CS305 – Programming Languages
Fall 2023-2024

HOMEWORK 1

Implementing a Lexical Analyzer (Scanner) for MailScript

Due date: October 23, 2023 @ 23:55

NOTE

Only SUCourse submission is allowed. No submission by e-mail. Please see the note at the end of this document for late submission policy.

1 Introduction

In this homework you will implement a scanner for ***MailScript*** language using flex. Your scanner will be used to detect the tokens in a MailScript program. MailScript is a scripting language that will be used for e-mail automation.

MailScript is loosely based on AppleScript, which is a scripting language that facilitates automated control over scriptable Mac applications. MailScript uses a block structure. Each block starts with the keywords "**Mail from**" followed by the e-mail of the user and a colon symbol. Each block ends with the keywords "**end Mail**". One can use the keyword "**send**" in order to send e-mails to certain users. Also they can create a sub-block called "**schedule**" and specify a certain time for the e-mails to be sent. One can "**set**" variables in and out of block. An example MailScript file is given below:

```
set Message ("Welcome to CS305.")
set Name ("Deniz")

Mail from derya@mail.com:

    send ["Hello!"] to [("Ayse", ayse@mail.com),
```

```
    (mehmet@mail.com), ("Mehmet", mehmet@mail.com)]

    schedule @ [03/10/2021, 16:00]:
        send [Message] to [("Beril", beril@mail.com.tr)]
        send ["Thank you
        very much."] to [(Name, deniz@mail.com.tr)]
    end schedule

end Mail


Mail from derya@sabanciuniv.edu:

    schedule @ [02/10/2021, 16:00]:
        send ["Good morning!"] to [(ali@mail.com),
        ("Ferhat Kaya", ferhat@mail.com),
        ("Ali", ali@mail.com)]
    end schedule

end Mail

Mail from derya@mail.com:

    schedule @ [03/10/2021, 04:00]:
        send ["How are you?"] to [("Omer", omer@mail.com)]
    end schedule

end Mail
```

In a MailScript program, there may be keywords (e.g. send, set, ...), values (e.g. e-mails, dates, ...), identifiers (programmer defined names for variables, etc.) and punctuation symbols (like , : etc.). Your scanner will catch these language constructs (introduced in Sections 2, 3, 4) and print out the token names together with their positions (explained in Section 6) in the input file. Please see Section 7 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

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

Lexeme	Token	Lexeme	Token
Mail	tMAIL	schedule	tSCHEDULE
end Mail	tENDMAIL	end schedule	tENDSCHEDULE
send	tSEND	to	tTO
from	tFROM	set	tSET

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

Note that for "`end Mail`", the whitespace between "`end`" and "`Mail`" is not necessarily a single blank character. There can be any non–empty sequence of blanks, tab characters, newline characters between `end` and `Mail`. So the followings (except the last one) are all going to be detected as `tENDMAIL`:

```
end Mail
```

```
end       Mail
```

```
end


     Mail
```

```
endMail
```

The same also applies to the number of whitespace in "`end schedule`".

3 Operators & Punctuation Symbols

Below is the list of operators and punctuation symbols to be implemented, together with the corresponding token names.

Lexeme	Token	Lexeme	Token
,	tCOMMA	:	tCOLON
(	tLPR	)	tRPR
[	tLBR	]	tRBR
@	tAT		

4 Identifiers & Values

You need to implement identifiers and values (strings, e-mails, dates and time). Here are some rules you need to pay attention to. Below, we use "letters" to refer to set of lowercase letters `a,b,c,...,z` and capital letters `A,B,C,...,Z`. "Digits" refer to the set of digits `0,1,2,...,9`, "underscore" refers to the symbol `_` , and hyphen (dash) refers to the symbol `-` .

- An identifier consists of any combination of letters, digits and underscore character. However, it cannot start with a digit.
- The token name for an identifier is tIDENT.
- Anything inside a pair of quotation marks is considered as a string. However a string cannot contain a quotation mark itself. The empty string (an opening quote immediately followed by a closing quote) is also a string. The token name for a string is tSTRING.
- An e-mail address is made up of a local-part, an @ sign and a domain. It can be represented as local-part@domain.
- The local-part can only contain letters, digits, hyphens, underscores and dots. However the dots cannot be the first or the last characters of the local-part and cannot appear consecutively. For example, the following lexemes would NOT be recognized as an e-mail address: .example@mail.com, example.@mail.com, ex..ample@mail.com.
- The domain name should consist of 2 or 3 dot-separated labels: xxx.xxx or xxx.xxx.xxx. Each label can contain letters, digits and hyphens. However hyphens cannot be the first or last character. For example,

the following lexemes would NOT be recognized as an e-mail address: example@-mail.com, example@mail-.com.

- The token name for an e-mail address is tADDRESS. Some examples are given below:

```
example@mail.com
john.smith@new-mail.com.tr
User--123@a-really-long-domain.name
12345@12345.12345
```

- A date value will be given in any one of the following 3 formats:

  1. DD/MM/YYYY (e.g. 11/10/2021).
  2. DD-MM-YYYY (e.g. 11-10-2021).
  3. DD.MM.YYYY (e.g. 11.10.2021).

  All day, month and year slots can take any digit. In other words, at least for this homework, we do not pay attention to the things like there are 12 months, there can be at most 31 days in a month, etc. Hence the following example would also be considered a date value: 45/98/0290 (similarly 45-98-0290 and 45.98.0290). However, mixed seperators is not allowed for a date value. Hence, the following inputs would not be considered a date variable: 11.10/2021 , 11-10.2021. All dates that are detected according to these rules should be printed out in the following format: `(Day_DD_Month_MM_Year_YYYY)` (for example 11/10/2021 seen in the input would be reported as `(Day_11_Month_10_Year_2021)`). The token name for a date value is tDATE.

- A time value will be given in the following format: HH:MM (e.g. 16:30). All hour and minute slots can take any digit. In other words, at least for this homework, we do not pay attention to the things like the hour part cannot be 34, or the minutes part cannot be 95. Hence the following example would also be considered a time value: 25:90. Time values should be printed out in the following format: `(Hour_HH_Minute_MM)` (for example, 09:05 seen in the input would be reported as `(Hour_09_Minute_05)`). The token name for a time value is tTIME.

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

5 Comments

Comments in a MailScript program are used to provide explanations, notes, or descriptions for the code and not executed by the 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:

```
set Message ("Hi, I will be there tomorrow!")
// This is a line comment
Mail from username@sabanciuniv.edu:
    schedule @ [02/02/2021, 21:00]: //Another line comment
        send [Message] to [(veli@mail.com)]
    end schedule
end Mail
```

Figure 1: An example MailScript program (`test19.ms`)

- Multiline comments: These comments begin with /* and end with */. Anything enclosed within this comment block is considered a multi-line comment. This type of comment can span multiple lines. If you encounter another opening comment tag (/*) inside an existing multi-line 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 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 as illegal characters.

Example:

```
set Message ("This message sent via MailScript program.")
/*
    This is a multiline comment
    /*
        This is a nested multiline comment
    */

*/
set Name ("Ali")
Mail from username@sabanciuniv.edu:
    schedule @ [04/07/2024, 00:00]:
        send [Message] to [(Name, deniz@mail.com.tr)]
    end schedule
end Mail

/* This is a multiline comment with no closing tag
/* The rest of the program is treated as comment

Mail from username@sabanciuniv.edu:
    schedule @ [04/07/2024, 00:00]:
        send ["Hi"] to [(selin@mail.com)]
    end schedule
end Mail
```

Figure 2: An example MailScript program (`test20.ms`)

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 for the contents of the comments.

6 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 see Section 7 to see how the position information is reported together with the token names.

7 Input, Output, and Example Execution

Assume that your executable scanner (which is generated by passing your flex program through flex and by passing the generated lex.yy.c through the C compiler gcc) is named as 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**
identifier	⟨*row*⟩_tIDENT_(⟨*lexeme*⟩)
date	⟨*row*⟩_tDATE_(⟨*lexeme*⟩)
time	⟨*row*⟩_tTIME_(⟨*lexeme*⟩)
string	⟨*row*⟩_tSTRING_(⟨*lexeme*⟩)
e-mail address	⟨*row*⟩_tADDRESS_(⟨*lexeme*⟩)
any other token	⟨*row*⟩_⟨*token_name*⟩
illegal character	⟨*row*⟩_ILLEGAL_CHARACTER_(⟨*lexeme*⟩)

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

- ⟨*row*⟩ should be replaced by the location (line number) of the first character of the lexeme of the token.
- ⟨*lexeme*⟩ should be replaced by the lexeme of the token. There is a special case that needs to be handled for the lexemes of strings: **The lexeme of a tSTRING token does not include the quotation marks, but only what is enclosed between the quotation marks**.
- ⟨*token_name*⟩ is the token name for the current item. These are the remaining, fixed–lexeme tokens, hence here you will use token names like `tMAIL`, `tFROM`, etc. based on what you detect.

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 there will be no spaces inserted in the output by yourself. Instead, we will use underscore characters, and we will use only one underscore character to separate two pieces of information. Also, do not insert unneccessary newline characters. Each token must be on a separate line, and 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 it may seem harmless to insert a space, or an additional underscore in the output, 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 `test17.ms` has the content as given in Figure 3:

```
Mail from beril@mail.com:

    set News ("Hi, I will not be joining today's meeting.")
    schedule @ [19/12/2021, 08:30]:
        send [News] to [("Tugce", tugce@company-mail.com)]
    end schedule

end Mail
```

Figure 3: An example MailScript program (`test17.ms`)

Then the output of your scanner must be:

```
1_tMAIL
1_tFROM
1_tADDRESS_(beril@mail.com)
1_tCOLON
3_tSET
3_tIDENT_(News)
3_tLPR
3_tSTRING_(Hi, I will not be joining today's meeting.)
3_tRPR
4_tSCHEDULE
4_tAT
4_tLBR
4_tDATE_(Day_19_Month_12_Year_2021)
4_tCOMMA
4_tTIME_(Hour_08_Minute_30)
4_tRBR
4_tCOLON
5_tSEND
```

```
5_tLBR
5_tIDENT_(News)
5_tRBR
5_tTO
5_tLBR
5_tLPR
5_tSTRING_(Tugce)
5_tCOMMA
5_tADDRESS_(tugce@company-mail.com)
5_tRPR
5_tRBR
6_tENDSCHEDULE
8_tENDMAIL
```

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

```
:: schedule ]Ali @ from Mail
example@mail.edu
send
99/99/9999
.example@mail.com
example@-mail.com
```

Figure 4: An example MailScript program (`test18.ms`)

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

```
1_tCOLON
1_tCOLON
1_tSCHEDULE
1_tRBR
1_tIDENT_(Ali)
```

```
1_tAT
1_tFROM
1_tMAIL
2_tADDRESS_(example@mail.edu)
3_tSEND
4_tDATE_(Day_99_Month_99_Year_9999)
5_ILLEGAL_CHARACTER_(.)
5_tADDRESS_(example@mail.com)
6_tIDENT_(example)
6_tAT
6_ILLEGAL_CHARACTER_(-)
6_tIDENT_(mail)
6_ILLEGAL_CHARACTER_(.)
6_tIDENT_(com)
```

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
multiple-line comment
  // This looks like a single-line comment,
  // but it is inside of another comment.
  /* Even though this is also a comment,
  it's within another comment. */
*/
:: schedule ]Ali @ from Mail // last comment
```

Figure 5: An example MailScript program (`test21.ms`)

Then your scanner output should be the following:

```
8_tCOLON
8_tCOLON
8_tSCHEDULE
8_tRBR
8_tIDENT_(Ali)
```

```
8_tAT
8_tFROM
8_tMAIL
```

8 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.

9 Notes

- **Important**: Name your file as you are told and **don't zip it.** **[-10 points otherwise]**
- Do not copy-paste MailScript program fragments from this document as your test cases. Copy/paste from PDF can create some unrecognizable characters. Instead, all MailScript codes fragments that appear in this document are provided as a text file for you to use.
- Make sure you print the token names exactly 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 TA or from your 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 flow.sabanciuniv.edu), there can be incompatibilities once you transfer them to flow.sabanciuniv.edu. Since the grading will be done automatically on the flow.sabanciuniv.edu, we strongly encourage you

to do your development on flow.sabanciuniv.edu, or at least test your code on flow.sabanciuniv.edu before submitting it. If you prefer not to test your implementation on flow.sabanciuniv.edu, this means you accept to take the risks of incompatibility. Even if you may have spent hours on the 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 mins of delay.
- We will not accept any homework later than 500 mins after the deadline.
- SUCourse timestamp will be used for STF computation.
- If you submit multiple times, the last submission time will be used.