Compilation \ PA2 Documentation

Group members:

- Kalev Alpernas, user: kalevalp, ID: 304385727
- Ariel Stolerman, user: arielst1, ID: 039432489
- Vadim Stotland, user: stotland, ID: 314282682

Code structure:

IC.lex is the source for the lexical analyzer which is used to scan the input IC code. IC.cup and Library.cup are the source for the syntactic parser for the input IC file and the input IC-library (optional) file respectively. The parsers are used to scan the output tokens of the lexer, derive legal sequences of tokens according to the IC syntax and build an AST for the input. All the AST nodes are dynamic types of class ASTNode, determined with respect to the syntactic type they represent (e.g. class StaticMethod for a static-method node).

The compiler's main method runs over the input file using the Parser (/LibraryParser) class (and the Lexer class), with an optional pretty-printing of the AST. Lexical exceptions are handled with the LexicalError class, and Syntax exceptions are handled with the SyntaxError class.

Testing Strategy:

Correctness:

- Correct parsing of every derivation rule in the IC specification document:
  - A simple test case for each rule and version of a rule.
  - A few complex tests that use a number of different rules.
- Correct handling of operator order of precedence and associativity.

Soundness:

- Parser error handling – correct error report for any type of syntax error.
  - Check scenarios of multiple errors in the same file, same line, etc.
- Lexical analyzer error handling – based on error tests from PA1.

Project Hierarchy (under directory src)

Package IC:

- Class Compiler: contains the main method; as described above, takes an input IC program file (with an optional IC library file) and parses it, creating an AST for the input. With optional pretty-printing of the AST. Uses:
  - Package IC.Parser for classes Parser, Lexer, Token, Sym, LexicalError, SyntaxError and optional LibraryParser.
  - Package IC.AST for the Pretty-printing of the AST with class PrettyPrinter.
- Enumerated types used mainly in package IC.AST: BinaryOps, DataTypes, LiteralTypes, UnaryOps.

Package IC.AST:

- Class ASTNode: The extended classes for the AST build. Used in package IC.Parser.
- A group of classes extending class ASTNode, used for each syntactic type (e.g. Program, Literal etc.). Some of theses classes use the enumerated types in package IC.
- The Visitor interface: used for dynamic type handling, in this case implemented only by class PrettyPrinter.
- Class PrettyPrinter: implements class Visitor, used for pretty-printing the AST generated by the parsers.

Package IC.Parser:

- Class Lexer: The lexical analyzer class generated by JFlex from IC.lex. Extends java_cup.runtime.Scanner.
- Class Parser: The syntactic parser class generated by CUP from IC.cup. Extends java_cup.runtime.lr_parser.
- Class LibraryParser: The syntactic parser class generated by CUP from Library.cup. Extends java_cup.runtime.lr_parser.
- Class LexicalError: used for lexical errors, extends Exception.
- Class SyntaxError: used for syntax errors, extends Exception.
- Class Sym: independent definitions class.
- Class Token: used by class Lexer, extends java_cup.runtime.Symbol.
- Additions:
    - IC.cup: The cup source code from which class Parser is generated by CUP. Compiled separately.
    - Library.cup: The cup source code from which class LibraryParser is generated by CUP. Compiled separately.
    - IC.lex: The lex source code from which class Lexer is generated by JFlex. Compiled separately.

Grammar description:

IC.cup CUP grammar dump:

```
===== Terminals =====
[0]EOF [1]error [2]MULTIPLY [3]DIVIDE [4]PLUS
[5]MINUS [6]MOD [7]LP [8]RP [9]LB
[10]RB [11]LCBR [12]RCBR [13]SEMI [14]COMMA
[15]DOT [16]EQUAL [17]ASSIGN [18]GTE [19]GT
[20]LTE [21]LT [22]NEQUAL [23]LAND [24]LOR
[25]LNEG [26]BREAK [27]CONTINUE [28]EXTENDS [29]WHILE
[30]IF [31]ELSE [32]TRUE [33]FALSE [34]LENGTH
[35]NEW [36]NULL [37]RETURN [38]STATIC [39]THIS
[40]VOID [41]BOOLEAN [42]INT [43]CLASS [44]STRING
[45]CLASS_ID [46]ID [47]INTEGER [48]QUOTE [49]UMINUS
===== Non terminals =====
[0]program [1]classDecl [2]field_or_method [3]classDecl_list [4]field
[5]id_list [6]method [7]formal [8]formal_list [9]type
[10]stmt [11]stmt_list [12]location [13]expr [14]expr_list
[15]call [16]static_call [17]virtual_call [18]binop [19]unop
[20]literal
===== Productions =====
[0] program ::= classDecl_list
[1] $START ::= program EOF
[2] classDecl_list ::= classDecl
[3] classDecl_list ::= classDecl_list classDecl
[4] classDecl ::= CLASS CLASS_ID LCBR RCBR
[5] classDecl ::= CLASS CLASS_ID LCBR field_or_method RCBR
[6] classDecl ::= CLASS CLASS_ID EXTENDS CLASS_ID LCBR RCBR
[7] classDecl ::= CLASS CLASS_ID EXTENDS CLASS_ID LCBR field_or_method RCBR
[8] field_or_method ::= field
[9] field_or_method ::= method
[10] field_or_method ::= field_or_method field
[11] field_or_method ::= field_or_method method
[12] field ::= type id_list SEMI
[13] id_list ::= ID
[14] id_list ::= id_list COMMA ID
[15] method ::= STATIC VOID ID LP RP LCBR stmt_list RCBR
[16] method ::= STATIC type ID LP RP LCBR stmt_list RCBR
[17] method ::= STATIC VOID ID LP formal_list RP LCBR stmt_list RCBR
[18] method ::= STATIC type ID LP formal_list RP LCBR stmt_list RCBR
[19] method ::= VOID ID LP RP LCBR stmt_list RCBR
[20] method ::= type ID LP RP LCBR stmt_list RCBR
[21] method ::= VOID ID LP formal_list RP LCBR stmt_list RCBR
[22] method ::= type ID LP formal_list RP LCBR stmt_list RCBR
[23] formal_list ::= formal
[24] formal_list ::= formal_list COMMA formal
[25] formal ::= type ID
[26] type ::= INT
[27] type ::= BOOLEAN
[28] type ::= STRING
[29] type ::= CLASS_ID
[30] type ::= type LB RB
```

```
[31] stmt_list ::=
[32] stmt_list ::= stmt_list stmt
[33] stmt ::= location ASSIGN expr SEMI
[34] stmt ::= call SEMI
[35] stmt ::= RETURN SEMI
[36] stmt ::= RETURN expr SEMI
[37] stmt ::= IF LP expr RP stmt
[38] stmt ::= IF LP expr RP stmt ELSE stmt
[39] stmt ::= WHILE LP expr RP stmt
[40] stmt ::= BREAK SEMI
[41] stmt ::= CONTINUE SEMI
[42] stmt ::= LCBR stmt_list RCBR
[43] stmt ::= type ID SEMI
[44] stmt ::= type ID ASSIGN expr SEMI
[45] expr ::= location
[46] expr ::= call
[47] expr ::= THIS
[48] expr ::= NEW CLASS_ID LP RP
[49] expr ::= NEW type LB expr RB
[50] expr ::= expr DOT LENGTH
[51] expr ::= binop
[52] expr ::= unop
[53] expr ::= literal
[54] expr ::= LP expr RP
[55] call ::= static_call
[56] call ::= virtual_call
[57] static_call ::= CLASS_ID DOT ID LP RP
[58] static_call ::= CLASS_ID DOT ID LP expr_list RP
[59] virtual_call ::= ID LP RP
[60] virtual_call ::= ID LP expr_list RP
[61] virtual_call ::= expr DOT ID LP RP
[62] virtual_call ::= expr DOT ID LP expr_list RP
[63] expr_list ::= expr
[64] expr_list ::= expr_list COMMA expr
[65] location ::= ID
[66] location ::= expr DOT ID
[67] location ::= expr LB expr RB
[68] binop ::= expr PLUS expr
[69] binop ::= expr MINUS expr
[70] binop ::= expr MULTIPLY expr
[71] binop ::= expr DIVIDE expr
[72] binop ::= expr MOD expr
[73] binop ::= expr LAND expr
[74] binop ::= expr LOR expr
[75] binop ::= expr LT expr
[76] binop ::= expr LTE expr
[77] binop ::= expr GT expr
[78] binop ::= expr GTE expr
[79] binop ::= expr EQUAL expr
[80] binop ::= expr NEQUAL expr
[81] unop ::= MINUS expr
[82] unop ::= LNEG expr
[83] literal ::= INTEGER
[84] literal ::= QUOTE
[85] literal ::= TRUE
[86] literal ::= FALSE
[87] literal ::= NULL
```

Library.cup CUP grammar dump:

```
===== Terminals =====
[0]EOF [1]error [2]MULTIPLY [3]DIVIDE [4]PLUS
[5]MINUS [6]MOD [7]LP [8]RP [9]LB
[10]RB [11]LCBR [12]RCBR [13]SEMI [14]COMMA
[15]DOT [16]EQUAL [17]ASSIGN [18]GTE [19]GT
[20]LTE [21]LT [22]NEQUAL [23]LAND [24]LOR
[25]LNEG [26]BREAK [27]CONTINUE [28]EXTENDS [29]WHILE
[30]IF [31]ELSE [32]TRUE [33]FALSE [34]LENGTH
[35]NEW [36]NULL [37]RETURN [38]STATIC [39]THIS
[40]VOID [41]BOOLEAN [42]INT [43]CLASS [44]STRING
[45]CLASS_ID [46]ID [47]INTEGER [48]QUOTE [49]UMINUS
===== Non terminals =====
[0]libic [1]libmethod [2]libmethod_list [3]formal [4]formal_list
[5]type
```

```
===== Productions =====
[0] libic ::= CLASS CLASS_ID LCBR libmethod_list RCBR
[1] $START ::= libic EOF
[2] libmethod_list ::= libmethod
[3] libmethod_list ::= libmethod_list libmethod
[4] libmethod ::= STATIC VOID ID LP RP SEMI
[5] libmethod ::= STATIC type ID LP RP SEMI
[6] libmethod ::= STATIC VOID ID LP formal_list RP SEMI
[7] libmethod ::= STATIC type ID LP formal_list RP SEMI
[8] formal_list ::= formal
[9] formal_list ::= formal_list COMMA formal
[10] formal ::= type ID
[11] type ::= INT
[12] type ::= BOOLEAN
[13] type ::= STRING
[14] type ::= CLASS_ID
[15] type ::= type LB RB
```

Special parts of the grammar and comments (for both grammars):

- There are no conflicts between the two grammars.
- Many of the terminals in Library.cup are not used, but defined in order to allow use with the same Sym.java class as used by class Parser that is generated from IC.cup.

Here is a list of special non-terminals used in both grammars in order to build the AST; the ones mentioned here are of types that are not used directly as an ASTNode, including lists of dynamic ASTNode types:

- IC.cup\classDecl_list: of type List<ICClass>; used to derive a list of classes (ICClass) for deriving program (Program).
- IC.cup\field_or_method: of the new type IC.AST.FieldOrMethod; A data-type used for collecting fields and methods under classes (in any order of appearance) for deriving Class nodes in the AST. The fileds-list and the methods-list from the FieldOrMethod object are taken as fields for constructing class nodes.
- Library.cup\libmethod_list: of type List<Method>; used to derive a list of library methods (LibraryMethod) for deriving Library.IC\libic (the library class, of type ICClass).
- IC.cup\field: of type List<Field>; used to derive a list of fields that are all under the same type declaration (e.g. $int\ i, j, k;$). This list may contain only one field. All fields are added to field_or_method's fields-list.
- IC.cup\id_list: of type List<String>; used to derive a list of ID's for deriving field (all ID's from the same type).
- IC.cup\formal_list: of type List<Formal>; used to derive a list of formals (Formal) for deriving methods (StaticMethod or VirtualMethod in IC.cup, LibraryMethod in Library.cup).
- IC.cup\stmt_list: of type List<Statement>; used to derive a list of statements (Statement) for deriving methods (types listed in the previous bullet) or statements block (StatementsBlock).
- IC.cup\expr_list: of type List<Expression>; used to derive a list of expressions (Expression) for deriving calls (StaticCall or VirtualCall).

Special precedence options:

- The UMINUS unary operator has precedence over the binary operator MINUS.
- The "IF…ELSE…" statement has precedence over the sole "IF…" statement (without "ELSE…").

Feedback:

- We spent approximately 18-20 hours on all part of the assignment.
- The assignment was in an appropriate level of difficulty. The hardest part was getting started and understanding the way the parser integrates with the lexer. It came to our understanding that using class Token is unnecessary; object Symbol is already enough for holding all the information needed (e.g. using the "left" field for holding line number), and removing class Token may simplify future projects. Further explanation about this part in class may assist in the future.