COMP314W2
Assignment 1 - 2019

1. Purpose

The purpose of this assignment is to write a Java program which will allow a *s-grammar* (see slide 15 of the CFL slide set) G = (V,T,S,P) and a string to be inputted. The program should check that the grammar is a valid *s-grammar* and then parse the string to determine whether the string is a member of the language that is defined by the grammar.

All elements in the set V of variables and set T of terminals consist of **single** characters. Variables in V consist of upper-case alphabetic characters. Terminals in T consist of lower-case alphabetic characters and special characters like { = + etc. The set P of production rules do not include ϵ-productions, but otherwise can include any valid *s-grammar* rules.

2. Program Details

The program must have a GUI interface which will allow the grammar to be inputted. The program should then check that the grammar is a valid *s-grammar*. If it is, strings should be inputted and the program should then parse them, i.e. check whether they belong to the language that the grammar defines, by performing a leftmost derivation on them. The program must also have a facility to display each step of the leftmost derivation as it is occurring.

You might find it useful to check out *JFlap*'s *Grammar* option which has quite a nice interface. Any *s-grammar* (like grammars G_1 and G_2 in section 3 below) is actually usable in *JFLap* as an LL(1) grammar, since *s-grammars* are special cases of LL(1) grammars.

In addition, *s-grammars* and strings to be parsed must be able to be read in from a data file.

You should use a common internal representation for *s-grammars* and strings to be parsed, so that it will make no difference to your program whether they are inputted via the GUI interface or via a data file.

3. Testing

You must test your code on the following two *s-grammars*, though of course you can (and should) create or find others as well on which to test your code. Note that all productions in these grammars follow the requirements for *s-grammars* in that

- there is a single variable on the left hand side of a production.
- the right hand side of a production starts with a single terminal and is followed by 0 or more variables.
- if there are two or more productions with the same variable on the left hand side, the right hand of each of these productions starts with a different terminal.

G_1 is a *s-grammar*

$$G_1 = (\{ S,A,B \}, \{ a,b \}, S, P)$$

for the language $L = \{a^n b^{n+1} : n \geq 1\}$, with the elements of P being the productions

S	→	aAB
A	→	aAB | b
B	→	b

G_2 is a *s-grammar*

G_2 = ({ S,R,L,C,A,Y,P,Z,X,T,D }, { {,},x,=,y,+,z,i,t,e,d }, S, P }

for a fragment of a programming language, with the elements of P being the productions

S	→	{LR
R	→	}
L	→	xAYPZ | iXTLC
C	→	d | eLD
A	→	=
Y	→	y
P	→	+
Z	→	z
X	→	x
T	→	t
D	→	d

A program fragment takes the form

{ *Stmt* }

where *Stmt* can have the one of the forms

(i) `x = y + z`

(ii) **`if`** `x` **`then`** *`Stmt`* **`endif`**

(iii) **`if`** `x` **`then`** *`Stmt`* **`else`** *`Stmt`* **`endif`**

E.g a program fragment might be

```
{ if x then
    x = y + z
  else
    if x then
      x = y + z
    endif
  endif }
```

Obviously, since both variables and terminals are specified to consist of single characters only, any multi-character variables or terminals like *Stmt* or **endif** must be represented by single characters. The following scheme is used

word	**symbol**
Stmt	L
if	i
then	t
else	e
endif	d

The above program fragment is then represented by the string

{ixtx=y+zeixtx=y+zdd}

You could decide in your implementation that whitespace is not significant within strings to be parsed. If that were the case, then the string might have the (slightly) more user-friendly format

{ i x t x=y+z e i x t x=y+z d d }

4. Data Structures and Algorithms

In any leftmost derivation, the essential requirement is to replace the leftmost variable in the current sentential form with the right hand side of a production with that variable on the left hand side. In an *s-grammar*, the choice of which production to use is made very simple. If *a* is the next symbol in the input string being parsed, and *A* is the leftmost variable in the current sentential form, we have to find a production with *A* on the left hand side whose right hand side starts with *a*. By the nature of an *s-grammar* there can be only one such production. If there is no such production the string cannot be a member of the language that the grammar defines.

Design decisions you should consider include

- how sentential forms should be represented so that it is easy to find and replace the leftmost variable in the sentential form.

- how productions should be represented so that it is easy to find the production that is to be used in the replacement of the leftmost variable in the sentential form. The restriction that variables must consist of uppercase single characters means there can be no more than 26 variables here, but in a real grammar there may be dozens or even hundreds of variables. A method of representing and retrieving productions efficiently is therefore important.

5. Submission

The completed assignment is to be submitted as a single exported *Eclipse .zip* project file, though of course you can have multiple classes within the project. You must ensure that your project file can be imported into *Eclipse* and run without any changes needing to be made to it. The markers don't have the time to do this for you! The name of your project file **must** be *yoursurname_yourinitials_yourregno_Assignment_1.zip*. The main method through which the assignment is run **must** be in a class called *RunAssignmnent*.

You will need to submit the properly documented source code of your classes together with a record of the testing you undertook to show the correctness of your code, including the data files. These must all be in the *Eclipse* project file. Submit through *Moodle* by clicking on the *Assignment 1* topic on the front page and uploading your single zipped project file. You can resubmit as often as you like, but only the last submission is kept.

The deadline for submission is 23H55 on 29 September.