CS111 FINAL JEOPARDY: THE HOME VERSION

*The game that turns CS111 into CS***fun***11*

December 12, 2006

Arrays

[1] This is the Java expression that denotes the number of elements in the array `A`.

[2] This is one advantage of representing a sequence of elements as an array rather than as a list.

[3] Suppose that `B` is an array of booleans. This is a sequence of statements that swaps the contents of the first and last slots of `B`.

[4] This is a definition of a `concatAll()` method that concatenates all of the elements of an array of strings into a single string. For example, suppose `a` is defined as follows:

```
String [] a = {"ab","cde","","f","ghij"};
```

Then `concatAll(a)` returns the string `"abcdefghij"`.

[5] This is a definition of a class method satisfying the following contract:

`public IntList intArrayToList (int [] A);`
Returns an `IntList` containing all of the elements of `A` in the same order.

Objects

[**1**] A class declaration typically includes these entities, used to keep track of an object's state.

[**2**] This keyword is used to signify a variable or method that is not tied to a specific instance of a class.

[**3**] This is a list of **all** the different **kinds** of (1) methods and (2) variables that can be in a Java class declaration.

[**4**] This is displayed in the Java Console window by an animation that contains a single sprite create via `new TextSprite(2,1)`, where the `TextSprite` class is defined as follows:

```
public class TextSprite extends Sprite {

  private int x = 17;

  public TextSprite (int a, int b) {x = 10*a + b;}

  public void updateState() {x = x/2 - 1;}

  public void drawState() {
    if (x > 0) System.out.println(2*x);
  }
}
```

[**5**] This is displayed in the Java Console window when the `main` method of the following `Counter` class is executed:

```
public class Counter {

  private static int c = 0;
  private int i;

  public Counter () {c = c + 1; i = 0;}

  public int print () {
    i = i + 1;
    System.out.println("c = " + c + "; i = " + i);}

  public void main (String [] args) {
    Counter a = new Counter(); a.print(); a.print();
    Counter b = new Counter(); b.print(); a.print();
  }
}
```

Iteration/Recursion

[**1**] This special type of recursion can also be written as a **while** loop.

[**2**] This is the equivalent `while`-loop version of the following code:

```
for (int i = 0; i < a.length; i++) {
  s[i] = a[i] + b[i];
}
```

[**3**] This is the definition of a recursive method that satisfies the following contract:

`public int sumLengths(StringList strs);`
Returns the sum of the lengths of all the strings in the given StringList.

[**4**] Given the following method,

```
public static boolean isEven(int n) {
    return (n == 0) ? true : isOdd(n-1);
},
```

this is the definition of an `isOdd()` method that uses the `isEven()` method to determine if an integer is odd.

[**5**] This is the definition of a method, written with a `while` loop, that satisfies the following contract:

`public int fileSize(String fileName) throws IOException;`
Returns the number of characters in the file whose name is given. Passes along any `IOException` encountered when opening fileName.

Lists

[**1**] When defining a recursive list method, a good strategy is to assume you can successfully invoke the method on this part of the list.

[**2**] This is one advantage of storing a sequence of elements in a list as opposed to an array.

[**3**] This list is the result of applying the following `mystery()` method to the list `[2, 3, 9, 5, 6, 4]`:

```
public IntList mystery (IntList L){
  if(isEmpty(L)){
    return empty();
  } else if ((head(L) % 3) == 0) {
    return mystery(tail(L));
  } else {
    return prepend(2*head(L),
                   mystery(tail(L)));
  }
}
```

[**4**] How many *new* list nodes are created by the invocation `appendages(ns)`, where `ns` is the list `[1,2,3]`, and `appendages` is defined below:

```
public IntList appendages (IntList L) {
  if(isEmpty(L)) return L;
  else return append(L, appendages(tail(L)));
}

public IntList append (IntList L1, IntList L2) {
  if(isEmpty(L1)) return L2;
  else return prepend(head(L1), append(tail(L1), L2));
}
```

[**5**] This is the definition of a class method `digitsToInt()` that takes a list of single-character digit strings and returns the integer that corresponds to concatenating the digits in *in reverse order*.

E.g., if `digs` is the list `["5","3","7","2"]`, then `digitsToInt(digs)` should return 2735.

You may define only one method; no auxiliary methods are allowed. Assume all the usual list operations operate on `StringList`s.

Bugs That Bite

[**1**] This is a bug in the following array method.

```
public static int product (int [] a) {
  int result = 1;
  for (int i = 0; i <= a.length; i++) {
    result = a[i] * result;
  }
  return result;
}
```

[**2**] This is a bug in the following turtle method;

```
public int halfs (int len) {
  if (len > 0) {
    fd(len);
    int rest = halfs(len/2);
    bd(len);
    return rest + 1;
  }
}
```

[**3**] This is a bug in the following method to determine if an integer list is sorted:

```
public static boolean isSorted (IntList L) {
  if (isEmpty(L)) {
    return true;
  } else {
    return (head(L) <= head(tail(L)))
           && isSorted(tail(L));
  }
}
```

[**4**] These are *three* of the *four* bugs in the following class declaration:

```
public class Circle {

  private Point center;
  private int radius;

  public void Circle (int x, int y, int radius) {
    Point center = Point(x,y);
    radius = radius; }

  public void draw (Graphics g) {
    g.drawOval(center.x - radius, center.y - radius,
               2*radius, 2*radius);}
}
```

[**5**] These are *two* bugs in the following `isMember` method for determining if a given integer is in an array of integers sorted from low to high:

```
// Assume a is sorted from low to high
public static boolean isMember (int n, int [] a) {
  int i = a.length - 1;
  while ((n < a[i]) && (i >= 0)) {
    i--;
  }
  return (n == a[i]);
}
```

Potpourri

[**1**] These are 3 of the primitive (non-Object) types in Java.

[**2**] In the Java Execution Model, this special variable appears in frames for invocations of instance methods and constructor methods but not class methods.

[**3**] Julius Caeser left out this crucial CS111 problem solving step in his famous military strategy.

[**4**] This is the picture drawn in an applet by the following statements. (Indicate relevant coordinates and colors in your picture.)

```
Graphics g = getGraphics();
Point p1 = new Point(10, 20);
Point p2 = new Point(30, 60);
g.setColor(Color.red);
g.drawOval(p1.x, p1.y, p1.y, p1.y);
g.drawRect(p1.x, p1.y, p2.x, p2.y);
Polygon p = new Polygon():
p.addPoint(p1.x, p1.y);
p.addPoint(p2.x, p2.y);
p.addPoint(p1.x, p2.y);
g.setColor(Color.green);
g.fillPoly(p);
```

[**5**] This is an expression *exp* that makes the single statement

return *exp*;

equivalent to the following statement:

```
if (a) {
  if (b) return true;
  else if (c) return false
  else return true;
} else {
  return d;
}
```