CSE 305 Spring 2015

Assignment #2: Object-Oriented Top-down Parsing

Assigned: February 16, 2015

Due: Weds, Feb 25, 11:59 pm (Part I) and Sun, Mar 1, 11:59 pm (Part II)

Note: This assignment may be done by a pair of students.

This assignment is about top-down parsing and type-checking for a simple programming language, called TinyPL, whose grammar is given below in Chomsky notation with EBNF extensions:

```
program → 'begin' decls stmts 'end'

decls  →  [ 'int'  idlist ';' ] [ 'real' idlist ';' ] [ 'bool' idlist ';' ]

idlist → id {',' id}

stmts  → { stmt }

stmt   →  assign | cond | loop | cmpd

assign → var '=' expr ';'

loop   → while '(' expr ')' stmt

cond   → if '(' expr ')' stmt [ 'else' stmt ]

cmpd   → '{' stmts '}'

expr   → term [ ('+' | '-' | '||') expr ]

term   → factor [ ('*' | '/'  | '&&') term ]

factor → literal | '!' factor | '(' expr ')'
       | '(' expr  ('<=' | '>=' | '==' | '!=' | '<' | '>')  expr ')'

literal → 'int' | 'real' | 'true' | 'false' | 'id'
```

Part I: Due Weds, Feb 25, 11:59 pm

Write an object-oriented top-down recognizer in Java to recognize well-formed TinyPL programs according to the above grammar. This part does not require any type-checking; that is to be done in part II. Note the following:

1. All terminal symbols in the above grammar are given in quotes, and these are handled by the lexical analyzer. For simplicity, the lexical analyzer assumes that an identifier is a single letter; an int is an unsigned number; and a real is of the form digits.digits.

2. For each nonterminal of the TinyPL grammar, write one Java class, and put in its constructor the code for the production rule associated with the nonterminal.

3. Develop your code starting from the files in: Piazza→Resources→Homeworks→TinyPL.zip. There are four Java files: TinyPL.java, Lexer.java, Buffer.java and Token.java. You have to complete TinyPL.java with the code for the recognizer. The other three files pertain to lexical analysis – do not change these files.

4. The grammar exhibits the "dangling else" ambiguity. Resolve the ambiguity in your recognizer by matching each 'else' with the closest previous unmatched 'if'. Note: There is no need to rewrite the grammar.

5. Define four additional classes, called Int_Lit, Real_Lit, Bool_Lit, and Id_Lit corresponding to the literals for int, real, bool, and id respectively. Make them subclasses of class Literal. Objects of these classes will hold specific instances for the four kinds of literals: integers, reals, booleans, and identifiers respectively.

6. The input test cases will all be well-formed programs; no error-checking of the input is necessary.

7. Run your recognizer through JIVE and save the object diagram for each test case. The object diagram for the i-th test case should be named obj_i.png. Use the "Stacked" option for the diagrams.

8. Submit TinyPL.java and your object diagrams as separate files using the 'submit_cse305' command.

JIVE Object Diagrams

1. In drawing an object diagram (parse tree), JIVE will consider references (pointers) that emanate from fields of objects; it will not consider references from local variables of methods/constructors.

2. Once a parse-tree has been bound to a field of an object, one should not over-write it with another parse-tree. Doing so will cause the original parse-tree to become disconnected from the diagram.

Part II: Due Sun, Mar 1, 11:59 pm

Augment your recognizer of Part I with code for performing type-checking, as described below.

1. The input test cases will be well-formed TinyPL programs, but they may have type-related errors:
   (i) re-declaration of an identifier;
   (ii) use of an undeclared variable in a statement; and
   (iii) mismatch in the types of an operator and one of its operands or the presence of a non-boolean expression in the condition part of an 'if' or 'while' statement.

   Sample error messages are given on the next page.

2. Corresponding to the above three type-related errors, declare three exception classes called, respectively, Redeclaration, TypeUnknown, and TypeMismatch. Throw an exception when any of these errors is detected and pass a string describing the error. Handle the exception in 'main' and terminate parsing.

3. Declare in class TinyPL a static field ST (for symbol table) initialized to a Java 'HashMap'. Insert into ST each identifier found in the declarations – but first check if it is already in the table before inserting. Throw the 'Redeclaration' exception if any identifier is declared more than once, passing the name of the re-declared identifier as an argument to the exception.

4. Run your type-checker through JIVE and save the object diagram for each test case. The object diagram for the i-th test case should be named part2_obj_i.png. Use the "Stacked" option for the diagrams.

5. Submit TinyPL.java and your object diagrams as separate files using the 'submit_cse305' command.

Test Cases and Online Submission

1. Test cases for the two parts will be posted on Piazza.
2. Both students working on the assignment should make a submission and should write their names at the top of the file TinyPL.java. It is fine to work solo; write your name at the top of the file.

End of Assignment 2

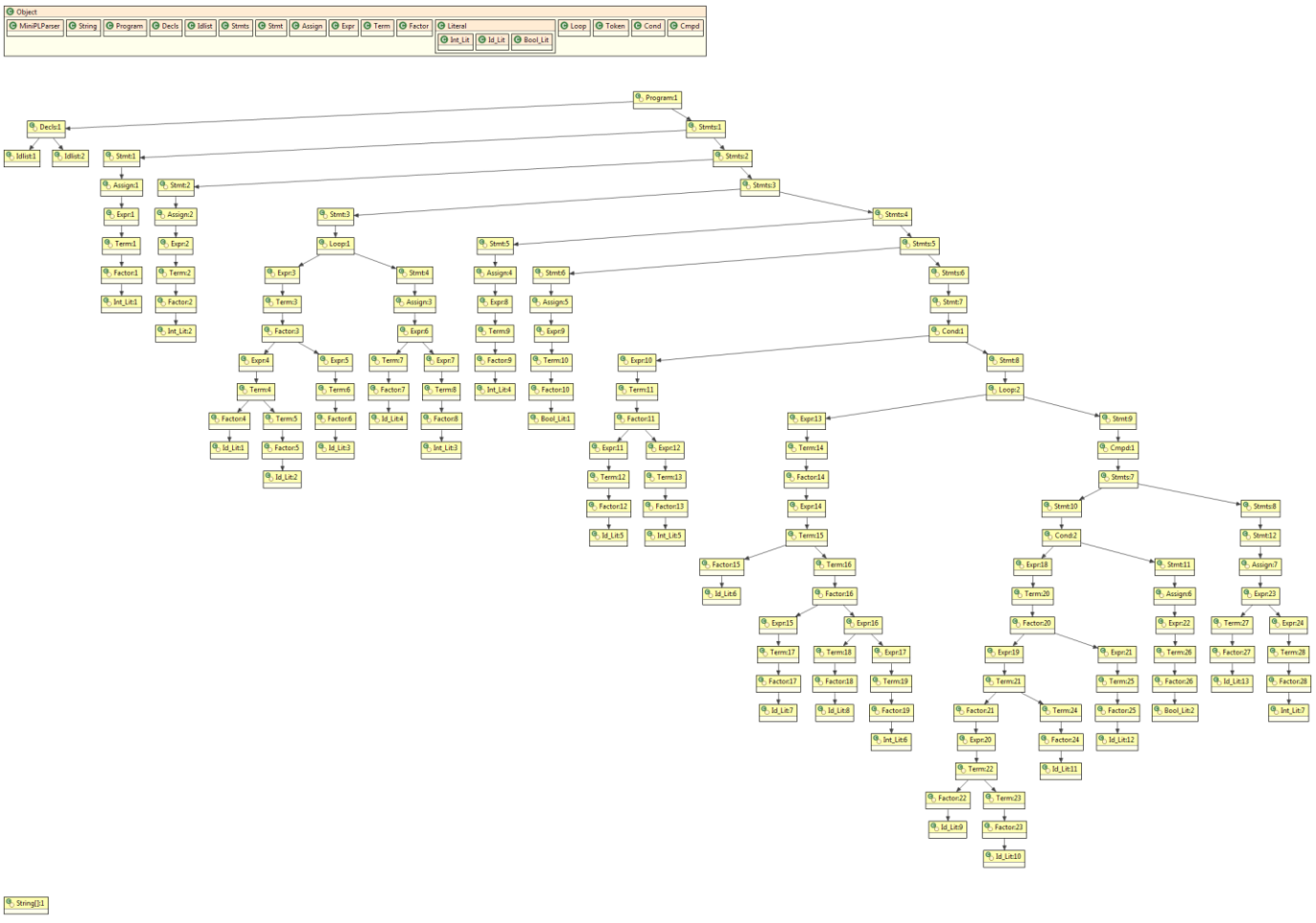
Sample Error Messages for Part II

Statement	Error Message	
int x, y, z, x;	Redeclaration: x	
int y; real x, y;	Redeclaration: y	
if (x * y) ...	TypeMismatch: if condition must have bool type	
while(a + b) …	TypeMismatch: while condition must have bool type	
x = x * 3.5	TypeMismatch: int * real	(if x was declared int)
x = 2 * x / 3.5	TypeMismatch: int / real	(if x was declared int)
x = x && true	TypeMismatch: real && bool	(if x was declared real)
int w = 10;	TypeUnknown: w on LHS of assignment	(if w was not declared)
int x = w;	TypeUnknown: w in expression	(if w was not declared but x was declared)
while (x > w) …	TypeUnknown: w in expression	(if w was not declared but x was declared)

Sample Well-Formed Tiny PL Program without Type Errors:

```
begin
        int s, i, n;
        bool p;

        n = 37;
        s = 1;

        while (s*s < n)  s = s + 1;

        i = 2;
        p  = true;

        if (n > 2)
                while (p && (i < s+1)) {
                        if ((n/i)*i == n)
                                p = false;
                        i = i + 1;
                }
end
```

The JIVE object diagram for the above program is shown on the next page.


Object
MiniPLParser
String
Program
Decls
Idlist
Stmts
Stmt
Assign
Expr
Term
Factor
Literal
Int_Lit
Id_Lit
Bool_Lit
Loop
Token
Cond
Cmpd
Program:1
Decls:1
Stmts:1
Idlist:1
Idlist:2
Stmt:1
Stmts:2
Assign:1
Stmt:2
Stmts:3
Expr:1
Assign:2
Stmt:3
Stmts:4
Term:1
Expr:2
Loop:1
Stmt:5
Assign:4
Stmt:6
Stmts:5
Factor:1
Term:2
Expr:3
Stmt:4
Expr:8
Assign:5
Stmts:6
Int_Lit:1
Factor:2
Term:3
Assign:3
Term:9
Expr:9
Stmt:7
Int_Lit:2
Factor:3
Expr:6
Factor:9
Term:10
Cond:1
Expr:4
Expr:5
Term:7
Expr:7
Expr:10
Stmt:8
Term:4
Term:6
Factor:7
Term:8
Factor:10
Term:11
Loop:2
Factor:4
Term:5
Factor:6
Id_Lit:4
Factor:8
Bool_Lit:1
Factor:11
Expr:13
Stmt:9
Id_Lit:1
Factor:5
Id_Lit:3
Int_Lit:3
Expr:11
Expr:12
Term:14
Cmpd:1
Id_Lit:2
Term:12
Term:13
Factor:14
Stmts:7
Factor:12
Factor:13
Expr:14
Stmt:10
Stmts:8
Id_Lit:5
Int_Lit:5
Term:15
Cond:2
Stmt:12
Factor:15
Term:16
Expr:18
Stmt:11
Assign:7
Id_Lit:6
Factor:16
Term:20
Assign:6
Expr:23
Expr:15
Expr:16
Factor:20
Expr:22
Term:27
Expr:24
Term:17
Term:18
Expr:17
Expr:19
Expr:21
Term:26
Factor:27
Term:28
Factor:17
Factor:18
Term:19
Term:21
Term:25
Factor:26
Id_Lit:13
Factor:28
Id_Lit:7
Id_Lit:8
Factor:19
Factor:21
Term:24
Factor:25
Bool_Lit:2
Int_Lit:7
Int_Lit:6
Expr:20
Factor:24
Id_Lit:12
Id_Lit:11
Term:22
Factor:22
Term:23
Id_Lit:9
Factor:23
Id_Lit:10
String[]:1