گزارش

امین ساوه دورودی 98243032

محمدرضا اسکینی 98243006

در فاز اول پروژه, پیاده سازی اسکنر تنها در جهت شناسایی توکن های تعریف شده از ما خواسته شده است. برای پیاده سازی syntax highlighter توضیح داده شده در پروژه, ابتدا نیاز است با استفاده از ابزار jflex, موارد موجود در پروژه را با استفاده از عبارات منظم و کلیدواژه های تعریف شده در مستندات jflex را تعریف کنیم. تا با کمک این ابزار بتوانیم, فایل جاوا اسکنر کامپایلر را بسازیم.

ابتدا یک فایل با پسوند flex میسازیم و طبق مستندات jflex, به تعریف موارد مورد نیاز میپردازیم.

```
package com.compiler;

%%

%class CoolScanner
%public
%unicode
%line
%column
%function nextToken
%type Token

%{
    private Token token(String token, Type type) {
        return new Token(token, type);
    }
%}
```

در این بخش ابتدا پکیجی که در تمامی این پروژه استفاده میشود, تعریف شده است که بعد از تبدیل به فرمت جاوا, در بالای کد اسکنر قرار گیرد. همچنین اسم کلاس اسکنر ما نیز CoolScanner تعریف شده است. برای دسترسی به توکن بعدی نیز از nextToken استفاده میکنیم. همچنین توکن ها در قالب کلاس Token برگردانده میشوند که این کلاس نیز پیاده سازی شده است که شامل دو Property String (مقدار توکن خوانده شده) و Type (نوع توکن) میباشد. Type نیز Enum ای است که پیاده سازی شده است تا نوع هر توکن مشخص شود.

در گام بعدی نیاز است ماکرو هایی با استفاده از عبارات منظم تعریف شوند تا در استیت ها با توجه به توکن های خوانده شده, بتوانیم آن ها را مدیریت کنیم و نوع توکن ها را مشخص کنیم.

در ابتدا طبق پروژه نیاز است, کلمات رزرو شده تعیین شوند.

```
// Reserved Keywords
Let = "let"
Void = "void"
Int = "int"
Real = "Real"
Bool = "bool"
String = "string"
Static = "static"
Class = "class"
For = "for"
Rof = "rof"
Loop = "loop"
Pool = "pool"
While = "while"
Break = "break"
Continue = "continue"
If = "if"
Fi = "fi"
Else = "else"
Then = "then"
New = "new"
Array = "Array"
Return = "return"
In_String = "in_string"
In_Int = "in_int"
Print = "print"
Len = "len"
ReservedKeywords = {Let} | {Void} | {Int} | {Real} | {Bool} | {String} | {Static} | {Class} |
{For} | {Rof} | {Loop} | {Pool} | {While} |{Break} | {Continue} | {If} | {Fi} | {Else} | {Then} |
 {New} | {Array} | {Return}|{In_String}|{In_Int}|{Len}|{Print}
```

از آنجا که کلمات رزرو شده تنها یک string هستند, پس هر ماکرو را به صورت بالا تعریف میکنیم و در آخر هرکدام از آنها میتوانند یک نوع Reserved Keyword باشند.

در گام بعدی نیاز است که identifiers مشخص شود. Identifires با حرف شروع میشوند و میتوانند شامل حروف و اعداد و خط فاصله (_) باشند. حروف به کار رفته نیز حساس به بزرگی و کوچکی هستند. همچنین محدودیت 31 کاراکتری نیز دارند. با استفاده از ماکرو های تعریف شده برای ارقام و حروف و خط فاصله, میتوانیم این ماکرو را به صورت زیر تعریف کنیم. همچنین محدودیت 31 کاراکتری نیز در آخر عبارت منظم گذاشته شده است.

```
//Identifires
Identifiers = {Letter} ({Underscore} | {Letter} | {Digit}) {0, 30}
```

در گام بعدی نیاز است ماکرو مربوط به اعداد صحیح, هگزا, اعشاری, نماد علمی تعریف شود.

با استفاده از عبارات منظم و به کمک متن پروژه, ماکرو های بالا را پیاده سازی میکنیم.

```
// Numbers
DecimalInteger = {Digit}+
HexaDecimal = "0""x"|"X"({Digit}| [a-fA-F])+
RealNumber = {Digit}+"."{Digit}*
ScientificNotation = ({RealNumber} | {DecimalInteger})("e"|"E")("+"|"-"){DecimalInteger}
```

در گام بعدی, مطابق با متن پروژه, به پیاده سازی کامنت ها به دو صورت یک خطی و چند خطی میپردازیم.

```
//Comment Section
MultiLineComment = "/*" [^*] ~"*/" | "/*"~"*/"
LineComment = "//" {InputCharacter}* {LineTerminator}?   //find all things after "//" expect enter or end
Comment = {LineComment} | {MultiLineComment}
```

کامنت یک خطی, ابتدا با // شروع میشود و تا آخر خط را نیز در برمیگیرد.

کامنت چند خطی, با /* شروع میشود و تا */ ادامه دارد.

با استفاده از عبارات منظم, ماکرو مربوط به کامنت نیز پیاده سازی شده است.

در گام بعدی, طبق جدول داده شده در متن پروژه , ماکرو عملگرها و علائم را تعریف میکنیم.

```
OperatorsAndPunctuation = {add} | {unaryminus} | {production} |  {division} | {additionassignment} | {subtractionassignment} |
                          {productionassignment} | {divisionassignment} | {increment} | {decrement} | {less} | {greater} |
                          {greaterequal} |{lessequal} | {notequal} | {equal} | {assignment} | {mod} | {logicaland} | {logicalor} | {bitwiseand}
                          {bitwiseor} | {stringliteral} |{bitwisexor} | {not} | {dot} | {colon} | {semicolon} | {openingbraces} |
                          {closingbraces} | {openingparenthesis} | {closingparenthesis} | {openingcurlybraces} | {closingcurlybraces}
```

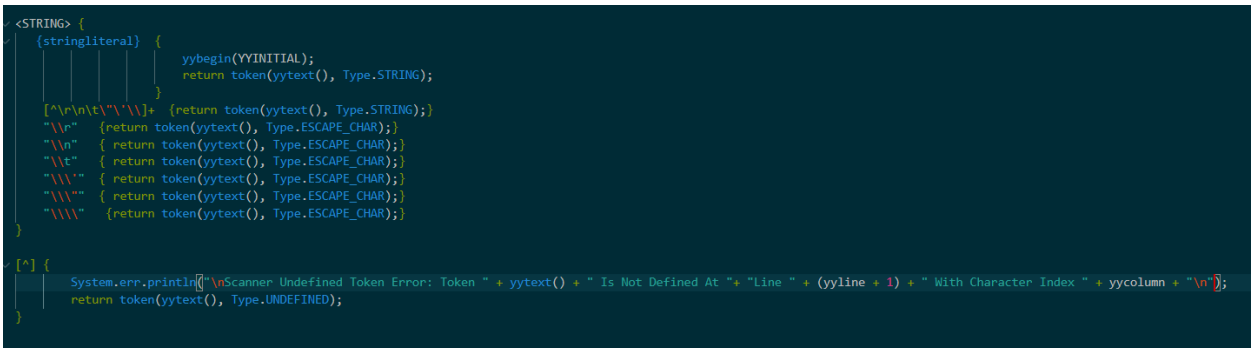
در گام بعدی با استفاده از ماکرو های تعریف شده, نیاز داریم استیت ابتدایی مان را کامل کنیم.

استیت ابتدایی به ما توکن خوانده شده را می دهد و سپس ما با استفاده از ماکرو های تعریف شده, نوع آنها را شناسایی میکنیم و به برنامه اصلی مان به صورت Token بر میگردانیم.

```
<YYINITIAL> {
    {StringLiteral} {
        yybegin(STRING);
        return token(yytext(), Type.STRING);
    }
    {Comment} {
        return token(yytext(), Type.COMMENT);
    }
    {ReservedKeywords} {
        return token(yytext(), Type.RESERVED_KEYWORD);
    }
    {OperatorsAndPunctuation} {
        return token(yytext(), Type.OPERATORS_AND_PUNCTUATION);
    }
    {Identifiers} {
        return token(yytext(), Type.IDENTIFIERS);
    }
    {DecimalInteger} {
        //ICV = Integer.valueOf(yytext());
        return token(yytext(), Type.INTEGER_NUMBER);
    }
    {RealNumber} {
        //RCV = Float.valueOf(yytext() + "f");
        return token(yytext(), Type.REAL_NUMBER);
    }
    {HexaDecimal} {
        //String absoluteStringValue = yytext().substring(yytext().indexOf("0") + 2);
        //ICV = Integer.parseInt(absoluteStringValue, 16);
        return token(yytext(), Type.HEX);
    }
    {ScientificNotation} {
        //RCV = Float.valueOf(yytext() + "f");
        return token(yytext(), Type.SCIENTIFIC_NOTATION);
    }
    {WhiteSpace} {
        return token(yytext(), Type.WHITESPACE);
    }
    [^] {
        System.err.println("\nScanner Undefined Token Error: Token " + yytext() + " Is Not Defined At "+ "Line " + (yyline + 1) + " With Character Index " + yycolumn + "\n");
        return token(yytext(), Type.UNDEFINED);
    }
}
```

از آنجا که در این فاز, تنها شناسایی نوع توکن های خوانده شده توسط اسکنر از ما خواسته شده است, پس نیازی به تعریف ICV و RCV و... نیست و در فازهای بعدی از آنها استفاده خواهد شد.

از آنجا که طبق متن پروژه هر String با " شروع و پایان می یابد و هر String میتواند شامل Escape Char نیز باشد, پس نیاز به تعریف یک استیت جدید برای مدیریت String میباشد. به طوری که با خواندن " String آغاز میشود و وارد استیت جدید میشویم و در آنجا به خواندن توکن ها می پردازیم تا دوباره به " برسیم که نشان دهنده انتهای String میباشد. با خواندن انتهای String, دوباره به استیت اصلی برمیگردیم و به خواندن و شناسایی بقیه توکن ها می پردازیم.



```
<STRING> {
   {stringliteral}  {
                        yybegin(YYINITIAL);
                        return token(yytext(), Type.STRING);
                    }
    [^\r\n\t\"\'\\]+  {return token(yytext(), Type.STRING);}
    "\\r"   {return token(yytext(), Type.ESCAPE_CHAR);}
    "\\n"   { return token(yytext(), Type.ESCAPE_CHAR);}
    "\\t"   { return token(yytext(), Type.ESCAPE_CHAR);}
    "\\\'"  { return token(yytext(), Type.ESCAPE_CHAR);}
    "\\\""  { return token(yytext(), Type.ESCAPE_CHAR);}
    "\\\\"   {return token(yytext(), Type.ESCAPE_CHAR);}
}

[^] {
        System.err.println("\nScanner Undefined Token Error: Token " + yytext() + " Is Not Defined At "+ "Line " + (yyline + 1) + " With Character Index " + yycolumn + "\n");
        return token(yytext(), Type.UNDEFINED);
}
```

سپس نیاز است که این کد توسط jflex به کلاس اسکنر (CoolCompiler) با پسوند جاوا تبدیل شود. ابتدا نیاز است jflex.jar دانلود شود.

سپس با استفاده از دستور java -jar jflex-full-1.8.2.jar Scanner.flex آن را تبدیل به فایل جاوا کنیم.

برای دسترسی به برخی از پراپرتی های کلاس اسکنر نیاز است آنها را به صورت دستی به صورت public قرار دهیم تا بتوانیم در کلاس های دیگر از آنها استفاده کنیم. به طور مثال برای دسترسی به شماره خط توکن خوانده شده (yyline), نیاز است در کلاس اسکنر, آن را public کنیم.

سپس در هر کلاسی از پکیج مورد نظر, میتوانیم از کلاس اسکنر و متد ها و Property های آن استفاده کنیم.

موارد پیاده سازی شده (کلاس Token و enum Type) نیز به صورت زیر می باشد.

```java
package com.compiler;
public enum Type {
    STRING,
    COMMENT,
    RESERVED_KEYWORD,
    OPERATORS_AND_PUNCTUATION,
    IDENTIFIERS,
    INTEGER_NUMBER,
    REAL_NUMBER,
    HEX,
    SCIENTIFIC_NOTATION,
    WHITESPACE,
    UNDEFINED,
    ESCAPE_CHAR,
}
```

```java
package com.compiler;
public class Token {
    private final String token;
    private final Type type;

    public Token(String token, Type type) {
        this.token = token;
        this.type = type;
    }

    public String getToken() {
        return token;
    }

    public Type getType() {
        return type;
    }
}
```

همانطور که در بالاتر هم توضیح داده شد، یک کلاس Type اضافه کردیم، که از این کلاس برای تشخیص نوع واژه دریافتی استفاده شده است. که با استفاده از همین Type میتوانیم تشخیص دهیم که از چه استایلی برای نمایش آن استفاده کنیم. درباره چگونگی استفاده، و نحوه تولید خروجی در ادامه توضیح میدهیم:

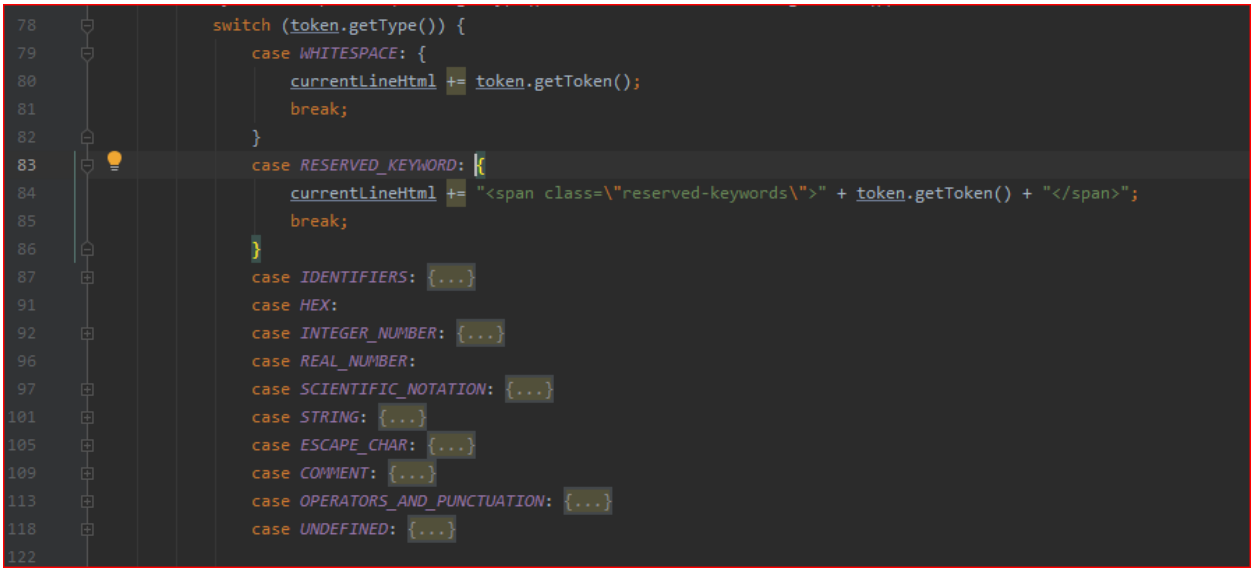
در ابتدا یک کلاس Main نوشته ایم که تقریبا تمام تعاملات را در این کلاس انجام می دهیم. که این کلاس دارای دو Property با نام های inputFile و outputFile می باشد، که به ترتیب، آدرس فایل های دریافتی (همان فایلی که کاربر به زبان Cool نوشته است) و آدرس فایل خروجی (که یک فایل html است) می باشد.

در ادامه یک متد scan داریم که در آن، فایل ورودی را خوانده، و با کمک کلاس CompilerScanner خروجی مناسب رو تولید کرده ایم.

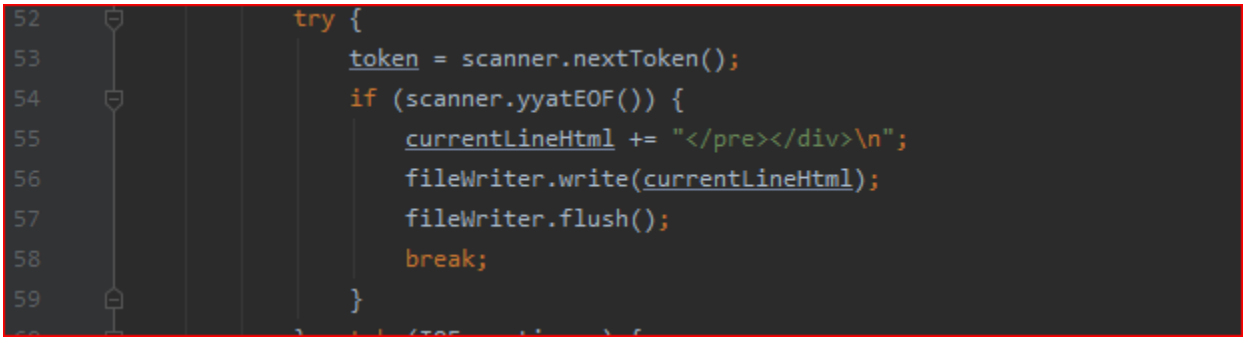
```
28      private void scan() throws IOException {
29          CompilerScanner scanner = new CompilerScanner(new FileReader(inputFile));
30          File output = new File(this.outputFile);
31          FileWriter fileWriter = new FileWriter(output);
32          fileWriter.write( str: "<html lang=\"en\">\n" +
33                  "  <head>\n" +
34                  "    <meta charset=\"UTF-8\" />\n" +
35                  "    <title>Compiler Cool Project</title>\n" +
36                  "    <link rel=\"stylesheet\" href=\"./styles.css\" />\n" +
37                  "    <meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0\" />\n" +
38                  "    <link\n" +
39                  "      href=\"https://fonts.googleapis.com/css?family=Karla\"\n" +
40                  "      rel=\"stylesheet\"\n" +
41                  "    />\n" +
42                  "  </head>\n" +
43                  "  <body>\n");
44          fileWriter.flush();
```

در ابتدای متد scan، مقداری کد html به اصطلاح hard code شده است، که در آن به فایل styles.css لینک شده است. (همچنین اینکه از یک فونت دیگر به نام 'Karla' استفاده کرده ایم که این یک تصمیم کاملا شخصی و سلیقه ای بوده است. به همین دلیل یک لینک به سایت Google Fonts هم در آن دیده میشود)

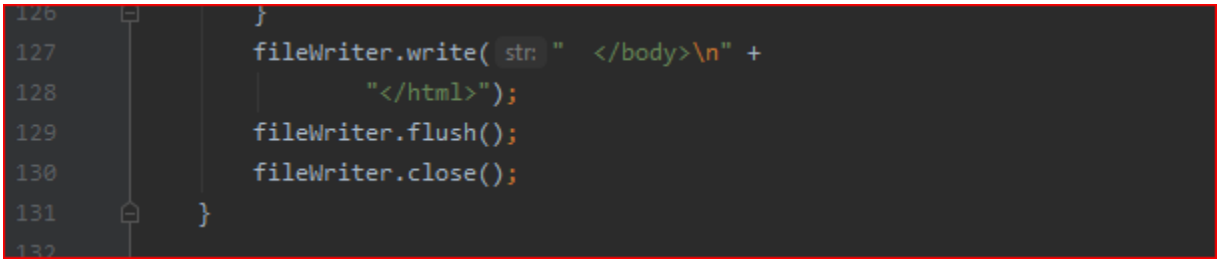
سپس واژه ها رو یکی پس از دیگری با استفاده از نمونه scanner ساخته شده، دریافت میکنیم. و با کمک یک switch case، پس از تشخیص نوع آن، کد html متناظر با آن را در خروجی چاپ میکنیم.



و اما switch case توضیح داده شده به صورت فوق می باشد، که همانطور که دیده میشود تمام حالت ها را به کمک Type بررسی کرده ایم. و در آن از کلاس های CSS ای که در فایل styles.css قرار داده ایم، استفاده شده است. که جلوتر به توضیح کامل آن نیز پرداخته شده است.



همانطور که در تصویر بالا مشخص شده است، برای خروج از فایل از yyatEOF استفاده کرده ایم، که تعیین میکند آیا به انتهای فایل رسیده ایم یا نه.



و در آخر نیز، آخرین تگ های html را بسته ایم. البته اگر این قسمت را هم انجام دهیم اتفاق خاصی رخ نمی دهد.

و در ادامه، برای راحتی کار در فایل CSS مان، یک سری متغیر تعریف کرده ایم:

```css
/* theme colors */
:root {
  --bg-color: #222222;
  --reserved-keywords: #fc618d;
  --identifiers: #ffffff;
  --integer-numbers: #f59762;
  --real-numbers: #f59762;
  --strings: #fce566;
  --special-characters: #ee82ee;
  --comments: #69676c;
  --operators-and-punctuations: #00ffff;
  --undefined-token: #ff0000;
}
```

که این کار باعث افزایش انعطاف کارمان هم میشود که میتوان به راحتی با عوض کردن چند متغیر کل تم کامپایلر را ادیت کرد.

```css
.identifiers {
  color: var(--identifiers);
}
.operators-and-punctuations {
  color: var(--operators-and-punctuations) !important;
}
.reserved-keywords {
  color: var(--reserved-keywords);
}

.strings {
  color: var(--strings);
}
.comments {
  color: var(--comments);
}
.undefined-token {
  color: var(--undefined-token);
}
.real-numbers {
  color: var(--real-numbers);
  font-style: italic;
}
.integer-numbers {
  color: var(--integer-numbers);
}
.line-number {
  width: 30px;
  display: inline-block;
  color: var(--comments);
}
.special-characters {
  color: var(--special-characters);
}
```

و دست آخر به نوشتن یک سری کلاس برای راحتی کار پرداخته ایم. که از آن همانطور که بالاتر هم گفته شد، برای استایل دهی استفاده شده است.

در استایل دهی و فایل html، از تگ div برای هر خط استفاده کرده ایم، که با توجه به اینکه display ای از نوع block دارد، پس از هر div، به خط بعدی میریم، که این کار به ما کمک میکند تا بتوانیم خطوط را مدیریت کنیم. و همچنین برای مدیریت indentation از تگ pre به جای p استفاده شده است.

تصویری از خروجی کد کامپایل شده:

```
// Main class
class Main{
    let int[] items;
    void printArray(){
            let int i;
            for (i=0;i<100;i++)
                    print(items[i]);
            rof
    }

    int main(){
            let int i;
            let int j;
            let int[] rawItems;
            let int arrayLenInHex;
            arrayLenInHex = 0x64;
            rawItems = new Array (int, arrayLenInHex);
            let int dummyScientificNumber;
            dummyScientificNumber = 10.2E+2;

            print("Please enter random numbers!\tWe just print them!");
            print("Max numbers counter: 100, \n(Enter \"-1\"to end sooner.)");
            for(i=0; i<arrayLenInHex; i = i+1)
                    let int x;
                    x = in_int();
                    if(x==-1) then
                            break;
                    else
                            rawItems[i]=x;
                    fi
            rof
            printArray();

            //undefined Token
            \
    }
}
```