Faculty of Engineering and Technology

Coursework Title: CalcX Language Interpreter Referral
A report and interpreter for an extension of a basic language (**Individual Work**)

Module Name:	Programming Language Theory
Module Code:	5129COMP
Level:	5
Credit Rating:	20
Weighting:	60% of module
Maximum mark:	100
Lecturer:	Dr Paul Bell
Contact:	If you have any issues with this coursework you may contact your lecturer. Contact details are: Email: p.c.bell@ljmu.ac.uk Room: 7.33
Hand-out Date:	17/02/2020
Hand-in Date:	~~03/07/2020 5pm~~ 07/08/2020 5pm
Hand-in Method:	Electronic submission via VLE, details below
Feedback Date:	Results notification day
Feedback Method:	Individual comments to each student, sent via VLE or Email

Introduction

This assignment is an **individual task and is different from the main coursework,** please read all details of this coursework spec. You will expand a given lexical analyser/parser (written in Java) to handle a simple language (CalcInt), as well as expanding a context free grammar for the language's specification. You should follow an established Software Development Lifecycle to design and develop an efficient solution to the problem specification.

Learning Outcomes to be assessed

LO1. Explain the key concepts in specifying and evaluating a programming language
LO2. Apply appropriate formal methods to specify a programming language
LO3. Synthesise appropriate algorithms and data structures to fulfil a problem specification
LO4. Appraise Imperative and Declarative programming paradigms as an appropriate mechanism for a variety of problem domains

Detail of the task

Problem Specification – CalcX Language

You will design and implement an extension to an interpreter for an extension to the CalcInt Language that is specified below. You are provided with an interpreter, written in Java, which is an interpreter for the current (unextended) version of CalcInt. You should use the provided Java interpreter (in CalcX.zip on Canvas) as a basis for your coding.

Upon entering source code into your program, the output that should be produced is:

1. Either:
    a. "Lexical analysis successful.\n" if the program is correctly passed through the lexical analyser, at which point it should output a list of tokens;
    b. "Error in lexical analysis of program.\n" after which your program should stop.
2. If the program passes lexical analysis, your program should output:
    a. "Parsing successful.\n" if the parsing phase of your interpreter successfully parses the output of your lexical analyser, otherwise:
    b. "Error in parsing of the program.\n" after which your program should stop.
3. If the input program successfully parses, and there are no other errors, then your program should interpret the parsed program and output the result.

CalcInt language specification

We first describe the CalcInt language informally before giving a context free grammar (CFG) and syntax directed translation scheme (SDTS) for it. You will be designing and implementing an interpreter for an extended version of CalcInt. This will involve modifying the CFG/SDTS and modifying the code of the interpreter in Java. You may make additional assumptions to the details given below, but you should make any such assumptions clear in your report.

Keywords in CalcInt Language are written entirely in upper-case (e.g. INT), whereas variable names are lower-case letters (such as "x", "hello" or "temp"). This makes lexical analysis slightly easier than otherwise. Whitespace is ignored in CalcInt (spaces, tabs or newlines).

CalcInt only contains the following data type:

- INT – Unsigned integer data type (e.g. 42, 0, 900, …) with the same range as the standard unsigned INT data type in Java.

Variables and their type must be declared first, *at the start of a program* and assigned an unsigned integer value. A semi-colon is used after the declaration of each variable. We will not be concerned with more complicated variable scope during this coursework. Attempting to declare an already declared variable should produce an error (e.g. "INT x = 7; INT x = 3;"). CalcInt language allows no user input, all data must be declared within the program source code directly.

CalcInt only has the following operations defined on variables:

- The two arithmetic operations of addition and subtraction (denoted "+" and "-" respectively).

There are no order of precedence of operations other than expressions are evaluated right to left, so an expression like "x+y-z+y" would be evaluated as x+(y-(z+y)). There is also no brackets (parenthesis) in the language.

A program (prog) in CalcInt is made up of declarations (decls), with each declaration (decl) being a single variable, which is assigned an integer value. The assignment operator is denoted '='. Each declaration is ended by a semicolon ';'. After the declarations, we have a colon character ':' followed by a single expression (expr).

An expression (expr) is a sequence of lexemes, each of which is a variable name or an operation, written in infix notation (for example x+y-z+x). Therefore, each operation has a variable on its left and right. Numerical values are not allowed in expressions, only variables and operations (so 'x+y-z' is fine, but not 'x+y-3' for example). At the end of the expression is a single '$' character to denote the end of the 'program'.

The following is an example valid program written in CalcInt.

INT x=7; INT y=3; : x+x-y$

The output of the program is to evaluate the expression according to the precedence rules explained earlier (right-to-left only). So for example, the program just written would produce output "11", since 7+(7-3) is 11. Remember that $ denotes the end of the program and does not affect the output.
We now give a Context Free Grammar (CFG) for CalcINT.

prog	→	decls ':' expr '$'
decls	→	decl decls
	|	ε
decl	→	'INT' var '=' intnum ';'
expr	→	var op expr
	|	var
var	→	[a-z]
intnum	→	[0-9][0-9]*
op	→	'+' | '-'

For example, a program (prog) is made up of declarations (decls), followed by a colon, followed by an expression (expr), followed by a dollar symbol etc.

We may also write this as a Syntax Directed Translation Scheme (SDTS) by adding semantic actions to be carried out at each stage of parsing (next page):

prog	→	decls ':' expr '$'	{ Print expr.val(); }
decls	→	decl decls	
	|	ε	
decl	→	'INT' var '=' intnum ';'	{ var.val() = intnum.val(); }
$expr_1$	→	var op $expr_2$	{ $expr_1$.val() = var.val() op $expr_2$.val(); }
	|	var	{ $expr_1$.val() = var.val(); }
var	→	[a-z]	
intnum	→	[0-9][0-9]*	
op	→	'+' | '-'	

You are provided with a Java interpreter which performs lexical analysis on an input string, creates a sequence of tokens (if the program is valid) and then parses the string, to output the correct value. You should run the program and pass in a few programs so see the output, as well as reading and understanding the code. The code is intentionally quite basic and does not output particularly helpful error messages or check for all possible incorrect programs.

Tasks

You are required to complete the following tasks as part of this coursework. Even if you plan to submit all tasks, it is *strongly recommended* to begin with the earlier tasks and build on these to reach the later tasks. Even if your code does not work correctly/compile, you should still submit it and may gain partial marks. In the details below, "*Report*" denotes tasks that should be included in your report and "*Program*" denotes something that you should program, by modifying/extending the given java files (contained within CalcX.zip). You should use separate sections in your report for each answered question (e.g. Basic I, Basic 3 etc.)

You should note that there are differences in several of the tasks compared with the non-referral coursework. You may modify your original coursework submission if you wish, in order to satisfy the requirements of the tasks given below. Please ensure that you check each task for any changes from the non-referral coursework.

Basic:

1. *Report*:
   a) Draw a parse tree for the following program by using the context free grammar of CalcInt given previously.
      INT hello = 5; INT test = 7; : hello - test $
   b) Explain whether the given context free grammar for CalcInt is *left-recursive*.
   c) Describe the difference between an *imperative* and *declarative* language and give two examples of each.
   d) Include in your report a class diagram for the CalcInt interpreter given (in CalcX.zip).

CalcInt Extension:

2. *Report&code*: In this task you will **extend the CalcInt language and interpreter**. You should add **two** new operations to the Context Free Grammar & Syntax Directed Translation Scheme (language design) and to the (Java) interpreter for CalcInt. These two operations are exponentiation "^" and absolute difference, denoted "~" which work on unsigned integers and which are applied in a right-to-left manner (as currently). Exponentiation is used to take one integer to the power of another (i.e.

2^4 = 2*2*2*2) and absolute difference works like subtraction followed by "absolute value" (so 2~5 = |2-5| = 3).

Secondly, you should modify the language design and then the interpreter so that expressions can include unsigned integers as well as variables as *part of the expression*. Therefore, the following code examples should all be valid in your new language:

- *INT x = 5; INT y = 7; : x^x~y $*
- *INT value = 5; : value + value^2 $*

In your report, you should copy the CFG&SDTS given previously and update them so that the new operations and the new form of the expressions are included. You should then modify the Java code of the interpreter to handle these changes. You will therefore need to modify the TokenType enum, the Lexical Analyser code and the Parser (as a minimum).
For additional marks, your interpreter should output appropriate and useful error messages for a variety of invalid program inputs.

Advanced:

3. *Report&code*: For this task, you should further extend CalcInt (both its description via the context free grammar & syntax directed translation scheme and the interpreter) to allow the following modification (you can modify the CFG&SDTS and interpreter that were already updated in Task 2 for this purpose): *CalcInt should allow multiple expressions, one after the other, rather than a single expression as currently*. Each expression should be separated by the next one by a colon. The following program shows an example of a program in this new language, containing three expressions:

   INT x = 6; INT y = 3; : x + y : x ~ x + y: y + y $

   The program should output the value of each of these expressions, separated by spaces. Therefore the program just given would output "9 3 6" for example.

Marking Summary

Depending on the tasks you attempt, this will necessarily limit the maximum marks available in the relevant components. Please consult the separate rubric for a detailed breakdown, but as a rough guide:-

- Attempting all of the **basic level** tasks only will limit your maximum project grade to 25%
- Attempting also the **intermediate** task will limit your maximum project grade to 65%
- Attempting the **advanced** task will allow you to be graded > 65%

Testing

Your program submissions will be tested in two ways. Firstly, some programs written in CalcInt Language will be entered into your interpreter, to determine if it correctly scans, parses and outputs the correct solution. Some of the input programs will purposely contain errors, to determine if your program correctly halts and outputs a reasonable error message. Secondly, your code will be read to determine how well it is written. If your program does not correctly scan, parse or interpret one of the scripts, then the marker will attempt to determine why, and partial marks will be given for reasonable attempts, so *you should still submit your code even if it does not work*.

What you should hand in

You should submit a zipped folder containing your single report (in PDF), together with a folder called CalcInt containing all your Java source files (as an Eclipse project). You should use the supplied Java source files (in CalcX.zip) to start you off, and extend them as necessary.

Marking Scheme/Assessment Criteria

Task	Assessment Criteria	% weighting
1	*a) Parse tree* *b) CalcInt left-recursion* *c) Declarative versus imperative languages* *d) Class diagram of CalcInt*	**10** **5** **5** **5**
2	**Design and implementation of CalcInt extension** *a) Context Free Grammar and Syntax Directed Translation Scheme for extended CalcInt language* **b)** *Implementation of extended CalcInt language interpreter in Java*	 **15** **25**
3	**Design and implementation of CalcBool language** *c) Context Free Grammar and Syntax Directed Translation Scheme for CalcBool language* **d)** *Implementation of CalcBool language interpreter in Java*	 **15** **20**
	TOTAL	**100**

Recommended reading:

Lecture slides, practical sheets and course textbook ("Compilers principals, techniques, and tools", Aho, Lam, Sethi, Ullman).

Extenuating Circumstances: If something serious happens that means that you will not be able to complete this assignment, you need to contact the module leader as soon as possible. There are a number of things that can be done to help, such as extensions, waivers and alternative assessments, but we can only arrange this if you tell us. To ensure that the system is not abused, you will need to provide some evidence of the problem. More guidance is available at https://www.ljmu.ac.uk/about-us/public-information/student-regulations/guidance-policy-and-process

Any coursework submitted late without the prior agreement of the module leader will receive 0 marks.

Academic Misconduct: The University defines Academic Misconduct as 'any case of deliberate, premeditated cheating, collusion, plagiarism or falsification of information, in an attempt to deceive and gain an unfair advantage in assessment'. The Faculty takes Academic Misconduct very seriously and any suspected cases will be investigated through the University's standard policy (https://www.ljmu.ac.uk/about-us/public-information/student-regulations/appeals-and-complaints). If you are found guilty, you may be expelled from the University with no award.

It is your responsibility to ensure that you understand what constitutes Academic Misconduct and to ensure that you do not break the rules. If you are unclear about what is required, please ask.

For more information you are directed to following the University web pages:

- Information regarding ***academic misconduct***: https://www.ljmu.ac.uk/about-us/public-information/student-regulations/appeals-and-complaints
- Information on ***study skills***: https://www2.ljmu.ac.uk/studysupport/
- Information regarding ***referencing***: https://www2.ljmu.ac.uk/studysupport/69049.htm