CS125 : Introduction to Computer Science

Lecture Notes #23
Picture Recursion

Lecture 23 : Picture Recursion

Today, we will attempt to draw some pictures using recursion. We will use recursion for these particular pictures because they will have a "recursive nature" to them – that is, a similar-but-smaller version of the picture, will be part of the larger picture, and so we could (potentially) treat the drawing of the similar-but-smaller version of the picture, as a subproblem, and handle it with a recursive call.

To start with, assume we have a method to print out the same character repeatedly, with no newline at the end:

```
// assume that num is non-negative
public static void printRepeatedChar(char item, int num)
{
   if (num > 0)
   {
      System.out.print(item);
      printRepeatedChar(item, num - 1);
   }
}
```

Certainly, you could write that using a loop as well, but as is the case for many of our examples, our goal is to demonstrate recursion, so we'll stick with the recursive version above.

So, for example, the call `printRepeatedChar('X', 7)` would print the following:

```
XXXXXXX
       /|\
        |
   next character to print, goes right here
```

Seven of the `'X'` character are printed. That is how this method works.

Now, suppose we would like to draw a triangle of a given height, that borders the left side of the monitor:

```
X
XX
XXX
XXXX
XXXXX
```

If we want to write a method to do this, that method would need to know the height of the triangle:

```
public static void drawTriangle(int height)
```

Now, having a negative height makes no sense at all, so we will at least assume the argument sent to that height parameter is non-negative. We *can* allow a height of `0` if we want – in that case, the method would do nothing (since there's nothing to draw) and then return. On the other hand, we could instead require a positive argument, and thus demand that if the client wants to draw a triangle, they should send in a height that would enable *something* to be drawn. Either way is okay, as long as we clearly state the method's expectations for anyone who wants to use the method. Our choice in these examples will be to only allow a positive height, i.e. we will NOT consider `0` a legal argument to the method.

Finally, note that a triangle of height `5` contains a triangle of height `4` within it:

```
X
XX
XXX
XXXX

-------------------- above: triangle of height 4
                     below: the base we add to get a triangle of height 5

XXXXX
```

This suggests that one way to draw this picture, is to *first* draw a triangle of one smaller height, and then to add the base to make it a triangle of the desired height. Above, to draw a triangle of height `5`, we first draw a triangle of height `4`, and then add the base to make it a triangle of height `5`.

That gives us the following code:

```
// rough draft #1
public static void drawTriangle(int height)  // height >= 1
{
   if (height > 1)
   {
      drawTriangle(height - 1);        // draw smaller triangle
      printRepeatedChar('X', height);  // draw base
   }
   else  // height == 1
   {
      printRepeatedChar('X', height);  // prints one 'X'
   }
}
```

Note that we passed `height` in as the argument to `printRepeatedChar(...)`, since in our triangle pictures, the width of the base was always equal to the height of the triangle.

We actually have an error in the code! If we pass in `5` to the height right now, we would end up with the following picture:

```
XXXXXXXXXXXXXXX
               ^ cursor would be right here, after last 'X'
```

Since `printRepeatedChar(...)` doesn't end a line, we need to end the lines ourselves. Once we fix that, we get the following code:

```
// rough draft #2
public static void drawTriangle(int height)  // height >= 1
{
   if (height > 1)
   {
      drawTriangle(height - 1);        // draw smaller triangle
      printRepeatedChar('X', height);  // draw base
      System.out.println();            // start new line of triangle
   }
   else  // height == 1
   {
      printRepeatedChar('X', height);  // prints one 'X'
      System.out.println();            // start new line of triangle
   }
}
```

Finally, note that the two lines in the `else` case, also appear in the `if`-case (only the comment is different); rather than duplicate that code in both cases, we could just pull it out of the conditional altogether:

```
// rough draft #3
public static void drawTriangle(int height)  // height >= 1
{
   if (height > 1)
   {
      drawTriangle(height - 1);     // draw smaller triangle
   }
   else  // height == 1
   {
      ;
   }
   printRepeatedChar('X', height);  // draw base
   System.out.println();            // start new line of triangle
}
```

and since the `else` case only has an empty statement now, it can be completely eliminated. Also, we don't need the curly braces for the `if` case, since there's only one statement between them:

```
// final version
public static void drawTriangle(int height)  // height >= 1
{
   if (height > 1)
      drawTriangle(height - 1);     // draw smaller triangle

   printRepeatedChar('X', height);  // draw base
   System.out.println();            // start new line of triangle
}
```

Next, suppose we would like to draw a pyramid instead of a triangle. For example, here is a picture of a pyramid of height `5`:

```
    X
   XXX
  XXXXX
 XXXXXXX
XXXXXXXXX
```

Note that the picture is basically just two triangles facing in opposite directions, pushed together so that their vertical edges overlap:

```
    X   X                        X
   XX   XX                      XXX
  XXX   XXX       -------->     XXXXX
 XXXX   XXXX                  XXXXXXX
XXXXX   XXXXX                XXXXXXXXX
```

that means that the base of the pyramid, is equal to twice the base of a triangle of the same height, minus one. We subtract one because the vertical edges overlap, and so we should only count that once, not twice. Since the base of our triangle picture is equal to the height of our triangle picture, the pyramid's base – which is twice the triangle base, minus one – is equal to `2 * height - 1`.

Note also that the top four lines of that picture, are also a pyramid – the shape is the same; the picture is merely smaller.

```
   X
  XXX
 XXXXX
XXXXXXX
```

And similarly, for that pyramid of height `4`, its top three lines form a pyramid of height `3`:

```
  X
 XXX
XXXXX
```

In general, we would like to describe a pyramid of height `n`, as being a pyramid of height `n-1`, with a one-line base, just as we did for the triangle. For example, the pyramid of height `5` was just a pyramid of height `4`:

```
   X
  XXX
 XXXXX
XXXXXXX
```

followed by drawing the bottom line, the base of the pyramid of height `5`.

```
XXXXXXXXX
```

So it would seem like we can do the following:

```
DrawPyramid(int n)
   1) Draw a Pyramid of height n-1
   2) Draw the base of the pyramid of height n
```

As with the triangle, we will choose `n==1` as our base case, and assume no one will send in an argument that would result in *nothing* being printed. (That is, we'll assume no one will pass in an argument less than or equal to `0`. Again, be aware that this is not the *only* way it could be done; you could choose `n==0` as the base case, too, if you wanted, and have the method simply print nothing at all in that case.) This gives us the following code:

```
// rough draft #1
public static void DrawPyramid(int height)  // height >= 1
{
   if (height > 1)
   {
      DrawPyramid(height - 1);
      printRepeatedChar('X', 2 * height - 1);
      System.out.println();
   }
   else   // height == 1
   {
      printRepeatedChar('X', 2 * height - 1);  // could also hardcode 1
                                               //  for second argument
      System.out.println();
   }
}
```

However, there is an error with that code! If you run that with an initial argument of `5` you'll get this:

```
X
XXX
XXXXX
XXXXXXX
XXXXXXXXX
```

What went wrong? Well, since we have not discussed indentation at all, we are still assuming that the pyramid that we print will border the left-hand side of the monitor. And thus, we have made a mistake when choosing our subproblem. This is the pyramid of height `5`:

```
    X
   XXX
  XXXXX
 XXXXXXX
XXXXXXXXX
```

and this is the pyramid of height `4`, that we would expect if we passed `4` as the *initial* argument to the method:

```
   X
  XXX
 XXXXX
XXXXXXX
```

and that being the case, the pyramid of height 5 does NOT contain the pyramid of height 4. Rather, the pyramid of height 5 contains this picture:

```
    X
   XXX
  XXXXX
 XXXXXXX
```

If you look carefully, you see that that triangle, is indented one space over from the triangle of height 4 that we drew above it. *That* is the picture we want to draw recursively; we need our recursive call to NOT assume the pyramid should be bordering the left-hand side of our monitor.

So the lesson here is to be careful – sometimes, the subproblem we *think* solves our problem isn't really the one that solves our problem. We were close here, but not exact – we assumed the picture would magically be indented the right amount, when in reality, the computer will automatically print adjacent to the left-hand side of the monitor unless we state otherwise.

The only way to send information to a method is via the parameters, so we will need to add an extra parameter in our examples – a parameter that will store the amount of indentation we want. Our problem is therefore converted from "print a pyramid of a given height" to "print a pyramid of a given height and a given indentation". Here is our code with the indentation modification made:

```
// rough draft #2
public static void DrawPyramid(int height, int indent)  // height >= 1,
{                                                       // indent >= 0
   if (height > 1)
   {
      DrawPyramid(height - 1, indent + 1);
      printRepeatedChar(' ', indent);          // indent the requested amount
      printRepeatedChar('X', 2 * height - 1);
      System.out.println();
   }
   else   // height == 1
   {
      printRepeatedChar(' ', indent);          // indent the requested amount
      printRepeatedChar('X', 2 * height - 1);  // could also hardcode 1
                                               //  for second argument
      System.out.println();
   }
}
```

and as with the printing of the triangle, we can note that the two cases are the same except that the `if` case has a recursive call before everything else. Therefore, we can just put that one line in the conditional, and have everything else out of the conditional entirely:

```
// final version
public static void DrawPyramid(int height, int indent)  // height >= 1,
{                                                       // indent >= 0
   if (height > 1)
      DrawPyramid(height - 1, indent + 1); // draw smaller pyramid

   printRepeatedChar(' ', indent);         // indent the requested amount
   printRepeatedChar('X', 2 * height - 1);
   System.out.println();
}
```

If we want to draw an 'X', we have a similar issue. Actually, first, we need to decide on the actual problem:

```
*       *           *         *
 *     *             *       *
  *   *               *     *
   * *                 *   *
    *                   * *
   * *                 *   *
  *   *               *     *
 *     *             *       *
*       *           *         *
```

If we have an 'X' of height nine, which of the above do we get? The example on the left has the two lines of the 'X' cross in the middle, but the example on the right does not. Futhermore, do we allow an 'X' of height ten, or no? If we do, we'd need to match it to our choice above, so we get one of these two:

```
*       *           *         *
 *     *             *       *
  *   *               *     *
   * *                 *   *
    *                   * *
    *                   * *
   * *                 *   *
  *   *               *     *
 *     *             *       *
*       *           *         *
```

Let's simplify it for now, and stick with only odd heights and odd widths, meaning we will implement our 'X' in the first upper left example above.

In that case, our goal is accomplished by printing the first line of the 'X', then recursively printing a smaller 'X', and then printing the last line of the 'X':

```
*       *      <----- first this line
 *     *        -------|
  *   *                |
   * *                 |
    *                  |--- this part is handled by a recursive call
   * *                 |
  *   *                |
 *     *         ------|
*       *      <----- finally this line
```

But we will need to indent the recursive call's output, so that the smaller 'X' starts one space over to the right. This gives us the following code:

```
public static void PrintX(int height, int indent)
{
   if (height == 1)
   {
      printRepeatedChar(' ', indent);
      System.out.println('*');
   }
   else
   {
      printRepeatedChar(' ', indent);
      System.out.print('*');
      printRepeatedChar(' ', height - 2);
      System.out.println('*');

      PrintX(height - 2, indent + 1);

      printRepeatedChar(' ', indent);
      System.out.print('*');
      printRepeatedChar(' ', height - 2);
      System.out.println('*');
   }
}
```

As a final example, let us consider the drawing of a fan blade:

```
 *****
  ****
   ***
    **
     *
     **
     ***
     ****
     *****
```

That is a fan blade whose blade widths are each 5...but within it, there is a fan blade whose blade widths are each 4.

```
 *****
  ****   ------------|
   ***               |
    **               |
     *               |----  here is a smaller fan blade
     **              |
     ***             |
     **** -----------|
     *****
```

So we can do the same sort of thing as in the previous two examples – we make a recursive call to print the smaller picture, but we'll need to be able to indent the smaller picture when we print it. That gives us the following code:

```
public static void PrintFanBlade(int width, int indent)
{
   if (width == 1)
   {
      printRepeatedChar(' ', indent);
      System.out.println('*');
   }
   else // width > 1
   {
      printRepeatedChar(' ', indent);
      printRepeatedChar('*', width);
      System.out.println();
      PrintFanBlade(width - 1, indent + 1);
      printRepeatedChar(' ', indent + width - 1);
      printRepeatedChar('*', width);
      System.out.println();
   }
}
```