Syntax and semantics of Mini-Java (Compilers Spring 2012)

Mini-Java is a simplified (and slightly modified) subset of Java.
Mini-Java programs do not use packages (nor other component/library services).
All fields are **private**, and all methods are **public**.
The **main** method has no parameters, and is the only "**static**" method allowed.
Classes have no constructors, and no overloading is allowed in Mini-Java.
The Mini-Java statement **System.out.println** `(...)` can only print integers.
The Mini-Java construct *e*.**length** only applies to expressions of type *T* [] (where *T* is a type). There are only one-dimensional arrays. Array types are compatible only if they have the same element type.
Note that **System**, **out**, **println**, **main**, and **length** are here treated as "special literals" (in Java terminology), i.e., you cannot use them as identifiers in Mini-Java programs.
Mini-Java includes a C-style **assert** statement (if an assertion fails the system prints out a diagnostic message). The **void** type can only be used as a return type for a method.
Otherwise, the meaning of a Mini-Java program is given by its meaning as a Java program.

Context-free syntax notation for Mini-Java

The syntax definition is given in so-called *Extended Backus-Naur* form (EBNF).
In the following Mini-Java grammar, the notation *n*`*`, where *n* is a symbol, means 0, 1, or more repetitions of the symbol *n*.
Parentheses may be used to group together a sequence of related symbols.
Brackets ("[" "]") may be used to enclose optional parts (i.e., zero or one occurrence).
Reserved keywords are marked bold (as "**bold**").
Operators, separators, and other single or multiple character tokens are enclosed within quotes (as "`&&`").
The syntax given below does not specify the precedence of operators. However, Mini-Java expressions use the same precedences as Java.

<program> `::=` *<main class>* *<class declaration>*`*`

<main class> `::=` **class** *<identifier>* "`{`" **public static void main** "`(`" "`)`" "`{`" *<statement>*`*` "`}`" "`}`"

<class declaration> `::=` **class** *<identifier>* [**extends** *<identifier>*] "`{`" *<declaration>*`*` "`}`"

<declaration> `::=` *<variable declaration>* | *<method declaration>*

<method declaration> `::=` **public** *<type>* *<identifier>* "`(`" [*<formals>*] "`)`" "`{`" *<statement>*`*` "`}`"

<variable declaration> `::=` *<type>* *<identifier>* "`;`"

<formals> `::=` *<type>* *<identifier>* ("`,`" *<type>* *<identifier>*)`*`

<type> `::=` *<simple type>* | *<array type>*

<simple type> `::=` **int** | **boolean** | **void** | *<type identifier>*

<array type> `::=` *<simple type>* "`[`" "`]`"

<type identifier> `::=` *<identifier>*

<statement> `::=` **assert** "(" expr ")" |
<local variable declaration> |

"{" <*statement*>* "}" |
if "(" <*expr*> ")" <*statement*> |
if "(" <*expr*> ")" <*statement*> **else** <*statement*> |
while "(" <*expr*> ")" <*statement*> |
System.out.println "(" <*expr*> ")" ";" |
<*lvalue*> "=" <*expr*> ";" |
return <*expr*> ";" |
<*method invocation*> ";"

<*local variable declaration*> **::=** <*variable declaration*>

<*method invocation*> **::=** <*expr*> "." <*identifier*> "(" [<*expr*> ("," <*expr*>)*] ")"

<*lvalue*> **::=** <*expr*>

<*expr*> **::=** <*expr*> <*binary operator*> <*expr*> |
<*expr*> "[" <*expr*> "]" |
<*expr*> "." **length** |
new <*simple type*> "[" <*expr*> "]" |
new <*type identifier*> "(" ")" |
"!" <*expr*> |
"(" <*expr*> ")" |
<*identifier*> | <*integer literal*> |
this | **true** | **false** |
<*method invocation*>

<*binary operator*> **::=** "&&" | "||" | "<" | ">" | "==" | "+" | "-" | "*" | "/" | "%"

Lexical issues

Identifiers: An identifier is a sequence of letters, digits, and underscores, starting with a letter. Uppercase letters are distinguished from lowercase.
Integer literals: A sequence of decimal digits is an integer constant that denotes the corresponding integer value.
Comments: In a Mini-Java program, a comment may appear *between* any two tokens.
There are two forms of comments: one starts with "/*", ends with "*/", can extend over multiple lines, and may be nested.
The other comment alternative begins with "//" and goes only to the end of the line.

Sample Program

```
class Factorial {
  public static void main () {
    System.out.println (new Fac ().ComputeFac (10));
  }
}
class Fac {
  public int ComputeFac (int num) {
    assert (num > -1);
    int num_aux;
    if (num == 0)
      num_aux = 1;
    else
      num_aux = num * this.ComputeFac (num-1);
    return num_aux;
```

```
    }
}
```

This Mini-Java definition was modified from the one given in [A.W. Appel, 2002].
