Welcome to the compiler project!

In this project, you are supposed to design a compiler for an Object Oriented Language called Cool. Cool stands for ***Classroom Object Oriented Language*** and is designed to be implemented with reasonable effort in a one-semester course. During this project, we will cover most of its features. Note that we have changed official Cool syntax a little bit!

The whole project consists of three main phases:

1- Scanner (or Lexical Analyzer) 2- Parser 3- Code Generator

The output of this phase – with little changes – is the input for the next phase. The final program containing all phases must be able to get a Cool program and generate codes ready to be run. Notice that if you do not implement a phase properly with minimum requirements, the next phase will be in great chaos.

Phase 1

At this phase, only the scanner is required, meaning that your program should be capable of getting a stream of characters (not necessarily a Cool program) and break it to the tokens. Your scanner must not consider the whitespaces out of string constants and should distinguish keywords, integer/real/string/boolean constants, operators and identifiers. Furthermore, it must just ignore the token if not in the sets above.

In this phase, you should create a syntax highlighter for the language we describe later in this document. To make your syntax highlighter, you should design your scanner first and then use your designed scanner in your syntax highlighter.

Reserved Keywords

Cool language has following keywords.

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
print	len				

Identifier

A sequence of letters, digits and underline starting with a letter. Cool is case sensitive. The identifiers can be at most 31 characters long.

Numbers

The specifications of the numbers are described below.

Type	Description
Decimal Integer	A sequence of digits from 0-9 Example: 1642, 134, 546
Hexadecimal	A sequence of digits and characters from A/a to F/f staring with 0X/0x. Example: 0x0, 0X12aE
Real Numbers	A sequence of digits having a '.' in between. Note that there can be no digit after '.' Example: 0.12 ✓, 12. ✓, .12 ×
Scientific Notation	A real or integer number following an 'E' or 'e' and an integer with an optional plus or minus sign. Examples: 12.2e+2 ✓, 12.E2 ✓, 1.2E-1 ✓, .12E3 ×

Comments

Comments have two types: ones that start with // and ones that start with /* and end with */ and can span in multiple lines.

Operators and Punctuation

The language also contains following symbols which must have black color.

Description	Symbol	Description	Symbol
add	+	unary minus	-
production	*	division	/
addition assignment	+=	subtraction assignment	-=
production assignment	*=	division assignment	/=
increment	++	decrement	--
less	<	less equal	<=
greater	>	greater equal	>=
not equal	!=	equal	==
assignment	<-	mod	%
logical and	&&	logical or	||
bitwise and	&	bitwise or	|
string literal	“	bitwise xor	^
not	!	dot	.
colon	,	semicolon	;
opening braces	[	closing braces	]
opening parenthesis	(	closing parenthesis	)
opening curly braces	{	closing curly braces	}

Strings

Like C language, strings start and end with “. Strings can have special characters.

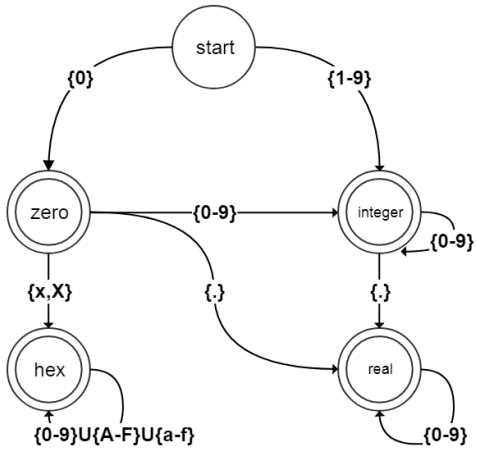
Special Characters (Escape Characters)

A character preceded by a backslash (\) is an escape character and has special meaning to the compiler.

Special Characters you have to consider:

\n, \t, \r, \’, \”, \\

Part of Scanner Graph



How to do this phase?

At first, you should download/install and learn jflex. Jflex is a lexical analyzer generator (scanner generator). You mostly use regular expressions (regex) in the jflex file. Suppose that your jflex file has scanner.flex name. With the following command you can generate your scanner. You can read jflex documentation here.

```
java -jar jflex-1.8.2.jar scanner.flex
```

After that you should have a Main.java file that use your scanner to read the tokens. Then, you should use the scanner and build a Html file that has the tokens with the colors that we mention below.

Note: You should use #222222 background color for your html output.

Token	Style (color and format)
Reserved Keywords	#FC618D
Identifiers	#FFFFFF
Integer Numbers	#F59762
Real Numbers	#F59762 and *italic*
Strings	#FCE566
Special Characters	#EE82EE and *italic*
Comments	#69676C
Operators and Punctuations	#00FFFF
Undefined Token	#FF0000

An example html output for following code:

```
// 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 (for example a single backslash "\" is undefined)
    \
  }
}
```

Notes

- The due date is Aban 30th.
- Your program must output an HTML file that highlights text based on rules described above.
- Your html must be readable. If your html output does not have indentations like input file, there is no problem. Just make sure that the output is readable.
- Showing indentations like input file and showing line numbers has bonus points up to 10% of total points!
- What you must upload is a .zip file containing your program, and a .pdf or .md report file explaining how to run the project.
- In case of using any resources, mention them in your report file.
- You can use python language for this phase but it is not recommended! Maybe you have no problem for this phase but you must use PGen tool in the next phase. You can't use parser generator of PGen tool in next phase because it generates java classes. If you choose python, you should read PGen generated parse tables and implement the parser based on them in next phase. It is a bit harder than using java generated parsers but it is your choice! We do not recommend python!
- This phase of the project can be done in groups of two.