3.

a.

Explain what is meant by *the dangling-else ambiguity*, and describe the solutions which can be employed to overcome it.

[10 marks]

b.

Consider the following grammars, which have non-terminals {E, C, F} and terminals {**id**, **or**, **and**} with start symbol E. Each grammar generates the same language, which may be thought of as that of Boolean expressions with E as expression, C as conjunct, and F as factor.

Grammar A	Grammar L	Grammar R
E → E **or** E	E → E **or** C	E → C **or** E
E → E **and** E	E → C	E → C
E → **id**	C → C **and** F	C → F **and** C
	C → F	C → F
	F → **id**	F → **id**

i. Write down parse trees for the expression a **or** b **or** c for each of these grammars.

ii. Explain in what way these grammars would be unsuitable for implementing Boolean expressions with C/C++ style of semantics by a top-down parser.

iii. Show how grammar L can be transformed in a systematic way into a new grammar which avoids these problems.

iv. Extend the new grammar so that it can handle fully parenthesised expressions.

[15 marks]

4.

a.

The following two augmented grammars are expressed in yacc:

i)
```
list  : list INT {print($2);}
      | INT
```

ii)
```
list  : INT list {print($1);}
      |
```

From each derive a parse tree for the sentence 1 2 3 (which when tokenised becomes INT INT INT). By traversing that tree show the results of executing the print actions.

[7 marks]

b.

Consider the following definition

> An *atom* is either **a** or **b**
>
> A *list* is a left parenthesis, followed by zero or more atoms or lists, followed by a right parenthesis.

An example of a list is `( () ((a) (a b)) )`

i. Define a grammar for *list* in yacc.

ii. Carefully construct a parse tree using your grammar for the example list given.

[18 marks]

2.

a.

Explain the terms

shift-reduce conflict
reduce-reduce conflict

as met in yacc, and also explain what if anything should be done about removing the cause of any such conflicts. [6 marks]

b.

The following S-R table was derived from output produced by yacc when given the input:

```
s : s ',' s
s : 'x'
```

	input	Action		GOTO
state	x	,	$end	s
0	s1			2
1	r2	r2	r2	
2		s3	accept	
3	s1			4
4		s3	r1	

i. Explain how the table entries s1 and r2 are to be interpreted. Also explain how a blank entry is to be interpreted.

ii. Show the moves which the parser would make when parsing the sentence

x_1 , x_2 , x_3

[Hint: remember to initialise the stack with $ 0, corresponding to state 0.]

iii. Construct the corresponding parse tree, *labelling each mode with the number of the move from the parse.*

[10 marks]

During the processing of the grammar, yacc reported a conflict.

iv. Explain the nature of this conflict.

v. Show how the grammar could be modified to remove the conflict, without affecting the structure of the parse tree.

vi. Justify your modification, possibly by comparing its effects with that of a similar modification which you could have made.

[9 marks]

CONTINUED

Computer Science B224 Systems Software

The use of electronic calculators is *NOT* permitted.

Answer Question 1, which is worth 50 marks, and two others.

1.

a.

Consider the C/C++ statement

```
do
        sum = sum + 100;
while (sum <= 1000);
```

i. By reference to the above example, explain the terms *lexeme, keyword, token, token type*, and *token value*.

ii. Show the kind of code which a simple compiler might generate for a stack machine from the statement. [6 marks]

b.

Explain the terms:
recursive-descent compiler, top-down parsing, and *bottom-up parsing.* [6 marks]

c.

i. Draw a syntax diagram for the (generic) C/C++ *do-while* statement. You may assume that syntax diagrams for *statement* and *expression* (including truth-valued expressions) exist.

ii. Suppose that the existing syntax diagrams for *expression* do not cover the boolean operators || (or) and && (and). Show how, using the diagrams defining expression, you would incorporate the boolean operators with their C/C++ priorities into expressions. State any assumptions you make.

iii. Annotate your syntax diagrams so as to generate code for a simple stack-based machine.

Sketch an implementation of your syntax diagrams as procedures (functions) in a recursive-descent compiler. Briefly describe the purpose of any other procedures or variables to which your implementation refers. In the case of variables, for each one you should make it quite clear whether it is local or global, and explain why. You may refer to `newlabel()`, a function which returns a unique label each time it is called.

Your implementation should work with nested constructs, and you should indicate how your design allows this. [19 marks]

d.

i. Explain the purpose and use of *pseudovariables* in *yacc*.

ii. Show how the do-while statement can be implemented using yacc, assuming that yacc code to compile *statement* and *expression* exists.

iii. Explain how priorities and associativity can be given to operators in yacc, using as an example the boolean operators || (or) and && (and). [10 marks]

e.

Explain the purpose and structure of a *makefile*, illustrating your answer with short examples. [5 marks]

f.

Comment on the ambiguity or otherwise of the C/C++ conditional expression, which is of the form *expression ? expression : expression*. [4 marks]
