Ollscoil na hÉireann, Gaillimh *GX_____*
National University of Ireland, Galway

Semester I Examinations 2007 / 2008

Exam Code(s) 3IF1
1EM1

Exam(s) 3rd Year Examination in Information Technology
Erasmus

Module Code(s) CT331

Module(s) Programming Paradigms

Paper No.
Repeat Paper

External Examiner(s) Professor J.A. Keane
Internal Examiner(s) Professor G. Lyons
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, distinguish between the imperative and declarative programming paradigms. *(6)*

(ii) Describe what is meant by syntax and semantics. *(4)*

With respect to syntax, what syntactic elements do the languages Python and VB.NET use to group code into blocks? Provide sample code fragments to illustrate your answer. *(6)*

(iii) VB.NET is an example of an event-driven paradigm. Using examples from VB.NET, describe the main features of the event-driven paradigm. *(4)*

Describe the three main approaches to event processing that can be taken by an event-driven language. *(6)*

Q. 2. (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)*

(ii) Control in SCHEME and PROLOG is mainly achieved through recursion. With the aid of an example in SCHEME describe: *(2)*

- (a) what is meant by recursion? *(3)*
- (b) what is the difference between tail recursion and non-tail recursion? *(3)*
- (c) what is the role of the call stack (run time stack) when using non-tail recursive functions? *(2)*
- (d) show the contents of the call stack (run time stack) for a small sample run of the recursive code you develop. *(2)*

(iii) With the aid of an example, describe a representation of binary trees in SCHEME. *(3)*
Write code in SCHEME to implement a depth first search using the binary tree representation you developed. *(5)*
Explain the approach taken. *(2)*

Q.3. (i) Describe the main features of the functional programming paradigm. *(2)*

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

Distinguish between the SCHEME primitives `car` and `cdr` by writing sequences of `cars` and `cdrs` to extract the symbol "**and**" from the following expressions: *(6)*

(a) `(this and that them they those)`
(b) `(those they (this and that) then)`
(c) `((then those) they (this and that))`

(ii) Distinguish between the two main data objects in SCHEME: atoms and lists. *(4)*

Distinguish between the SCHEME primitives `cons`, `list` and `append` by explaining what the output from each of the following SCHEME expressions is: *(6)*

(a) `(cons 1 '(a b c))`
(b) `(list 1 '(a b c))`
(c) `(append '(1) '(a b c))`
(d) `(list '(3 2) '(1 2 3))`
(e) `(append '(3 2) '(1 2 3))`
(f) `(cons '(3 2) '(((1 2 3))))`

(iii) Write a function in SCHEME which, when passed a list and a number, adds the number to the first element of the list if that element is a number, otherwise it returns the list unchanged. *(8)*
e.g. if the function is named `sum_it`:

```
(sum_it 3 '(1 2 a b)) returns (4 2 a b)
(sum_it 3 '(a b 1 2)) returns (a b 1 2)
```

Explain the approach taken. *(2)*

Q. 4. (i) Distinguish between strong and weak typed languages using the following as an example which shows sample output in a strong and weak typed language: *(8)*

```
Python >>>> 2+ "day"
→ TypeError: unsupported operand type(s) for +: 'int'
   and 'str'

Perl:DB<1> 2+"day"
→ 2day
```

(ii) Explain the difference between collateral, sequential and recursive bindings. *(6)*

Given the following code fragments in SCHEME, distinguish between the binding times in the SCHEME special forms `let*` and `let,` identifying the type of binding that is used and explaining the output in each case. *(6)*

```
(define x 100)
(let ( (x 2)
       ( y (+ x x))
     )
  (+ x y))
```

```
(define x 100)
(let* ( (x 2)
        ( y (+ x x))
      )
  (+ x y))
```

(iii) The following VB.NET procedure inserts a value into an integer array, where the procedure is passed the array (`A`), the upper bound of the array (`ub`), the size of the array (total space allocated to the array) and the item to insert (`item`):

```
Private Sub Insert(ByRef A() As Integer, ByRef ub As
Integer, ByVal size As Integer, ByVal item As Integer)

        If ub = size-1 Then
            MsgBox("Error: Not enough Space")
        Else
            ub = ub + 1
            A(ub) = item
        End If
 End Sub
```

Using some sample data explain the difference between passing data to the procedure by value (`ByVal`) and passing by reference (`ByRef`). What is the effect of using the following procedure definition: *(10)*

```
Private Sub Insert(ByVal A() As Integer, ByVal ub As
Integer, ByVal size As Integer, ByVal item As Integer)
```

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

(a) Just In Time Compilation. *(2)*
(b) Lexical Analysis. *(2)*
(c) Error Recovery. *(2)*
(d) Semantic Analysis. *(2)*
(e) 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)*

- identifiers
- `if` keyword
- positive real numbers

(iii) The following specifies a simple grammar which should describe a restricted set of algebraic expressions:

Grammar = {N, T, S, P}
N = {<goal>, <expr>, <term>, <factor>}
T = {`2, 4, 6, +, ×` }
S = <goal>
P =

<goal> ::= <expr>
<expr> ::= <term> | <term> × <expr>
<term> ::= <factor> | <factor> + <term>
<factor> ::= 2 | 4 | 6

(a) Outline the general steps in Recursive Descent Top Down Parsing to recognise (parse) expressions. *(2)*
(b) By parsing the sample expression `2+4*6`, obtain a parse tree and abstract syntax tree and discuss the problem(s) with the productions P. *(4)*
(c) Rewrite the productions to correct the error(s) in the productions. *(4)*