National University of Singapore

CS4212 Project Assignment 1

AY 2015/2015 Semester 1

Due Date: 14^{th} September 2014 (23:59 Hrs)

(Version 2, updated 22^{nd} August 2014)

1 Introduction

In this assignment, you are required to construct a parse tree for a small object-based programming language called JLite.

The syntax description of JLite and a sample program are provided in Appendix **??**.

You are required to build the lexer and the parser for JLite using `Ocaml` tools, specifically, `ocamllex` and `ocamlyacc`. At the end of the assignment, you are required to produce at least the following (please refer to the latter part of this assignment for more information):

1. An input `.mll` file to `ocamllex` encoding the regular definitions and actions for tokens to be recognized for programs written in JLite.
2. An input `.mly` file to `ocamlyacc` encoding the grammar specification for JLite syntax.
3. An output from the parser generated by `ocamlyacc` representing the parse trees for the processed JLite programs.

The parser must accept those *and only* those syntactically valid programs spelt out in the grammar specification, as shown in Appendix **??**.

You may wish to access the folder named *Assignment 1 resources* in the *Project Resources* folder in IVLE to jumpstart the development of your programs. In addition to some skeleton codes for lexer and parser, the resource also contain a file `jlite_structs.ml` which details the datatype declarations for the parse trees which you need to construct. Note: You are not allowed to modify the datatype declarations.

Together with these datatype declations are a set of "pretty-print" functions which enable you to print the parse trees.

2 Testing of Your Programs

You are required to create some sample programs to test your product.

3 Submission of your product

Please **submit a zipped item**, named after you, containing the following documents to `IVLE CS4212` website under the "`Project Assignment 1 Submission`" folder.

1. Your product.
   - A completed file named `jlite_lexer.mll`.
   - A completed file named `jlite_parser.mly`.
   - A completed file named `jlite_main.ml` which is the main program of your product.
2. Three (or more) sample programs that you have tried on your product
3. A document, named `Axxxxxxx_pja1_readme.txt`, describing your product, the content of your submission, and any important information which you would like to share with us. Here, `Axxxxxxx` stands for your matriculation number. (It might not begin with an 'A'; in that case, please use the correct alphabet.)

4 Late Submission

We try to discourage you from submitting your assignment beyond deadline. This is to ensure that you have time to prepare for other modules, as well as time for other assignments handed out in this module.

Any submission after the deadline will have its mark automatically deducted by certain percentages. Your submission time is determined by the time recorded in IVLE submission folders. If you have multiple submissions, we will take the latest submission time. Any submission to places other than the appropriate submission folders (such as to the instructor's email account) will be discarded and ignored.

Here is the marking scheme:

Submit by 23:59HRS of	Maximum Mark	If your score is ... %	It becomes ... %
14th Sept	100	80	80
16th Sept	80	80	64
17th Sept	60	80	48
18th Sept	0	-	0

A Specification of JLite Syntax

A.1 Lexical Issues

- *id* ∈ **Identifiers**:

  An *identifier* is a sequence of alphabets, digits, and underscore, **starting with a lower letter**. Except the first letter, uppercase letters are not distinguished from lowercase.

- *cname* ∈ **Class Names**:

  A *class name* is a sequence of alphabets, digits, and underscore, **starting with an uppercase letter**. Except for the first letter, uppercase letters are not distinguished from lowercase.

- **Integer Literals**:

  A sequence of decimal digits (from 0 to 9) is an integer constant that denotes the corresponding integer value. Here, we use the symbol `INTEGER_LITERAL` to stand for an integer constant.

- **String Literals**:

  A string literal is the representation of a string value in a JLite program. It is defined as a quoted sequence of ascii characters (Eg: ``` ``this is a string'' ```) where some constraints hold on the sequence of characters. Specifially, some special characters such as double quotes have to be represented in the string literal by preceding them with an escape character, backslash ("\"). More formally, a string literal is defined as a quoted sequence of: escaped sequences representing either special characters ($\backslash\backslash, \backslash n, \backslash r, \backslash t, \backslash b$) or the ascii value of an ascii character in decimal or hexadecimal base (e.g. $\backslash 032, \backslash x08$); characters excluding double quote, backslash, new-line or carriage return. Here, we use the symbol `STRING_LITERAL` to stand for any string constant.

- **Boolean Literals**:

  Believe it or not, there are only two boolean literals: `true` and `false`.

- **Binary Operators** :

  Binary operators are classified into several categories:

  1. **Boolean Operators** include conjunction and disjunction.
  2. **Relational Operators** are comparative operators over two integers
  3. **Arithmetic Operators** are those that perform arithmetic calculations.

  In addition to this categorization, each binary operator is associated with its own associativity rule; two distinct binary operators are related by a precedence relation.

- **Unary Operators** :

  There are only two unary operators:

  1. `!`. This is a negation operator, which aims to negate a Boolean value.
  2. `-`. This is a negative operator, which aims to negate an integer value.

- **Class constructor** : There is **no** class constructor. Given the following declaration of a class, say `Box`,

```
class Box {
   Int x ;
   Int y ;
   Int z ;
   Box b1 ;
}
```

The call `new Box()` will create an object instance, and initalize all its attributes, through *shallow* initialization. Thus, for the given example, the attributes are initialized as follows:

```
x = 0 ; y = 0 ; z = 0 ; b1 = NULL
```

- **Comments**:

  A *comment* may appear between any two tokens. There are two forms of comments: One starts with `/*`, ends with `*/`, and may run across multiple lines; another begins with `//` and goes to the end of the line.

A.2 A Sample Program

```
class Factorial {
     void main (Int a) {
        println(new Fac().computeFac(a)) ;
        return ;
     }
}

class Fac {

/* This is the factorial function that is not written
   in tail-recursive manner. (A tail recursive factorial
   function is expected to take in two parameters.)
   Can your compiler optimize it to a tail-recursive function?
*/

     Int computeFac(Int num) {
        Int num_aux;
        if (num < 1)  // shouldn't it be num <= 1?
           num_aux = 1 ;
        else
           num_aux = num * (this.computeFac(num-1)) ;
        return num_aux ;
     }
}
```

A.3 Grammar

The grammar in BNF notation is provided in the following page.

$$
\begin{array}{rcl}
\langle Program \rangle & \rightarrow & \langle MainClass \rangle \ \langle ClassDecl \rangle \ ^{\star} \\
\langle MainClass \rangle & \rightarrow & \texttt{class} \ \langle cname \rangle \ \{ \ \texttt{Void main (} \ \langle FmlList \rangle \ \texttt{)} \ \langle MdBody \rangle \ \} \\
\langle ClassDecl \rangle & \rightarrow & \texttt{class} \ \langle cname \rangle \ \{ \ \langle VarDecl \rangle \ ^{\star} \ \langle MdDecl \rangle \ ^{\star} \ \} \\
\langle VarDecl \rangle & \rightarrow & \langle Type \rangle \ \langle id \rangle \ \texttt{;} \\
\langle MdDecl \rangle & \rightarrow & \langle Type \rangle \ \langle id \rangle \ \texttt{(} \ \langle FmlList \rangle \ \texttt{)} \ \langle MdBody \rangle \\
\langle FmlList \rangle & \rightarrow & \langle Type \rangle \ \langle id \rangle \ \langle FmlRest \rangle \ ^{\star} \mid \epsilon \\
\langle FmlRest \rangle & \rightarrow & \texttt{,} \ \langle Type \rangle \ \langle id \rangle \\
\langle Type \rangle & \rightarrow & \texttt{Int} \mid \texttt{Bool} \mid \texttt{String} \mid \texttt{Void} \mid \langle cname \rangle \\
\langle MdBody \rangle & \rightarrow & \{ \ \langle VarDecl \rangle \ ^{\star} \ \langle \ Stmt \rangle \ ^{+} \ \} \\
\langle Stmt \rangle & \rightarrow & \texttt{if (} \ \langle Exp \rangle \ \texttt{)} \ \{ \ \langle Stmt \rangle \ ^{+} \ \} \ \texttt{else} \ \{ \ \langle Stmt \rangle \ ^{+} \ \} \\
 & & \mid \texttt{while (} \ \langle Exp \rangle \ \texttt{)} \ \{ \langle Stmt \rangle \ ^{\star} \} \\
 & & \mid \texttt{readln (} \ \langle id \rangle \ \texttt{)} \ \texttt{;} \mid \texttt{println (} \ \langle Exp \rangle \ \texttt{)} \ \texttt{;} \\
 & & \mid \langle id \rangle \ \texttt{=} \ \langle Exp \rangle \ \texttt{;} \mid \langle Atom \rangle \texttt{.} \langle id \rangle \ \texttt{=} \ \langle Exp \rangle \ \texttt{;} \\
 & & \mid \langle Atom \rangle \ \texttt{(} \ \langle ExpList \rangle \ \texttt{)} \mid \texttt{return} \ \langle Exp \rangle \ \texttt{;} \mid \texttt{return ;} \\
\langle Exp \rangle & \rightarrow & \langle BExp \rangle \mid \langle AExp \rangle \mid \langle SExp \rangle \\
\langle BExp \rangle & \rightarrow & \langle BExp \rangle \ \texttt{||} \ \langle Conj \rangle \mid \langle Conj \rangle \\
\langle Conj \rangle & \rightarrow & \langle Conj \rangle \ \texttt{\&\&} \ \langle RExp \rangle \mid \langle RExp \rangle \\
\langle RExp \rangle & \rightarrow & \langle AExp \rangle \ \langle BOp \rangle \ \langle AExp \rangle \mid \langle BGrd \rangle \\
\langle BOp \rangle & \rightarrow & \texttt{<} \mid \texttt{>} \mid \texttt{<=} \mid \texttt{>=} \mid \texttt{==} \mid \texttt{!=} \\
\langle BGrd \rangle & \rightarrow & \texttt{!} \langle BGrd \rangle \mid \texttt{true} \mid \texttt{false} \mid \langle Atom \rangle \\
\langle AExp \rangle & \rightarrow & \langle AExp \rangle \ \texttt{+} \ \langle Term \rangle \mid \langle AExp \rangle \ \texttt{-} \ \langle Term \rangle \mid \langle Term \rangle \\
\langle Term \rangle & \rightarrow & \langle Term \rangle \ \texttt{*} \ \langle Ftr \rangle \mid \langle Term \rangle \ \texttt{/} \ \langle Ftr \rangle \mid \langle Ftr \rangle \\
\langle Ftr \rangle & \rightarrow & \texttt{INTEGER\_LITERAL} \mid \texttt{-} \langle Ftr \rangle \mid \langle Atom \rangle \\
\langle SExp \rangle & \rightarrow & \texttt{STRING\_LITERAL} \mid \langle Atom \rangle \\
\langle Atom \rangle & \rightarrow & \langle Atom \rangle \texttt{.} \langle id \rangle \mid \langle Atom \rangle \texttt{(} \ \langle ExpList \rangle \ \texttt{)} \\
 & & \mid \texttt{this} \mid \langle id \rangle \mid \texttt{new} \ \langle cname \rangle \texttt{()} \\
 & & \mid \texttt{(} \ \langle Exp \rangle \ \texttt{)} \mid \texttt{null} \\
\langle ExpList \rangle & \rightarrow & \langle Exp \rangle \ \langle ExpRest \rangle \ ^{\star} \mid \epsilon \\
\langle ExpRest \rangle & \rightarrow & \texttt{,} \ \langle Exp \rangle
\end{array}
$$