Computer Science 330 Language Implementation Test 2001 Thursday 5th April 6.30-8.00pm

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 question sheets 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. Always show your working - most marks are for showing you know what you are doing, rather than just getting the right answer. Attempt all questions. Questions total 100 marks. The test counts for 10% of the total mark.

Question 1

Consider the following CUP grammar. Note that it is DIFFERENT from the one in the assignment.

```
terminal SEMICOLON, IF, THEN, ELSE, ELIF, FI, WHILE, DO, DONE, ERROR;
terminal String WORD;

non terminal
      StmtList, Stmt, ElseOpt, WordList;

start with StmtList;

StmtList::=
            Stmt
      |
            StmtList Stmt
      ;

Stmt::=
            WordList SEMICOLON
      |
            IF WordList THEN StmtList ElseOpt FI
      |
            WHILE WordList DO StmtList DONE
      ;

ElseOpt::=
            /* Empty */
      |
            ELIF WordList THEN StmtList ElseOpt
      |
            ELSE StmtList
      ;

WordList::=
            WORD
      |
            WORD WordList
      ;
```

Assume that

- Line breaks are represented by carriage returns “\r”, linefeeds “\n” or carriage return/linefeed pairs “\r\n”. Line breaks are consumed, and not returned by the lexical analyser.
- White space corresponds to blanks “ ”, or tabs “\t”. White space is consumed, and not returned by the lexical analyser.
- Any text starting with a “#” up to a line break is treated as a comment.
- The terminal symbols SEMICOLON, IF, THEN, ELSE, ELIF, FI, WHILE, DO, DONE correspond to “;”, “if”, “then”, “else”, “elif”, “fi”, “while”, “do”, and “done”.

- WORD corresponds to any sequence of visible textual ASCII characters other than “ ”, “#”, and “;”, that do not match a reserved word.
- ERROR is not a real terminal. It is returned by the lexical analyser whenever it cannot match a token.

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

```
if a b then c; fi
```

Show both the symbols and states on the stack, the current token, and the action performed at each stage. Note that the individual tokens are “if”, “a”, “b”, “then”, “c”, “;”, “fi”. (20 marks)

(b) Draw the full parse tree, showing all rules used in the above shift-reduce parse. (7 marks)

(c) (i) Indicate which nonterminals are nullable. (1 mark)

(ii) Draw the first graph, and compute the first sets for this grammar. (10 marks)

(iii) Draw the follow graph, and compute the follow sets for this grammar. (15 marks)

(d) Write a JLex program suitable for lexically analysing input for the above language. (14 Marks)

(e) Indicate how a parse tree could be built, and a treewalk of the parse tree performed for the above grammar, by:

(i) Adding in actions, etc for the rules for ElseOpt. (4 marks)

(ii) Indicating the code in the main program to perform the parse, create the parse tree, then invoke a method to perform a treewalk of the parse tree. (4 marks)

Question 2

Suppose that we have a computer language, with variable declarations similar to those in Java.

```
int a, b, c = a + b, d;
int[][] arrayA = new int[ 3 ][ 3 ], arrayB, arrayC = arrayA;
TextField textField = new TextField( “你好吗      ?” );
TextField[] result;
```

There are no modifiers. Types can be either identifiers or array types (treat primitive types such as “int” as identifiers). A declaration starts with a type, followed by a comma separated list of the identifiers being declared, possibly with initial values specified by expressions, and is terminated by a “;”.

Write a CUP grammar to parse any sequence of variable declarations. You may assume that grammar rules have been provided for expressions. You do not have to write any actions.

(25 Marks)

Appendix

Tables for the CUP grammar

Grammar Rules (Productions)

```
10: WordList ::= WORD WordList
9: WordList ::= WORD
8: ElseOpt ::= ELSE StmtList
7: ElseOpt ::= ELIF WordList THEN StmtList ElseOpt
6: ElseOpt ::=
5: Stmt ::= WHILE WordList DO StmtList DONE
4: Stmt ::= IF WordList THEN StmtList ElseOpt FI
3: Stmt ::= WordList SEMICOLON
2: StmtList ::= StmtList Stmt
1: StmtList ::= Stmt
0: $START ::= StmtList EOF
```

Action Table

```
From state #0
      IF:SHIFT(5) WHILE:SHIFT(4) WORD:SHIFT(1)
From state #1
      SEMICOLON:REDUCE(9) THEN:REDUCE(9) DO:REDUCE(9)
      WORD:SHIFT(1)
From state #2
      SEMICOLON:SHIFT(25)
From state #3
      EOF:REDUCE(1) IF:REDUCE(1) ELSE:REDUCE(1)
      ELIF:REDUCE(1) FI:REDUCE(1) WHILE:REDUCE(1)
      DONE:REDUCE(1) WORD:REDUCE(1)
From state #4
      WORD:SHIFT(1)
From state #5
      WORD:SHIFT(1)
From state #6
      EOF:SHIFT(8) IF:SHIFT(5) WHILE:SHIFT(4)
      WORD:SHIFT(1)
From state #7
      EOF:REDUCE(2) IF:REDUCE(2) ELSE:REDUCE(2)
      ELIF:REDUCE(2) FI:REDUCE(2) WHILE:REDUCE(2)
      DONE:REDUCE(2) WORD:REDUCE(2)
From state #8
      EOF:REDUCE(0)
From state #9
      THEN:SHIFT(10)
From state #10
      IF:SHIFT(5) WHILE:SHIFT(4) WORD:SHIFT(1)
From state #11
      IF:SHIFT(5) ELSE:SHIFT(12) ELIF:SHIFT(13)
      FI:REDUCE(6) WHILE:SHIFT(4) WORD:SHIFT(1)
From state #12
      IF:SHIFT(5) WHILE:SHIFT(4) WORD:SHIFT(1)
From state #13
      WORD:SHIFT(1)
From state #14
      FI:SHIFT(15)
From state #15
      EOF:REDUCE(4) IF:REDUCE(4) ELSE:REDUCE(4)
      ELIF:REDUCE(4) FI:REDUCE(4) WHILE:REDUCE(4)
      DONE:REDUCE(4) WORD:REDUCE(4)
```

```
From state #16
      THEN:SHIFT(17)
From state #17
      IF:SHIFT(5) WHILE:SHIFT(4) WORD:SHIFT(1)
From state #18
      IF:SHIFT(5) ELSE:SHIFT(12) ELIF:SHIFT(13)
      FI:REDUCE(6) WHILE:SHIFT(4) WORD:SHIFT(1)
From state #19
      FI:REDUCE(7)
From state #20
      IF:SHIFT(5) FI:REDUCE(8) WHILE:SHIFT(4)
      WORD:SHIFT(1)
From state #21
      DO:SHIFT(22)
From state #22
      IF:SHIFT(5) WHILE:SHIFT(4) WORD:SHIFT(1)
From state #23
      IF:SHIFT(5) WHILE:SHIFT(4) DONE:SHIFT(24)
      WORD:SHIFT(1)
From state #24
      EOF:REDUCE(5) IF:REDUCE(5) ELSE:REDUCE(5)
      ELIF:REDUCE(5) FI:REDUCE(5) WHILE:REDUCE(5)
      DONE:REDUCE(5) WORD:REDUCE(5)
From state #25
      EOF:REDUCE(3) IF:REDUCE(3) ELSE:REDUCE(3)
      ELIF:REDUCE(3) FI:REDUCE(3) WHILE:REDUCE(3)
      DONE:REDUCE(3) WORD:REDUCE(3)
From state #26
      SEMICOLON:REDUCE(10) THEN:REDUCE(10) DO:REDUCE(10)
```

Reduce (GoTo) Table

```
From state #0:
      StmtList:GOTO(6)
      Stmt:GOTO(3)
      WordList:GOTO(2)
From state #1:
      WordList:GOTO(26)
From state #2:
From state #3:
From state #4:
      WordList:GOTO(21)
From state #5:
      WordList:GOTO(9)
From state #6:
      Stmt:GOTO(7)
      WordList:GOTO(2)
From state #7:
From state #8:
From state #9:
From state #10:
      StmtList:GOTO(11)
      Stmt:GOTO(3)
      WordList:GOTO(2)
From state #11:
      Stmt:GOTO(7)
      ElseOpt:GOTO(14)
      WordList:GOTO(2)
From state #12:
      StmtList:GOTO(20)
      Stmt:GOTO(3)
      WordList:GOTO(2)
From state #13:
      WordList:GOTO(16)
```

```
From state #14:
From state #15:
From state #16:
From state #17:
      StmtList:GOTO(18)
      Stmt:GOTO(3)
      WordList:GOTO(2)
From state #18:
      Stmt:GOTO(7)
      ElseOpt:GOTO(19)
      WordList:GOTO(2)
From state #19:
From state #20:
      Stmt:GOTO(7)
      WordList:GOTO(2)
From state #21:
From state #22:
      StmtList:GOTO(23)
      Stmt:GOTO(3)
      WordList:GOTO(2)
From state #23:
      Stmt:GOTO(7)
      WordList:GOTO(2)
From state #24:
From state #25:
From state #26:
```

This page deliberately left blank

Computer Science 330

Language Implementation Test 2001

Thursday 5th April

Answer Booklet

Surname
Given Names
Student ID Number
Login Name

1(a)___________________________/20

1(b)____________________________/7

1(c) (i) _________________________/1

1(c) (ii)________________________/10

1(c) (iii) _______________________/15

1(d)___________________________/14

1(e) ____________________________/8

2 _____________________________/25

Total_________________________/100

Do Not write on this page.

It will not be returned to you.

Print your login name ______________________________

Question 1

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

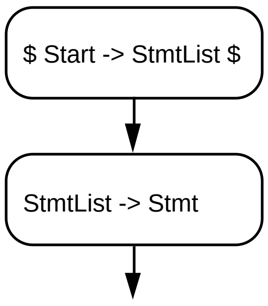
```
if a b then c; fi
```

Stack

$ 0							
$ 0	if 5						
$ 0							
$ 0							
$ 0							
$ 0							
$ 0							
$ 0							
$ 0							
$ 0							
$ 0							
$ 0							
$ 0							
$ 0							
$ 0							
$ 0							
$ 0	SL 6	$ 8					
$ 0	$S' -1						

Input	Action
if	Shift if 5
WORD a	Shift WORD 1
-	Reduce $S' -> SL $
	Accept

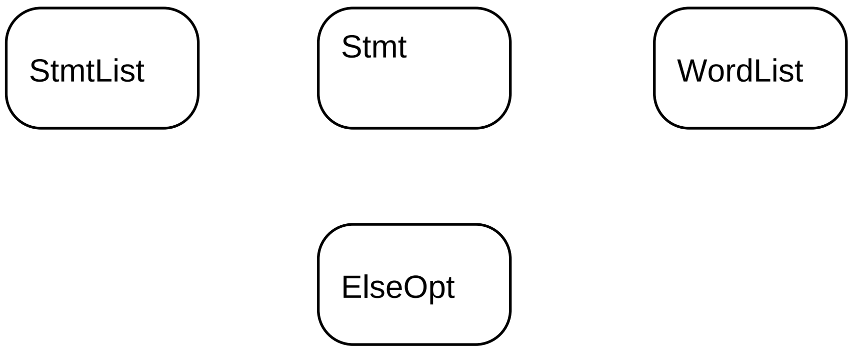
(b) Draw the full parse tree, showing all rules used in the above shift-reduce parse.



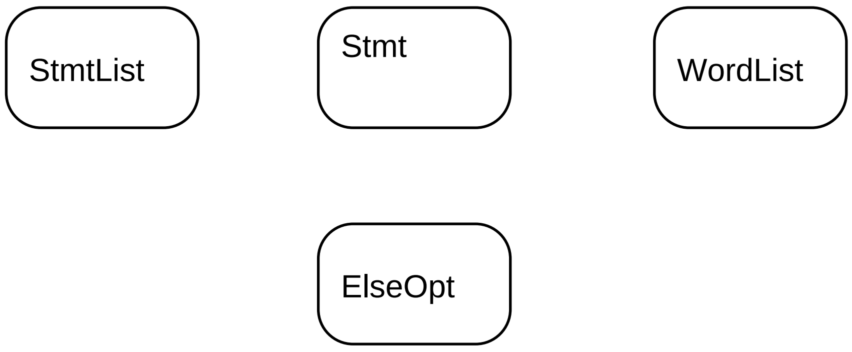
Print your login name ______________________________

(c)

Draw the first graph for this grammar.



Draw the follow graph for this grammar.



Indicate which nonterminals are nullable, and the first and follow sets for the grammar

Symbol	Nullable	First Set	Follow Set
StmtList			
Stmt			
ElseOpt			
WordList			

(d) Write a JLex program suitable for lexically analysing input for the above language.

```
import java.io.*;
import java_cup.runtime.*;
%%
%public
%type      Symbol
%char
%{
      public Symbol token( int tokenType ) {
            return new Symbol( tokenType,
                  yychar, yychar + yytext().length(), yytext() );
            }
%}
%init{

%init}
%eofval{

%eofval}
```

```
%state NORMAL, ERROR
%%

<NORMAL>if                           { return token( sym.IF ); }
<NORMAL>then                         { return token( sym.THEN ); }
<NORMAL>elif                         { return token( sym.ELIF ); }
<NORMAL>else                         { return token( sym.ELSE ); }
<NORMAL>fi                           { return token( sym.FI ); }
<NORMAL>while                        { return token( sym.WHILE ); }
<NORMAL>do                           { return token( sym.DO ); }
<NORMAL>done                         { return token( sym.DONE ); }
```

Print your login name ____

(e) (i) Actions, etc, for ElseOpt.

(e) (ii) Code in Main.main().

Question 2

Write a CUP grammar to parse any sequence of variable declarations. You may assume that grammar rules have been provided for expressions. You do not have to write any actions.

```
terminal


non terminal



start with DeclarationSequence;

DeclarationSequence::=
```