CS4158: A Parser for BUCOL Programs
Project 2023

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 and report back, **not create an executable**.

You should use either Lex and Yacc or Flex and Bison to develop this parser. It can be 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". As each phrase/line ends with a full-stop the first line of a program will be "START." And the last line will be "END."

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 integer are supported in BUCOL. In the individual declaration of each variable, the first string defines the capacity of the integer being declared. This string will always be represented by one of more S's. For example a variable declared as SSS specifies that a three-digit number can be held in the variable. A SSSSS specifies that a five-digit number can be held in the variable.

The identifier follows the size string, and is the name of the variable being declared. It can be any combination of alphabetic characters, digits, and hyphens, as long as it starts with a character, or an underscore and then a character, and is not a series of contiguous S's. Examples include:

- SSS G56-MH.
- SSSS _HTY.
- SSS J-.
- SSSSS S6787-7XX7.

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." statement.

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

- MOVE identifier TO identifier.
- MOVE integer TO identifier.

Here a value (specified by the integer or identifier after the MOVE keyword) is assigned to the variable, identified by the identifier at the end of the statement. Alternatively, an *assignment* statement can be of the forms:

- ADD integer 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.
SSS XY-1.
SSSS _Y.
SSSS Z.

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

MOVE 15 TO Z.
ADD Y TO Z.
PRINT XY-1;” + “;Y;”=”;Z.

END.
```

A in*valid example program* is as follows: there is a variable named SS and the printed variable XY is not declared

```
START.
SSS XY-1.
SSSS Y.
SSSS Z.
SSSS SS.

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

MOVE 15 TO Z.
ADD Y TO Z.
PRINT XY;” + “;Y;”=”;Z.

END.
```

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. Abdul will be there to offer support on this week and week 8, 10 and 11 (Thursday 09:00-11:00, CS3004B) and provide email support outside of those hours***, but you should have tried to work ahead in advance of the lab sessions/emails so that you make the most of your time with him.***

By Tuesday 17:00 of week 9, you should have completed the lexical scanner. That is, you should have produced a lexical analysis tool that identifies all the tokens of the language, showing this **by printing out the token types on recognition**. Abdul will assess this via your SULIS submission. He will mark your efforts out of 8.

By week 12, you will be expected to have completed the parser. The parser should report when 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 500000 TO Y' a warning flag should be raised (as Y is only declared as 'SSSS'). 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 larger than identifier 2 and issue a warning. (in the example above 'MOVE Z to XY-1' should cause this).

Abul will assess the completed parser (again ONLY), via your SULIS submission by Tuesday week 12, 17:00. He will mark your efforts out of 12. He will then combine your 2 marks to give a total mark for the project out of 20.

Please note that Abdul reserves the right to get in touch with you over email or face-to-face to check details of your implementation if he sees fit.

Schedule:

Week 7-8	Independent work towards lexer creation, with supporting labs
Week 9	Assessment of lexer by Abdul (Tuesday 17:00 submission deadline)
Week 10-11	Independent work towards complete parser with supporting labs
Week 12	Assessment of parser by Abdul (Tuesday 17:00 submission deadline)
On-going	Ocassional email support from Abdul at Abdul.Qayum@ul.ie