

RECIPECONVERTER

JAMES SCOTT-BROWN

AUGUST 5, 2009

Version 1.0 of recipeconverter.pl was released under GNU GPL on 5 August 2009 by James Scott-Brown (<http://www.jamesscottbrown.com/recipeconverter>). Here are more extensive comments about how it works.

FIRST, WE MUST GET the name of the file to process (passed as the first argument on the command line), and open it for reading:

```
2   $input=$ARGV[0];
   if (! $input){ $input='cheesecake.txt';}
   open RECIPE, "<$input";
```

To keep track of whether or not the line we're currently reading is inside a list of ingredients, we'll use a variable:

```
$inStep = 0;
```

WE'LL USE A LOOP to read the input file line-by-line. Each line is copied into a variable called \$line. It then has any terminal white-space, and anything that follows a hash, removed :

```
5 while (<RECIPE>){
   $line=$_;
7   chomp $line;
   ($line, $junk) = split('#', $line);
```

Every line is either a *note*, *whitespace*, *instruction*, or *ingredient*. Each type is to be processed differently.

If the first character is a hash, the line is a note, and is appended to the variable \$notes:

```
9 if ($line =~ s/^\#/g){
   $notes=$notes . $line . "\n\n";
11 }
```

If the line contains a colon, and is not within a list of ingredients, then it is an *instruction*. We split it into its label and instruction parts, remove terminal whitespace, and process any // characters so that they appear as newlines in the final document. We also set \$inStep to reflect the fact that we are now expecting a list of ingredients.

```

13 elseif($line =~ /\:/ && $inStep==0){
    $unrecordedData=1;
15     ($step, $instruction)=split(':', $line);
    chomp $instruction;
17
    @instructionLines=split('///', $instruction);
19     if ($instructions>1) { $instruction = "\\begin{tabular}{l}" . join('//
        ', @instructionLines) . "\\end{tabular}}" }
21
    $inStep=1;
}

```

If the line is just *whitespace*, we have either not started reading the actual instructions (in which case, there is nothing to do), or have just finished reading the list of ingredients used in one step. In the latter case, \$unrecordedData will have been set to one and we should format the list of ingredients for this step, producing a blob of L^AT_EXcode that we store in the hash %markedup, indexed by the label of that step. We must also reset \$inStep, and \$unrecordedData, to indicate that the current step has now finished. The array ingredients, which we use in the formatting is cleared to avoid listing the ingredients again in subsequent steps.

```

23 elseif ( !($line =~ /\S/) ){
25     if ($unrecordedData==1){
27         $markedup{$step}=' \left . \begin{array}{ll} \_ ' ;
        foreach (@ingredients) { $markedup{$step} .= " \mbox{" . $_ .
            "}" \_ \_ \_ \_ \_ " ; }
29         $markedup{$step} = substr $markedup{$step}, 0, -3; #remove
            terminal newline
        $markedup{$step} .= "&\_ \_ \_ \_ \_ \_ \end{array} \_ \_ \_ \_ \_ \_ \begin{
            tabular}{l}" . $instruction . '\end{tabular}}';
31
        $inStep=0;
33         @ingredients=();
        $unrecordedData=0;
35
        }
37
    }
}

```

And if the line isn't anything else, it must be an *ingredient*, in which case we just push it onto the ingredients array, to be processed when the next blank line is reached. At this stage, a label (like < A >) is treated the same as any other ingredient.

```

else{
41   push @ingredients , $line ;
}

```

NOW WE'VE READ ALL THE DATA, AND JUST NEED TO PRESENT IT NICELY. We have a separate block of code for each instruction, and wish to combine them. We do this by looking through each block of code, and replacing each label (such as $\langle A \rangle$) with its corresponding block of code (and deleting that block from the %markedup hash). To make sure that we don't miss any labels, we loop through every block in turn, and if we make any substitutions, then we set \$complete to 0, and so check them all again.

```

43 while ( $complete != 1){
    $complete=1;
45
    while((($key, $value) = each(%markedup)) {
47       if ( $markedup{$key} =~ /\mbox\{\langle(\w)\rangle\}/){
           $complete=0;
49
           ($before, $after)=split('\mbox{\langle' . $1 . "\rangle"',
           $markedup{$key});
51       $before = substr $before, 0, -1;
           $markedup{$key} = $before . $markedup{$1} . $after;
53
           delete ($markedup{$1});
55
           }
57     }
59 }

```

We print out the notes, wrapped in L^AT_EX commands for bold text, and followed by a 0.4cm vertical space:

```

chomp($notes);
61 print "\\textbf{$notes}\\vspace{0.4cm}\\n\\n" if
    $notes;

```

Finally, we print the formatted instructions:

```

63 print values %markedup;
65 }

```