Université d'Ottawa
Faculté de génie

École d'ingénierie et de
technologie de l'information

uOttawa

L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Information
Technology and Engineering

Introduction to Computer Science II (ITI 1121)
Midterm Examination

Instructor: Marcel Turcotte

February 2007, duration: 2 hours

Identification

Student name: _____

Student number: _____ Signature: _____

Instructions

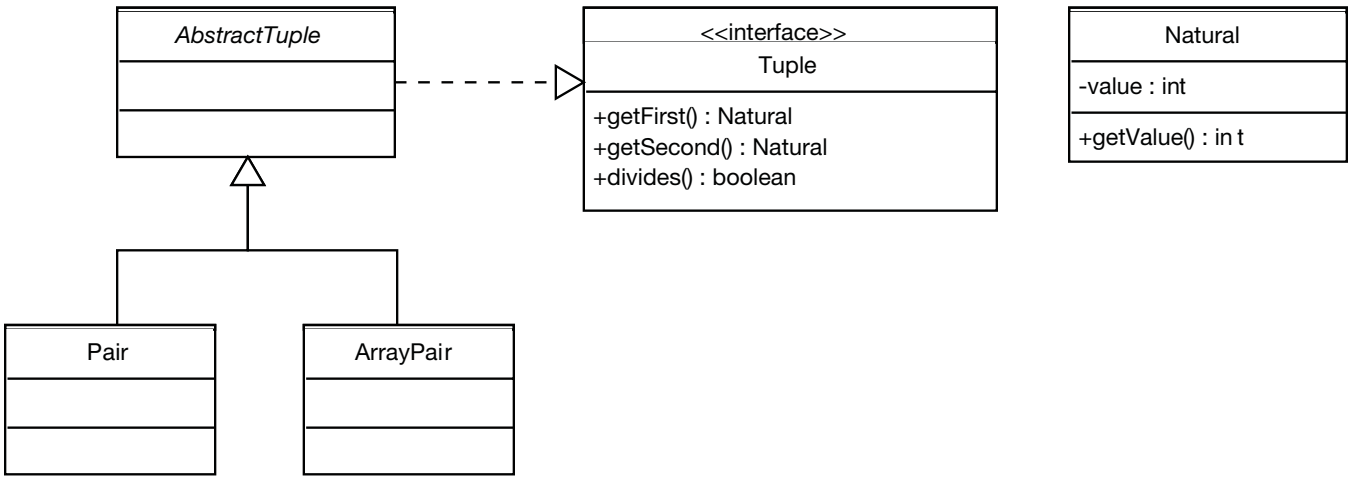
1. This is a closed book examination;
2. No calculators or other aids are permitted;
3. Write comments and assumptions to get partial marks;
4. Beware, poor hand writing can affect grades;
5. Do not remove the staple holding the examination pages together;
6. Write your answers in the space provided.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

Marking scheme

Question	Maximum	Result
1	36	
2	15	
3	15	
4	12	
5	22	
Total	**100**	

Question 1: (36 marks)

A tuple holds two **Natural** numbers (objects of the class **Natural**). All the tuples have a method **getFirst**, as well as a method **getSecond**, returning a reference to the first, and second, number of the tuple, respectively. A tuple has a method **divides** that returns **true** if the first element is a factor of the second element, and **false** otherwise.



For this question there is an interface named **Tuple**, an abstract class named **AbstractTuple**, and two concrete implementations, called **Pair** and **ArrayPair**. Their complete description can be found on the next pages. The implementation of the class **Natural** is given on page 18. The execution of the statements below produces the following output: `7 is a factor of 42`.

```
Natural n1, n2, n3;

n1 = new Natural( 7 );
n2 = new Natural( 17 );
n3 = new Natural( 42 );

Tuple t1, t2;

t1 = new Pair( n1, n2 );
t2 = new ArrayPair( n1, n3 );

if ( t1.divides() ) {
    System.out.println( t1.getFirst() + " is a factor of " + t1.getSecond() );
}

if ( t2.divides() ) {
    System.out.println( t2.getFirst() + " is a factor of " + t2.getSecond() );
}
```

Make sure to include the constructors and access methods that are necessary for the execution of the above statements.

A. Implement the interface **Tuple**. The interface declares 3 methods. There are two access methods, named **getFirst** and **getSecond**, and both return a reference to a **Natural** object. Finally, the interface also declares a method called **divides** that returns a **boolean** value. (8 marks)

B. Write the abstract class named **AbstractTuple**, which implements the interface **Tuple**. The class **AbstractTuple** has a concrete implementation of the method **divides**, which returns **true** if the first element of the tuple is a divisor of the second element, and **false** otherwise. A divisor of n is a number that divides n without leaving a remainder. (8 marks)

C. Write a concrete implementation of the class **AbstractTuple** called **Pair**. The implementation has two instance variables that are references to the first and second element of this tuple. Add all the necessary constructors and access methods. (10 marks)

D. Write a concrete implementation of the class **AbstractTuple** called **ArrayPair**. The implementation uses an array of size 2 to store references to the first and second element of this tuple. Add all the necessary constructors and access methods. (10 marks)

Question 2: (15 marks)

Complete the implementation of the static method **boolean isPalindrome(CharReader r)**. Let's define a **palindrome** as a word or a phrase that reads the same forward and backward if the punctuation symbols and spaces are ignored. Examples of palindromes include:

- i prefer pi
- never odd or even
- was it a cat i saw

For this question, the data is read using a **CharReader** object. Therefore, you don't know the actual size of the input and it is impossible to read it again, once read. Follow all the directives.

- **boolean isPalindrome(CharReader r)**. Write a stack-based algorithm that returns **true** if the whole word or phrase specified by the reader is a palindrome according to the above definition, and **false** otherwise;
- The parameter of the method is a **CharReader**. A **CharReader** has **exactly** two instance methods (and no public variables). Once the data has been read it is impossible to read it again.
  - **boolean hasMoreChars()**; returns **true** if the reader has more characters to return, that is if a call to **char nextChar()** would succeed, and **false** otherwise;
  - **char nextChar()**; returns the next character of the input.
- You cannot use arrays or strings to store the characters that are read from the input, you have to use implementations of a stack;
- Assume the existence of a class **StackImpl** that implements the interface **Stack**. For this question, a **Stack** stores characters.

```
public interface Stack {
    public abstract boolean isEmpty();
    public abstract char peek();
    public abstract char pop();
    public abstract void push( char element );
}
```

- **StackImpl** can store an arbitrarily large number of characters;
- **Character.isLetter(char ch)** determines if the specified character, **ch**, is a letter.

```
public static boolean isPalindrome( CharReader reader ) {




}
```

Question 3: (15 marks)

The class **DynamicArrayStack** below uses the technique seen in class, as well as in the assignment 3, to increase or decrease, its physical size according to the needs of the application.

- **DynamicArrayStack** uses an array to store the elements of this stack;
- The interface **Stack** and its implementation, **DynamicArrayStack**, have a formal parameter type (in other words, the implementation uses the concept of generics types, introduced in Java 1.5);
- The initial capacity of this array is given by the first parameter of the constructor;
- The physical size of the array is increased by a fixed amount (**increment**) when the method **void push(E elem)** is called and the array is full;
- The physical size of the array is decreased by a fixed amount (**increment**) during a call to the method **E pop()** if the number of free cells becomes **increment** or more;
- The **increment** is given by the second parameter of the constructor;
- The instance variable **top** designates the top element (i.e. the cell where the last element was inserted, or -1 if the stack is empty).

A. Correct at least 5 mistakes (compile-time or runtime errors) in the partial implementation. (5 marks)

B. Complete the partial implementation of the class **DynamicArrayStack** given the above information. (10 marks)

```
public class DynamicArrayStack<E> implements Stack<E> {

    // Instance variables
    private static E[] elems;    // Stores the elements of this stack
    private static int top = -1; // Designates the top element
    private final int capacity;  // Memorizes the initial capacity
    private final int increment; // Used to increase/decrease the size

    public DynamicArrayStack( int capacity, int increment ) {
        E[] elems = (E[]) new Object[ capacity ];
        this.capacity = capacity;
        this.increment = increment;
    }

    // Returns true if this stack is empty;
    public boolean isEmpty() {
        return top == 0;
    }

    // Continues on the next page ...
```

DynamicArrayStack (continued)

```
public void push( E element ) {

    if ( ___________________________ ) {
        increaseSize();
    }

    elems[ top ] = element;
    top++;
}

private void increaseSize() {

    E[] newElems;
    int newSize;

    newSize= elems.length + increment;

    newElems = new E[ newSize ];

    for ( int i=0; i<elems.length; i++ ) {
        newElems[ i ] = elems[ i ];
    }

    ______________________________;
}

// Continues on the next page ...
```

DynamicArrayStack (continued)

```
    public E peek() {

        return ______________________________;
    }

    // Complete the implementation of pop()

    public E pop() {


        E saved;
        saved = elems[ top ];


        elems[ top ] = ________________;


        top--;


        return saved;
    }


    private void decreaseSize() {

        E[] newElems;
        int newSize;

        newSize = elems.length - increment;

        if ( newSize < capacity ) {
            newSize = capacity;
        }

        ________________________________________;

        for ( int i=0; i<=top; i++ ) {
            newElems[ i ] = elems[ i ];
        }

        elems = newElems;
    }

} // End of DynamicArrayStack
```

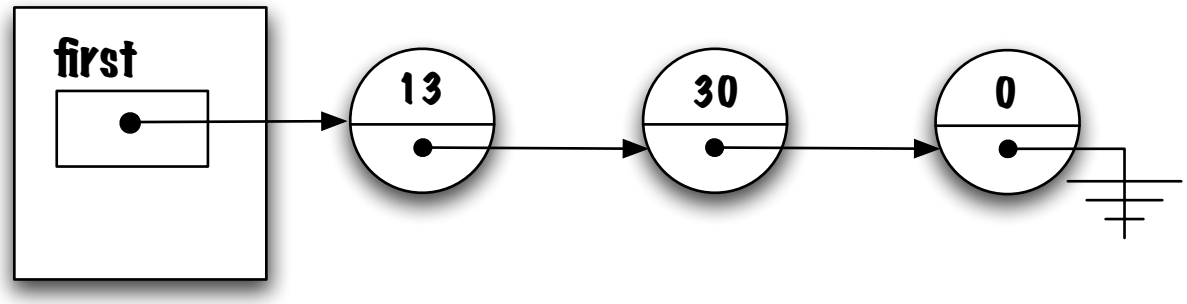
DynamicArrayStack (continued)

Complete the implementation of the method **main**. It declares a stack of Integer objects, creates a new stack of Integer objects, pushes 20 elements onto the stack, removes and prints those elements.

```
public class Test {

    public static void main( String[] args ) {

        // Declare a reference to a stak of Integer objects

        _____________ s;

        // Create an instance of DynamicStack to store Integer objects

        s = ______________________________________;

        for ( int i=0; i<20; i++ ) {
            s.push( new Integer( i ) );
        }

        while ( ! s.isEmpty() ) {
            // Declares an Integer

            _______________ elem;


            // Removes an element from the stack


            elem = _____________;

            System.out.println( elem );
        }
    }

} // End of Test
```

Question 4: (12 marks)

Complete the implementation of the class **Time** below. Linked elements are used to store the hours, minutes and seconds of this **Time** object. Specifically, the class **Time** has a single instance variable, called **first**. The variable **first** is a reference to an **Elem** object, which is used to store the hours of this **Time** value. The variable **next** of that element designates an **Elem** object, which is used to store the minutes. The variable next of that **Elem** object designates the last element, which is used to store the seconds of this **Time** value. For instance, the creation of this object: **new Time(13, 30, 0)**, has the following associated memory diagram.



Here, the values that are stored in the **Elem** objects are of type **int** (a primitive type).

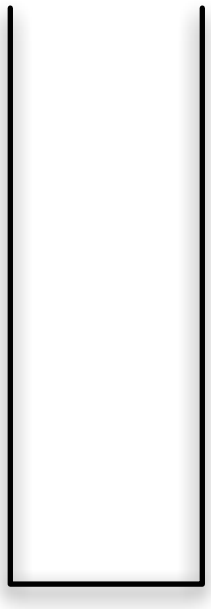
```
public class Time {
    private static class Elem {
        private int value;
        private Elem next;
        private Elem( int value, Elem next ) {
            this.value = value;
            this.next = next;
        }
    }
    private Elem first;

    public Time( int hours, int minutes, int seconds ) {









    }
```

```
    public int getHours() {









    }

    public int getMinutes() {









    }

    public int getSeconds() {









    }
} // End of Time ;-)
```

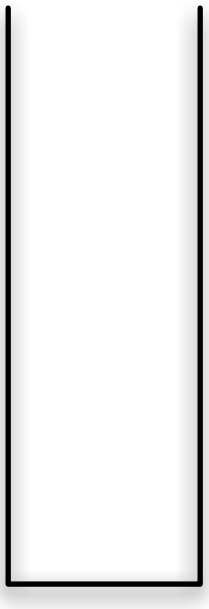
Question 5: (22 marks)

A. Use the stack-based algorithm seen in class to evaluate the following postfix (RPN) expression and show the content of the stack immediately before and after processing each operator. The notation $-_x$ is used to denote the first, second and third subtraction of the expression. (8 marks)

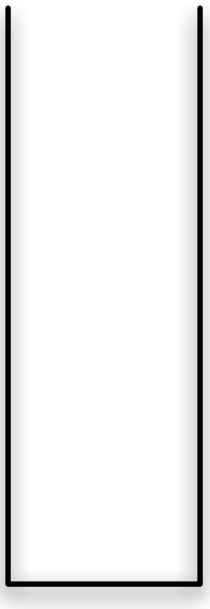
$$8 \quad 2 \quad -_a \quad 10 \quad 8 \quad 1 \quad -_b \quad -_c \quad /$$



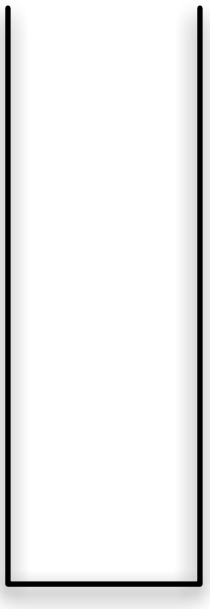
Before $-_a$



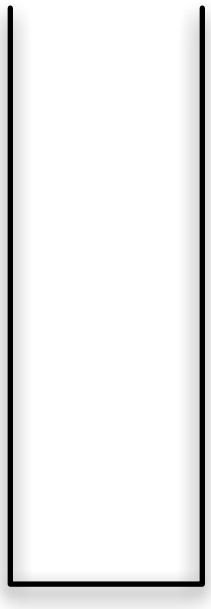
After $-_a$



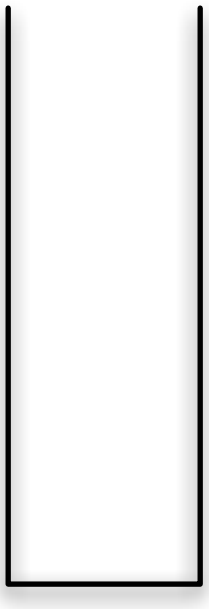
Before $-_b$



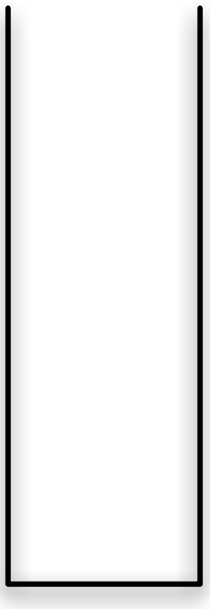
After $-_b$



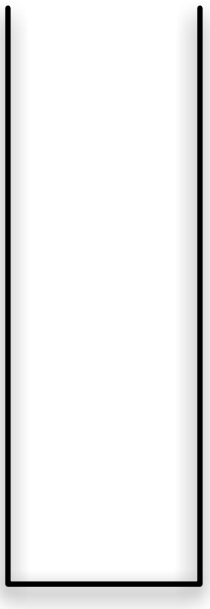
Before $-_c$



After $-_c$



Before /



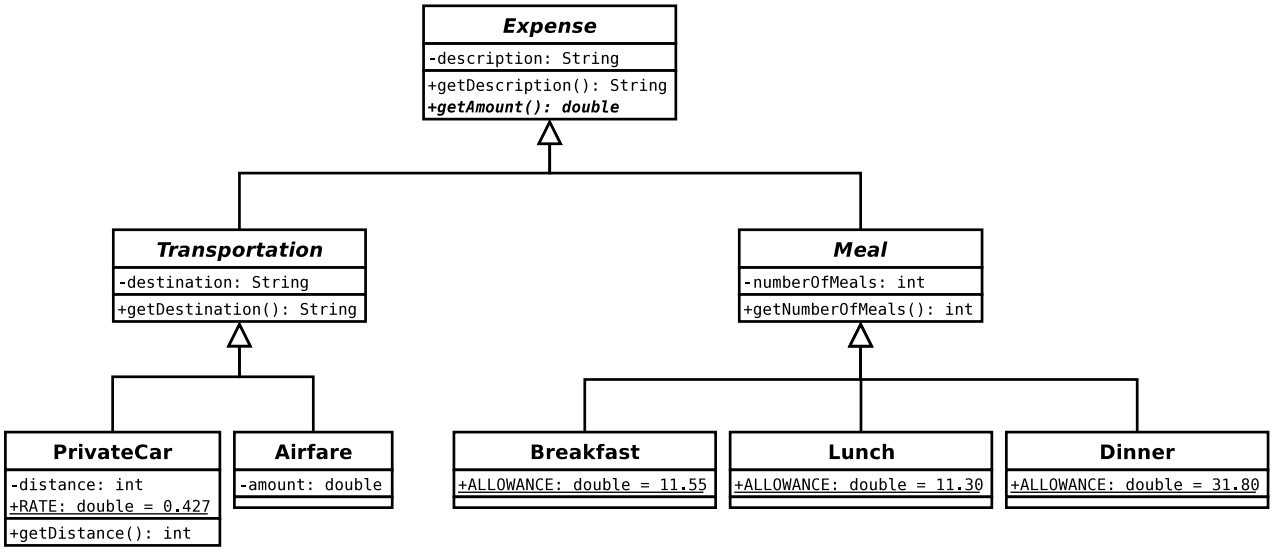
After /

B. Write the following infix expression in postfix (RPN) notation. (4 marks)

$$15 - 3 * 5 + 8/2$$

Solution:

C. The following class hierarchy is part of a software system for tracking professional expenses.



Specification:

- All expenses have a description (a character string);
- **Transportation** and **Meal** are sub-classes of **Expense**;
- **Expense**, **Transportation** and **Meal** are abstract classes;
- **PrivateCar** and **Airfare** are sub-classes of the class **Transportation**; **Breakfast**, **Lunch** and **Dinner** are sub-classes of the class **Meal**;
- All the transportation expenses have a destination (a character string);
- The class **Transportation** implements the interface **Taxable** (not shown in the above diagram);
- The interface **Taxable** declares the method **double getTax()**;
- A transportation expense using a private car has a distance (of type int);
- A transportation expense by air has a fixed amount (of type double) specified when a new transportation expense is created;
- All the meal expenses have an attribute which represents the number of meals;
- All the expenses have a method to calculate the amount represented by this expense:
  - The amount for a transportation expense using a private car is a fixed rate times the distance traveled;
  - The amount for a transportation expense by air is a fixed amount (specified when a new transportation expense is created);
  - The amount for a meal expense is the number of meals times a fixed rate. The rate depends on the kind of meal: Breakfast, Lunch or Dinner;

Which statements are valid? (10 marks)

(a) `Transportation t = new Transportation( "ISMB 2007", "San Fransisco" );`

(b) `Expense e = new Transportation( "ISMB 2007", "San Fransisco" );`

(c) `Taxable t = new Transportation( "ISMB 2007", "San Fransisco" );`

(d)
```
Expense e = new Airfare( "ISMB 2007", "San Fransisco", 789.0 );
double a = e.getAmount();
```

(e) `Taxable t = new Airfare( "ISMB 2007", "San Fransisco", 789.0 );`

(f)
```
Taxable t = new Airfare( "ISMB 2007", "San Fransisco", 789.0 );
double x = t.getTax();
```

(g)
```
Expense e = new Airfare( "ISMB 2007", "San Fransisco", 789.0 );
double x = e.getTax();
```

(h) **Taxable t = new Taxable(); double x = t.getTax();**

(i)
```
Expense e = new Airfare( "ISMB 2007", "San Fransisco", 789.0 );
String s = e.getDestination();
```

(j)
```
Expense t = new Airfare( "ISMB 2007", "San Fransisco", 789.0 );
double x = t.getAmount();
```

A Natural

```
/* The Natural class wraps a value of the primitive type int in an
 * object. Furthermore, the value is greater than or equals to
 * zero. An object of type Natural contains a single field whose type
 * is int.
 */

public class Natural {

    private int value; // instance variable

    /* Constructs a newly allocated Natural object that represents
     * the specified int value. Throws IllegalArgumentException if the
     * specified value is negative.
     */

    public Natural( int value ) {
        if ( value < 0 ) {
            throw new IllegalArgumentException( "less than zero" );
        }
        this.value = value;
    }

    /* Returns the value of this Natural as an int.
     */

    public int getValue() {
        return value;
    }

    /* Returns a String representation of this Natural.
     */

    public String toString() {
        return Integer.toString( value );
    }
}
```