Ollscoil na hÉireann, Gaillimh GX 1480
National University of Ireland, Galway

Semester I Examinations, 2003/2004

Exam Code(s)	3IF1 1MF2 Erasmus
Exam(s)	3rd Year Examination in Information Technology M.Sc. Degree (Software Design & Development) (Stream II) Erasmus
Module Code(s)	CT331
Module(s)	Programming Paradigms
Paper No.	
Repeat Paper	Special Paper
External Examiner(s)	Professor P. Nixon
Internal Examiner(s)	Professor G. Lyons Ms. J. Griffith
Instructions:	Answer **THREE** questions. All questions carry equal marks.
Duration	**2hrs**
No. of Answer books	
Requirements:	
Handout	
MCQ	
Statistical Tables	
Graph Paper	
Log Graph Paper	
Other Material	
No. of Pages	4
Department(s)	Information Technology

Q. 1. (i) What is meant by a *programming paradigm? (4)*
Distinguish between the imperative and declarative programming paradigms. Use sample code from a language of each of the two paradigms to support your answer. *(6)*

(ii) Describe the main features of the event-driven paradigm. *(4)*
Describe the three main approaches to event processing that can be taken by event-driven languages. *(6)*

(iii) MS Visual Basic provides a large amount of flexibility with respect to the approaches which can be taken when declaring variables. Discuss the advantages and disadvantages of this flexibility, listing some of the different approaches which can be taken. *(10)*

Q. 2. (i) What is meant by programming language abstraction? *(4)*
With respect to procedural abstraction, discuss the problems that can arise with *side-effects* and *aliasing*. Use examples to support you answer. *(6)*

(ii) With respect to data abstraction, what is meant by a *binding*? *(2)*
Describe the four distinct binding times that can exist. *(4)*
With the aid of an example distinguish between the binding times in the SCHEME special forms `let*` and `let`. *(4)*

(iii) Discuss the main features of the concurrent programming paradigm. *(4)*
With respect to the concurrent paradigm, distinguish between the shared memory model and the message passing model for both communication and synchronisation. *(6)*

Q.3. (i) Describe what is meant by *each* of the following with respect to program translation:

a) error recovery and error repair. *(5)*

b) symbol tables. *(5)*

(ii) Explain what is meant by a Finite State Automaton (FSA). *(2)*
Illustrate, with the aid of a diagram, a FSA which combines automata to recognise the following lexical tokens: *(8)*

- identifiers (token: `ID`).
- if keyword (token: `IF`).
- positive integers (token: `NUM`).
- positive reals (token: `REAL`).

You can assume only valid strings will be read.

(iii) The following production rules, P1 and P2, are two alternative production rules for P in the grammar given which should describe a restricted set of algebraic expressions. By parsing sample strings and obtaining parse trees, discuss the problems, if any, with P1 and P2: *(10)*

Grammar = {N, T, S, P}
N = {<goal>, <expr>, <term>, <factor>}
T = {a, b, c, +, ×}
S = <goal>

P1 =
<goal> ::= <expr>
<expr> ::= <term> | <term> × <expr>
<term> ::= <factor> | <factor> + <term>
<factor> ::= a | b | c

P2 =
<goal> ::= <expr>
<expr> ::= <expr> + <expr> | <expr> × <expr> | <factor>
<factor> ::= a | b | c

Q.4. (i) Describe the main features of the functional programming paradigm. *(5)*
Describe the main data structure available in the functional programming language SCHEME, outlining a representation for the data structure. *(5)*

(ii) With the aid of examples, describe the SCHEME primitives `list` and `append`. *(4)*
Write a function in SCHEME which when passed an element and a list removes all occurrences of the element from the list. *(3)*
Explain the approach taken. *(3)*

(iii) Describe a representation of binary trees in SCHEME. *(4)*
Write code in SCHEME to implement a breadth first search using the binary tree representation developed. *(3)*
Explain the approach taken. *(3)*

Q. 5. (i) Control in SCHEME and PROLOG is mainly achieved through recursion. Distinguish between tail recursive and non-tail recursive functions by writing a tail recursive and non-tail recursive function in SCHEME which returns the reverse of the top level elements in a list. *(6)*

e.g.,
```
(reverse '(a  b  c  d))
```
returns the list `(d c b a)`

For both functions, explain the approach taken. *(4)*

(ii) Describe the main features of the logic programming paradigm. *(4)*
With the aid of examples, distinguish between *facts*, *relations*, *rules* and *queries* in PROLOG. *(6)*

(iii) Describe, with the aid of examples, the list data structure in PROLOG, outlining its representation and syntax. *(4)*
Write code in PROLOG to merge two lists, explaining the steps taken in developing the code. *(6)*