

## Final Project, Part B: Parser

Due: Last day of regular classes: May 8, 2008, MIDNIGHT

For your final assignment, write a parser for the Java-- (Java minus-minus) language. Your parser will read program statements from a file and report whether the program is a valid Java--program. If it is not, your parser will print out an appropriate error message.

The Java-- language is a very primitive language, it consists only of assignment statements. Assignment statements have the form

$$\langle \text{assignment} \rangle ::= \langle \text{identifier} \rangle = \langle \text{expression} \rangle ;$$

Expressions in Java-- are just identifiers and integer literals separated by plus signs. So

$$x12=4;$$
$$y44 = x12 + 45;$$
$$zzz = 34567+ 555 + y44;$$

are legal assignments.

As you already know, a lexical analyzer breaks a program up into tokens -- the smallest units of the program that can have meaning. If your programming language were English, the lexical analyzer would identify the individual words and punctuation marks in a document.

The parser determines whether the tokens have been arranged to form valid statements in the language. So if your programming language were English, the parser would check your sentences for grammatical correctness.

You wrote a lexical analyzer for the first part of this assignment. Now you will use your lexical analyzer as a basis for writing a Java-- parser.

## ***Errors***

The parser should print error messages when it encounters a statement it cannot parse. For example, the statement

```
xyz 2;
```

should result in an error message of the form

```
Parse error: expecting assignment operator.
```

because after "` `xyz", the only legal input would be an assignment operator (=). The complete list of errors you should report are:

- expecting identifier: when an unexpected token is encountered at the beginning of a statement
- expecting assignment operator: when an unexpected token is encountered after an initial identifier
- expecting operand: when an unexpected token is found after an equals sign or a plus sign
- expecting plus sign or semicolon: when an unexpected token is found after an integer or variable in an expression.
- identifier undefined: when a variable is used that hasn't been previously assigned a value.

After you encounter an error, your program can stop parsing and exit. You need not attempt to recover from the error and continue parsing.

## ***Symbol Table***

In order to keep track of the variables in the program you are parsing, you'll need to create a symbol table. A symbol table stores the names of variables together with the addresses they've been assigned in memory. Since you are only parsing, and not interpreting, you really just need to track the names of variables so that you can report when an identifier is used before its defined.

## ***Reading from a File***

Your program should get its input from a file. The user should be allowed to input any file name at the beginning of the program. You should modify your lexer (`readInput`) so that it reads in multiple lines of text into a single long string. Be sure and put a space or tab between the code of each line.