The Difference Between Getting a Program to Work and Getting it Right

Introduction

Although **Principles of Program Design** by M.A. Jackson was written in 1975, it is still considered to be one of the best books on designing programs. However, the examples given in the book use a programming notation that is archaic and difficult to understand by modern programmers. So I converted the programs to a modern programming language – XSLT.

The words in this document are from the book, with some minor alterations. This document contains just the first few pages of the book. It is intended to give you a flavor for the excellent concepts presented in the book.

Intricacy and Complexity are Vices; Clarity and Simplicity are Virtues

It is widely recognized that intricacy and complexity are programming vices; the virtues are clarity and simplicity. As we build ever larger and more powerful systems it becomes ever more important that those systems, and the components of which they are made, should be *transparently simple and self-evidently correct*. As Professor Dijkstra points out (**Structured Programming**, Academic Press 1972):

> If the chance of correctness of an individual component equals p, the chance of correctness of a whole program, composed of N such components, is something like
>
> $$P = p^N$$
>
> As N will be very large, p should be very, very close to 1 if we desire P to differ significantly from zero!

Example: Suppose the chance of correctness of an individual component is 99% (p=0.99) and the program is composed of 500 such components (N=500). The chance of correctness of the program is

$$P=(0.99)^{500}$$
$$P = 0.007$$

There is less than a 1% chance that the program is correct!

The purpose of Jackson's book is to present a coherent method and procedure for designing systems, programs and components which are transparently simple and self-evidently correct.

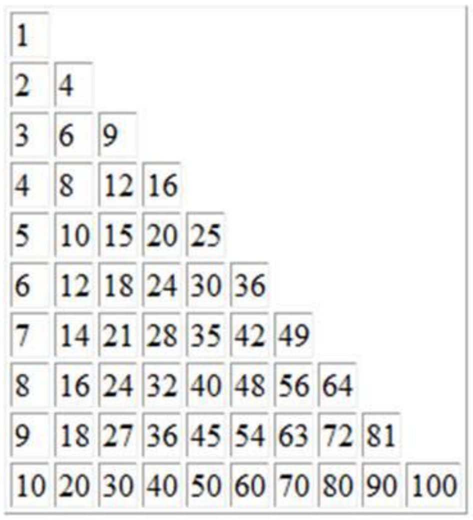
Getting a Program to Work Versus Getting it Right

The beginning of wisdom for a programmer is to recognize the difference between getting his program to work and getting it right. A program which does not work is undoubtedly wrong; but a program which does work is not necessarily right. It may still be wrong because it is hard to understand; or because it is hard to maintain as the problem requirements change; or because its structure is different from the structure of the problem; or because we cannot be sure that it does indeed work.

Problem: Multiplication Table

If a program is wrongly designed we will not be saved by the fact that each individual part is well formed. As an illustration, consider the following trivial problem.

A multiplication table is to be generated and printed. The required output is:



1									
2	4								
3	6	9							
4	8	12	16						
5	10	15	20	25					
6	12	18	24	30	36				
7	14	21	28	35	42	49			
8	16	24	32	40	48	56	64		
9	18	27	36	45	54	63	72	81	
10	20	30	40	50	60	70	80	90	100

Here is a very badly designed program to solve this problem:

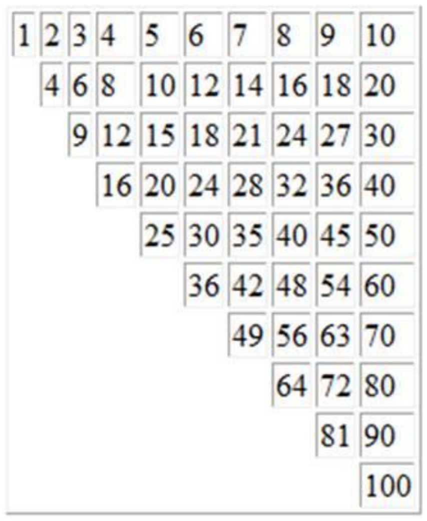
```
<!-- Print the lower-left triangular half of the multiplication table -->

<xsl:template match="/">
   <html>
      <body>
         <table border="1">
            <xsl:sequence select="p:START()" />
         </table>
      </body>
   </html>
</xsl:template>

<xsl:function name="p:START">
   <tr>
      <td>1</td>
      <xsl:for-each select="(2 to 10)">
         <td></td>
      </xsl:for-each>
   </tr>
   <xsl:for-each select="(1 to 9)">
      <xsl:variable name="line-no" select="." />
```

```
            <tr>
                <xsl:sequence select="p:LINE($line-no)" />
                <xsl:variable name="spaces" select="10 - ($line-no + 1)" />
                <xsl:for-each select="(1 to $spaces)">
                    <td></td>
                </xsl:for-each>
            </tr>
        </xsl:for-each>

    </xsl:function>

    <xsl:function name="p:LINE">
        <xsl:param name="line-no-minus-one" />

        <xsl:variable name="line-no" select="$line-no-minus-one + 1" />
        <xsl:for-each select="(0 to $line-no-minus-one)">
            <xsl:variable name="col-no" select="." />
            <td>
                <xsl:value-of select="p:NUM($line-no, $col-no)" />
            </td>
        </xsl:for-each>

    </xsl:function>

    <xsl:function name="p:NUM">
        <xsl:param name="line-no" />
        <xsl:param name="col-no-minus-one" />

        <xsl:variable name="col-no" select="$col-no-minus-one + 1" />
        <xsl:value-of select="$line-no * $col-no" />

    </xsl:function>
```

The program was designed by drawing a flowchart, and coded from the flowchart. It works correctly, producing the required output. The coding itself is well-formed; the xsl:for-each statements are correctly coded repetitions and the rest of the logic is sequential flow. And yet the structure is hideously wrong.

Consider what changes we would need to make to the program if the problem were changed in any of the following ways:

> Print the upper-right triangular half of the table
> instead of the lower-left triangular half; that is, print:



1	2	3	4	5	6	7	8	9	10
	4	6	8	10	12	14	16	18	20
		9	12	15	18	21	24	27	30
			16	20	24	28	32	36	40
				25	30	35	40	45	50
					36	42	48	54	60
						49	56	63	70
							64	72	80
								81	90
									100

The program requires substantial changes to solve the new problem:

```
<!-- Print the upper-right triangular half of the table -->

<xsl:template match="/">
   <html>
      <body>
         <table border="1">
            <xsl:sequence select="p:START()" />
         </table>
      </body>
   </html>
</xsl:template>

<xsl:function name="p:START">

   <xsl:for-each select="(0 to 8)">
      <xsl:variable name="line-no" select="." />
      <tr>
         <xsl:variable name="spaces" select="$line-no" />
         <xsl:for-each select="(1 to $spaces)">
            <td></td>
         </xsl:for-each>
         <xsl:sequence select="p:LINE($line-no)" />
      </tr>
   </xsl:for-each>
   <tr>
      <xsl:for-each select="(1 to 9)">
         <td></td>
      </xsl:for-each>
      <td>100</td>
   </tr>
```

```
</xsl:function>

<xsl:function name="p:LINE">
   <xsl:param name="line-no-minus-one" />

   <xsl:variable name="line-no" select="$line-no-minus-one + 1" />
   <xsl:for-each select="($line-no-minus-one to 9)">
      <xsl:variable name="col-no" select="." />
      <td>
         <xsl:value-of select="p:NUM($line-no, $col-no)" />
      </td>
   </xsl:for-each>

</xsl:function>

<xsl:function name="p:NUM">
   <xsl:param name="line-no" />
   <xsl:param name="col-no-minus-one" />

   <xsl:variable name="col-no" select="$col-no-minus-one + 1" />
   <xsl:value-of select="$line-no * $col-no" />

</xsl:function>
```

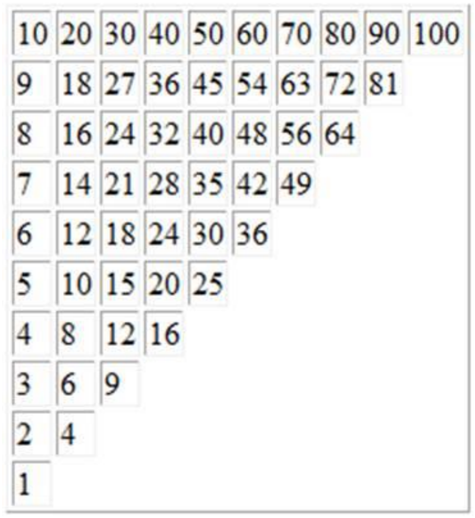
The change affects only the choice with each line of which numbers are to be printed and which omitted; instead of beginning at column=1 and continuing to print up to and including column =line-no, we want to begin with column=line-no and continue up to an including column=10. We ought to be able to make a localized change to the program – perhaps to the fifth statement of p:LINE

```
1       <xsl:function name="p:LINE">
2          <xsl:param name="line-no-minus-one" />
3
4          <xsl:variable name="line-no" select="$line-no-minus-one + 1" />
5          <xsl:for-each select="(0 to $line-no-minus-one)">
6             <xsl:variable name="col-no" select="." />
7             <td>
8                <xsl:value-of select="p:NUM($line-no, $col-no)" />
9             </td>
10         </xsl:for-each>
11
12      </xsl:function>
```

– but we cannot. The changes needed in the program amount to almost a complete rewriting.

Here's another change we might like to make to the problem:

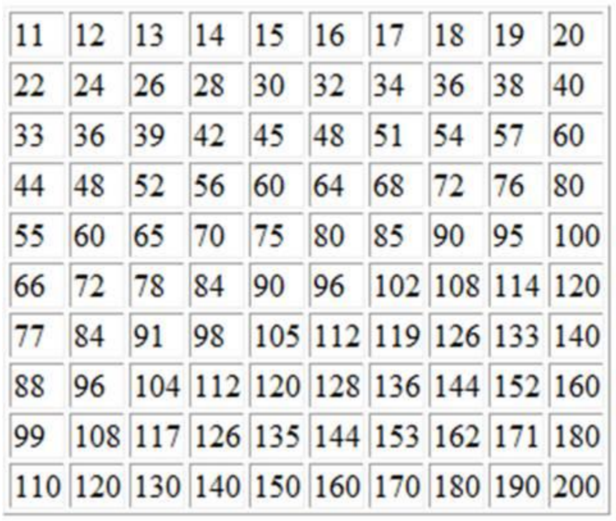
Print the lower-left triangular half of the table, but upside down; that is, with the multiples of 10 on the first line and 1 on the last line:



10	20	30	40	50	60	70	80	90	100
9	18	27	36	45	54	63	72	81	
8	16	24	32	40	48	56	64		
7	14	21	28	35	42	49			
6	12	18	24	30	36				
5	10	15	20	25					
4	8	12	16						
3	6	9							
2	4								
1									

And here's still another change we might like to make to the problem:

Print the right-hand continuation of the complete table; that is, print:



11	12	13	14	15	16	17	18	19	20
22	24	26	28	30	32	34	36	38	40
33	36	39	42	45	48	51	54	57	60
44	48	52	56	60	64	68	72	76	80
55	60	65	70	75	80	85	90	95	100
66	72	78	84	90	96	102	108	114	120
77	84	91	98	105	112	119	126	133	140
88	96	104	112	120	128	136	144	152	160
99	108	117	126	135	144	153	162	171	180
110	120	130	140	150	160	170	180	190	200

All of these changes are awkward – or as awkward as changes can be when the problem is so trivial and the program is so small.

The essence of the difficulty is this. We wanted to make simple and localized changes to the specification: to alter the choice of numbers to be printed within each line; to alter the order of printing the lines; to alter the choice and values of numbers to be printed in each line. We therefore looked to make similarly localized changes to the program: where is the component which determines the choice of numbers to be printed? where is the component which determines the order of lines to be printed? And the answers were not so simple as we hoped.

In short, the program structure does not match the problem structure. The program should instead have been written as follows:

```
<!-- Print the lower-left triangular half of the table -->

<xsl:template match="/">
   <html>
      <body>
         <table border="1">
            <xsl:sequence select="p:TABLE()" />
         </table>
      </body>
   </html>
</xsl:template>

<xsl:function name="p:TABLE">

   <xsl:for-each select="(1 to 10)">
      <xsl:variable name="line-no" select="." />
      <xsl:sequence select="p:LINE($line-no)" />
   </xsl:for-each>

</xsl:function>

<xsl:function name="p:LINE">
   <xsl:param name="line-no" />

   <tr>
      <xsl:for-each select="(1 to $line-no)">
         <xsl:variable name="col-no" select="." />
         <td>
            <xsl:value-of select="p:NUM($line-no, $col-no)" />
         </td>
      </xsl:for-each>
      <xsl:variable name="spaces" select="10 - $line-no" />
      <xsl:for-each select="(1 to $spaces)">
         <td></td>
      </xsl:for-each>
   </tr>

</xsl:function>

<xsl:function name="p:NUM">
   <xsl:param name="line-no" />
   <xsl:param name="col-no" />
```

```
        <xsl:value-of select="$line-no * $col-no" />

    </xsl:function>
```

The function p:TABLE processes the whole table. The function p:LINE processes each line. The function p:NUM processes each number. p:TABLE executes p:LINE 10 times. Each line consists of line-no numbers, and p:LINE executes p:NUM line-no times. There is a perfect correspondence between the program structure and the structure of the problem.

Let's see what changes are needed in the new program with this change to the problem:

> Print the upper-right triangular half of the table
> instead of the lower-left triangular half.

The change in the problem involves a change in the choice of numbers to be printed in each line. With our new program, the changes are localized to the p:LINE function:

```
    <!-- Print the upper-right triangular half of the table -->

    <xsl:template match="/">
        <html>
            <body>
                <table border="1">
                    <xsl:sequence select="p:TABLE()" />
                </table>
            </body>
        </html>
    </xsl:template>

    <xsl:function name="p:TABLE">

        <xsl:for-each select="(1 to 10)">
            <xsl:variable name="line-no" select="." />
            <xsl:sequence select="p:LINE($line-no)" />
        </xsl:for-each>

    </xsl:function>

    <xsl:function name="p:LINE">
        <xsl:param name="line-no" />

        <tr>
            <xsl:variable name="spaces" select="$line-no - 1" />
            <xsl:for-each select="(1 to $spaces)">
                <td></td>
            </xsl:for-each>
            <xsl:for-each select="($line-no to 10)">
            <xsl:variable name="col-no" select="." />
            <td>
               <xsl:value-of select="p:NUM($line-no, $col-no)" />
            </td>
         </xsl:for-each>
      </tr>

   </xsl:function>

   <xsl:function name="p:NUM">
      <xsl:param name="line-no" />
      <xsl:param name="col-no" />

      <xsl:value-of select="$line-no * $col-no" />

   </xsl:function>
```

Let's see what changes are needed in the new program with this change to the problem:

> Print the lower-left triangular half of the table, but
> upside down; that is, with the multiples of 10 on the
> first line and 1 on the last line.

The change in the problem involves a change in the order of printing the lines. With our new program, the changes are localized to the p:TABLE function:

```
   <!-- Print the lower-left triangular half of the table, but
        upside down -->

   <xsl:template match="/">
      <html>
         <body>
            <table border="1">
               <xsl:sequence select="p:TABLE()" />
            </table>
         </body>
      </html>
   </xsl:template>

   <xsl:function name="p:TABLE">

      <xsl:for-each select="reverse(1 to 10)">
         <xsl:variable name="line-no" select="." />
         <xsl:sequence select="p:LINE($line-no)" />
      </xsl:for-each>

   </xsl:function>

   <xsl:function name="p:LINE">
```

```
        <xsl:param name="line-no" />

        <tr>
            <xsl:for-each select="(1 to $line-no)">
                <xsl:variable name="col-no" select="." />
                <td>
                    <xsl:value-of select="p:NUM($line-no, $col-no)" />
                </td>
            </xsl:for-each>
            <xsl:variable name="spaces" select="10 - $line-no" />
            <xsl:for-each select="(1 to $spaces)">
                <td></td>
            </xsl:for-each>
        </tr>

    </xsl:function>

    <xsl:function name="p:NUM">
        <xsl:param name="line-no" />
        <xsl:param name="col-no" />

        <xsl:value-of select="$line-no * $col-no" />

    </xsl:function>
```

Let's see what changes are needed in the new program with this change to the problem:

> Print the right-hand continuation of the complete
> table.

The change in the problem just involves a change in the choice of numbers to be printed in each line. With our new program, the changes are localized to the p:LINE function:

```
   <!-- Print the right-hand continuation of the complete
          table -->

    <xsl:template match="/">
        <html>
            <body>
                <table border="1">
                    <xsl:sequence select="p:TABLE()" />
                </table>
            </body>
        </html>
    </xsl:template>

    <xsl:function name="p:TABLE">

        <xsl:for-each select="(1 to 10)">
            <xsl:variable name="line-no" select="." />
            <xsl:sequence select="p:LINE($line-no)" />
        </xsl:for-each>

    </xsl:function>

    <xsl:function name="p:LINE">
        <xsl:param name="line-no" />

        <tr>
            <xsl:for-each select="(11 to 20)">
                <xsl:variable name="col-no" select="." />
                <td>
                    <xsl:value-of select="p:NUM($line-no, $col-no)" />
                </td>
            </xsl:for-each>
        </tr>

    </xsl:function>

    <xsl:function name="p:NUM">
        <xsl:param name="line-no" />
        <xsl:param name="col-no" />

        <xsl:value-of select="$line-no * $col-no" />

    </xsl:function>
```