GX 0279

Ollscoil na hÉireann, Gaillimh
National University of Ireland, Galway

Semester I Examinations 2005 / 2006

Exam Code(s)	3IF 3IF Repeat Erasmus
Exam(s)	3rd Year Examination in Information Technology 3rd Year Examination in Information Technology (Repeat) Erasmus
Module Code(s)	CT331
Module(s)	Programming Paradigms
Paper No. Repeat Paper	
External Examiner(s) Internal Examiner(s)	Professor J.A. Keane Dr. M. Madden Ms. J. Griffith

Instructions: Answer **THREE** questions.
All questions carry equal marks.

Duration	**2 hours**
No. of Pages	5
Department(s)	Information Technology
Course Co-ordinator(s)	

Requirements:
MCQ
Handout
Statistical Tables
Graph Paper
Log Graph Paper
Other Material

Q. 1. (i) What is meant by a *programming paradigm? (4)*
With the aid of examples from programming languages of your choice, describe what is meant by:

(a) Syntax and semantics. *(3)*

(b) Keywords and reserved words. *(3)*

(ii) Discuss the main features of the concurrent programming paradigm. *(4)*
With the aid of the following fragment of code, describe what is meant by the term *non-determinism* in relation to concurrent implementations. In the sample code given, assume that the two blocks of code surrounded by the begin and end delimiters can be evaluated concurrently. *(6)*

```
x := 0;
cobegin
     begin x := 1; x := x + 1 end;
     begin x := 2; x := x + 1 end;
coend;
print(x);
```

(iii) Describe the main features of the event-driven paradigm. *(4)*

Discuss the following statement with respect to variable declarations in versions of Visual Basic, up to and including Visual Basic 6.0. *(6)*

> "Many poor features often remain in new revised versions of a programming language to ensure that old programs can be used without modification".

Q.2. (i) With the aid of examples, distinguish between the two main data objects in SCHEME: atoms and lists.*(4)*
With the aid of examples, distinguish between the SCHEME primitives `car`, `cdr` and `cons`. *(6)*

(ii) With the aid of an example, explain what is meant by "fully parenthesised prefix notation" with respect to the functional programming language SCHEME. *(2)*

Write a function in SCHEME which, when passed a list of numbers, finds the maximum (largest) number in the list. *(4)*

e.g., `(max_lst '(3 5 4 19 7))` returns `19`

Explain the approach taken. *(4)*

(iii) Write a function in SCHEME which, when passed an element and a list, searches the list and removes all occurrences of the element from the list. *(6)*

e.g. `(remove 'c '(a b c c b a)))` returns `(a b b a)`

Explain the approach taken. *(4)*

Q. 3. (i) Explain the difference between collateral, sequential and recursive bindings. *(6)*

With the aid of an example distinguish between the binding times in the SCHEME special forms `let*` and `let`. *(4)*

(ii) With respect to control abstraction, give an example of implicit and explicit sequence control in a programming language of your choice. *(4)*

What is meant by the "dangling else" problem? *(2)*
Given the following fragment of code, explain the flow of control through the code, giving the value of the variables after each statement, paying particular attention to a possible "dangling else" problem. *(4)*

```
void main()
{
      int x, y, z;
      y = 1;
      if (y == 0)
        x =3;
      else
        x = 1;
      printf("\n The value of x is %d", x);
      z = (y < 0);
      if (z == 1)
        if (y > -5)
           x = 3;
      else x = 5;
      printf("\n z is %d and x is %d", z, x);
}
```

(iii) With respect to procedural abstraction, distinguish between *reference parameters* (pass by reference) and *value parameters* (pass by value). Use the following fragment of code to explain your answer indicating the values of `m` and `n` when first assuming that the programming language supports pass by value and then assuming that the programming language supports pass by reference. *(10)*

```
Sub Procedure ex1(m As integer, n As integer)
      n = 1
      n = m + n
End Sub
```

In the calling environment the procedure is called with:

```
      i = 5
      ex1(i, i)
```

Q. 4. (i) Describe the main features of the logic programming paradigm. *(4)*

With the aid of examples, distinguish between *facts*, *relations*, *rules* and *queries* in PROLOG. *(4)*

With the aid of an example, show how implicit and explicit unification occurs in PROLOG. *(2)*

(ii) 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)*

(iii) Control in SCHEME and PROLOG is mainly achieved through recursion. Distinguish between tail recursive and non-tail recursive functions by writing both a tail recursive and non-tail recursive version of a function in SCHEME that returns the reverse of the top level elements in a list. *(6)*
e.g.,

```
(reverse_it '(a b c d))
```

returns the list `(d c b a)`

For both functions, explain the approach taken and explain the difference between the tail recursive and non-tail recursive versions. *(4)*

Note that you must write your own functions and not make use of the built-in reverse function in SCHEME.

Q.5. (i) Describe what is meant by *each* of the following with respect to program translation:

(a) Lexical Analysis. *(3)*
(b) Symbol Tables. *(3)*
(c) Cascading Errors. *(2)*
(d) Peephole Optimisation. *(2)*

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

- positive and negative integer numbers (e.g. 23, -14, etc.).
- positive and negative real numbers (e.g. 2.333, -101.1, etc.).

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 the sample strings:

a + b × c
a × b × c

obtain parse trees and 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