NUI Galway
OÉ Gaillimh

# ***Semester 1 Examinations 2009 / 2010***

| | |
|---|---|
| **Exam Code(s)** | 3IF1 |
| **Exam(s)** | 3rd B.Sc. (Information Technology) |
| **Module Code(s)** | CT331 |
| **Module(s)** | Programming Paradigms |
| Paper No. | I |
| External Examiner(s) | Prof. Michael O'Boyle |
| Internal Examiner(s) | Professor Gerard J. Lyons |
| | *Ms. Josephine Griffith |
| | *Dr. Jim Duggan |

**Instructions:**

Answer 3 questions. All questions will be marked equally.
Use a separate answer book for each section. At least one question must be answered from each section.

| | |
|---|---|
| **Duration** | 2hrs |
| **No. of Pages** | 5 |
| **Department(s)** | Information Technology |
| **Requirements** | None |

**OLLSCOIL NA hÉIREANN**
NATIONAL UNIVERSITY OF IRELAND, GALWAY

**SEMESTER I, WINTER 2009-2010 EXAMINATION**

**Third Year Examination in Information Technology**

*Programming Paradigms (CT331)*

Prof. Michael O'Boyle
Prof. Gerard J. Lyons
Ms. Josephine Griffith
Dr. Jim Duggan

**Time Allowed: 2 hours**

Answer THREE questions.
Use separate answer books.
At least ONE question must be answered from each section.

**SECTION A**

**Q. 1.** **(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)** Write PROLOG code to reverse the items (top level only) in a list, writing a tail recursive and non-tail recursive version of the code. *(8)*

Explain the steps taken for both versions. *(2)*

**Q.2.** **(i)** With the aid of an example, explain what is meant by "fully parenthesised prefix notation" with respect to the functional programming language SCHEME. *(4)*

**(ii)** 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 'a '(nice example))`
(b) `(list 'a '(nice example))`
(c) `(append '(a) '(nice example))`
(d) `(list '(a b) '(c d) '(e f))`
(e) `(append '(a b) '(c d) '(e f))`
(f) `(cons '(a b) '((c d e)))`

**(iii)** Control in SCHEME 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 finds the factorial (n!) of an integer number *n* where:

$n! = n \times n\text{-}1 \times n\text{-}2 \times \ldots \times 1$

For both functions, explain the approach taken and explain the difference between the tail recursive and non-tail recursive versions. *(10)*

**(iv)** Write a function in SCHEME which checks if two occurrences of a given list occur beside each other in a second list, returning `#t` or `#f` *(6)*
e.g.,

```
(two_conseq? '(a b) '( (x y) (a b) (a b)))
```
returns `#t`

```
(two_conseq? '(a b) '((a b)  (x y) (a b)))
```
returns `#f`

Explain the approach taken. *(4)*

**Q. 3.** **(i)** What is meant by a programming paradigm? *(4)*

Distinguish between the imperative and declarative programming paradigms, highlighting the main features of each paradigm. Use sample code from a language of each of the two paradigms to support your answer. *(6)*

**(ii)** With respect to program translation describe the purpose of the lexical analysis stage. *(2)*

Illustrate, with the aid of a diagram, a FSA which recognises strings of the form `c.(a|d).(a|d)*.r`, e.g., the FSA should recognise `car`, `cdr`, `cadr`, etc. *(8)*

**(iii)** With respect to program translation describe the purpose of the syntactic analysis stage. *(2)*

The given grammar describes a restricted set of algebraic expressions. By parsing the sample string:

`x + y × z`

obtain a parse tree and highlight the problems, if any, with the grammar. *(8)*

```
Grammar = {N, T, S, P}
N = {<goal>, <expr>, <term>, <factor>}
T = {x, y, z, +, ×}
S = <goal>

                          P=
<goal> ::= <expr>
<expr> ::= <term> | <term> × <expr>
<term> ::= <factor> | <factor> + <term>
<factor> ::= x | y | z
```

## SECTION B

**Q. 4.** **(i)** For a Student class, define a property that can set or retrieve the StudentNumber. Summarise the benefits of properties in object-oriented languages. (*6*)

**(ii)** Define the format of a delegate, and explain how it is used in C#. Highlight the difference between an event and a delegate. (*6*)

**(iii)** Using delegates, write a publish/subscribe program where subscribers are updated with the latest football scores. The publisher should keep a list of delegates, and provide subscribers with a mechanism to register. When a score occurs, all subscribers should be notified automatically. (*18*).

**Q. 5.** **(i)** Explain the difference, using examples, between a class constructor and an object constructor. (*6*)

**(ii)** Propose a solution that would ensure that only one object of a given type is created. Use the example of a **PrintSpooler**, which receives a request from an object that needs a printing service, and returns a reference to the single available **Printer** object. (*12*)

**(iii)** Expand on the solution in (ii), so that the PrinterSpooler can keep track of 10 printers, and it returns a reference to the first available printer. (*12*)

**Q. 6.** **(i)** Define an iterator, and summarise why it is a useful feature of C#. (*6*).

**(ii)** For a Student (ID, Name) object with a collection of Result (Subject Code, Grade) objects, use the default iterator to return each result object from the collection. (*15*)

**(iii)** Extend the solution in **(ii)** by developing custom iterators that implement the IEnumerator interface (e.g. public Object Current; public bool MoveNext(); public void Reset()). These should return the results in descending order. Assume that – if needed – a sorting object is available that will sort a list of results into descending order [i.e. there is no requirement to implement a sorting algorithm]. (*9*).