به نام خدا

فاز1

پروژه طراحی کامپایلر

سارینا نعمتی

98243090

کیانا باقری

98243095

- **فایل flex**

در این فایل 3 قسمت اصلی داریم که با %% جدا میشوند.

1. در قسمت اول، user code است. در این بخش قسمت هایی که در فایل جاوا نیاز داریم را قرار میدهیم. مثل کتابخانه ها، package ها و امثالهم.
2. در ابتدای قسمت دوم نام کلاس مد نظر را قرار دادیم. در این کد، نام کلاس ما، Scanner است. (%class Scanner) سپس با %public آن را پابلیک کردیم. هر کدام از خط های زیر آن نشان دهنده چیزی است. مثلا %line به ما کمک میکند که اگر در جایی error داشتیم، شماره خط آن را به ما بدهد. %state STRING برای ما تعریف میکند که یک استیت به نام STRING در قسمت سوم کد نیاز داریم. همچنین اسم تابع اصلی برنامه ما، nextToken است که با %function nextToken معرفی شده.

در {%}% وقتی کدی نوشته میشود، معادل دقیق آن در جاوا تولید میشود. یعنی در این قسمت کد جاوا زده میشود. ما کلاس Symbol را در اینجا تعریف کردیم. در این کلاس ما نیاز به جنس token، مقدار آن و شماره خطش داریم. سپس constructor و getter های آن را تعریف کردیم.

در خطوط بعد، توکن های مد نظر در پروژه و regex ها را تعریف کرده ایم:

- ReservedKeywords: در اینجا کلماتی که مخصوص زبان هستند را با | (or) نشان دادیم. یعنی مثلا اگر کلمهbool را ببیند، میداند که رزرو شده و مخصوص زبان است.
- Indentifier: برای تعریف این، با توجه به اینکه خود پروژه با ما گفته با یک حرف شروع شده و ترکیبی از حروف، اعداد و _ است که بیشتر از 31 کاراکتر ندارد، پس ابتدا regex برای اعداد داریم که از 0تا9 هستند. (Digits = [0-9]) سپس برای حروف هم میتوانند هم uppercase و lowercase باشد پس: Letters = [A-Za-z] در نهایت برای تعریف identifier داریم:

({Letters})({Underscore}|{Digits}|{Letters}){0,30}

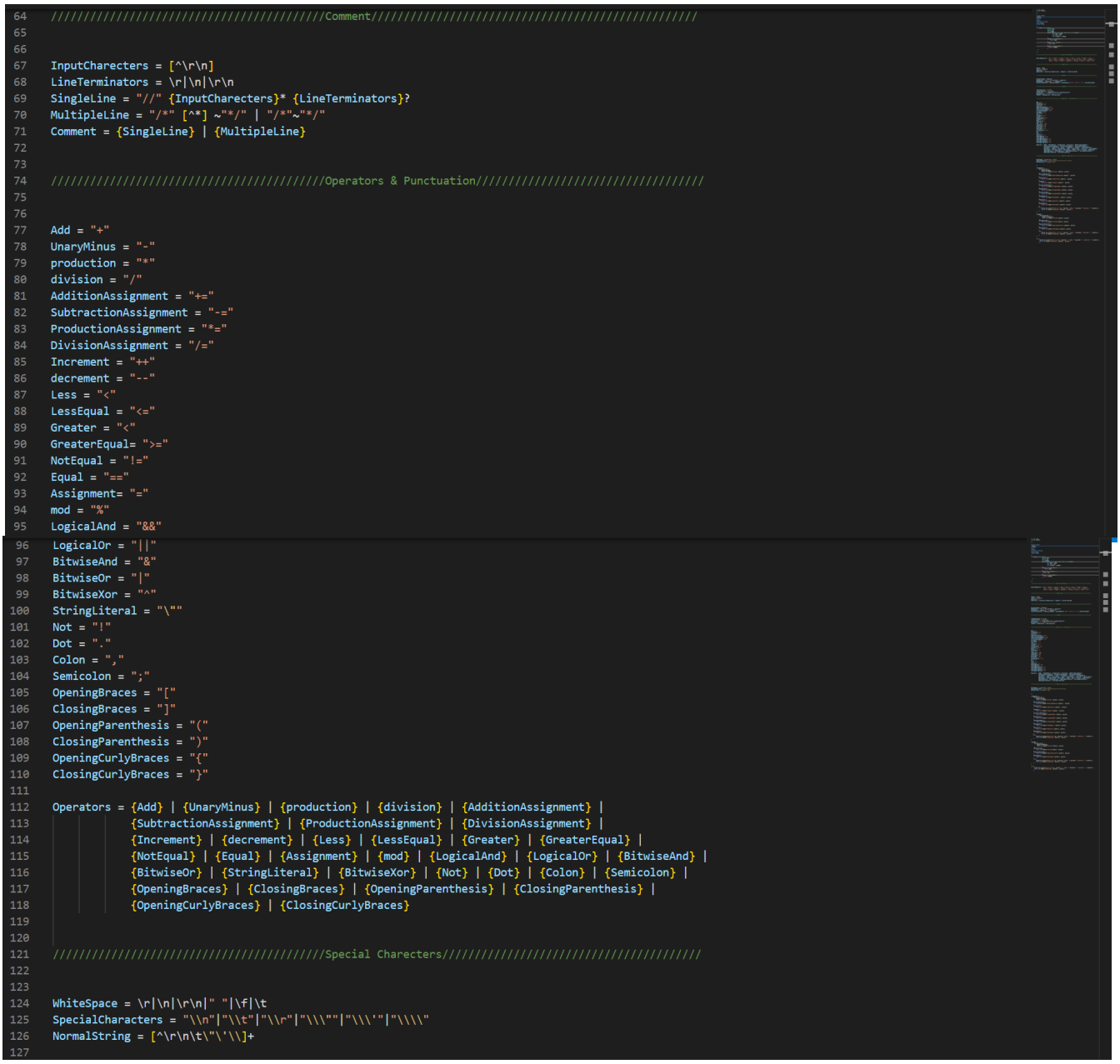
- Operators & Punctuations: همانند ReservedKeywors تعریف میکنیم و و با or نشان میدهیم.
- DecimalInteger: باتوجه به صورت پروژه: +{Digits}
- RealNumbers: با یک یا بیشتر رقم آغاز شده، نقطه می آید و سپس میتواند بعد آن رقم باشد یا نه. *({Digits}) "." +({Digits})
- Hexadecimal: ابتدای آن با 0X یا 0x شروع شده و پس از آن باید یک یا بیشتر رقم یا حرف داشته باشیم. +({Digits} | [A-Fa-f])("0X" | "0x")
- ScientificNotation: شروعش با realNumbers یا decimalInteger است. سپس e|E آمده و میتوانیم علامت - + داشته باشیم و سپس تعدادی رقم بعد آن

داریم. ({DecimalInteger}|{RealNumbers})("E"|"e")("-"|"+") {DecimalInteger}

- Comment: همانند توضیحات داخل ویدیو
- NormalString: رشته های عادی که نباید کاراکترهای خاص زبان باشند. [^\r\n\t\"\'\\]+
- WhiteSpace: واضحه!
- SpecialCharacters: واضحه!

3. در قسمت سوم، باید استیت ها و کار هایی که در هر استیت انجام میشود را بنویسیم. در اینجا برای هر نوعی از regex ها که تعریف شده در قسمت2، یک حالت در نظر گرفته و symbol مربوطه را برمیگردانیم. (این symbol شامل شماره خط، نوع و مقدار آن است.) اگر symbol وجود داشته باشد که برای ما قابل شناسایی نباشد، به ما اروری با شماره خط و column و خود کاراکتر را برمیگرداند و در دسته undefined قرار میگیرد.

```
// user code
package Phase1;

%%

%class Scanner
%public
%unicode
%line
%column
%function nextToken
%type Symbol
%state STRING

 %{
     public class Symbol{
                 String type;
                 Object val;
                 int lineNum;
                 public Symbol(String type, Object val, int lineNum){
                         this.type = type;
                         this.val = val;
                         this.lineNum = lineNum;
                 }
                 public String getType() {
                    return type;
                 }
                 public Object getVal() {
                   return val;
                 }
                 public int getLineNum() {
////////////////////////////////////////Reserved Keywords////////////////////////////////////

ReservedKeywords = "let" | "static" | "while" | "new" | "void" | "class" | "break" | "func" |
                   "int" | "for" | "continue" | "return" | "real" | "len" | "if" | "inputStr" |
                   "bool" | "loop" | "range" | "inputInt" | "string" | "print" | "else" | "in"


////////////////////////////////////////Identifier/////////////////////////////////////////////

Digits = [0-9]
Letters = [A-Za-z]
Underscore = "_"
Identifier = ({Letters})({Underscore} | {Digits} | {Letters}){0,30}

////////////////////////////////////////Numbers/////////////////////////////////////////////////

DecimalInteger = {Digits}+
Hexadecimal = ("0x" | "0X")([A-Fa-f] | {Digits})+
RealNumbers = ({Digits})+ "." ({Digits})*
ScientificNotation = ({DecimalInteger} | {RealNumbers}) ("E" | "e") ("-" | "+") {DecimalInteger}
```



```
////////////////////////////////////////Comment//////////////////////////////////////////////////


InputCharecters = [^\r\n]
LineTerminators = \r|\n|\r\n
SingleLine = "//" {InputCharecters}* {LineTerminators}?
MultipleLine = "/*" [^*] ~"*/" | "/*"~"*/"
Comment = {SingleLine} | {MultipleLine}


////////////////////////////////////////Operators & Punctuation//////////////////////////////////


Add = "+"
UnaryMinus = "-"
production = "*"
division = "/"
AdditionAssignment = "+="
SubtractionAssignment = "-="
ProductionAssignment = "*="
DivisionAssignment = "/="
Increment = "++"
decrement = "--"
Less = "<"
LessEqual = "<="
Greater = "<"
GreaterEqual= ">="
NotEqual = "!="
Equal = "=="
Assignment= "="
mod = "%"
LogicalAnd = "&&"
LogicalOr = "||"
BitwiseAnd = "&"
BitwiseOr = "|"
BitwiseXor = "^"
StringLiteral = "\""
Not = "!"
Dot = "."
Colon = ","
Semicolon = ";"
OpeningBraces = "["
ClosingBraces = "]"
OpeningParenthesis = "("
ClosingParenthesis = ")"
OpeningCurlyBraces = "{"
ClosingCurlyBraces = "}"

Operators = {Add} | {UnaryMinus} | {production} | {division} | {AdditionAssignment} |
            {SubtractionAssignment} | {ProductionAssignment} | {DivisionAssignment} |
            {Increment} | {decrement} | {Less} | {LessEqual} | {Greater} | {GreaterEqual} |
            {NotEqual} | {Equal} | {Assignment} | {mod} | {LogicalAnd} | {LogicalOr} | {BitwiseAnd} |
            {BitwiseOr} | {StringLiteral} | {BitwiseXor} | {Not} | {Dot} | {Colon} | {Semicolon} |
            {OpeningBraces} | {ClosingBraces} | {OpeningParenthesis} | {ClosingParenthesis} |
            {OpeningCurlyBraces} | {ClosingCurlyBraces}


////////////////////////////////////////Special Charecters//////////////////////////////////////


WhiteSpace = \r|\n|\r\n|" "|\f|\t
SpecialCharacters = "\\n"|"\\t"|"\\r"|"\\\""|"\\\'"|"\\\\"
NormalString = [^\r\n\t\"\'\\]+

```

```
%%

<YYINITIAL> {
    {StringLiteral} {
        yybegin(STRING);
        return new Symbol("String", yytext(), yyline);
    }
    {ReservedKeywords} {
        return new Symbol("ReservedKeywords",yytext() , yyline);
    }
    {Identifier} {
        return new Symbol("Identifiers",yytext() , yyline);
    }
    {Comment} {
        return new Symbol("Comment",yytext() , yyline);
    }
    {ScientificNotation} {
        return new Symbol("IntegerNumber",yytext(), yyline);
    }
    {Hexadecimal} {
        return new Symbol("IntegerNumber",yytext(), yyline);
    }
    {DecimalInteger} {
        return new Symbol("IntegerNumber",yytext(), yyline);
    }
    {RealNumbers} {
        return new Symbol("RealNumber", yytext(), yyline);
    }
    {Operators} {
        return new Symbol("Operators",yytext(), yyline);
    }
    {WhiteSpace} {
        return new Symbol("WhiteSpace",yytext(), yyline);
    }
   [^] {
        System.out.println("Error at line: "+yyline + "index: "+ yycolumn + "character = "+ yytext());
        return new Symbol("Undefined", yytext() , yyline) ;
    }
}

<STRING> {
     {StringLiteral}  {
         yybegin(YYINITIAL);
         return new Symbol("String",yytext(), yyline);
     }
    {NormalString} {
        return new Symbol("String",yytext(), yyline);
    }
    {SpecialCharacters} {
        return new Symbol("SpecialCharacters",yytext(), yyline);
    }
    {WhiteSpace} {
        return new Symbol("WhiteSpace",yytext(), yyline);
    }
    [^] {
        System.out.println("Error at line: "+yyline + "index: "+ yycolumn + "character = "+ yytext());
        return new Symbol("Undefined", yytext() , yyline) ;
    }
}

[^] {
     System.out.println("Error at line: "+yyline + "; index: "+ yycolumn + "; character = "+ yytext());
     return new Symbol("Undefined", yytext() , yyline) ;
```

- **فایل java**

در ابتدا آدرس فایل ورودی را برای تست و آدرس فایل خروجی که html بوده و برنامه ما در آن مینویسد را وارد میکنیم. از کلاس symbol نوشته شده در فایل scanner استفاده کرده. با کمک یک حلقه while(true) در هر خط symbol ها را چک کرده و با کمک تابع coloring و نوع symbol، آن را رنگ مد نظر پروژه میکنیم. در اولین if این حلقه، چک میشود که اگر فایل ما به آخر رسیده بود، تگ ها بسته شوند. در if بعدی، چک میکنیم که در خط اول نباشیم و سپس در فایل خروجی، تگ های مورد نظر را مینویسیم. سپس میخواهیم به خط بعدی برویم، پس باید خط قبلی را برابر خطی که الان بودیم قرار دهیم، سپس برای نوشتن شماره خط در ابتدای هر line، باید currLine+1 را بنویسیم که خطوط ما از 1شروع شود. در آخر هم باتوجه با نوع symbol که تابع coloring آن را تشخیص میدهد، symbol خوانده شده را رنگ میکنیم. در آخر هم تگ ها را میبندیم.

خروجی نمونه برنامه در شکل صفحه آخر آورده شده.

```
package Phase1;

import java.io.*;

public class Main {
    static String inputFile = "/Users/pouriya/Downloads/Scanner 2/src/Phase1/test/input.txt";
    static String outputFile = "/Users/pouriya/Downloads/Scanner 2/src/Phase1/test/output.html";
    static Scanner.Symbol symbol = null;

    public static String coloring(String currentLineHtml){
        switch (symbol.getType()) {
            case "WhiteSpace" -> currentLineHtml += symbol.getVal();
            case "Undefined" -> currentLineHtml += "<p3 style=\"color:  #FF1818 ;\">" + symbol.getVal() + "</p3>";
            case "String" -> currentLineHtml += "<p3 style=\"color:  #00C897;\">" + symbol.getVal() + "</p3>";
            case "ReservedKeywords" -> currentLineHtml += "<p3 style=\"color:  #B667F1 ;\">" + symbol.getVal() + "</p3>";
            case "Identifiers" -> currentLineHtml += "<p3 style=\"color:  #FFFFFF ;\">" + symbol.getVal() + "</p3>";
            case "RealNumber" -> currentLineHtml += "<em><p3 style=\"color:  #FFC300 ;\">" + symbol.getVal() + "</p3></em>";
            case "IntegerNumber" -> currentLineHtml += "<p3 style=\"color:  #FFC300 ;\">" + symbol.getVal() + "</p3>";
            case "SpecialCharacters" -> currentLineHtml += "<em><p3 style=\"color:  #B8FFF9;\">" + symbol.getVal() + "</p3></em>";
            case "Operators" -> currentLineHtml += "<p3 style=\"color:  #90E0EF ;\">" + symbol.getVal() + "</p3>";
            case "Comment" -> currentLineHtml += "<p3 style=\"color:  #69676C;\">" + symbol.getVal() + "</p3>";
        }
        return currentLineHtml;
    }

    public static void main(String[] args) {
        try {
            Scanner scanner = new Scanner(new FileReader(inputFile));
            File outFile = new File(outputFile);
            FileWriter fileWriter = new FileWriter(outFile);

            int lastLine = -1;
            String currentLineHtml = "";
```

```
            fileWriter.write("""
                <html lang="en">
                  <head>
                <style>body {padding: 40px;background-color: #222222}</style>    <meta charset="UTF-8" />
                    <meta/>
                  </head>
                  <body>
                """);
            fileWriter.flush();

            while (true) {
                try {
                    symbol = scanner.nextToken();
                    if (scanner.yyatEOF()) {
                        currentLineHtml += "</pre></div>\n";
                        fileWriter.write(currentLineHtml);
                        fileWriter.flush();
                        break;
                    }

                } catch (IOException e) {
                    e.printStackTrace();
                }

                int currLine = symbol.getLineNum();
//                System.out.println(currLine);

                if (currLine > lastLine) {

                    if (lastLine != -1) {
                        fileWriter.write(currentLineHtml);
                        fileWriter.flush();
                    }
```

```
                    lastLine = currLine;
                    currentLineHtml = "<div><pre><b><p3 style=\"color:  #4863A0;\">" + (currLine+1) + "</p3></b>";

                }

//                System.out.println(symbol.getType() + ": " + symbol.getVal());
                currentLineHtml = coloring(currentLineHtml);

            }

            fileWriter.write("  </body>\n" +
                    "</html>");
            fileWriter.flush();
            fileWriter.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

خروجی نمونه :

```
class Main {
    let int[] items;

    func printArray () void {
        let int i;
        for (i = 0 ; i < 100 ; i++)
            print (items[i]);
    }

     func main() int{
         let int i, j;
         let int[] rawItems;
         let int arrayLenInHex;

         arrayLenInHex = 0x64;
         rawItems = new int[arrayLenInHex];

         let int dummyscientificNumber;
         dummyScientificNumber = 10.2E+2;

         print("Please enter random numbers.");
         print("Max numbers counter: 100,  \n (Enter  \" -1 \" to end sooner.)");

         for (i=0 ; i < arrayLenInHex; i = i +1){
```

```
14
15        arrayLenInHex = 0x64;
16        rawItems = new int[arrayLenInHex];
17
18        let int dummyscientificNumber;
19        dummyScientificNumber = 10.2E+2;
20
21        print("Please enter random numbers.");
22        print("Max numbers counter: 100,  \n (Enter  \" -1 \" to end sooner.)");
23
24        for (i=0 ; i < arrayLenInHex; i = i +1){
25            let int x;
26            x = inputInt();
27            if (x == -1)
28                break;
29            else
30                rawItems [i] = x;
31        }
32        printArray ();
33       }
34    }
35}
```