Midterm Exam
CS 102 Computer Science II

KEY

Thursday February 27, 2013
Instructor Muller
Boston College
Spring 2014

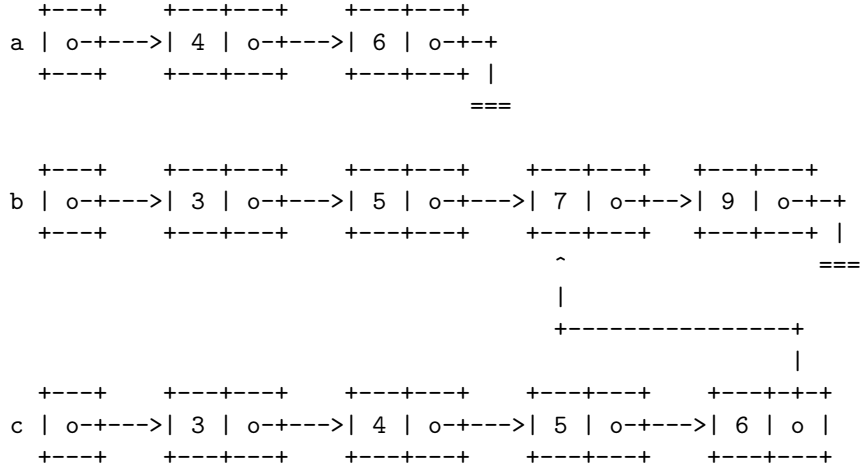
Before reading further, please arrange to have an empty seat on either side of you if you can. Now that you are seated, please write your name **on the top of the back of this exam**.

This is a closed-book and closed-notes exam but you may have one 8.5 by 11 sheet of notes. Computers, calculators and books are prohibited. **Do any 2 of the 3 problems, circling the numbers of the problems that you want graded.** Partial credit will be given so be sure to show your work. **Please try to write neatly.**

Problem	Points	Out Of	
1		8	(2 of 3)
2		8	(2 of 3)
3		8	(2 of 3)
Total		16	

1. (8 Points) In problem set 5, we implemented a non-recursive version of mergesort that sorted items stored in an array. Lists of sortable items might also be stored using linked allocation. Consider the simple implementation of lists of integers on the attached document. If `a` and `b` are `IntLists` in ascending order, then `IntList.merge(a, b)` should return an `IntList` that is the result of merging the elements of `a` and `b` in ascending order. For example:

```
IntList a = new IntList(4, new IntList(6, null)),
        b = new IntList(3, new IntList(5, new IntList(7, new IntList(9, null)))),
        c = IntList.merge(a, b);
```



Implement the static `IntList.merge` function.

Answer:

```
public static IntList merge(IntList a, IntList b) {
  if (IntList.isEmpty(a))
    return b;
  else
    if (IntList.isEmpty(b))
      return a;
    else {
      int m = a.getInfo(),
          n = b.getInfo();
      if (m < n)
        return new IntList(m, IntList.merge(a.getNext(), b));
      else
        return new IntList(n, IntList.merge(a, b.getNext()));
      }
}
```

2. (ADTs: 8 Points) You've set up an ADT shop. A client comes to you looking for Java software supporting *pairs* (a, b). For example, the client's code may need to pair a `Double` with a `String` or a `Vehicle` with an `Employee`. The client code will need to be able to make a pair using a two-argument constructor and retrieve each of the two components of a pair. The client code will need to be able to retrieve a string representation of a pair and to swap the components of a pair.

   The client indicates that all of the items that they might want to put in a pair are of a reference type `T` that implements the `Comparable<T>` interface. The client requires the pairs to be comparable to one another too because they may be stored in an order-sensitive data structure such as a binary heap.

   Design and implement an ADT for this client. As usual, you should use an `interface` for the API and a `class` to implement the interface. Note that an ADT `MyADT` that has one generic component `T` can be specified with an interface `MyADT<T>` while an ADT with two generic components can be specified with an interface using two type variables `MyADT<S, T>`.

   Answer:

```
public interface Pair<S extends Comparable<S>, T extends Comparable<T>> {
  public S getLeft();
  public T getRight();
  public String toString();
  public Pair<T, S> swap();
  public int compareTo(Pair<S, T> other);
}

public class PairC<S extends Comparable<S>, T extends Comparable<T>>
                  implements Pair<S, T> {
  private S left;
  private T right;

  public PairC(S left, T right) {
    this.left = left;
    this.right = right;
  }

  public S getLeft()  { return this.left; }
  public T getRight() { return this.right; }

  public String toString() {
   String leftString = this.getLeft().toString(),
          rightString = this.getRight().toString();
   return "(" + leftString + ", " + rightString + ")";
  }

  public Pair<T, S> swap() {
    return new PairC(this.getRight(), this.getLeft());
  }

  public int compareTo(Pair<S, T> other) {
    return this.getLeft().compareTo(other.getLeft());
  }
}
```

3. (8 Points Total)

(a) (4 Points) Let X be a set and let $R \subseteq X \times X$ be a relation on X. Then

i. R is *reflexive* iff $\forall x \in X.(x, x) \in R$,

ii. R is *symmetric* iff $\forall x, y \in X$, if $(x, y) \in R$ then $(y, x) \in R$, and

iii. R is *antisymmetric* iff $\forall x, y \in X$, if $(x, y) \in R$ and $(y, x) \in R$ then $x = y$, and

iv. R is *transitive* iff $\forall x, y, z \in X$, if $(x, y) \in R$ and $(y, z) \in R$ then $(x, z) \in R$.

A relation R on X that is *reflexive*, *antisymmetric* and *transitive* is said to be a *partial order* on X.

Let $X = \{\star, \diamond, \bullet\}$. Each question is worth 1 point.

Answers in blue:

i. Show a relation on X that is transitive but not reflexive.

$\{\}$

ii. Show a relation on X that is a partial order on X.

$\{(\star, \star), (\diamond, \diamond), (\bullet, \bullet)\}$

iii. Show a different partial order on X.

$\{(\star, \star), (\diamond, \diamond), (\bullet, \bullet), (\star, \diamond)\}$

iv. Show a relation on X that is neither symmetric nor antisymmetric.

$\{(\star, \diamond), (\star, \bullet), (\bullet, \star)\}$

(b) (4 Points) Consider a **MaxPQ** and the *binary heap* that would implement it. Show all of the successive complete binary trees that would result from the letter-by-letter insertion of

B O S T O N

Circle the complete binary trees that are well-formed binary heaps.

Answer: MaxPQs annotated with *.