CS4158: A Parser for JIBUC Programs
Project 2018

Overview
Your task is to develop a Parser (and associated Lexical analysis tool) for a syntactically limited language. This tool should report back to tell the programmer when the program is correctly and incorrectly formed with respect to its structure. **It only needs to analyse the program, not create an executable.**

You should use either [Lex and Yacc] or [Flex and Bison] to develop this parser. It should run on Red Hat Linux or Windows and a Gnu C or C++ compiler should be used.

Language Syntax:

The language is not case sensitive. Programs are monolithic starting with the keyword "START." and ending with the keyword "END-NOW." (the full stops are important)

Statements in the program can be divided into declarations, assignments, inputs, and outputs.

Declarations are contained just below the START. keyword. Only variables of type double are supported in JIBUC. In the individual declaration of each variable, the first string defines the capacity of the double being declared. This string will always be represented by one of more X's, with possibly a dash in the middle. For example a variable declared as XXX specifies that a three-digit whole number can be held in the variable. A XXX-XX specifies that a three-digit number with two decimal places can be held in the variable.

The identifier follows the size string and is the name of the variable being declared. It will be any combination of alphabetic characters, dashes and digits, under seven characters in length, as long as it starts with a character and is not a series of contiguous X's. Examples include:

- XXX G56-MH
- XX-XX HTY
- XXX J-
- XXXX-X U6787-77

After the declarations, comes the main body of code, signified by the keyword "MAIN.". This body of code consists of one or more statements after this keyword. The program ends with the "END-NOW." statement.

The *assignment* statements are of either of the following forms:

- identifier EQUALS-TO identifier.
- Identifier EQUALS-TO-VALUE double.

Here a value (specified by the double after the EQUALS-TO-VALUE or the identifier after the EQUALS-TO keyword) is assigned to the variable, identified by the identifier at the start of the statement. Alternatively, an *assignment* statement can be of the forms:

- ADD double TO identifier.
- ADD identifier TO identifier.

The *input* statement is of the following form:

- INPUT identifier1; identifier2; identifier3.

This read statement above will take in 3 values which will be held in the variables identified by identifier1, identifier 2 and identifier3. However read statements are not just limited to taking in 3 values. They may take in any number of values required by the program.

The *output* statement is of the following form

- PRINT “I’m printed out”; identifier1.

It starts with the keyword ‘Print’ and then contains any combinations of either identifiers or text (enclosed in quotation marks), separated by semi-colons. The print statement above prints out the string ‘I’m printed out’, followed by whatever value is held by identifier 1.

A *valid example program* is as follows

```
START.
XX-XX Y-S.
XX-XX Z.
X-XX XY-1.

MAIN.
PRINT “Please enter a number? ”.
INPUT Y-S.

Z EQUALS-TO 15.00.
ADD Y-S TO Z.
PRINT “Result is ”; Z.

END-NOW.
```

A in*valid example program* is as follows: the printed variable XY is not declared

```
START.
XXX XY-1.
XXXX Y.
XXXX Z.

MAIN.
PRINT “Please enter a number? ”.
INPUT Y.

Z EQUALS-TO Z.
ADD Y TO Z.
PRINT XY;” + “;Y;”=”;Z.

END-NOW.
```

(note that I have used all capitals in my examples for legibility, but programmers are not required to do so)

Deadlines and Marks:
This project should be completed after 5 weeks. It is intended that you do this in your own time and in the lab hours that are associated with the module. During the lab hours (weeks 7 to the start of week 12) Ashish will be there to offer support***, but you should have tried to work ahead in advance of the lab sessions so that you make the most of your time with him.***

By your 2nd practical (Tuesday week 8, or Monday week 9), you should have completed the lexical scanner. That is, you should be able to produce a lexical analysis tool that identifies all the tokens of the language, showing this by printing out the token type on its recognition. Ashish will assess this (ONLY) during the practical hours of Tuesday week 8 or Monday week 9 by bringing some JIBUC program files to the session and asking you to run them through your 'lexer'. He will mark your efforts out of 8.

By Tuesday week 12, you will be expected to have completed the parser. The parser should report that it has been presented with a well formed/not well formed program. It should also flag an error if the program attempts to assign a value to a variable that is not declared or assign a value to a variable which is bigger than its declared capacity. For example, using the program above, if a program tried to 'MOVE 5000.00 TO Y-S' a warning flag should be raised (as Y-S is only declared as 'XX.XX'). For top marks the parser should detect if, when you move a value from an identifier 1 to an identifier 2, identifier 1 is declared to be inconsistent with the size of identifier 2 and issue a warning. (in the example above 'MOVE Z to XY-1' should cause this). Note that all assignments to variables, either from another identifier or from a double have to be an exact match for the receiving variable's size.

Ashish will assess the completed parser (again ONLY) during Tuesday of week 12. He will work with the class to identify suitable times for assessment and mark your efforts out of 12. He will then combine your 2 marks to give a total mark for the project out of 20.

Schedule:

Tuesday Week 7, Monday week 8	Independent work towards lexer
Tuesday Week 8, Monday week 9	Assessment of lexer by Ashish
Tuesday Week 9, Monday week 10	Independent work towards complete parser
Tuesday Week 10, Monday week 11	Independent work towards complete parser
Tuesday Week 11, Monday week 12	Independent work towards complete parser
Tuesday Week 12	Assessment of parser by Ashish