CM20029: Assignment 1

Set by:	J. P. Fitch
Set on:	27 February 2004
Due Date and Place:	11 May 2004, Pigeon Holes
Time estimate:	10 hours
Project Environment:	C and LEX/Flex OR Java and Jflex
Project Value:	$\frac{1}{2}$ of CM20029 coursework

Introduction

One cannot understand the reality of writing language recognition schemes without writing a small parser, at the very least. This assignment should ensure that you can write a lexer using a standard tool LEX, Flex, or JFlex and then use it in a simple top-down parser, such as might be employed as a user interface to a computer system.

Part 1

Using LEX and associated C, or Jflex and associated JAVA, write a lexical analyser and symbol table for a subset of a C-like language which has the following lexical elements:

- Comments starting with a `//` and lasting to the end of the line,
- Variables made from upper, lower-case letters, underscore and digits only, but starting with a lower-case letter or underscore
- Integers in decimal notation
- Strings, enclosed in `"` characters, but no escaped characters (*i.e.* no `\n`)
- The keywords `if else while int void char return`
- The operators and punctuation `+ - ( ) , ; = * / || &&`

The resulting program should be capable of reading an ASCII text and creating a stream of tokens (type/lexeme pairs), which can then printed in some simple representation or subjected to further computation. A possible top-level testing function in C will be provided, together with a suitable header file. You should assume that the tokens once created are passed to the next stage and the memory should not be reused.

Part 2

Using ANSI C, or JAVA, write a recursive descent parser for the language defined by the following grammar which creates a parse tree. You must **not** use a parser generator for this. The grammar can be modified if necessary to allow a recursive descent parser, but the language must not change.

goal	⇒	expression ;
goal	⇒	ifstatement ;
expression	⇒	term + term
expression	⇒	term − term
expression	⇒	term
term	⇒	factor * factor
term	⇒	factor / factor
term	⇒	factor
factor	⇒	variable
factor	⇒	integer
factor	⇒	string
factor	⇒	(expression)
ifstatement	⇒	**if** (expression) goal **else** goal
ifstatement	⇒	**if** (expression) goal

where variable, integer and string are as defined in part 1 of the assignment. You will need to use your lexer. Not all tokens recognised by the lexer are used in the parser, but your solution should deal with all the tokens.

The resulting program should be capable of reading an ASCII text and creating a printout of the parse tree in some format.

Material to be handed In

You should submit program listings, together with your testing. You may also include notes on your solution, restrictions and extensions, grammar changes or difficulties, as a separate document or as comments as appropriate.

Note: Program listings should be in fixed spaced fonts. Failure to do this makes it hard to identify spaces and layout, and will lose marks.

Mark Scheme

Lexer:	30
Parser:	40
Testing:	20
Evidence of working:	10
Total:	100

Notes on Marking Scheme

A first class mark can be obtained by writing a complete lexer/parser which has been tested and which satisfies the usual software requirements of being readable and clearly maintainable.

Second class performance would typically have an almost working program, or have a complete parser or lexer with a partial solution to the other component, with some clear testing, but lacking either the completeness or clarity one expects from the best attainable standard.

Third class marks would be awarded to a program that while showing signs of working does not fit the above properties.

Marks below a classified grade may be awarded if none of the above apply.