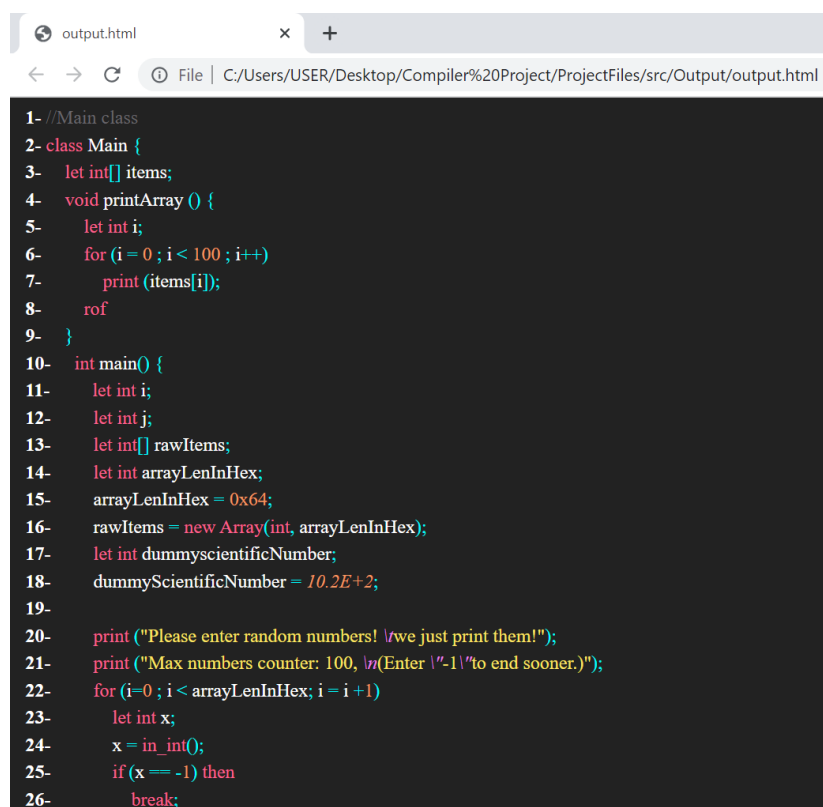
در فایل Output.html کد های مربوط به HTML از قبل نوشته شده و کد هایی که ما اضافه میکنیم به قبلی ها append میشود . در حلقه ی دوم ، ما روی کلید های هش مپ پیمایش میکنیم. مقدار هر کلید ، یک ArrayList شامل توکن های موجود در آن خط است. ابتدا یک رشته تعریف میکنیم که مقدارش یک تگ p میباشد . بعد شماره ی آن خط را در یک تگ span گذاشته و به آن رشته اضافه میکنیم. سپس با پیمایش روی

ArrayList مربوط به آن خط ، هر توکن را در یک تگ span قرار میدهم و کلاس آن تگ را برابر type آن توکن قرار میدهم. استایل مربوط به کلاس های مختلف فایل HTML در فایل CSS در پوشه ی output قرار دارد که استایل مشخص شده در فایل صورت پروژه را به توکن ها میدهد. پس از اتمام حلقه ی داخلی ، یک خط از ورودی به طور کامل خوانده شده است. حال تگ p را میبندیم و این خط را به fileWriter در فایل مینویسیم. پس ساختار کلی به این صورت است که هر خط ورودی ، یک تگ p است که شماره ی خط و توکن های موجود در آن خط ، هر کدام در یک تگ span به صورت جداگانه درون آن تگ p قرار میگیرند.

بعد از اتمام حلقه ی خارجی ، کل توکن ها در فایل خروجی نوشته شده اند و الان کافی است که تگ های body و html بسته بشوند و آن ها هم به انتهای فایل خروجی اضافه شوند.

به این ترتیب ، در فایل Output.html تمام توکن ها با استایل و رنگ مشخص شده نوشته شده اند ، همینطور از آنجایی که ما با شناسایی whitespace ، یک توکن با همین type را برمیگردانیم و هنگام نوشتن در فایل خروجی ، آنجا هم به ازای آن توکن ، whitespace در فایل خروجی مینویسیم ، اینتدنت های موجود در فایل ورودی ، در فایل خروجی هم وجود دارد و همینطور توکن هایی که در فایل ورودی در یک خط هستند ، در فایل خروجی نیز در یک خط خواهند بود.

تصاویر پایین فایل output.html میباشد.



```
1- //Main class
2- class Main {
3-     let int[] items;
4-     void printArray () {
5-         let int i;
6-         for (i = 0 ; i < 100 ; i++)
7-             print (items[i]);
8-         rof
9-     }
10-    int main() {
11-        let int i;
12-        let int j;
13-        let int[] rawItems;
14-        let int arrayLenInHex;
15-        arrayLenInHex = 0x64;
16-        rawItems = new Array(int, arrayLenInHex);
17-        let int dummyscientificNumber;
18-        dummyScientificNumber = 10.2E+2;
19-
20-        print ("Please enter random numbers! \nwe just print them!");
21-        print ("Max numbers counter: 100, \n(Enter \"-1\" to end sooner.)");
22-        for (i=0 ; i < arrayLenInHex; i = i +1)
23-            let int x;
24-            x = in_int();
25-            if (x == -1) then
26-                break;
```

```
output.html x +
File | C:/Users/USER/Desktop/Compiler%20Project/ProjectFiles/src/Output/output.html
11- let int i;
12- let int j;
13- let int[] rawItems;
14- let int arrayLenInHex;
15- arrayLenInHex = 0x64;
16- rawItems = new Array(int, arrayLenInHex);
17- let int dummyscientificNumber;
18- dummyScientificNumber = 10.2E+2;
19-
20- print ("Please enter random numbers! \nwe just print them!");
21- print ("Max numbers counter: 100, \n(Enter \"-1\" to end sooner.)");
22- for (i=0 ; i < arrayLenInHex; i = i +1)
23-     let int x;
24-     x = in_int();
25-     if (x == -1) then
26-         break;
27-     else
28-         rawItems [i] = x;
29-     fi
30- rof
31- printArray ();
32-
33- //Undefined Token (for example a single backslash "\" is undefined)
34- \
35- }
36- }
```