Semester 1 Examinations 2010 / 2011

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)	*Dr. Jim Duggan
	Mr. Alan Cunningham

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	6 (Including cover page)
Department(s)	Information Technology
Requirements	None

OLLSCOIL NA hÉIREANN
NATIONAL UNIVERSITY OF IRELAND, GALWAY

SEMESTER I, WINTER 2010-2011 EXAMINATION

Third Year Examination in Information Technology

Programming Paradigms (CT331)

Prof. Michael O'Boyle
Dr. Jim Duggan
Mr. Alan Cunningham

Time Allowed: 2 hours

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

SECTION A

Q1 **(i)** With respect to features of the functional programming paradigm, and with the with the aid of examples, describe what is meant by each of the following statements:

(a) Programs are constructed as a composition of functions. (4)
(b) Control is mainly achieved through recursion. (3)
(c) Theoretically, functions have no side effects. (3)

(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) Distinguish between the SCHEME primitives car and cdr by writing sequences of cars and cdrs to extract the symbol “and” from the following expressions: (4)

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

(iv) 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:

(a) (sum_it 3 ‘(1 2 a b)) returns (4 2 a b)
(b) (sum_it 3 ‘(a b 1 2)) returns (a b 1 2)
Explain the approach taken. (2)

Q2 **(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)

Q3 **(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)

(a) if keyword
(b) positive real numbers
(c) all strings that contain "acat" as a substring, with an alphabet of acgt

(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

SECTION B

Q4 **(i)** For an Account class, define a property that can set or retrieve the AccountNumber (as a String). 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 weather information. The publisher should keep a list of delegates, and provide subscribers with a mechanism to register their interest in a particular city. When the weather changes, all subscribers to that city should be notified automatically. A simple data structure (struct) should be used to pass information, and it should capture important weather-related information (*18*).

Q5 **(i)** Examine the following code segment, and determine what output will be printed from each statement. (*6*)

```
String a = “HelloWorld”;
String b = “HelloWorld”;
String c = “HELLOWORLD”;

Console.WriteLine(a == b);
Console.WriteLine(b==c);
```

(ii) Explain the use of regular expressions in C#, and show give a simple example of a function that determines whether a string is uppercase. (*6*)

(iii) Using a regular expression, write a function that validates a mobile phone number. The format of the number is as follows:

Mobile Phone Number = Start Character Symbol + Country Code + Hyphen + Prefix + Hyphen + Main Number
Start Character Symbol = +
Country Code = 1{digit}3 // between 1 and three digitas
Prefix = 2{digit}2 // three digits
Hyphen = -
Main Number = 7{digit}7
Digit = [0-9]

For example, a valid phone number would be +353-86-1111111. *(18)*

Q6 **(i)** Explain the difference between interface and implementation inheritance. (*8*)

(ii) Summarise the key ideas behind the adapter design pattern. Use a class diagram to show its structure, and a collaboration diagram to highight its behaviour. (*8*)

(iii) Use the adapter design pattern to create a Last in First Out (LIFO) container object, which is "wrapped" around C#'s ArrayList class. The interface for the LIFO list is:

```
public interface LIFO
{
        public void insert (Object o);
        public Object getFirstElement();
        public Object removeFirstElement();
        public int getListSize();
}
```

For the ArrayList, the following property and methods are relevant. Also, an index should be used to retrieve a value from an ArrayList. For example, the first element of an ArrayList x is x[0].

Properties:
Count – Returns the number of elements in the ArrayList

Methods
Add – Add an object to the end of the ArrayList *(14)*