CS305 – Programming Languages
Spring 2018-2019

Homework 1

Due date: March 4, 2019 @ 23:55

Implementing a Lexical Analyzer (Scanner) for CSML

1 Introduction

In this homework you will implement a scanner by using flex for an XML based language which we call ***CSML***, standing for *Course Schedule Markup Language*. The language is used to give the time table for courses scheduled at our university. In addition, some constraints can also be given on the schedule, such as two particular courses must not overlap (since there are students taking both of these courses).

The scanner you will implement will produce the tokens in a CSML file.

CSML has an XML based syntax. We will first briefly explain the XML based structure of CSML in general. An XML file consists of *element*s. For example in CSML, we have elements like *course*, or *meeting*, etc. Each element has a "start tag" and an "end tag". For an *element*, the start tag has the syntax `<element>` and the end tag has the syntax `</element>`. For example, the start tags and the end tags for elements *course* and *meeting* are `<course>`, `</course>`, `<meeting>`, `</meeting>`.

An element is everything between from start tag to end tag. For example, a *course* element is given in CSML as everything between the start tag `<course>` and the end tag `</course>`. Anything between `<course>` and `</course>` belongs to this *course* element.

In general for XML files, an element can contain other elements in it (e.g. `<course> ... <meeting> ... </meeting> <meeting> ... </meeting> ...</course>`). In addition, and element can have attributes, (e.g. `<course name="Programming Languages" code="CS305">`) and can have text. However, CSML elements do not contain text. You can simply assume that

there are only tags and attributes of tags in CSML files.

In addition to start and end tags of elements, we also have so–called *self-closing* tags in XML. The syntax for self–closing tags is `<element/>`. The self–closing tags can be used to introduce an instance of an element with no inner elements but with some attributes. As an example for this type of tag consider `<meeting time=08:40/>` which starts and immediately ends a *meeting* element (no other element is given inside this meeting element), but this meeting element has an attribute named *time* and the value of this attribute is 08:40.

Your scanner will catch the language constructs (introduced in Sections 2.1, 2.2, 3.1, 3.2) and print out the token and their positions (explained in Section 5) in the given CSML 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.

Below is a simple CSML file, providing examples of the kind of elements (i.e. tags) existing in CSML, together with examples of attributes of these elements.

```
<course code="CS305" name='Programming Languages' type="Lecture">
    <class section="0" instructor='Husnu Yenigun' crn=20258>
        <meeting day=R start=08:40 end=10:30/>
        <meeting start=08:40 end=09:30 day=F/>
    </class>
</course>
<course code="CS301R" name="Algorithms-Recitation" type="Recitation">
    <class section="0" instructor='Husnu Yenigun' crn=20257>
        <meeting start=17:40 day=M end=18:30/>
    </class>
</course>
<constraint>
    <item code="CS305"/>
    <item crn=20257/>
</constraint>
```

2 Tags

Each tag consists of tag names and optionally attributes. In this section you can get the information about tag symbols and tag names. Attributes are introduced in Section 3.

2.1 Tag Symbols

There are three types of tags: *start* (e.g. `<element>`), *end* (e.g. `</element>`) and *self-closing* (e.g. `<element/>`). Starting and ending symbols identify the type of tag. Therefore the analyzer will catch the symbols explicitly. Below is the list of the symbols and corresponding token names.

Lexeme	Token
`<`	`tOPEN`
`>`	`tCLOSE`
`</`	`tEND`
`/>`	`tSELF`

2.2 Tag Names

In CSML, tag names are predefined. Below is the list of tag names and token names that will be implemented.

Lexeme	Token
`course`	`tCOURSE`
`class`	`tCLASS`
`meeting`	`tMEETING`
`constraint`	`tCONSTRAINT`
`item`	`tITEM`

3 Attributes

Attributes have three parts: keyword, assignment operator and value. In this homework each attribute consists of two tokens: keyword and value tokens. Keyword and assignment operator will be caught as one token. Please see Section 3.1, 3.2 for details.

3.1 Attribute Keywords

Below is the list of attribute keywords that will be implemented.

`name`	`code`	`instructor`	`type`	`crn`
`section`	`capacity`	`start`	`end`	`day`

Since there is no token for the assignment operator, this operator is assumed to be a part of the keyword. We will also use the convention that the token name for a keyword is formed by appending upper case keyword to `t`. For example, for the keywords `name=` and `start=`, we will use the token names `tNAME` and `tSTART`, respectively.

3.2 Attribute Values

In this homework there are four different types of attribute values: string, number, time, day. The rule for string is the following:

> A string starts with quotation mark and ends with same mark.
> It consists of alpha numeric words (e.g. `"PL"`, `'CS 305'`, ...).

The token name for a string is `tSTRING`.

You need to implement positive integers as `tNUM`.

Time will be between from 00:00 to 23:59 and the corresponding token name will be `tTIME`.

Lexemes and token names for days are listed below:

Lexeme	Token
`M`	`tMON`
`T`	`tTUE`
`W`	`tWED`
`R`	`tTHU`
`F`	`tFRI`

Moreover, for all values you need to store the actual lexemes associated with the tokens. You are required to output the lexemes for the identifiers and the positions of these lexemes. For numbers you will output both the lexemes and the corresponding values along with their positions in your output (See Section 6).

4 Complete List of Tokens and Lexemes

Token	Lexeme	Token	Lexeme
tOPEN	<	tCLOSE	>
tEND	</	tSELF	/>
tCOURSE	course	tCLASS	class
tMEETING	meeting	tCONSTRAINT	constraint
tITEM	item	tNAME	name=
tCODE	code=	tINSTRUCTOR	instructor=
tTYPE	type=	tCRN	crn=
tSECTION	section=	tCAPACITY	capacity=
tSTART	start=	tEND_A	end=
tDAY	day=	tSTRING	anything in quotations
tMON	M	tNUM	any positive integer
tTUE	T	tWED	W
tTHU	R	tFRI	F
tTIME	Any time between 00:00 and 23:59		

5 Positions

You must keep track of the position information. For each token that will be reported, the position (line number) of the first character of the lexeme of the token is considered to be the position of the token. Please see 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 by passing the generated `lex.yy.c` through the C compiler `gcc`) is named as `CSMLscanner`. Then we will test your scanner on a number of input files using the following command line:

```
CSMLscanner < test17.csml
```

As a response, your scanner should print out a separate line for each language construct it catches in the input file (`test17.csml` in the example above). The output format for a single is given below for each token separately:

Token	**Output**
string, number, time	$\langle row \rangle \langle space \rangle \langle token\_name \rangle \langle space \rangle (\langle lexeme \rangle)$
others	$\langle row \rangle \langle space \rangle \langle token\_name \rangle$

Here, $\langle row \rangle$ gives the location of the first character of the lexeme of the construct and $\langle token\_name \rangle$ is the token name for the current item. $\langle space \rangle$ just denotes that you will print out a space characters, and $\langle lexeme \rangle$ will display the lexeme of the token. For string tokens quotation mark will be eaten up.
For example let us assume that `test.csml` has the following content:

```
<course code="CS305">
</course>
```

Then the output of your scanner must be:

```
1 tOPEN
1 tCOURSE
1 tCODE
1 tSTRING (CS305)
1 tCLOSE
2 tEND
2 tCOURSE
2 tCLOSE
```

Note that, the content of the test files need not to be a complete or grammatically correct CSML file for this homework. If the content of a test file is the following:

```
</ course />
```

(which is not well–structured CSML file but everything in it can be converted into valid CSM tokens) then your scanner should not complain about anything and output the following information:

```
1 tEND
1 tCOURSE
1 tSELF
```

However, you can assume that we will use test files that consist of lexemes of valid tokens only. For example, we will not try to fool your scanner by providing a text like `"sample text'` which is against the rules explained above for forming a valid string.

7 How to Submit

Submit only your flex file on SUCourse. The name of your flex file must be

`id-hw1.flx`

where `id` is your five digit student id.

8 Notes

- **Important**: SuCourse's clock may be off a couple of minutes. Take this into account to decide when to submit.
- No homework will be accepted if it is not submitted using SUCourse.
- Note that, Windows ports of flex are available, either native or based on `cygwin`. Although it is discouraged, you may use Windows version on your PC. However, we want to remind you that, your homework will be evaluated on the machine, `flow.sabanciuniv.edu`. Hence we recommend that you, at least, test your implementation on flow before submitting.
- 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.