

Heriot-Watt University
School of Mathematical and Computer Sciences
Compilers (F23PF2)
Tutorial Sheet 2

Andrew Ireland

Exercise 1

Write a LEX specification to recognise the language which describes sentences which consist of one or more x's followed by zero or more y's.

Exercise 2

Write a LEX specification that will count the number of words and lines that appear within a file. Ensure that both counts are displayed once the lexer terminates.

Exercise 3

Using the table-driven top-down parser illustrated in lecture attempt to parse the following sentences:

2.1 ()

2.2 ((x))

2.3 (x , x)

Exercise 4

Using the table-driven bottom-up parser illustrated in lecture attempt to parse the following sentences:

3.1 (x,())

3.2 (x,(x))

3.3 (x , (x,x))

Exercise 5

Consider the following grammar:

$$\begin{aligned}\langle S \rangle &::= \langle S \rangle * \langle T \rangle \mid \langle T \rangle \langle S \rangle \# \langle T \rangle \\ \langle T \rangle &::= @ \langle P \rangle \mid @ \langle T \rangle \\ \langle P \rangle & \quad x \mid y\end{aligned}$$

Transform these grammar rules so that they are amenable to the recursive descent parsing technique.

Exercise 6

Consider the grammar given in the “Recursive Descent Parsing” lecture notes”, *i.e.*

$$\begin{aligned}\langle \text{Prop} \rangle &::= \langle \text{Prop} \rangle \text{ and } \langle \text{Term} \rangle \mid \\ & \quad \langle \text{Prop} \rangle \text{ or } \langle \text{Term} \rangle \mid \\ & \quad \langle \text{Term} \rangle \\ \langle \text{Term} \rangle &::= \text{not } \langle \text{Prop} \rangle \mid \\ & \quad (\langle \text{Prop} \rangle) \mid \\ & \quad \text{true} \mid \text{false}\end{aligned}$$

Extend this grammar to include an operator called `implies`, *i.e.* logical implication. In addition, extend the corresponding C code parser given in the lecture notes. Don't forget, all the relevant C code can be found in: <http://www.cee.hw.ac.uk/~air/compilers/programs/>

Exercise 7

Extend the YACC propositional interpreter given in lecture to include the `nand` operator, *i.e.* where `A nand B` is equivalent to `not(A and B)`.

Exercise 8

Extend the YACC propositional interpreter given in lecture to include `implies`, *i.e.* the implications operator. Hint, remember that the conventional definition for `A implies B` is `not(A) or B`.

Exercise 9

Using a simple imperative program demonstrate how a block list and symbol table mechanism can be used to identify multiple declarations for an identifier at the same depth of lexical nesting.