CS111 JEOPARDY: THE HOME VERSION

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

October 11, 2006

Conditionals/Recursion

[**1**] This is the part of a recursive method that guarantees the recursion will terminate.

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

`return` *exp*;

equivalent to the following statement:

```
if (b == false) {
   return true;
 } else {
   return false;
 }
```

[**3**] This is the output of the following code snippet when `x` has the value `16`:

```
if ((x > 5) && (x <= 15)) {
   System.out.println("Swiss Cheese");
  } else {
     if ((x % 2)== 0) {
       System.out.println("American Cheese");
     } else {
          System.out.println("Cheese Whiz");
     }
  }
```

[**4**] Given the recursive method below, this is the value of `tunnel(10)`.

```
public static int tunnel (int n) {
   if (n<=1) {
      return 0;
   } else {
      return 1 + tunnel(n/2);
   }
}
```

[**5**] This is a Buggle method that will drop bagels all the way to the wall, which is an unknown distance away. Bagels are dropped in all cells in front of the Buggle, but not under the Buggle's current cell. This method maintains a position and heading invariant.

Worlds

[1] This is how to set Buggle becky's position at (2,2) in BuggleWorld (assume that Becky is already created as a Buggle object).

[2] Suppose that `w` is a `PictureWorld` picture of the wedge shown below in Figure 1. This is a `PictureWorld` expression that denotes the picture in Figure 2.

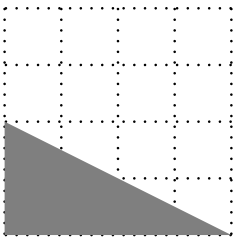

Figure 1

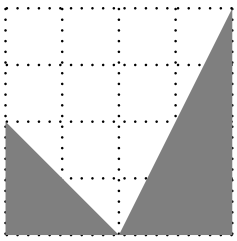

Figure 2

[3] This is a definition of the buggle method satisfying the following contract:

```
public void forwardTurningLeft(int n);
```
Moves the buggle forward a total of `n` spaces, turning left whenever the buggle encounters a wall. (Turning does not count as "moving forward a space".)

[4] This is a definition of the buggle method satisfying the following contract:

```
public boolean canGoForwardBy (int n);
```
Returns `true` if the buggle would not encounter a wall in `forward(n)`, and `false` otherwise. Executing this method should leave the state of the buggle unchanged.

[**5**] This is the picture drawn by invoking the turtle method `pattern(40)` on a new turtle, where `pattern` is defined as follows:

```
public void pattern (int n) {
  if (n < 10) {
    fd(n)
  } else {
    pattern(n/2);
    lt(90);
    fd(n);
    bd(n);
    rt(90);
    pattern(n/2);}}
```

Bugs That Bite

[**1**] This is a bug in the following Buggle method:

```
public void jello (int n) {
  if (n = 0) {
    dropBagel();
  } else {
    forward();
    jello(n - 1);
    backward();
  }
}
```

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

```
public void crazy (int n) {
  if (n > 0) {
    fd(n);
    lt(45);
    crazy(n);
    rt(45);
  }
}
```

[**3**] This is a bug in the following Buggle method:

```
public void dance (int n, Color c) {
  if ((n > 1) && !(isFacingWall())){
    forward();
    dance(c, n-2);
    backward();
  }
}
```

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

```
public int spiral (int n) {
  if (n == 0) {
    return 0;
  } else {
    fd(n); lt(90);
    spiral(n/2);
    rt(90); bd(n);
  }
}
```

[**5**] This is the bug in the code below:

```
public int eatBagels() {
  if (isFacingWall()) { return 0; }
  else { forward();
         eatBagels();
         int count =  eatBagels();
         if (isOverBagel()) { pickUpBagel();
                              backward();
                              return 1 + count;  }
         else { backward(); return count; }
  }
}
```

Potpourri

[**1**] In the Java Execution Model, this is created when an instance method is invoked.

[**2**] This is the object that all sticker superheroes have in hand.

[**3**] Mystery audio question! Identify the song!

[**4**]
This is the value of the expression **legal(methodB(4))**, given the two contracts:

`public int methodB (int n);`
Returns n multiplied by 5.

`public boolean legal(int n);`
Returns `true` if n is greater than or equal to 21 and `false` otherwise.

[**5**] This is what `countBagels()` returns for the following method definition given the configuration shown on the board.

```
public int countBagels() {
  int n = 0;
  if (isFacingWall()) {
     return n;
  } else {
     forward();
     if (isOverBagel()){ n = n + 1;}
     countBagels();
     backward();
     return n;
  }
}
```