CS305 – Programming Languages

Spring 2024-2025

HOMEWORK 1

Implementing a Lexical Analyzer (Scanner) for Meeting Scheduler (MS)

Due date: March 2nd, 2025 @ 23:55

NOTE

Only SUCourse submissions are allowed. Submissions by e-mail are not allowed. Please see the note at the end of this document for the late submission policy.

1 Introduction

In this homework, you will use flex to implement a scanner for (**MS**) language. Your scanner will be used to detect the tokens in an MS program. MS is a scripting language that will be used for meeting schedules.

The MS program comprises certain types of tokens. Your scanner should catch them. The details will be explained in the upcoming sections.

An example MS program is given below:

```
Meeting "Meeting with the boss"
    meetingNumber = 12057
    description = "My monthly meetings with the boss.
                   We discuss the general monthly plans."
    startDate = 17.04.2025
    endDate = 17.04.2025
    startTime = 14.40
    endTime = 16.30
    locations = FENSG032
    isRecurring = yes
    frequency = monthly
    repetitionCount = 12
endMeeting
```

```

Meeting "Important project"
    meetingNumber = 13204
    description = "The meetings for our important project"
    startDate = 19.04.2025
    startTime = 09.00
    endDate = 26.04.2025
    endTime = 17.00
    locations = FENSG032,FASSG018,Company_Lounge
    isRecurring = no
    subMeetings
       Meeting "Design Meetings"
          meetingNumber = 13217
          description = "The meetings for the design"
          startDate = 19.04.2025
          startTime = 09.00
          endDate = 19.04.2025
          endTime = 16.00
          locations = FASSG018
          isRecurring = yes
          frequency = daily
          repetitionCount = 3
       endMeeting

       Meeting "Implementation Meetings"
          meetingNumber = 14935
          description = "The meetings for the implementation"
          startDate = 22.04.2025
          startTime = 10.00
          endDate = 22.04.2025
          endTime = 16.00
          locations = FENSG032
          isRecurring = yes
          frequency = daily
          repetitionCount = 4
       endMeeting

       Meeting "Presentation"
          meetingNumber = 23734
          description = "The presentation of the
```

```
                              results of the project"
            startDate = 26.04.2025
            startTime = 13.00
            endDate = 26.04.2025
            endTime = 17.00
            locations = Company_Lounge
            isRecurring = no
        endMeeting
    endSubMeetings
endMeeting
```

Your program should read and detect the tokens in such MS programs.

Note: In this homework, we are only implementing a scanner. Since the purpose of the scanner is to detect tokens (it does not have anything to do with the grammatical structure of the input), even grammatically incorrect inputs can be successfully processed by the scanner.

For example, when the following (grammatically wrong) MS program is processed by the scanner, there will be no complaints, and the tokens appearing in this program will be detected.

```
description = 11.11.1111
locations = 89561
startTime = FMAN1099
meetingNumber = yes
endDate = weekly
repetitionCount = "The Best Course Ever"
```

Note: The indentation is unimportant; any number of tabs or new lines can be used between the lexemes of tokens. It does not change anything in the end, as long as the lexemes are separated from each other with at least one blank space or tab or new line.

Also note that, in some cases, the lexemes of different tokens may not be separated by a space. For example, in

```
locations=89561
```

we have 3 tokens with lexemes `locations`, `=`, and `89561`, without any space between them.

In an MS program, there may be keywords (e.g., startTime), integers (e.g., 12057), strings (e.g., the name and the description of the meetings), dates (e.g., 17.04.2025), time (e.g., 09.30), identifiers (e.g., room names like FENSG032 or Company_Lounge), assign operator (=), and comma (,), and many others. Your scanner will catch these language constructs (introduced in Sections 2, 3) and print out the token names together with their positions (explained in Section 5) in the input file. Please see Section 6 for an example of the required output. The following sections provide extensive information on the homework. Please read the entire document carefully before starting your work on the homework.

2 Keywords, Punctuations & Values

Below is the list of keywords to be implemented, together with the corresponding token names.

Lexeme	**Token**
Meeting	tSTARTMEETING
endMeeting	tENDMEETING
subMeetings	tSTARTSUBMEETINGS
endSubMeetings	tENDSUBMEETINGS
meetingNumber	tMEETINGNUMBER
description	tDESCRIPTION
startDate	tSTARTDATE
endDate	tENDDATE
startTime	tSTARTTIME
endTime	tENDTIME
locations	tLOCATIONS
isRecurring	tISRECURRING
frequency	tFREQUENCY
repetitionCount	tREPETITIONCOUNT

Lexeme	**Token**	**Lexeme**	**Token**
daily	tDAILY	weekly	tWEEKLY
monthly	tMONTHLY	yearly	tYEARLY
yes	tYES	no	tNO
=	tASSIGN	,	tCOMMA

MS is case–sensitive. Only the lexemes given above are used for the keywords.

3 Identifiers, Strings, Integers, Date & Time

Token Type	**Token**	**Token Type**	**Token**
identifier	tIDENTIFIER	string	tSTRING
integers	tINTEGER	date	tDATE
time	tTIME		

You need to implement identifiers, strings, integer values, date and time tokens. Here are some rules you need to pay attention to.

- An identifier consists of any combination of letters, digits, and underscore characters. However, it must start with a letter.
- The token name for an identifier is tIDENTIFIER.
- A string is a sequence of characters enclosed by a pair of double quote symbols (e.g., the name or a description of a meeting). Any symbol (except the quote symbol) can be used inside a string.
- The token name for a string is tSTRING.
- An integer is a non–negative integer that can possibly start with 0.
- The token name for an integer is tINTEGER.

- Date should be given in the following style: DD.MM.YYYY

  Example date lexemes are : 12.11.2025, 04.12.2026, 13.04.2024, 01.01.2025, 34.72.0125

  Note that, as long as a lexeme follows the DD.MM.YYYY pattern, we consider it as a date. For the time being, we do not care if they really correspond to an actual date.

- The token name for a date object is tDATE.

- Time should be given in the following style: HH.MM

  Example time lexemes are : 12.11, 04.12, 04.09, 34.72

  Note that, as long as a lexeme follows the HH.MM pattern, we consider it as a time. We do not care if they really correspond to an actual time expression.

- The token name for a time object is tTIME.

- Any character that is not a whitespace character and which cannot be detected as part of the lexeme of a token must be considered an illegal character and must be printed out together with an error message (see below, for example, execution part about the error message details). The whitespace characters that should not be considered illegal characters are space or blank ( ), tab (`\t`), and newline (`\n`).

4 Comments

Comments in an MS program are used to provide explanations, notes, or descriptions for the MS program. There are two ways to indicate comments:

- Single line comments: A comment that starts with // and extends up to the end of the line. Anything following // on the same line is considered a comment.

  Example:

```
    // Initializing the meeting locations

    locations FENSG032, FMANG050  // 2 rooms are used
```

- Multiline comments: These comments begin with /* and end with */. Anything enclosed within this comment block is considered a multiline comment. This type of comment can span multiple lines. If you encounter another opening comment tag (/*) inside an existing multiline comment block, it should be considered as the start of a separate comment. The number of possible nested comments can vary; that is, it cannot be known in advance, and it is essential to treat each nested comment as a new comment block, each with its own content and its corresponding closing tag (*/). A multiline comment does not have to have a closing tag. In such a case, your scanner does not complain about it and should treat the remainder of the program as if it's still within the comment block. A closing tag (*/) without a corresponding opening tag is not related to comments and should be considered illegal characters.

  Example:

```
/*
    This is a multiline comment
    /*
        meetingNumber = 365
        description = "What a beautiful life!"
        isRequrring = false
    */
        startDate = 01.01.0001
        startTime = 00.00
        endDate = 21.12.2012
        endTime = 11.11
*/
```

A comment may appear on any line of a program. Anything that is commented out will be eaten up by your scanner silently. That is, no information will be produced for the comments or the comments' contents.

However, the lines used by the comments will be taken into consideration. For example, for the following MS program, an error will be produced for the illegal character #, and the line for the error must be 5:

```
1    /*
2        This is a multiline comment
3    */
4    // Another comment  /* this does not start a block comment
5    #
```

5 Positions

You must keep track of the position information for the tokens. For each token that will be reported, the line number at which the lexeme of the token appears is considered to be the position of the token. Please look at Section 6 to see how the position information is reported together with the token names.

6 Input, Output, and Example Execution

Assume that your executable scanner (which is generated by passing your flex program through flex and bypassing the generated lex.yy.c through the C compiler gcc) is named MSScanner. Then, we will test your scanner on a number of input files using the following command line:

```
MSScanner < test17.ms
```

As a response, your scanner should print out a separate line for each token it catches in the input file. The output format for a token is given below for each token separately:

Detected	**Output**
tokens from section 2	⟨*row*⟩_⟨*token*⟩
tokens from section 3	⟨*row*⟩_⟨*token*⟩_(⟨*lexeme*⟩)
illegal character	⟨*row*⟩_`ILLEGAL_CHARACTER`_(⟨*lexeme*⟩)

There are placeholders like ⟨*row*⟩ , ⟨*lexeme*⟩, ⟨*token*⟩ in the table above.

- ⟨*row*⟩ should be replaced by the location (line number) of the first character of the lexeme of the token.
- ⟨*token*⟩ is the token name for the detected token. The names of the tokens to be used here are given in Sections 2, 3) above.
- ⟨*lexeme*⟩ should be replaced by the lexeme of the token.

Update: For string lexemes, which are located between two double quotation mark, if the closing quotation mark does not appear, all remaining characters should be detected as string. And, at the end, the program should print a right parenthesis after all characters. If there is a case like:

```
    Meeting "Daily

v
```

The expected outcome is the following:

```
1_tSTARTMEETING
1_tSTRING("Daily

v)
```

Please note the use of underscore (_) characters as spaces in the output to be generated. **Please use no spaces, and use only a single underscore character as indicated in the output explanations and as given in the examples.** The spaces in the lexemes of the strings will stay as they are. There is no need to modify them. However, other than that, you will not insert spaces in the output. Instead, we will use underscore characters and use only one underscore character to separate two pieces of information. Also, do not insert unnecessary newline characters. Each token must be on a separate line; there should not be a line without any information.

The reason for this is the following. We automatically compare your output with the correct output. Although inserting a space or an additional underscore in the output may seem harmless, an automatic comparison would indicate a difference between your output and the correct output, which can cost you some points in the grading.

As an example, let us assume that the following is our MS program:

```
Meeting "Office Hour"
    meetingNumber = 36452
    description = "I must attend"
    startDate = 25.02.2025
    endDate = 26.02.2025
    startTime = 18.40
    endTime = 13.30
    locations = Zoom
    isRecurring = yes
    frequency = weekly
    repetitionCount = 13
    subMeetings
        Meeting "Office Hour of Tuesday"
            meetingNumber = 364521
            description = "I must attend"
            startDate = 25.02.2025
            endDate = 25.02.2025
            startTime = 18.40
            endTime = 19.30
            locations = Zoom
            isRecurring = no
        endMeeting

        Meeting "Office Hour of Wednesday"
            meetingNumber = 364522
            description = "I must attend"
            startDate = 26.02.2025
            endDate = 26.02.2025
            startTime = 12.40
            endTime = 13.30
            locations = Zoom
            isRecurring = no
        endMeeting
    endSubMeetings
endMeeting
```

Then the output of your scanner must be:

```
1_tSTARTMEETING
1_tSTRING_("Office Hour")
2_tMEETINGNUMBER
2_tASSIGN
2_tINTEGER_(3645)
3_tDESCRIPTION
3_tASSIGN
3_tSTRING_("I must attend")
4_tSTARTDATE
4_tASSIGN
4_tDATE_(25.02.2025)
5_tENDDATE
5_tASSIGN
5_tDATE_(26.02.2025)
6_tSTARTTIME
6_tASSIGN
6_tTIME_(18.40)
7_tENDTIME
7_tASSIGN
7_tTIME_(13.30)
8_tLOCATIONS
8_tASSIGN
8_tIDENTIFIER_(Zoom)
9_tISRECURRING
9_tASSIGN
9_tYES
10_tFREQUENCY
10_tASSIGN
10_tWEEKLY
11_tREPETITIONCOUNT
11_tASSIGN
11_tINTEGER_(13)
12_tSTARTSUBMEETINGS
13_tSTARTMEETING
13_tSTRING_("Office Hour of Tuesday")
14_tMEETINGNUMBER
```

```
14_tASSIGN
14_tINTEGER_(364521)
15_tDESCRIPTION
15_tASSIGN
15_tSTRING_("I must attend")
16_tSTARTDATE
16_tASSIGN
16_tDATE_(25.02.2025)
17_tENDDATE
17_tASSIGN
17_tDATE_(25.02.2025)
18_tSTARTTIME
18_tASSIGN
18_tTIME_(18.40)
19_tENDTIME
19_tASSIGN
19_tTIME_(19.30)
20_tLOCATIONS
20_tASSIGN
20_tIDENTIFIER_(Zoom)
21_tISRECURRING
21_tASSIGN
21_tNO
22_tENDMEETING
24_tSTARTMEETING
24_tSTRING_("Office Hour of Wednesday")
25_tMEETINGNUMBER
25_tASSIGN
25_tINTEGER_(364522)
26_tDESCRIPTION
26_tASSIGN
26_tSTRING_("I must attend")
27_tSTARTDATE
27_tASSIGN
27_tDATE_(26.02.2025)
28_tENDDATE
28_tASSIGN
28_tDATE_(26.02.2025)
```

```
29_tSTARTTIME
29_tASSIGN
29_tTIME_(12.40)
30_tENDTIME
30_tASSIGN
30_tTIME_(13.30)
31_tLOCATIONS
31_tASSIGN
31_tIDENTIFIER_(Zoom)
32_tISRECURRING
32_tASSIGN
32_tNO
33_tENDMEETING
34_tENDSUBMEETINGS
35_tENDMEETING
```

Note that the test file content need not be a complete or correct MS program. If the content of a test file is the following:

```
Meeting "Daily Meetings"
    startDate 23.12.2022
    endTime 13.35
```

Then your scanner should not complain about anything and output the following information:

```
1_tSTARTMEETING
1_tSTRING_("Daily Meetings")
2_tSTARTDATE
2_tDATE_(23.12.2022)
3_tENDTIME
3_tTIME_(13.35)
```

Also note that some of the test cases may have nested comment lines. In that case, you should handle them. If the content of a test file is the following:

```
/*
    This is a multiline comment starting
```

```

        Meeting "This meeting will never be shown."
            startTime = 11.30

        This is a multiline comment ending
    */
        endMeeting // This part should be detected!
```

Then your scanner output should be the following:

```
9_tENDMEETING
```

7 How to Submit

Submit only your flex file (**without zipping it**) on SUCourse. The name of your flex file must be:

username-hw1.flx

where username is your SUCourse username.

If your username is ***husnu.yenigun***, then the name of your flex file must be ***husnu.yenigun-hw1.flx***. It must not be husnuyenigun-hw1.flx or something else.

If you fail to submit your file with this correct name, you will get -20 points.

8 Notes

- **Important**: Name your file as you are told and **don't zip it.**
  [**-20 points otherwise**]
- Do not copy-paste MS program fragments from this document as your test cases. Copying/pasting from PDFs can create some unrecognizable characters. Instead, all MS code fragments that appear in this document are provided as a text file for you to use.
- Make sure you print the token names precisely as it is supposed to be. Also, make sure you use underscore characters (not spaces), as explained above. You will lose points otherwise.

- No homework will be accepted if it is not submitted using SUCourse.
- You may get help from our TAs, LA, or friends. However, **you must write your flex file by yourself**.
- Start working on the homework immediately.
- If you develop your code or create your test files on your own computer (not on cs305.sabanciuniv.edu), there can be incompatibilities once you transfer them to the cs305 machine. Since the grading will be done automatically on the cs305 machine, we strongly encourage you to develop it on the cs305 machine or at least test your code on the cs305 machine before submitting it. If you prefer not to test your implementation on the cs305 machine, this means you accept to take the risks of incompatibilities. Even if you have spent hours on homework, you can easily get 0 due to such incompatibilities.

LATE SUBMISSION POLICY

Late submission is allowed subject to the following conditions:

- Your homework grade will be decided by multiplying what you get from the test cases by a "submission time factor (STF)".
- If you submit on time (i.e., before the deadline), your STF is 1. So, you don't lose anything.
- If you submit late, you will lose 0.01 of your STF for every 5 minutes of delay.
- We will not accept any homework later than 500 minutes after the deadline.
- SUCourse+'s timestamp will be used for STF computation.
- If you submit multiple times, the last submission time will be used.