Computer Science 330 Language Implementation Test

6.20-8.00pm Thursday 5th April 2007

Course Compsci 330/Compsci 601
Surname
Given Names
Student ID Number
Login Name
Normal Signature

1 _____ /31

2(a) _____ /25

2(b) _____ /8

2(c)_____ /8

2(d) _____ /8

3_____ /20

Total _____ /100

Start reading 6.20p.m. Write your name on all sheets of your answer book. Start writing your answers at 6.30pm. Stop writing at 8.00p.m.

Remove the staple fastening the appendices to the answer book, but do not remove the staples from the answer book. Read the questions carefully. Hand in your answer book at the front of the class. Attempt all questions. Questions total 100 marks. The test counts for 20% of the total mark.

Question 1 — 31 Marks

(a) Write JFlex regular expressions to match the following tokens. You may declare support regular expressions if you need them.

(i) A word composed of an upper case alphabetic character, followed by zero or more lower case alphabetic characters. For example, `Ronny` or `Sarah` or `B` but not `Tien-Wei` or `MacDonald` or `B2`.

(ii) A binary integer, with a prefix of `0b` or `0B`, followed by one or more binary digits. For example, `0b1010111` or `0B1010111` but not `0b123456789` or `0b`.

(iii) A floating point number, with an integer part, fractional part, and an exponent with an explicit sign for the exponent. For example, `123.456e+23` or `123.456e-23` but not `1.23e5` or `-1.23e-5` or `123e-5` or `3.14159`.

(iv) A decimal number with exactly three digits. The leading digit may be `0`. For example, `012` or `120` but not `1234` or `12`.

(v) A decimal number, with the first digit nonzero, and at most three digits. For example, `123` or `12` but not `012` or `1234`.

(vi) A sequence of one or more single decimal digits, with "," characters between each digit. For example, `1,2,3` or `2` but not `1,,2` or `12,34` or `,4`.

(vii) A nonzero decimal number, and commas grouping the digits into blocks of three, as is normally done when writing numbers in English. For example, `1,200,001` or `12,345,678` or `12` but not `012,333` or `1234`.

(3 marks each)

Print your login name_____

(b) Find and correct at least five different kinds of errors in the following fragment of JFlex code. ("..." just means omitted code). Assume spaces and line breaks are not syntactically important. Assume comments cannot be nested.

```
...
newline     =     \r|\n|\r\n
space       =     [\ |\t]
ident       =     [A-Za-z][A-Za-z0-9]+
%state NORMAL, COMMENT
%init{
      yybegin( COMMENT );
%init}

%%
<NORMAL> {
      {ident}     { return token( sym.IDENT ); }
      if          { return token( sym.IF ); }
      else        { return token( sym.ELSE ); }
      ...
      /*          { yybegin( COMMENT ); }
      .           { return token( sym.error ); }
      {space}     { }
      {newline}   { linecount++; }
      }
<COMMENT> {
      */          { yyend( COMMENT ); }
      {newline}   { linecount++; }
      .           { return token( sym.error ); }
      }
<<EOF>>           { return token( sym.EOF ); }
```

1

2

3

4

5

(10 marks)

Question 2 **41 marks**

Consider the CUP grammar in the appendices. The terminal symbols COMMA, LEFT, RIGHT, LEFTSQ, RIGHTSQ, BAR correspond to "`,`", "`(`", "`)`", "`[`", "`]`", "`|`", and IDENT corresponds to an identifier.

(a) Using the information provided in the appendices, perform a shift-reduce LALR(1) parse of the valid input

```
f( [ X | Y ], [ X, Y ] )
```

Show both the symbols and states on the stack, the current token, and the action performed at each stage.

Note that some grammar rules are left recursive, while others are right recursive. (25 marks)

Stack

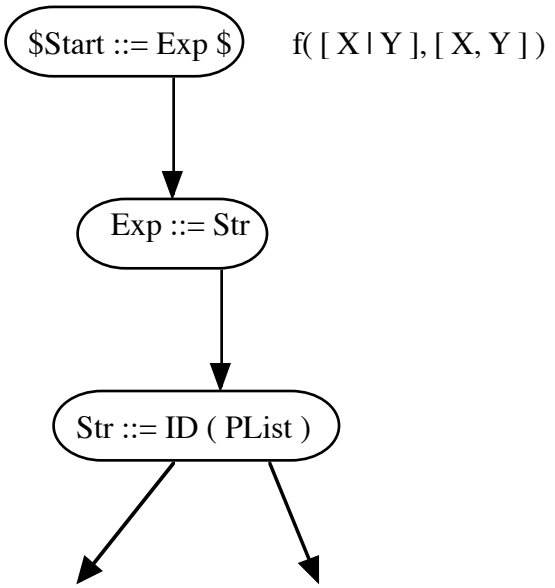
$0								
$0	ID 4							
$0	ID 4	(14						
$0	ID 4	(14	[5					
$0	ID 4	(14	[5	ID 4				
$0								
$0								
$0								
$0								
$0								
$0								
$0								
$0								
$0								
$0								
$0								
$0								
$0								
$0								
$0								
$0								
$0								
$0								
$0								
$0								
$0								
$0								
$0								
$0								
$0								
$0	S 1							
$0	E 3							
$0	E 3	$ 20						
$0	$St -1							

Input	Action	
ID f		Shift ID 4
(		Shift (14
[		Shift [5
ID X		Shift ID 4
|	Reduce E ::= ID	
	Reduce E ::= S	Shift E 3
		Shift $ 20
$	Reduce $St ::= E $	Shift $St -1
	Accept	

Print your login name_____

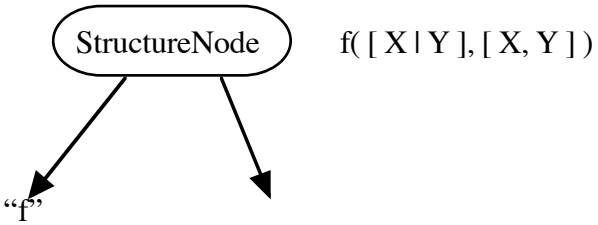
(b) Draw the full parse tree, showing all rules used in the above shift-reduce LALR(1) parse.

(8 marks)



(c) Draw the abstract syntax tree, as specified by the actions associated with the rules. Assume ParamListNode is composed of a vector, containing all the children.

(8 marks)



Print your login name_____

Question 3 8 marks

In assignment 1, question 2, you had to implement a simple compiler that generated assembly language for constructs such as if statements.

Write Java code to complete the implementation of an IfThenElseStmtNode.

```
package node.stmtNode;
import env.*;
import code.*;
import node.exprNode.*;

public class IfThenElseStmtNode extends StmtNode {

      private ExprNode cond;
      private StmtNode thenPart;
      private StmtNode elsePart;

      public IfThenElseStmtNode(
            ExprNode cond,
            StmtNode thenPart,
            StmtNode elsePart ) {
            this.cond = cond;
            this.thenPart = thenPart;
            this.elsePart = elsePart;
            }

      public String toString() {
```

```
            }

      public void genDeclCode( Env env ) {
            thenPart.genDeclCode( env );
            elsePart.genDeclCode( env );
            }

      public void genCode() {
            Code.enter();
            Code.labelDefn( "if" );
```

```
            Code.instrn( "blbc", "$t0", "else" );
            Code.labelDefn( "then" );
```

```
            Code.instrn( "br", "end" );
```

```
            elsePart.genCode();
            Code.labelDefn( "end" );
            Code.exit();
            }
      }
```

Question 4 20 marks

A typical "switch" statement in some computer language is:

```
switch ( Expr ) {
      case -1, +3, 5 .. +8, 10:
            Stmt
      case -2, -4, 12 .. 14, +20, 30:
            Stmt
      default:
            Stmt
      }
```

In general, a "switch" statement includes a list of one or more "case" statements, then an optional "default" statement. The "default" statement, if it exists, must occur at the end.

A "case" statement such as

```
      case -2, -4, 12 .. 14, 20, 30:
            Stmt
```

includes a comma separated "case expression list" containing one or more alternative "case expression"s, for example

```
-2, -4, 12 .. 14, +20, 30
```

Each "case expression" in the "case expression list" can be either a "signed integer", such as

```
-4
```

or a "range" of signed integers, such as

```
12 .. 14
```

A "signed integer" is an "integer constant", optionally preceded by a "+" or "-" sign, for example `-2` or `+20` or `30`.

A default statement is of the form

```
default:
      Stmt
```

Write a CUP grammar definition for a "switch" statement for the language. You do not have to write any actions.

You do not have to write the grammar for statements or expressions. You should assume an "integer constant" is a terminal symbol.

Print your login name_____

Print your login name_____

Appendices

The CUP grammar

```
terminal
      LEXERROR,
      COMMA,      LEFT, RIGHT,      LEFTSQ,     RIGHTSQ,    BAR;
//    ,           (           )           [           ]     |

terminal String IDENT;        //    [A-Za-z][A-Za-z0-9]*

nonterminal ParamListNode
      ParamList;
nonterminal ExprNode
      Expr, List, ElementListOpt, ElementList, Structure;

start with Expr;

Expr::=
            IDENT:ident
            {:
            RESULT = new NameNode( ident );
            :}
      |
            Structure:structure
            {:
            RESULT = structure;
            :}
      |
            List:list
            {:
            RESULT = list;
            :}
      ;

Structure::=
            IDENT:ident LEFT ParamList:paramList RIGHT
            {:
            RESULT = new StructureNode( ident, paramList );
            :}
      ;

ParamList::=
            Expr:expr
            {:
            RESULT = new ParamListNode( expr );
            :}
      |
            ParamList:paramList COMMA Expr:expr
            {:
            paramList.addElement( expr );
            RESULT = paramList;
            :}
      ;

List::=
            LEFTSQ ElementListOpt:elementList RIGHTSQ
            {:
            RESULT = elementList;
            :}
      ;
```

```
ElementListOpt::=
            ElementList:elementList
            {:
            RESULT = elementList;
            :}
      |
            /* Empty */
            {:
            RESULT = new EmptyListNode();
            :}
      ;

ElementList::=
            Expr:expr
            {:
            RESULT = new NonEmptyListNode( expr, new EmptyListNode() );
            :}
      |
            Expr:expr COMMA ElementList:elementList
            {:
            RESULT = new NonEmptyListNode( expr, elementList );
            :}
      |
            Expr:expr BAR Expr:tail
            {:
            RESULT = new NonEmptyListNode( expr, tail );
            :}
      ;
```

Tables for the CUP grammar

Grammar Rules (Productions)

```
[0] $START ::= Expr EOF
[1] Expr ::= IDENT
[2] Expr ::= Structure
[3] Expr ::= List
[4] Structure ::= IDENT LEFT ParamList RIGHT
[5] ParamList ::= Expr
[6] ParamList ::= ParamList COMMA Expr
[7] List ::= LEFTSQ ElementListOpt RIGHTSQ
[8] ElementListOpt ::= ElementList
[9] ElementListOpt ::=
[10] ElementList ::= Expr
[11] ElementList ::= Expr COMMA ElementList
[12] ElementList ::= Expr BAR Expr
```

Action Table

```
From state #0
      LEFTSQ:SHIFT(state 5) IDENT:SHIFT(state 4)
From state #1
      EOF:REDUCE(rule 2) COMMA:REDUCE(rule 2) RIGHT:REDUCE(rule 2)
      RIGHTSQ:REDUCE(rule 2) BAR:REDUCE(rule 2)
From state #2
      EOF:REDUCE(rule 3) COMMA:REDUCE(rule 3) RIGHT:REDUCE(rule 3)
      RIGHTSQ:REDUCE(rule 3) BAR:REDUCE(rule 3)
From state #3
      EOF:SHIFT(state 20)
From state #4
      EOF:REDUCE(rule 1) COMMA:REDUCE(rule 1) LEFT:SHIFT(state 14)
      RIGHT:REDUCE(rule 1) RIGHTSQ:REDUCE(rule 1) BAR:REDUCE(rule 1)
From state #5
      LEFTSQ:SHIFT(state 5) RIGHTSQ:REDUCE(rule 9) IDENT:SHIFT(state 4)
From state #6
      COMMA:SHIFT(state 10) RIGHTSQ:REDUCE(rule 10) BAR:SHIFT(state 11)
From state #7
      RIGHTSQ:REDUCE(rule 8)
From state #8
      RIGHTSQ:SHIFT(state 9)
From state #9
      EOF:REDUCE(rule 7) COMMA:REDUCE(rule 7) RIGHT:REDUCE(rule 7)
      RIGHTSQ:REDUCE(rule 7) BAR:REDUCE(rule 7)
From state #10
      LEFTSQ:SHIFT(state 5) IDENT:SHIFT(state 4)
From state #11
      LEFTSQ:SHIFT(state 5) IDENT:SHIFT(state 4)
From state #12
      RIGHTSQ:REDUCE(rule 12)
From state #13
      RIGHTSQ:REDUCE(rule 11)
From state #14
      LEFTSQ:SHIFT(state 5) IDENT:SHIFT(state 4)
From state #15
      COMMA:SHIFT(state 17) RIGHT:SHIFT(state 18)
From state #16
      COMMA:REDUCE(rule 5) RIGHT:REDUCE(rule 5)
From state #17
      LEFTSQ:SHIFT(state 5) IDENT:SHIFT(state 4)
From state #18
      EOF:REDUCE(rule 4) COMMA:REDUCE(rule 4) RIGHT:REDUCE(rule 4)
      RIGHTSQ:REDUCE(rule 4) BAR:REDUCE(rule 4)
From state #19
      COMMA:REDUCE(rule 6) RIGHT:REDUCE(rule 6)
From state #20
      EOF:REDUCE(rule 0)
```

Reduce (GoTo) Table

```
From state #0:
      Expr:GOTO(3)
      List:GOTO(2)
      Structure:GOTO(1)
From state #1:
From state #2:
From state #3:
From state #4:
From state #5:
      Expr:GOTO(6)
      List:GOTO(2)
      ElementListOpt:GOTO(8)
      ElementList:GOTO(7)
      Structure:GOTO(1)
From state #6:
From state #7:
From state #8:
From state #9:
From state #10:
      Expr:GOTO(6)
      List:GOTO(2)
      ElementList:GOTO(13)
      Structure:GOTO(1)
From state #11:
      Expr:GOTO(12)
      List:GOTO(2)
      Structure:GOTO(1)
From state #12:
From state #13:
From state #14:
      ParamList:GOTO(15)
      Expr:GOTO(16)
      List:GOTO(2)
      Structure:GOTO(1)
From state #15:
From state #16:
From state #17:
      Expr:GOTO(19)
      List:GOTO(2)
      Structure:GOTO(1)
From state #18:
From state #19:
From state #20:
```
