UNIVERSITY OF DUBLIN

TRINITY COLLEGE

CS3BA71

Faculty of Engineering and Systems Sciences

School of Engineering

BA (Mod) Computer Science
Junior Sophister Examination

Trinity Term, 2000

3BA7 — Software Engineering and Compiler Design

Tuesday 6 June, 2000 Examination Hall 9:30 - 12:30

Dr JA Redmond, Dr DM Abrahamson

Attempt five questions, at least two from each section.
Please use separate answer books for each section.

Section A

1. i. Responsibility-Driven Design with Classes, Responsibilities and Collaborations technique (RDD with CRC—Wirf's Brock et al) can be represented in 26 steps in six phases. Give the six phases and discuss briefly what each phase does.

   ii. Comment on RDD with CRC as an object-oriented design method based on your experience (or not) of using it in the Lift and Car Rally assignments. If your team did not use RDD with CRC give the reasons why and explain and justify the method the team did use.

... / Contd

1 Contd / ...

iii. Give four items you would like to see in an automated object-oriented design package for RDD with CRC.

iv. Give four rules-of-thumb which are useful in RDD with CRC.

2. i. Briefly give the structure of the Coverdale Systematic Approach to Task Completion (SA).

ii. Indicate how your team used (or not) SA to control your Lift and Car Rally assignments. If not, indicate why and describe the process which the team actually followed.

iii. Describe the team processes and how the teams operated in these two assignments, commenting on any evolution which occurred in the team.

iv. Comment on What Went Well and Why (WWW & W) with each assignment and on What Could Be Improved and How (WCBI & H).

3. Discuss three of the following four paradigms of the software development process giving a concise discussion of each, its advantages and disadvantages, what problems it sets out to solve, and its likely future prospects:

i. The waterfall software lifecycle
ii. Rapid Incremental Application Development (RIAD)
iii. The transformational approach
iv. Exploratory programming (also called Structured Growth)

4. i. Discuss Mantei's three programming teams giving the management structure, communication channels, advantages and disadvantages of each. Summarize the characteristics of each in a table.

   ii. You have been appointed project manager of a project which is estimated to take three years and it is estimated that the final product will consist of two million source lines of code. Assuming a productivity rate of 10 source lines of completed code per day per programmer, estimate the number of programming teams needed, show how you would structure them and estimate the amount of administrative staff necessary.

   iii. Which of Mantei's three teams corresponds to the structure of your team during the Lift and Car Rally assignments? What characteristic(s) of this team type did your team obviously exhibit?

Section B

5. Using finite state techniques, design a simple lexical analyzer to recognize XML tokens for *Comments and Character Data Sections* described by the following grammar; a complete set of test inputs (designed to visit all non-error entries in the transition table) should be included with the design description.

S	::=	(#x20 | #x9 | #xD | #xA)+
Comment	::=	'<!--' ((Char - '-') | ('-' (Char - '-')))* '-->'
CDSect	::=	CDStart CData CDEnd
CDStart	::=	'<![CDATA['
CData	::=	(Char* - (Char* ']]>' Char*))
CDEnd	::=	']]>'

where `Char` represents any ascii character in the range `#x21` to `#xFF` or a sequence of white space characters `S` as defined above.

... / Contd

5 Contd / ...

For example, given the input:

<!--CDATA sections are used to escape markup within text-->

<![CDATA[<greeting>Hello, World!</greeting>]]>

the lexical analyzer should return the following two tokens:

(Comment, 'CDATA sections are used to escape markup within text')

(CDATA, '<greeting>Hello, World!</greeting>')

6. Consider the following translation grammar with starting symbol **<S>**:

```
<S>  ->  <E> {RESULT}
<E>  ->  <E> + <T>
<E>  ->  <T>
<T>  ->  <T> * <F>
<T>  ->  <F>
<F>  ->  <P> ↑ <F>
<F>  ->  <P>
<P>  ->  ( <E> )
<P>  ->  Const
```

where **Const** is a lexical token representing an integer constant,
and ↑ is the exponentiation operator.

i By eliminating left recursion, find an equivalent LL(1) grammar that generates the same language (sequences of terminal symbols), compute the selection set for each production in the grammar and finally add attributes and associated attribute evaluation rules so that the action symbol **{RESULT}** will have an inherited attribute that is equal to the numerical value of the arithmetic expression generated by <E>.

ii. Draw a fully decorated derivation tree for the expression:

2 ↑ 3 ↑ 2

7. i. Show clearly why the following two productions for an if-statement are not LL(1), and outline a simple technique that may be used to process them in a pushdown machine:

```
if <condition> then <statement> else <statement>
if <condition> then <statement>
```

ii. In relation to translation grammars, show by example the differences between inherited and synthesized attributes of non-terminal symbols. How are these two attribute types handled during recursive descent parsing?

8. i. Describe in detail the design and use of the procedure skip_to in global error recovery during recursive-descent parsing.

ii. Design an attributed translation grammar for a loop statement of the form:

```
for (init ; update ; final) do statement;
```

where `init` specifies an initial value for the control variable, `update` specifies an increment or decrement operation on the control variable, and `final` specifies a terminating condition for the loop. For example, if `n` is greater than zero,

```
for (index:=0 ; index:=index+1 ; index<n) do …;
```

specifies a loop that should be executed `n` times.

Discuss the function of the action symbols along with the form of object code that should be produced by the code generator.