CS-4100/5100
HW05

LL(1) Grammar for Jack

The basic Jack language contains the following binary operators:

```
{'+', '-', '*', '/', '&', '|', '<', '>', '='}
```

It also contains the following unary operators:

```
{'-', '~'}
```

There is no defined order of operations for any of these operators beyond the unary operators being higher precedence than any of the binary operators. Furthermore, while the unary operators are intrinsically right associative (because they are both prefix operators), there is no defined associativity for the others.

We will extend the Jack language to include more of the traditional operators found in various languages.

Problem #1

Modify the Jack Grammar to support the following operators

```
LEVEL  1: {'()', '[]'}
LEVEL  2: {'-', '+', '~', '!'}
LEVEL  4: {'*', '/', '%'}
LEVEL  5: {'-', '+'}
LEVEL  6: {'<<', '>>', '<<<', '>>>'}
LEVEL  7: {'<', '>', '<=', '>='}
LEVEL  8: {'==', '!='}
LEVEL  9: {'&'}
LEVEL 10: {'^'}
LEVEL 11: {'|'}
LEVEL 12: {'&&'}
LEVEL 13: {'^^'}
LEVEL 14: {'||'}
```

Note that the equality relational operator, '==', replaces the original '='. This is to eventually permit assignment operators to serve as expression operators as well.

Your new grammar should have each of these operators as a distinct lexical element (with the exception that the overloaded operators, namely '+' and '-', cannot be distinguished as to which they are by the lexer – the parser must do this).

CS-4100/5100
HW05

LL(1) Grammar for Jack

Each level should comprise a different token (with the operators being the allowed lexemes for that token). Classify the '+' and '-' operators as Level 5 tokens and the '~' and '!' as the only Level 2 tokens. Then simply write the grammar such that a unary operator can be either a Level 2 or a Level 5 token.

Your grammar should impose the precedence and associativity of each of the operators as defined in the Jack Operator Precedence Table. Your grammar is unrestricted in the sense that it does not matter if it is left or right recursive, but it must be unambiguous.

Be sure to first identify your lexical elements (tokens/lexemes) and then write your grammar in terms of those tokens. In general, if a single lexeme is the only option for the next token, it is acceptable to use the literal lexeme (such as '{') in the grammar rule. This allows you to have a catchall SYMBOL token, for instance, than includes those lexemes that can only be used in this way. But if more than one lexeme is acceptable, then a token name should be used instead, keeping in mind that anyplace a token name is used, it must be possible to use ANY lexeme categorized as that token in that position. The more you can use token names, the simpler it will be to write your parser (which is not part of this assignment).

Problem #2

Now modify your grammar so that it is back-track free (i.e., LL(1)). You must maintain the defined precedence, but you may sacrifice the defined associativity (this will be addressed in a later assignment). Since this may result in changes to your lexical element definitions, be sure to include a complete copy of this section even if it happens to be identical to that from Problem #1.

Submission

Your submission will be a single ZIP file with a single PDF file containing the grammars for each problem. It is highly recommended that you not do this by hand, but rather type it into a Word document or even a text file. Cut and paste can be a big time-saver here.

Include a header with the required information and that will be sufficient as far as homework format is concerned (other than clearly indicating the problem and which part is the Lexical Elements and which part is the Grammar.