NUI Galway
OÉ Gaillimh

Semester 1 Examinations 2012 / 2013

	3BCT1
Exam Code(s)	3IF1
	1EM1
Exam(s)	3rd B.Sc. Computer Science and Information Technology
	3rd B.Sc. (Information Technology)
	Erasmus
Module Code(s)	CT331
Module(s)	Programming Paradigms
Paper No.	I
External Examiner(s)	Prof. Michael O'Boyle
Internal Examiner(s)	Prof. Gerard Lyons
	Dr. Michael Madden
	*Ms. Josephine Griffith
	*Dr. Jim Duggan

Instructions:

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

Duration	2hrs
No. of Pages	7 (Including cover page)
Department(s)	Information Technology
Requirements	None

PTO

OLLSCOIL NA hÉIREANN
NATIONAL UNIVERSITY OF IRELAND, GALWAY

SEMESTER I, WINTER 2012-2011 EXAMINATION

Third Year Examination in Computer Science and Information Technology

Programming Paradigms (CT331)

Prof. Michael O'Boyle
Prof. Gerard Lyons
Dr. Michael Madden
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.

PTO

SECTION A

Q1 **(a)** Distinguish between the SCHEME primitives `car` and `cdr` by writing sequences of `car` and `cdr` to extract the number "2" from the following lists: *(3)*

(i) `(+ two and 2)`
(ii) `(and then (2 more) please)`
(iii) `(and (twice (2 3 4) plus two (too)))`

(b) Distinguish between the SCHEME primitives `cons`, `list` and `append` by showing, and explaining, the output from each of the following SCHEME expressions: *(4)*

(i) `(cons '(a b) '(c d))`
(ii) `(list 'a 'b '(c d))`
(iii) `(append '(a b) '(c d) '(e f))`
(iv) `(list (list 'a) (cons 'b (append '(c) '(d) '(e))))`

(c) Write a recursive function in SCHEME which performs a linear search of a list of numbers when passed a list and an item. You can assume that the data in the list is in ascending sorted order. The function should return #f or #t. The function should stop searching when the item is found or when some number greater than the item is found or when the end of the list is reached without finding the item. For example, if the function is called linsearch:
`(linsearch '10 '(2 4 6 8))` returns `#f`
`(linsearch '4 '(2 4 6 8))` returns `#t`

Explain the approach taken, highlighting the base cases and the reduction stage. *(8)*

(d) Write both a non-tail recursive and a tail-recursive function in SCHEME which counts the number of occurrences of an item in a list, returning this number. You can assume valid input and that there are no sublists. For example, if the function is named countoccurs:
`(countoccurs 'a '(a b b a))` returns `2`

Explain the approach taken highlighting the base case and reduction stages for each function. *(10)*

PTO

Q2 Given the following set of relations in PROLOG which specify some courses taken by some students:

```
takes(john, ct101).
takes(kate, ct101).
takes(john, ct103).
takes(ciara, ct229).
takes(kate, ma101).
takes(shane, ct101).
takes(ciara, ct230).
```

(a) With the aid of the above relations in the PROLOG database answer the following questions:

(i) Show how unification occurs in PROLOG, and the resulting output, using the following query *(2)*:

```
?- takes(Y, ct101).
```

(ii) Show how unification occurs in PROLOG, and the resulting output, using the following query *(2)*:

```
?- takes(ciara, X).
```

(iii) Write the additional line(s) of code needed in the database to represent a student named `ann` who takes `ct101` and `ma190`. *(1)*

(iv) Add a rule to the database which states that two people are classmates if they take the same class. *(2)*

(b) Describe, with the aid of examples, the list data structure in PROLOG, outlining its representation and syntax. *(2)*

(c) Write code in PROLOG to merge two lists, explaining the steps taken in developing the code. *(6)*

(d) 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. Explain the steps taken for both versions. *(10)*

PTO

Q3 **(a)** Explain what is meant by a Finite State Automaton (FSA) by drawing an FSA to recognise strings that have no consecutive 1s with an alphabet of `{1,0}`. For example

(i) `01101` is not a valid string
(ii) `1001` is a valid string
(iii) `1` is a valid string
(iv) `0` is a valid string

Illustrate how your FSA works given the above sample strings. *(8)*

(b) Given the following grammar :

G = {N, T, S, P}
T = {a, b, c}
N = {X, Y}
S = Y
P =
<Y> ::= a<X>
<X> ::= b<X>
<X> ::= c

(i) Identify the terminals, non-terminals, productions and starting production in the grammar. *(1)*
(ii) What type of strings does the grammar generate and recognise? *(2)*
(iii) Draw the FSA that can be used to recognise the strings of the form identified in part (ii). *(4)*

(c) The following grammar describes a restricted set of assignment expressions using the operators of addition (+) and multiplication (×).

G = {N, T, S, P}
T = { =, +, ×, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b}
N = {<R>, <E>, <id>, <num> }
S = R
P =

<R> ::= <id> = <E>
<E> ::= <id> | <num>
<E> ::= <E> × <E> | <E> + <E>
<num> ::= 0|1|2|3|4|5|6|7|8|9
<id> ::= a|b

Given the following string:

a = a + b × 3

(i) Show how two different parse trees are possible when parsing this string using the above grammar. *(5)*
(ii) Modify the grammar to correct the problem and, using the same sample string, show how only one parse tree results from parsing the string with the new grammar. *(5)*

PTO

SECTION B

Q4 (a) Describe the key idea of a delegate, and show how it is defined in C#.

(5)

(b) Define a delegate that can be used to send information on a sports event to subscribers. The information is specified as a *value type* with the following fields: Event ID, Time, Team Description, Scorer.

(8)

(c) Based on (b), implement a publisher/subscriber solution whereby subscribers can join/leave a publisher service, and are notified automatically once an event has happened. Make use of a List structure to store all subscribers. A list of all events should also be stored by the publisher, on a last-in first-out basis.

(12)

Q5 (a) Describe the purpose of the *Command Pattern*, show its overall structure, and discuss its advantages. As part of the answer, include a class diagram and a collaboration diagram.

(10)

(b) Using the Command Pattern, implement a solution in C# for the following stock control transactions.

- Create a class called StockListener, which creates an **Invoker**, and calls a **Client** with information when a stock change (increase or decrease) has happened. This information includes the StockId(String) and the amount it has changed by (integer). The **Client** will return an ICommand object (supertype), which the **StockListener** then passes on to the **Invoker**. [The Client also creates a **Receiver** object, and passes this to the concrete ICommand object].

- The **StockListener** has two methods called *ProcessBatch()* and *Rollback()*, and these methods call the Invoker to carry out these operations. The Invoker should arrange for the logic of both batch updates and complete rollback of all transactions.

- For the **Invoker**, use a List class to store information on all ICommand objects. All commands should be able to (1) Execute and (2) Rollback.

- Two concrete commands objects should be created, one to increment the stock, the other to decrement the stock. All stock changes are controlled with the **Receiver** object.

(15)

PTO

Q6 (a) Explain the following RegEx sequences:

- [abc]
- [^abc]
- [a-z]
- \d
- \D

(8)

(b) Write a function, using regular expressions, that checks whether or not a string contains only lowercase characters (i.e. if all are lowercase, the function returns true, otherwise false is returned).

(5)

(c) Summarise the main benefits of overriding operators in C#.

Create a class that contains a one-dimensional array of integers. Override both the + and - operators so that:

- An array can be added to the current array
- An array can be subtracted from the current array

Also, a complete list of all previous states of the array should be stored.

(12)