Introduction to Computer Science II (CSI 1101)
FINAL EXAMINATION

Instructor: Marcel Turcotte

April 2002, duration: 3 hours

Identification

Student number: _____ section: _____

Student name: _____ designated seat: _____

Instructions

1. This is a closed book examination.
2. No calculators or other aids are permitted.
3. Write your answers in the space provided. Use the backs of pages if necessary. You may **not** hand in additional pages.
4. Write comments and assumptions to get partial marks.
5. Beware, poor hand writing can affect grades.
6. Do not remove the staple holding the examination pages together.

Marking scheme

Question	Maximum	Actual
1	6	
2	3	
3	3	
4	14	
5	14	
6	12	
7	6	
8	10	
9	12	
10	6	
11	10	
12	4	
Total	**100**	

Question 1 (6 marks)

Given the following binary sequence:

$$(1101)_2$$

What decimal value does it represent for each of the following 3 interpretations:

a) as an unsigned number (positional system).

b) as a ONE's complement number, encoded with 4 bits.

c) as a TWO's complement number, encoded with 4 bits.

Question 2 (3 marks)

Convert the following unsigned decimal number to binary.

$$(67.5625)_{10} = (\qquad\qquad\qquad\qquad\qquad\qquad)_2$$

Question 3 (3 marks)

Showing all the steps in your computation, multiply the following unsigned binary numbers. All intermediary calculations must be carried out in binary.

$$\begin{array}{cccccccc} & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ \times & & & & 1 & 1 & 0 & 1 \\ \hline \end{array}$$

Question 4 (14 marks)

In this question there is:

- an interface called **Queue**:

```
public interface Queue {
    public boolean isEmpty(); // returns true if the queue is empty
    public void enqueue(Object object); // adds an object at the rear of the queue
    public Object dequeue() // removes and returns the front object
                throws java.util.NoSuchElementException;
}
```

- a class called **Q** that implements the interface **Queue**.

In the class **Q**, write an instance method to compare two **Queue** instances. To write this method you can only use the (3) methods declared in the interface **Queue**. Two queues are equal if they contain the same elements, in the same order. The two queues must not be changed by this method.

```
public class Q implements Queue {
    // ...
    public boolean equals(Object object) {
        // answer:
```

Question 4 (continued)

```
    } // end of method equals
} // end of class Q
```

Question 5 (14 marks)

Given the linked list implementation below, write a **recursive** instance method that compares two lists. Two lists are equal if they contain the same elements, in the same order — two empty lists are considered equal. Complete the method **equals(Node p, Node q)**.

```
public class LinkedList {
    private class Node {
        public Object value;
        public Node next;
        public Node(Object value, Node next) {
            this.value = value;
            this.next = next;
        }
    }
    private Node head;

    public boolean equals(Object object) {
        if (! (object instanceof List))
            return false;

        LinkedList other = (LinkedList) object;
        return equals(head, other.head);
    }

    private boolean equals(Node p, Node q) {
        // answer:
```

Question 5 (continued)

```
    } // end of the method equals
}
```

Question 6 (12 marks)

Given the linked list implementation below, write an instance method that merges two **ordered** lists. For example, if the method **merge** is called for the following list, **l1**:

```
l1 -> a -> d -> e -> null
```

with the list **l2** as an argument:

```
l2 -> b -> d -> f -> h -> null
```

i.e. the following call is made:

```
l1.merge(l2);
```

then the list **l1** is changed to:

```
l1 -> a -> b -> d -> d -> e -> f -> h -> null
```

but **l2** remains unchanged:

```
l2 -> b -> d -> f -> h -> null
```

Elements of the lists are compared using the method **public int compareTo(Object)** from the interface **Comparable**.

```
public class DoublyLinkedList implements OrderedList {
    private class Node {
        public Comparable value;
        public Node previous;
        public Node next;

        public Node(Comparable value, Node previous, Node next) {
            this.value = value;
            this.previous = previous;
            this.next = next;
        }
    }

    private Node head;
    private Node tail;
```

Question 6 (continued)

```
    public void merge(DoublyLinkedList l2) {
        // answer:
```

```
    } // end of the method merge
} // end of class DoublyLinkedList
```

Question 7 (6 marks)

Here is a simple stack-based language to draw lines on a (10×10) two dimensional space . The operations of the language are **C** and **L**:

C: duplicates the top of the stack. For example, if the stack contains:

```
8 <- top of the stack
5
3
```

the operation **C** changes the stack to be:

```
8 <- top of the stack
8
5
3
```

L: draws a line from (x_1, y_1) to (x_2, y_2). The coordinates are read from the stack. Once the line has been drawn, the coordinates (x_2, y_2) are put back onto the stack. For example, given the following stack,

```
8 <- top of the stack
2
3
1
5
3
```

the operation **L** draws a line from $(1, 3)$ to $(2, 8)$. After the execution of the operation, the stack contains the following elements:

```
8 <- top of the stack
2
5
3
```

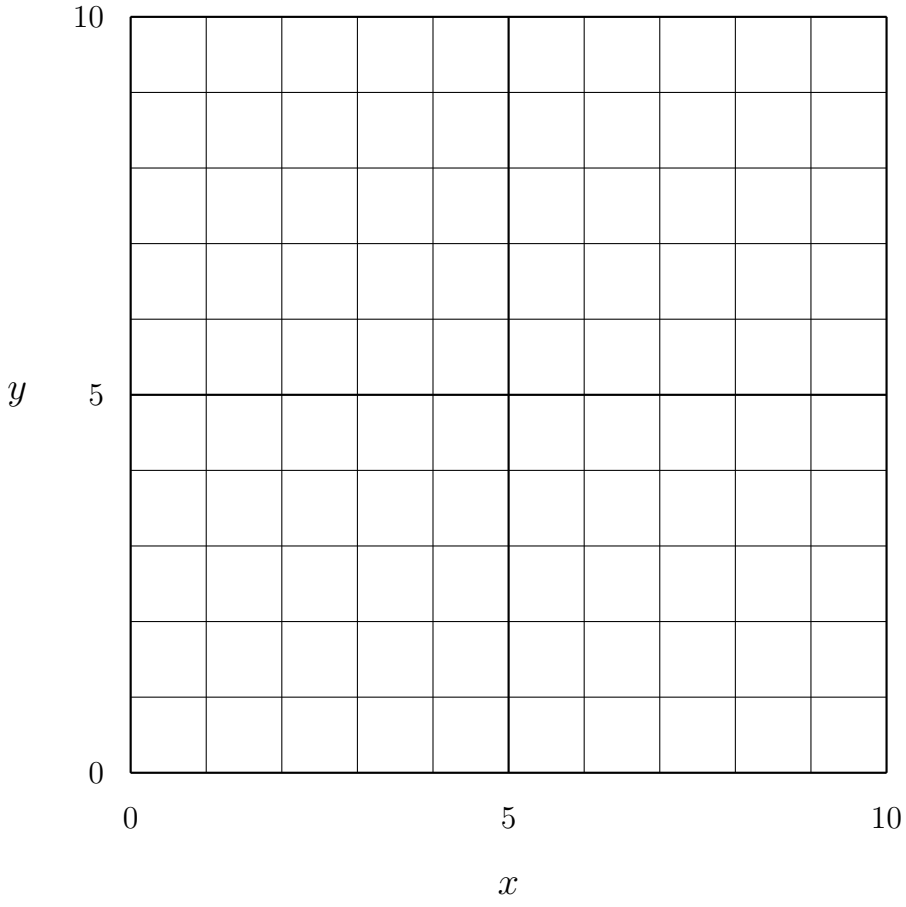
A class that implements these operations (an interpreter for this language) can be found on the next page. It uses a class called **Stack** that implements the following methods:

int peek(); Looks at the element at the top of this stack without removing it from the stack.

int pop(); Removes and returns the element at the top of this stack.

int push(int element); Pushes an element onto the top of this stack.

```
class S {

    private Stack s;

    public S() {
        s = new Stack();
    }

    public void execute(String program) {
        for (int i=0; i<program.length(); i++) {
            char c = program.charAt(i);

            if (Character.isDigit(c))
                s.push(Character.digit(c,10));
            else
                switch (c) {
                case 'L':
                    L();
                    break;
                case 'C':
                    C();
                    break;
                }
        }
    }

    private void C() {
        s.push(s.peek());
    }

    private void L() {
        int y2 = s.pop();
        int x2 = s.pop();
        int y1 = s.pop();
        int x1 = s.pop();

        line(x1, y1, x2, y2);

        s.push(x2);
        s.push(y2);
    }

    public static void line(int x1, int y1, int x2, int y2) {
        // draws a line from (x1,y1) to (x2,y2)
    }
}
```



Onto the above grid, draw the lines produced by the execution of the following sequence of operations:

```
1 C 7 4 L 2 7 L 5 C L
```

i.e., the result of executing the following statements:

```
S machine = new S();
machine.execute("1C74L27L5CL");
```

In the space below, show the content of the stack after each call to L.

Question 7 (continued)

Question 8 (10 marks)

In this question there is:

- an interface called **Iterator**,

```
public interface Iterator {
    public boolean hasNext(); // Returns true if the iteration has more elements,
                              // i.e., returns true if next would return an element.
    public Object next();     // Returns the next element in the iteration.
}
```

- a class called **CircularQueue**,
  - that implements the methods **enqueue(Object)** and **serve()** (dequeue).
  - the elements of the queue are **Objects** that are stored in a circular array.
  - the empty queue is represented with **front = -1** and **rear = -1**.

The class **CircularQueue** contains a partial implementation of the interface **Iterator**. Fill in the (5) missing elements in the class `CircularQueue` on the next page.

The execution of the following statements:

```
CircularQueue q = new CircularQueue();

for (int i=0; i<4; i++)
    q.enqueue("elem-" + i);

Iterator j = q.iterator();
while (j.hasNext())
    System.out.print(" " + j.next());
```

should display:

```
elem-0 elem-1 elem-2 elem-3
```

```
import java.util.NoSuchElementException;

public class CircularQueue {

    private static final int MAX_QUEUE_SIZE = 4;

    private Object[] q;
    private int front, rear;

    public CircularQueue () {
        q = new Object[MAX_QUEUE_SIZE];
        front = -1;
        rear  = -1;
    }

    public void enqueue (Object o) {
        if (front == -1) {
            front = 0;
        }
        rear = (rear + 1) % MAX_QUEUE_SIZE;
        q[rear] = o;
    }

    public Object serve () {
        if (front == -1)
            throw new NoSuchElementException();

        Object returnValue = q[front];
        if (front == rear) {
            front = -1;
            rear  = -1;
        } else {
            front = (front + 1) % MAX_QUEUE_SIZE;
        }
        return returnValue;
    }

    // continues over the next column

    public Iterator iterator() {
        return new _____________________________________________ ;
    }

    private class CircularQueueIterator _________________________ {
        public int current;

        public CircularQueueIterator() {
            current = -1;
        }

        public Object next() {

            if (current == rear)
                throw new NoSuchElementException();

            if (current == -1) {
                current = front;
            } else
                current = _____________________________________ ;

            return _____________________________________________ ;
        }

        public boolean hasNext() {
            return (current == -1 && ___________________________ ) ||
                    current != rear;
        }
    }  // end of class CircularQueueIterator
} end of class CircularQueue
```

Question 9 (12 marks)

In this question there is:

- an interface called **Purchase**:

```
public interface Purchase {
    public double getPrice();
    public boolean isTaxable();
}
```

- a class called **Work**, to represent works of art:

```
public class Work {

    private String title;
    private String author;

    public Work(String author) {
        this.author = author;
    }

    public String getAuthor() {
        return author;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getTitle() {
        return title;
    }
}
```

Write a class called **Album** such that:

- **Album** is a subclass of **Work**.
- **Album** implements the interface **Purchase**.
- An **Album** has a list of songs.
- Accordingly, the class has a method to add a new song to the list of songs and a method that allows to retrieve the ith song (song at position i of the list of song).
- If the title of the **Album** has not been specified then the title corresponds to the title of the first song.
- The price of an album is calculated from the price per song, which is a parameter passed to the constructor of the class. For example, the price per song could be $2.10, with 9 songs in the album, the total price would be $18.90.

Hint: *use the class* ***LinkedList*** *or* ***ArrayList*** *to store the list of songs.*

Question 9 (continued)

Question 10 (6 marks)

Convert the following Java program into the Assembly language for the toy computer, TC-1101, discussed in class. The mnemonics available are: CLA, INC, HLT, LDA, STA, ADD, SUB, DSP, JMP, JZ and JN.

You do not have to give the Assembly language instructions that reserve storage for the variables — you may assume this has already been done.

```
while ( p > r ) {
    r = r + 2;
}
q = 1;
```

Question 11 (10 marks)

A class called **IntelligentGadget** uses an instance of the class **LightBulb** to indicate its status (on or off).

A **LightBulb** has a certain life time, which is modeled with an instance variable called **life**, which is initialized to a random value (0..1000) when a new light bulb is created.

(a) Create a new exception type, subclass of **Exception**, and called **WornOutLightBulbException**.

(b) Modify the instance method **on()**, of the class **LightBulb**, so that if the number of times the method **on()** has been called exceeds its life time then an exception of type **WornOutLightBulbException** will be thrown.

(c) Modify the instance method **powerOn()** so that it catches any exception of type **WornOutLightBulbException** and create a new **LightBulb** instance to replace the one that has just worn out.

The class definitions can be found on the next page. Write your answers in the space below.

```
public class IntelligentGadget {

    private LightBulb indicator;

    public IntelligentGadget {
        indicator = new LightBulb();
    }

    public void powerOn() {
        indicator.on();
    }

    public void powerOff() {
        indicator.off();
    }
}

public class LightBulb () {

    private boolean on;
    private int count;
    private int life;

    public LightBulb() {
        on = false;
        count = 0;
        life = (int) (Math.random() * 1000.0);
    }

    public void on() {
        count = count + 1;
        on = true;
    }

    public void off() {
        on = false;
    }
}
```

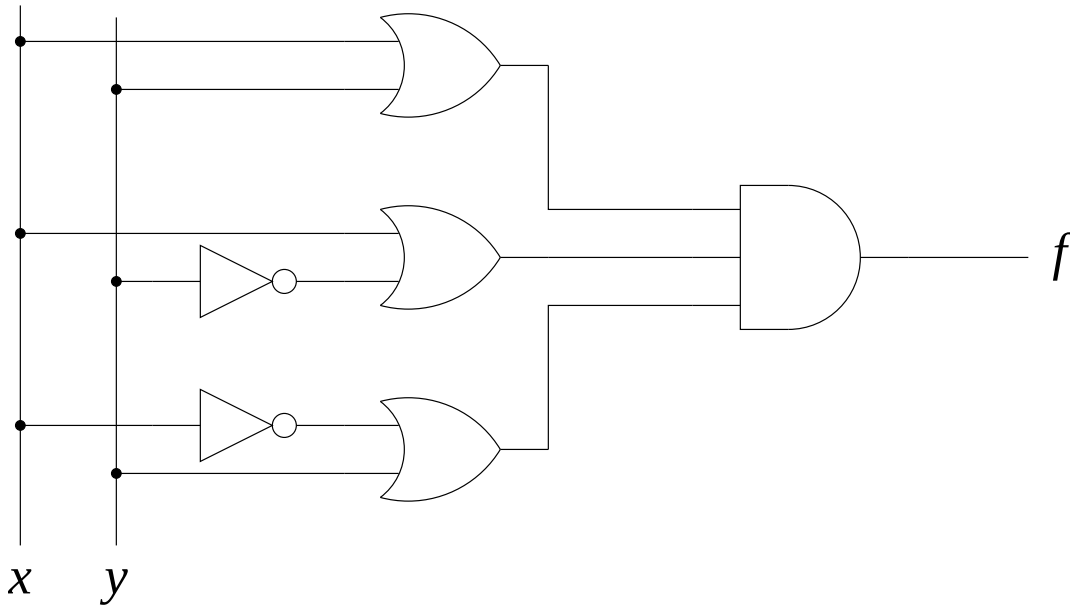
Question 12 (4 marks)

(a) Give the Canonical Product of Sum (CPOS) for the following truth table. Do not simplify your answer.

A	B	C	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

$f =$

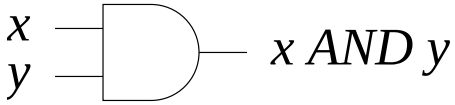
(b) Give the Boolean expression that corresponds to the following logic circuit:

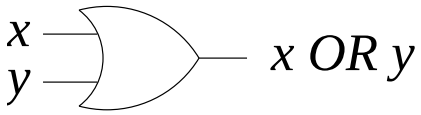


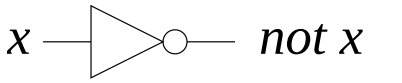
$f =$

Appendix

Circuit symbols to be used for question 12 (b):







(blank space)