Sabancı University
Faculty of Engineering and Natural Sciences

CS305 Programming Languages

Homework 3

Due: 18-03-2019

1 Introduction

In this homework you will implement a simple parser for CSML language. The parser will check if a given CSML file has any syntax error with respect to the grammar given in Section 3.

2 The Language

The CSML language for which you will implement a parser is described in this section. Here is an example CSML file in this language to give you an idea how a CSML file looks like.

```
<course code="CS305" name='Programming Languages' type="Lecture">
  <class section="0" instructor='Husnu Yenigun' crn=20258 capacity=60>
    <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>
```

Below is the detailed syntactic features of the CSML language.

1. A CSML *program* consists of a list of top level *elements*.
2. The sequence of top level elements maybe empty, or contains one or more top level elements.

3. A top level element is either a *course element*, or a *constraint element*.

4. A *course element* consists of a *course opening tag* and a *course closing tag*. Between a *course opening tag* and a *course closing tag*, there is non–empty list of *class* elements.

5. A *course opening tag* consists of a `tOPEN` token, `tCOURSE` token, a non–empty list of *course attributes* and a `tCLOSE` token.

6. A *course attribute* gives either

   - the code of the course (e.g. `code="CS305"`), or
   - the name of the course (e.g. `name="Programming Languages"`), or
   - the type of the course (e.g. `type="Lecture"`)

   Note that, your grammar should allow these attributes to be given in any order, it should not force every attribute to be used (i.e. it is okay if some attributes are missing), and it should not restrict the multiple use of an attribute (i.e. it is okay if an attribute is seen more than once).

7. A *course closing tag* is simply `</course>`.

8. A *class* element consists of a *class opening tag* and a *class closing tag*. Between a *class opening tag* and a *class closing tag*, there is a non–empty list of *meeting* elements.

9. A *class opening tag* consists of a `tOPEN` token, `tCLASS` token, a non–empty list of *class attributes*.

10. A *class ending tag* consists of `tCLOSE` token.

11. A *class attribute* gives either

    - the section information for the class (e.g. `section="0"`), or
    - the instructor information for the class (e.g. `instructor="Husnu Yenigun"`), or
    - the CRN of the class (e.g. `crn=20258`), or
    - the capacity of the class (e.g. `capacity=60`)

    Note that, your grammar should allow these attributes to be given in any order, it should not force every attribute to be used (i.e. it is okay if some attributes are missing), and it should not restrict the multiple use of an attribute (i.e. it is okay if an attribute is seen more than once).

12. A *class closing tag* is simply `</class>`.

13. A *meeting element* consists of a `tOPEN` token, `tMEETING` token, a non–empty list of *meeting attributes* and a `tSELF` token.

14. A *meeting attribute* gives either

    - the start time information for the meeting (e.g. `start=17:40`), or
    - the end time information for the meeting (e.g. `end=18:30`), or
    - the day information for the meeting (e.g. `day=M`)

    Note that, your grammar should allow these attributes to be given in any order, it should not force every attribute to be used (i.e. it is okay if some attributes are missing), and it should not restrict the multiple use of an attribute (i.e. it is okay if an attribute is seen more than once).

15. A *constraint element* consists of *constraint opening tag* (which is simply `<constraint>`) and a *constraint closing tag* (which is simply `</constraint>`). Between a *constraint opening tag* and a *constraint closing tag*, there is be non–empty list of *item* elements.

16. An *item element* consists of a `tOPEN` token, `tITEM` token, an *item attribute* and a `tSELF` token.

17. An *item attribute* gives either

    - a code (e.g. `code="CS305"`), or
    - a CRN (e.g. `crn=20257`)

A context free grammar for the syntax explained above given in Section 3.

3 CFG for the CSML language

In this section, we give the context free grammar that you will use.

```
prog              -> elementList
elementList       -> ϵ | element elementList
element           -> beginCourse  classList endCourse
                  |  beginConstraint itemList endConstraint
beginCourse       -> tOPEN tCOURSE  courseAttrList  tCLOSE
endCourse         -> tEND tCOURSE tCLOSE
courseAttrList    ->  courseAttr  | courseAttr courseAttrList
courseAttr        -> tCODE tSTRING | tNAME tSTRING |tTYPE tSTRING
classList         ->  class |  class classList
class             -> beginClass classAttrList endClass  meetingList closeClass
beginClass        -> tOPEN tCLASS
endClass          -> tCLOSE
closeClass        -> tEND tCLASS tCLOSE
classAttrList     -> classAttr | classAttr classAttrList
classAttr         -> tSECTION tSTRING
                  |tINSTRUCTOR tSTRING
                  |tCRN tNUM
                  |tCAPACITY tNUM
meetingList       ->  meeting |  meeting  meetingList
meeting           ->  beginMeeting  meetingAttrList  endMeeting
beginMeeting      -> tOPEN tMEETING
endMeeting        -> tSELF
meetingAttrList   -> meetingAttr |  meetingAttr  meetingAttrList
meetingAttr       -> tDAY day | tSTART tTIME | tEND_A tTIME
day               -> tMON | tTUE | tWED | tTHU | tFRI
beginConstraint   -> tOPEN tCONSTRAINT tCLOSE
endConstraint     -> tEND tCONSTRAINT tCLOSE
itemList          ->  item | item  itemList
item              ->  beginItem  itemAttr  endItem
beginItem         -> tOPEN tITEM
endItem           -> tSELF
itemAttr          -> tCODE tSTRING | tCRN tNUM
```

4 Scanner

In this section we give the token specifications that you will use to implement your scanner.

Regular expression	Token
"<"	tOPEN
">"	tCLOSE
"/>"	tSELF
"</"	tEND
course	tCOURSE
meeting	tMEETING
item	tITEM
code=	tCODE
section=	tSECTION
start=	tSTART
day=	tDAY
type=	tTYPE
constraint	tCONSTRAINT
name=	tNAME
instructor=	tINSTRUCTOR
crn=	tCRN
end=	tEND_A
capacity=	tCAPACITY
class	tCLASS
M	tMON
T	tTUE
R	tTHU
W	tWED
F	tFRI
([0-1][0-9]|2[0-3]):[0-5][0-9]	tTIME
'[^']*'	tSTRING
["][^"]*["]	tSTRING
0|([1-9][0-9]*)	tNUM

5 Output

Your parser must print out `OK` and produce a new line, if the input is grammatically correct. Otherwise, your parser must print out `ERROR` and produce a new line.

6 How to Submit

Submit your bison file named as `id-hw3.y`, and flex file named as `id-hw3.flx` where `id` is your student ID. **Do not compress** your flex and bison files. We will compile your files by using the following commands:

```
    flex id-hw3.flx
    bison -d id-hw3.y
```

```
gcc -o id-hw3 lex.yy.c id-hw3.tab.c -lfl
```

So, make sure that these three commands are enough to produce the executable parser. If these commands are not enough to produce an executable on flow machine, then you will get 0 grade from this homework. If we assume that there is a text file named test, we will try out your parser by using the following command line:

```
id-hw3 < test
```

If the file `test` includes a grammatically correct CSML file then your output should be `OK` otherwise, your output should be `ERROR`.

7 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.
- You must write your files by yourself.
- Start working on the homework immediately.