The correct generation and typesetting of an index for a Greek document prepared with LaTeX is not a trivial task at all. First of all, the program `makeindex` cannot handle Greek letters, but even if the `.idx` is writen using the babel latin-transcription, the generated `.ind` file does not follow the order of the Greek alphabet. On the other hand, `makeindex` is doing a fine job for English. This means that if we let `makeindex` think that the Greek words are actually strange English words, we can use it to generate correct Greek indices. Let's be more concrete. Every line of an `.idx` file has the following format: `\indexnetry{word}{Page number}`, with the obvious meaning. If `word` is some Greek word, then in order to allow correct sorting we must sort index entries by using a transliteration that will correspond
α to a, β to b, γ to c, etc. Now, if we transform the above line into `\indexentry{trans.word@word}{Page number}`, it is possible to get a correct index. So, the first thing we need is a routine that will perform this transliteration. The following routine, `g2e`, accepts one argument, a word, and returns a transliteration of it. It can handle properly words writen with the Greek alphabet and word writen using the babel Greek transliteration. The routine is very simple: it splits the word into an array of characters, and creates a new word by transliterating each character of the original word. Special care has been taken to correctly process the letter ``Capital Alpha with accute'', since the ISO8859-7 and the Windows-1253 encodings have reserved different slots for this character.

```
<Routine that performs transliteration>=
sub g2e
{
   $word = $_[0];
   $tword = "";
   foreach $_ (split(//, $word))
   {
     if    (/^a|^A|^α|^A|^ά|^\xA2|^\xB6/) { $tword .= "a";}
     elsif (/^b|^B|^β|^B/)                { $tword .= "b"; }
     elsif (/^d|^D|^δ|^Δ/)                { $tword .= "d"; }
     elsif (/^e|^E|^ε|^E|^έ|^Έ/)          { $tword .= "e"; }
     elsif (/^i|^I|^ι|^I|^ί|^Ί|^ΐ/)      { $tword .= "i"; }
     elsif (/^o|^O|^ο|^O|^ό|^Ό/)          { $tword .= "o"; }
     elsif (/^p|^P|^π|^Π/)                { $tword .= "p"; }
     elsif (/^g|^G|^γ|^Γ/)                { $tword .= "c"; }
     elsif (/^z|^Z|^ζ|^Z/)                { $tword .= "f"; }
     elsif (/^h|^H|^η|^H|^ή|^Ή/)          { $tword .= "g"; }
     elsif (/^j|^J|^θ|^θ/)                { $tword .= "h"; }
     elsif (/^k|^K|^κ|^K/)                { $tword .= "j"; }
     elsif (/^l|^L|^λ|^Λ/)                { $tword .= "k"; }
     elsif (/^m|^M|^μ|^M/)                { $tword .= "l"; }
     elsif (/^n|^N|^ν|^N/)                { $tword .= "m"; }
     elsif (/^x|^X|^ξ|^Ξ/)                { $tword .= "n"; }
     elsif (/^r|^R|^ρ|^P/)                { $tword .= "q"; }
     elsif (/^s|^c|^S|^σ|^ς|^Σ/)          { $tword .= "r"; }
     elsif (/^t|^T|^τ|^T/)                { $tword .= "s"; }
     elsif (/^u|^U|^υ|^Y|^ύ|^Ύ|^ΰ/)       { $tword .= "t"; }
     elsif (/^f|^F|^φ|^Φ/)                { $tword .= "u"; }
     elsif (/^q|^Q|^χ|^X/)                { $tword .= "v"; }
     elsif (/^y|^Y|^ψ|^Ψ/)                { $tword .= "w"; }
     elsif (/^w|^W|^ω|^Ω|^ώ|^Ώ/)          { $tword .= "x"; }
     elsif (/^'|^`|^~|^<|^>/)             {                }
```

```
      else                                { $tword .= $_   }
   }
   return $tword;
}
```

Now, that we have solved one problem we must face the others, namely the modification of the `.idx` file and the subsequent modification of the `.ind` file. (This of course means that we use `makeindex` to actually generate the index.) Before we do anything we must get the various command line arguments. Our program accepts at most three command line arguments: `mkgrkindex.pl [-s A|a] [-l] index`. With the `-s` switch one can specify which index style he wants to be used: `A` stands for an index where each group of entries that start with the same letter has as group-header this letter in uppercase form; and `a` means that the group-headers are lowercase letter. The absence of this switch denotes that the index will be just an ordinary LaTeX index, i.e., without a group-header. The `-l` specifies that this is an index that has only Latin words. The absence of this switch means that the index is a Greek one. Finally, `index` denotes the name of the `.idx` file; users may omit the file extension. The first think we do is to check whether there any command line arguments. In case there aren't or there more than three, we just print a usage message and exit. Otherwise, we print the version number and we process the command line arguments.

```
<Check for command line arguments>=
$argc = @ARGV;

if ($argc == 0 || $argc > 4 ) # no command line arguments or more than 3
{                             # arguments
   die "Usage: mkgrkindex.pl [-s A|a] [-l] index\n";
}
else
{
   print "This is mkgrkindex (previously known as mkindex) version 2.2\n";
   <Process command line arguments>
}
```

Since we don't know apriori the number of arguments we set the following global variables: `$style` and `$is_latin`. For variable `$style` a negative value means that a lowercase letter will precede each group of words beginning with the same letter, a positive value means that the letter will be uppercase and the value 0 means that there will be no letter preceding each group of words. While for variable `$is_latin` a positive value means that the index is non-Greek script index and a negative or zero means it is a Greek script index. The last thing we must do is to check whether the index file exists.

```
<Process command line arguments>=

   $style = 0;
   $is_latin = 0;
   <Get command line arguments>
   <Check if .idx file exists>
```

In order to get the various command line arguments we use a simple `while` loop that check each element of the array `@ARGV`. We check for both switches and we get the name of the file that contains the index.

```
<Get command line arguments>=

   SWITCHES:
   while($_ = $ARGV[0])
   {
      shift;
      if (/^-s/)
      {
         if ($ARGV[0] eq "a")
         {
            $style = -1;
         }
         elsif ($ARGV[0] eq "A")
         {
            $style = 1;
         }
         else
         {
            die "$ARGV[0]: Illegal argument for switch -s\n";
         }
         shift;
      }
      elsif (/^-l/)
      {
         $is_latin = 1;
      }
      elsif (/^-\w+/)
      {
         die "$_: Illegal command line switch!\n";
      }
      else
      {
         $file = $_;
      }
   }
   die "No index file name specified!\n" if $file eq "";
```

In order to check whether the index file exists, we simply use the `-e` operator. First we check if `$file` exits, if doesn't then probably file `$file.idx` exists, i.e., the user hasn't typed the filename extension. In case this doesn't exist, then we are sure there is no index file. After these checks, variable `$file` contains the file name without extension.

```
<Check if .idx file exists>=
   if (! (-e $file))
   {
       die "$file: no such file!\n" if $file =~ /.+\..+/;
       die "$file.idx: no such file!\n" if (! (-e "$file.idx"));
       $index_file = "$file.idx";
   }
   else
   {
```

```
        $index_file = $file;
        $file = $1 if $index_file =~ /(.+)\..+/;
    }
```

We have now all the information we need in order to modify the index file. First we must rename the index file, since we are going to modify it. Next, we open the renamed file for reading and we open a fresh file into which we will write the modified index. These changes must be done only if this is a Greek index, i.e., if the value of `$is_latin` is equal to zero. The next think we do is to modify the index entries. If an index entry appears in the `\frontmatter` of a LaTeX document, then its page number comes out as a Latin number. Since we are preparing a document with the `greek` option of the babel package this number comes out as an argument of the command `\textlatin`, e.g., `\textlatin{ix}`. This means that we must modify the page information too. However, this change requires to rescan the whole index file. So, we delete the original index file, rename the modified one and then correct the page numbers, by using the same method.

```
<Modify LaTeX index file>=

$old_file="$index_file.old";

if ($is_latin == 0)
{
   rename $index_file, $old_file;
   open(OLD, "<$old_file") || die "Can't open file $old_file\n";
   open(NEW, ">$index_file") || die "Can't open file $index_file\n";
   <Modify index entries>
   close OLD;
   close NEW;
   unlink $old_file;
}

rename $index_file, $old_file;
open(OLD, "<$old_file") || die "Can't open file $old_file\n";
open(NEW, ">$index_file") || die "Can't open file $index_file\n";

<Correct page numbers>

close OLD;
close NEW;
unlink $old_file;
```

If the value of variable `$is_latin` is positive there is nothing to do. But, if it is zero, then we change each index line. Nikos Platis has suggested a better method to parse individual lines. Since this method is better, I have decided to use it. The method uses `split` to split what goes inside the curly brackets into simpler parts. This function takes two arguments, a regular expression and a string, and if the regular expression matches somewhere in the string, the returns what goes before and after the substring that matched. The method tries to first split the string at the `|` symbol. Note that the expression `(?<!")\x7C` matches any `|` that is not preceeded by a quotation mark (the

original algorithm did not take care of this detail). Then it splits what goes before | at each occurence of the ! symbol. Finally, each part of these substrings is splitted at the point where the @ symbol is found. The last thing is to make all necesary transformations and to assemble the index entry.

<Modify index entries>=

```
while (<OLD>)
{
    #\x7B = {, \x7D = },\x40 = @, \x7C = |, \x21 = !, \x28 = (, \x29 = )
    chomp;
    #Nikos Platis has suggested the replacement of the original code with a better
    #parsing method. The code that follows is essentially Nikos's code with
    #some minnor modifications, as explained in the documentation.
    $newentry = "";
    /^\\indexentry\x7B(.+)\x7D(.+)/;
    $fullentry = $1;
    $page = $2;
    ($indexentry, $format) = split(/(?<!")\x7C/, $fullentry);
    @entryparts = split(/(?<!")\x21/, $indexentry);
    $numparts = @entryparts;
    for ($i = 0; $i < $numparts; $i++)  {
      ($x, $y) = split(/(?<!")\x40/, @entryparts[$i]);
      if ($i > 0)  {
        $newentry .= "!"
      }
      $tx = g2e($x);
      if ($y) {
        $newentry .= "$tx\@$y"
      }
      else {
        $newentry .= "$tx\@$x";
      }
    }
    print NEW "\\indexentry{$newentry";
    if ($format) {
      print NEW "|$format"
    }
    print NEW "}$page\n"
}
```

Correcting the page numbers is very easy. We check each line and if one happens to have the form `\indexentry{w}{\textlatin{p}}`, then we replace it with `\indexentry{w|textlatin}{p}`. Of course this does not take care of cases where the | operator is already in use, but then we make the assumption that definitions and such stuff do not appear in the foreword of a document.

<Correct page numbers>=

```
while(<OLD>)
{
    chomp($_);
    if (/\\indexentry\x7B(.+)\x7D\x7B\\textlatin\s*?\x7B(\w+)\x7D\x7D/)
    {
       print NEW "\\indexentry{$1|textlatin}{$2}\n";
    }
```

```
    else
    {
       print NEW "$_\n";
    }
}
```

The `.ind` file will be generated by invoking program `makeindex`. In case the name of the program is different in the user's installation, he/she must change the assignment at the beginning of the program. As we said, the user can choose between three different types of indices. Indices of type "A" and "a", assume that LaTeX and `makeindex` have access to files `uppercase-headers.ist` and `lowercase-headers.ist`. So, make sure these files are properly installed. Depending on the value of variable `$style`, we generate the appropriate index.

```
<Generate .ind file>=

if ($style < 0)
{
   system("$makeindex -s lowercase-headers.ist $file");
}
elsif ($style > 0)
{
   system("$makeindex -s uppercase-headers.ist $file");
}
else
{
   system("$makeindex $file");
}
```

Now the we have created the `.ind` file, if the user has chosen either the `A` or `a` option we must modify the file so that the English letters become Greek ones. The mechanism is very simple: we employ the inverse of the mapping we employed in subroutine `g2e`. In order to do this we use a hash table. In case the `$is_latin == 1`, then the only thing we need to do is to translate the word ``symbol'' into Greek. Before, we do anything further we must open the file and to rename the old index file.

```
<Modify the .ind file>=

<Declare hash table>
$ind_file = "$file.ind";
$old_file="$ind_file.old";

if ($is_latin == 0)
{
   rename $ind_file, $old_file;
   open(OLD, "<$old_file") || die "Can't open file $old_file\n";
   open(NEW, ">$ind_file") || die "Can't open file $ind_file\n";
   <Correct header letters>
   close OLD;
   close NEW;
   unlink $old_file;
}
```

```
rename $ind_file, $old_file;
open(OLD, "<$old_file") || die "Can't open file $old_file\n";
open(NEW, ">$ind_file") || die "Can't open file $ind_file\n";
<Translate the word symbol>
close OLD;
close NEW;
unlink $old_file;
```

We declare a hash table in which we will store the inverse transliteration table.

```
<Declare hash table>=

%e2g = (
          'a' => 'α', 'A' => 'A',
          'b' => 'β', 'B' => 'B',
          'c' => 'γ', 'C' => 'Γ',
          'd' => 'δ', 'D' => 'Δ',
          'e' => 'ε', 'E' => 'E',
          'f' => 'ζ', 'F' => 'Z',
          'g' => 'η', 'G' => 'H',
          'h' => 'θ', 'H' => 'θ',
          'i' => 'ι', 'I' => 'I',
          'j' => 'κ', 'J' => 'K',
          'k' => 'λ', 'K' => 'Λ',
          'l' => 'μ', 'L' => 'M',
          'm' => 'ν', 'M' => 'N',
          'n' => 'ξ', 'N' => 'Ξ',
          'o' => 'ο', 'O' => 'O',
          'p' => 'π', 'P' => 'Π',
          'q' => 'ρ', 'Q' => 'P',
          'r' => 'σν', 'R' => 'Σ',
          's' => 'τ', 'S' => 'T',
          't' => 'υ', 'T' => 'Y',
          'u' => 'φ', 'U' => 'Φ',
          'v' => 'χ', 'V' => 'X',
          'w' => 'ψ', 'W' => 'Ψ',
          'x' => 'ω', 'X' => 'Ω'
        );
```

In order to translate the word symbol we must just scan the entire `.ind` file and to find the word `symbols` or `Symbols`. This isn't difficult since each header line is of the form `{\hfil word \hfil}`, where `word` is either the a letter, the word `symbols` or the word `Symbols`. Once we find the word we replace it with its translation.

```
<Translate the word symbol>=

while (<OLD>)
{
   if (/^\x7B\\hfil (\w)ymbols \\hfil\x7D/)
   {
       if ($1 eq "s")
       {
              print NEW "{\\hfil \\textgreek{\\textbf{s'umbola}}";
              print NEW "\\hfil}\\nopagebreak\n";
```

```
        }
        elsif ($1 eq "S")
        {
               print NEW "{\\hfil \\textgreek{\\textbf{S'umbola}}";
               print NEW "\\hfil}\\nopagebreak\n";
        }
        else
        {
           die "Illegal header $1 in .ind file\n";
        }
    }
    else
    {
       print NEW;
    }
}
```

Correcting the header letters means to scan the whole index file and replace the English letters with Greek ones. We find lines of the form `{\hfil a \hfil}\nopagebreak` and we replace the English letter with the corresponding Greek by using the hash `%e2g`.

```
<Correct header letters>=

while (<OLD>)
{
     if (/^{\\hfil (\w?) \\hfil}/)
     {
        $lettergr = $e2g{$1};
        print NEW "{\\hfil $lettergr \\hfil}\\nopagebreak\n";
     }
     else
     {
        print NEW ;
     }
}
```

Let's summarize. This program is makes it possible to use program `makeindex` as if this program was writen for Greek people only! The program works as follows: (a) it check the command line arguments, (b) it modifies the LaTeX generated index, (c) generates the `.ind` according to the user's wishes, and (d) corrects the `.ind` file if necessary, i.e., if the user has asked for some alphabetic headers in his/her index.

```
<*>=
#!/usr/bin/env perl
#
#(c) Copyright 1998-2009   Apostolos Syropoulos
#                          asyropoulos@yahoo.com
#
# The LaTeX Project Public License (lppl)
# This software is copyright but you are granted a license which gives you,
# the "user" of the software, legal permission to copy, distribute, and/or
# modify the software. However, if you modify the software and then distribute
# it (even just locally) you must change the name of the software, or use other
# technical means to avoid confusion.
```

```
#
$makeindex = "makeindex"; #name of the index generation utility
<Routine that performs transliteration>
<Check for command line arguments>
<Modify LaTeX index file>
<Generate .ind file>
if ($style != 0)
{
   <Modify the .ind file>
}
__END__
```