هدف این پروژه طراحی یک اسکنر (تحلیل‌گر لغوی) است.

این پروژه می‌تواند به صورت گروهی انجام شود. برای این منظورشما باید تا تاریخ ۱۴ اسفند موارد زیر را اعلام کنید:

- لیست نام و شماره دانشجویی اعضای گروه
- زبان برنامه نویسی مبدا
- زبان برنامه نویسی که قرار است اسکنر به آن زبان نوشته شود

لازم به ذکر است که برای اعلام این موراد کافی است یکی از اعضا گروه به f.ahamadi.moughari@gmail.com ایمیل بزند.

تعداد اعضای گروه می تواند از ۱ تا ۴ نفر باشد.

اسکنر مورد نظر را می توانید به صورت دستی پیاده‌سازی کنید یا از ابزارهای موجود مانند FLEX (یا ابزارهای مشابه مانند LEX, JLEX, ...) استفاده کنید.

برای استفاده از FLEX می‌توانید از لینک‌های زیر راهنمایی بگیرید. هم‌چنین با جستجوی مناسب در اینترنت می‌توانید راهنما و نمونه‌های مناسبی بیابید.

http://alumni.cs.ucr.edu/~lgao/teaching/flex.html

http://web.mit.edu/gnu/doc/html/flexfitoc.html#SEC1

https://github.com/westes/flex

هم‌چنین یک مقدمه کوتاه در مورد FLEX را می‌توانید در زیر بیابید.

Introduction to Flex

Flex allows you to implement a lexical analyzer by writing rules that match on user-defined regular expressions and performing a specified action for each matched pattern. Flex compiles your rule file (e.g., \lexer.flex") to C source code implementing a finite automaton recognizing the regular expressions that you specify in your rule file. Fortunately, it is not necessary to understand or even look at the automatically generated (and often very messy) file implementing your rules.

Rule files in flex are structured as follows:

%{
Declarations
%}
Definitions
%%
Rules
%%
User subroutines

The Declarations and User subroutines sections are optional and allow you to write declarations and helper functions in C. The Definitions section is also optional, but often very useful as definitions allow you to give names to regular expressions. For example, the definition

DIGIT [0-9]

allows you to define a digit. Here, *DIGIT* is the name given to the regular expression matching any single character between 0 and 9. The following table gives an overview of the common regular expressions that can be specified in Flex:

x	*the character "x"*
"x"	*an "x", even if x is an operator.*
\x	*an "x", even if x is an operator.*
[xy]	*the character x or y.*
[x-z]	*the characters x, y or z.*
[^x]	*any character but x.*
.	*any character but newline.*
^x	*an x at the beginning of a line.*
<y>x	*an x when Flex is in start condition y.*
x$	*an x at the end of a line.*
x?	*an optional x.*
*x**	*0,1,2, ... instances of x.*
x+	*1,2,3, ... instances of x.*
x|y	*an x or a y.*
(x)	*an x.*
x/y	*an x but only if followed by y.*
{xx}	*the translation of xx from the definitions section.*
x{m,n}	*m through n occurrences of x*

The most important part of your lexical analyzer is the rules section. A rule in Flex specifies an action to perform if the input matches the regular expression or definition at the beginning of the rule.
The action to perform is specified by writing regular C source code. For example, assuming that a digit represents a token in our language, the rule:

```
{DIGIT} {
coolfiyylval.symbol = inttable.addfistring(yytext);
return DIGITfiTOKEN;
}
```

records the value of the digit in the global variable and returns the appropriate token code.
An important point to remember is that if the current input (i.e., the result of the function call to *yylex*()) matches multiple rules, Flex picks the rule that matches the largest number of characters. For instance, if you define the following two rules

```
[0-9]+ {// action 1}
[0-9a-z]+ {// action 2}
```

and if the character sequence 2a appears next in the file being scanned, then *action 2* will be performed since the second rule matches more characters than the first rule. If multiple rules match the same number of characters, then the rule appearing first in the file is chosen.
When writing rules in Flex, it may be necessary to perform different actions depending on previously encountered tokens. For example, when processing a closing comment token, you might be interested in knowing whether an opening comment was previously encountered. One obvious way to track state is to declare global variables in your declaration section, which are set to true when certain tokens of interest are encountered. Flex also provides syntactic sugar for achieving similar functionality by using state declarations such as:

%Start COMMENT

which can be set to true by writing *BEGIN(COMMENT)*. To perform an action only if an opening comment was previously encountered, you can predicate your rule on *COMMENT* using the syntax:

<COMMENT> {
// the rest of your rule ...
}

There is also a special default state called *INITIAL* which is active unless you explicitly indicate the beginning of a new state. You might find this syntax useful for various aspects of this assignment, such as error reporting.