

music - a *troff* preprocessor for printing music scores

User Manual

Eric Foxley

Departments of Mathematics and Computer Science
University of Nottingham
Nottingham NG7 2RD, UK
ef @ cs.nott.ac.uk

The *music* preprocessor provides a language for describing music scores, which can then be processed to produce output suitable for the *troff* typesetting system and its other preprocessors, which run under the UNIX† operating system. This document describes the basic facilities available in the *music* preprocessor, and gives examples of its use.

1. Introduction

The output

Cock of the North (Kevin)

A

A A D A B7 E7

A A D A G A

was created from an input file containing

```
..MS
title = "\fBCock of the North\fp \fI(Kevin)\fp";
timesig = 6 8; autobeam; key = a.

e< 'A' d< | # the "<" indicates a short note
c "A" d c c b a |
a "A" c e f↑> "D" e | # the ↑ indicates above the stave
\ 1 | # repeat bar 1
b "B7" c b b "E7" e d |
\ 1, 2 | # repeat bars 1 to 2
c> "A" c b "G" g= b | # the ">" indicates a long note, the "=" a natural
a>. "A" a> :| # the ">." indicates a dotted long note
..ME
```

The *music* system is another *troff*^{4,8} preprocessor. It passes most of its input through untouched, but translates data between lines “.MS” and “.ME” into commands suitable for the *pic*^{5,6} preprocessor, which can then draw the necessary pictures. Text outside and inside the *music* system can use the full features of the other *troff* preprocessors such as *eqn*³ and *tbl*⁷ if required. A typical UNIX command would then be

```
music source.file | pic | troff -ms
```

or

```
music source.file | pic | eqn | troff -ms
```

if the facilities of *eqn* are required.

† UNIX is a trademark of Bell Laboratories.

The particular rules for layout of the musical symbols are based on examples given in Stone¹⁰ and in the Oxford Concise Dictionary of Music⁹ subject to interpretation and variation. Suggestions for alternative rules would be appreciated, and could perhaps be built into the system as options. For a more general discussion in this area, the reader is referred to the paper by John Gourlay¹ for discussion of languages for representing music scores, and to the annual report from CCARH² for information on the current state of computer applications in music printing and in general musicology.

2. The input language

The basic input for the musical score is contained between lines “.MS” and “.ME”, and consists of header information describing the output format required, and input defaults, followed by the score details. All text not within a “.MS” to “.ME” section is passed straight through unchanged.

2.1. Header information

The header sets up variables such as the piece title, output width, time signature and key, each of which is specified by an entry of the form

```
<identifier> = <value>
```

The header items are separated by semi-colons, and terminated by a full-stop. A typical header for a straightforward example might be

```
title = "Twinkle twinkle little star";
timesig = 4 4;
key = d.
```

In all straightforward cases a short header is acceptable, since most items default to sensible values. However, the header items have to allow for many variations in output format, and examples of the major possibilities are shown in the following example:

Titles

```
title = "..."; # printed at top left of output; default is to have no title
ctitle = "..."; # printed at the top centre of the output, if given
rtitle = "..."; # printed at the top right of the output, if given
```

Time signature

```
timesig = 3 4; # sets the default note length to semi-breve divided by 4
# (the second number),
# and the default bar length to this times 3
# (the first number)
# the musical length of bars is checked against this
# "timesig = c;" or "timesig = 4 4 c;" sets to 4 4,
# and is printed as "c"
# "timesig = c|;" or "timesig = 2 2 c;" sets to 2 2
# "timesig = c.;" or "timesig = 3 4 c;" sets to 3 4
```

Keys

```
key = g; # the key of the piece is G for both input and output
# all F notes on input now default to F-sharp
keyin = f; # the input key can be specified distinctly from
keyout = a; # the output key to produce transposed output

transpose = 1 -1; # instead of using "keyin" and "keyout",
# this option specifies the number of additional
# sharps and octave displacement in the
# output key compared with the currently set input key
```

Octave

```
octave = s; # sets the default input octave to that within the
# stave of the treble clef
# see text for details of other options
```

Number of bars

```
bars = 8; # the number of bars in the piece
        # used for checking, incorrect value produces a warning
```

Dimensions

```
width = 6.5i; # the printed stave width in inches
            # the current troff default width is 6 inches
            # it can also be given in centimetres as "width = 16.51c"
height = .25i; # the height of a 5-bar stave
isg = .20i;   # increase the spacing to be left between staves,
            # the "inter-stave-gap" by 0.20 inches
```

Signature

```
sig = ckt.   # the "c" is "print a clef at the left of each stave"
            # the "k" is "print the key on each stave"
            # the "t" is "print the time-signature on the first stave"
            # the default is to have all
```

The header items are in any order, separated by semi-colons; the last is terminated by a full stop. The defaults are set in a file called *mus_default*, of which details are given in section 5. All unspecified items default from any previous header. An empty header is indicated by a full stop. Further header items will be described later.

Note that all text from any “#” symbol to end-of-line is ignored, and that if the last character of a line is the escape character “\”, then the next line is treated as a continuation of the current line.

2.2. Score details

The header is terminated by a full stop, and is followed by the score. The score consists of notes interspersed with bar-lines. There is a warning if the sum of the note lengths in any bar does not add up to the required bar length as deduced from the time signature, or if the total number of bars does not agree with that specified in the header. Both of these checks have been found to be useful.

The pitches of the notes are typed as lower-case letters relative to the current key, as in

```
d d a a | b b a | g g f f | e e d
```

Sharps and flats of the current key are omitted. Other required accidentals are typed at the first occurrence in the bar using “+” for sharp, “-” for flat, and “=” for natural. For example, “g+” represents g-sharp, “e-” represents e-flat, and “f=” represents f-natural in a key such as D. A “+” symbol against an already sharpened note is ignored; a “+” symbol against a note which is sharp in this key, but which occurred with a natural earlier in the bar, cancels the natural accidental. For double sharp, use two plus symbols, as in “g++”, and for double flat two minus symbols as in “g--”. On output, the computer will print only the necessary accidentals, omitting, for example, accidental signs on all but the first occurrence of a given accidental within a bar. A cancelling accidental will be printed if the note is used in the following bar.

The length of a note defaults to the value indicated by the denominator of the time signature, and is thus a quaver if the denominator is 8, a crochet if it is 4, and so on. To specify other lengths, symbols “>” after the note double its length, symbols “<” halve it, “.” increases it by 50%, and “..” increases it by 75%. Thus in 4/4 time, “c>.” represents a dotted minim, and in 6/8 time “c<<” represents a demi-semi-quaver. As an example, the source

```
..MS
timesig = 4 4;
key = d;
bars = 4.
d d a a | b b a> | g< g< g< g< f. f< | e e d> |
..ME
```

produces the output



The length of the default note for input may be changed by using these note duration symbols in association with the note given as the key in the header. Thus if the key is specified as “d>”, the key is D, and the default note length is double that which would otherwise be expected. If the above example is repeated changing two header entries to halve both the bar-length and the default note-length, the file becomes

```
..MS
timesig = 2 4;
key = d<; # half-length default note
bars = 4.
d d a a | b b a> | g< g< g< g< f. f< | e e d> |
..ME
```

and the resulting output is



For this effect, the key must be set **after** the time signature in the header, since the “timesig” entry itself resets the default note-length. The full-stop cannot be used to increase the default note-length by 50%, the default can be changed only by factors of 2.

For each note in the source, the letter (and possible accidental) specifying the pitch can be followed by an indication of octave displacement. Symbols “↑” after the pitch indicate an octave upward, symbols “↓” indicate an octave down*. The default octave can be set in different ways. It can be set to that from middle C to the B above using the header entry

```
octave = c
```

In this case “c↑↑” is two octaves above middle C, and “b↓” is one note below middle C. Thus in the key of G, the score

```
g< d< b↓< d< g< b< c↑< d↑< | e↑ e↑ d↑> | c↑ c↑ b b | a a g> |
```

produces the output



The octave symbols(s) must at present precede the note duration symbol(s). The default octave can be moved up or down an octave by appending “↑” or “↓” symbols to the note in the key definition in the header. The previous example, if the default key is specified as

```
key = g↑; # g up an octave
```

we obtain the output



* At Nottingham the ↑ symbol is typed as the circumflex symbol “ˆ”, and the ↓ symbol as the small letter “v”. The particular character is a compile-time definition.

The header entry

octave = s;

sets the default octave to that contained within the treble clef stave, from F above middle C to the E above that. The header entry

octave = b;

sets the default octave to that contained within the bass clef stave, from the A an octave and a half below middle C to the G below middle C. The header entry

octave = k;

sets the default octave to be that starting at the current key note. The entry

octave = c;

causes the default octave to be that starting from middle C. The entry

octave = p;

causes the octave of each note to be chosen to make its pitch as close as possible to that of the previous note. Thus in this mode the notes

a b c d | e f g a | a g f e | d c b a |

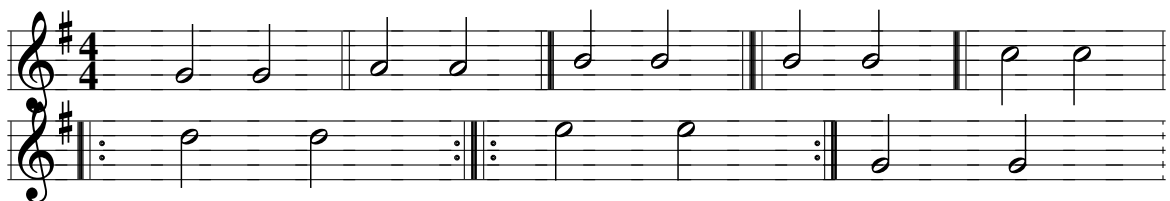
would give a run up and down an octave scale. In this mode, the very first note is set according to the “octave = s” rule. Note that although this option in general minimises the typing of input, it has the unfortunate side effect that octave errors now propogate throughout the rest of the piece.

Bar-lines

Bars are delimited by bar-lines. A limited variety of bar-lines is available, some indicated by an obvious construction, others by a letter following the bar symbol. A selection of these is indicated by

g> g> || a> a> |! b> b> |! b> b> !|
c↑> c↑> |: d↑> d↑> :|: e↑> e↑> :| g> g> |H

which produces



Note that the system is sufficiently intelligent to replace for example the “:|:” symbol if it occurs at the end of a stave by a “:|” symbol at the end of that stave, and a “|:” symbol at the beginning of the next stave.

Rests

Rests are indicated by the underscore symbol “_”, with lengths specified as for notes using the symbols “>”, “<” and “.”. The default length is the same as the value set for notes. The input

a> _> | a _ c.. _<< | a< _< c< _< a _ |

gives the output



Full bar rest symbols are centred in the bar; part bar rests are positioned proportionally as for notes.

Beams

To cause notes shorter than crochets to be joined under a common beam, the notes to be joined are put within square brackets, as in

```
[ a b c ] [ d e f ]
```

A beam will correctly fail if it includes any notes longer than or equal to a crochet; at the moment, it also fails if it includes any rests. The latter restriction should be removed soon. A beam cannot cross a bar-line.

As an example of beams, the source

```
timesig = 4 4; key = d<.  
d↓> [ d↓< e↓< f< g< ] a> a> | [ b. a< g< b. ] a>> |  
g> [ g. g< ] f> [ f g ] | [ a< g< a< b< ] [ a< g< f< e↓< ] d↓>> |!
```

appears as



To simplify the typing of input in straightforward cases, a system for automatically inserting beams is available, and can be invoked by inserting the additional line

```
autobeam;
```

in the music heading. The automatic beaming uses rules from the Oxford Concise Dictionary of Music⁹ which may not always be exactly what is required. The description given there has been interpreted as meaning that beams will not normally cross the divisions of a bar as given by the denominator of the time signature in simple time, and a quarter of that figure in compound time. Thus in 2/2 or 6/8 time, beams will not cross the midpoint of the bar, while in 3/4 or 9/8 they will not cross the one-third points.

Automatically generated beams will never over-ride beams inserted manually. Having been invoked, auto-beaming can be switched off by the header

```
autobeam = -1;
```

As a further facility, the auto-beamer can be instructed never to let a beam cross for example a “2 times default note-length” interval using

```
autobeam = 2;
```

to allow half-bars to be beamed in 4/4 time.

Different formulae for the slope of the beam are built into the program; formula number 2 can be accessed by the header entry

```
beamslope = 2;
```

Details are the formulae are available from the author. Other formulae could easily be added; suggestions are welcome.

Ties

Ties are indicated by putting parentheses (round brackets) around the notes to be joined under the tie. Their production is through the use of splines in *pic*; an angular version is available if splines are not provided. The system decides on details of the exact position of the tie; improvements to allow more detailed user specification await decisions on a clean syntax. The input

```
g | ( e> a> | c> g^> ) | ( e> ( e> ) | ( e> ) e> ) |!
```

gives



See the appendix at the end of this document for further examples. The header entry

```
ties = o;
```

causes ties to be above the notes;

```
ties = u;
```

sets them below the notes.

Grouped notes

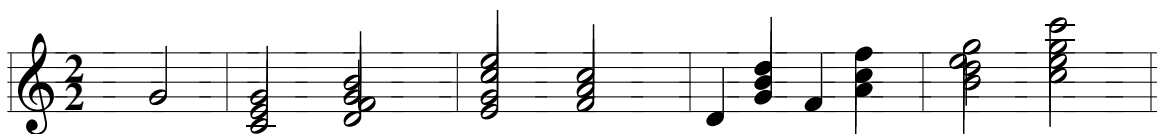
Notes to be joined together on a single stick are grouped by curly braces. If a number of groups are to be beamed, the sequence of symbols is as in

```
[ { c e g } { f a c } { b d g } ]
```

The input

```
timesig = 2 2; key = c;  
octave = s;  
bars = 4.  
g | { c↓ e↓ g } { d↓ f g b } | { e↓ g c e } { f a c } |  
d↓< { g< b< d< } f< { a< c< f↑< } | { b d e g↑ } { c e g↑ c↑ } |!
```

produces



There can be any number of notes in a group, and they can be given in any order; they must all be of the same length. If the lengths differ, the length of the first is taken as the length of all of them.

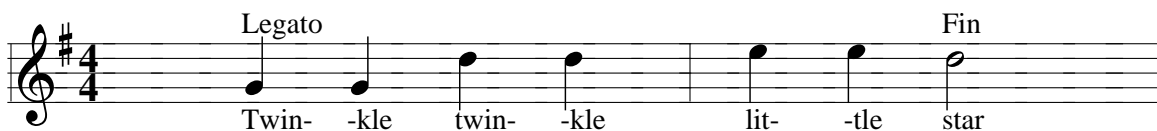
Text

The title of the piece as (and if) given with the keyword *title* in the header information appears at the top left corner of the output. Other titles can be given to appear at the the centre and at the top right-hand corner of the output using the keywords *ctitle* and *rtitle* in the header.

Additional items of text can be given to be positioned relative to any note or bar. For text associated with notes, the text required is contained within quotes to indicate its positioning. Single quotes ('Moderato') indicate text to be positioned above the staff, left justified and starting at the horizontal coordinate of the note; text in double quotes ("Tinkle") indicates text to be positioned below the staff; and text between "@" signs (@3@) indicates text to be positioned close to the note. Text of the last form will be positioned just above the note if it has a downward stick, and *vice versa*. Thus the input

```
g 'Legato' "Twin-" g "-kle" d "twin-" d "-kle" |  
e "lit-" e "-tle" d> 'Fin' "star" |
```

generates the output



The particular string @3@ is interpreted to imply a triplet, three notes in the time of two. The spacing of the notes on the score and the checking of bar-lengths takes this into account. Full treatment of general tuples appears later in this document.

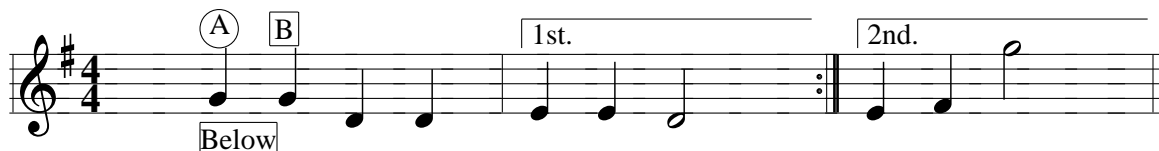
Text can also be associated with a bar. It is given after the bar-line, and is printed lined up with the start of the bar; it will be above the staff if contained in single quotes, and below if in double quotes.

Text strings starting with the escape character “\” are treated specially. Any text string starting “\ b” appears in a box, any starting “\ c” appears in a circle. and text starting “\ 1” or “\ 2” above the staff is

used for alternative bars on repeats. The second type, starting “\ c”, will be expected to be single character strings. For example, the file

```
g '\ cA' "\ bBelow" g '\ bB' d d | e '\ 1st.' e d> :| e '\ 2nd.' f g↑> |!
```

produces the output



Text starting “\ >” is assumed to be a *diminuendo*; its width will stretch over two notes if written “\ > 2n”, over two bars if written “\ > 2b”, and over 2 inches if written “\ > 2i”.

Text between @ symbols starting “\ .”, “\ -”, “\ <”, “\ >” is interpreted as accent symbols and positioned accordingly; see the appendix for examples.

It is hoped that most recognised musical symbols (such as pause and segno) will eventually be included in the standard system; such additional characters are then accessed by the usual methods. They may be either symbols generated as *pic* macros, or as additional symbols in the font. In the latter case they are accessed by methods appropriate to *troff*, typically being indicated either by a 2-letter code preceded by the characters “\ (“, or by a change to the music font using “\ f(MU” Users of the Sonata font can now access certain specific symbols directly by the strings “\ segno”, “\ pause”, “\ mordant”, “\ invmord” (inverted mordant), “\ turn”, “\ bowup”, “\ bowdown” (violin bowing symbols), “\ ped” (Pedal), “\ dc” (D.C. or da capo), “\ ds”, etc. The system recognises “\ +” anywhere in text to represent the sharp symbol, “\ -” for flat, and “\ =” for natural.

If text is given as two strings, such as

```
"1. G" "2. Em"
```

the two text strings “1. G” and “2. Em” appear one above the other, vertically aligned at the left, as in



Duplicate copies of earlier bars

The notation

```
\ 4 |
```

cause a duplicate of bar number 4* to be inserted at this point. The similar notation

```
\ 1, 3 |
```

causes bars 1 to 3 to be duplicated at this point. Numbering starts at bar 0 for the lead-in notes, and bar 1 for the first full bar. The notation

```
\ -1 |
```

causes a duplicate of the previous bar to be inserted (current position minus one), and

```
\ -4, -1 |
```

inserts copies of the preceding four bars. The notation

```
\ 'Legato' + 2, 'Legato' + 4
```

* This bar number must already have been entered, i.e. we must be currently positioned at bar number 5 or beyond.

duplicates source from the bar two beyond the most recent which contains the text 'Legato' to the bar four beyond it. Such a bar must occur earlier in the current “.MS” to “.ME” section. Only complete bars can be copied.

It is unwise to use absolute bar numbers in cases where an extra bar could perhaps be added or deleted at a later stage. Relative bar references (using the -4 or 'Legato' notations) are safer in such circumstances. References to bars not yet encountered are unacceptable.

Changes of signature

The notation

```
\ timesig = 3 4.
```

occurring at the start of a bar resets the time signature, default note length and bar length on the fly to a new value, causing the new value of the time signature to be printed on the staff. The input and/or output keys can be reset similarly using, for example

```
\ key = g<.
```

to reset both input and output key, or

```
\ keyout = d.
```

to reset only the output key. If such a change alters the pitch of the output key, the new key signature will be printed on the staff at that point; if it is used to change only the default note length or octave on input, no key signature is printed.

The notation

```
\ barno = 25.
```

moves the source to the start of bar 25; if the specified bar number is ahead of the current bar, any intervening bars are filled with rests. If the bar number given is less than the current bar, it is assumed that a further part is being added; see multi-part music below for details.

The notation

```
\ sticks = u.
```

causes the sticks of all following notes to be forced upwards until cancelled by either

```
\ sticks = d.
```

to send sticks downwards, or

```
\ sticks = x.
```

to leave sticks free to move in either direction.

Any of the above items can be combined as in a normal header, separated by semi-colons and terminated by a full-stop, as in

```
\ sticks = d; keyout = d.
```

With the exception of “\ sticks =”, the above notations starting with the escape character “\” are not guaranteed except when used at the start of a bar.

Controlling the number of bars on a staff

The notation

```
\ bps = 4.
```

sets the current “bars-per-staff” value to the given integer. The system will attempt to fit the given number of bars onto each staff. Within one staff, as far as is possible, the physical width of the bars will be proportional to their musical length. The “bps” facility can be used to lay out varying specified numbers of bars on each staff. This can however be achieved by a further facility; following any barline, the header item

```
\ endstave.
```

can be added.

If a group of bars finishing with an *endstave* is being copied, but the copied section is not required to end the stave, the entry

```
\ continue.
```

overrides the copied *endstave*.

Controlling the width

If, say, music is being printed at 8 bars per stave, but there are 4 bars left over at the end, those bars would normally be spread to fit the full width of the page. To reduce their width, use the inserted header

```
\ endstave; width = 4.5i.
```

to terminate the last full staff. This will decrease the width of the last staff to 4.5 inches. To reset back to the original width, use

```
\ width = 0.
```

Source from a separate file

The “.MS” line can be replaced by

```
..MS < filename
```

causing music input to be taken from the named file. The file would normally start with a “.MS” line. This enables music source to be easily tested before insertion into the main text. The line can be extended by header items terminated by a full stop. For example

```
..MS < filename keyout = b.
```

will cause the music in the named file to be printed in the key of B. Any header items given on the “.MS < file” line and following the filename are read **after** the first heading of the file which is being read. Thus if the named file contains a piece of music which is required to be printed in several keys, or in several different layouts, this can usually be done with extra header items in this way. The use of the escape character at the end of a line enables several header items to be given, as in

```
..MS < file keyout = g↑; rtitle = "arranged Foxley"; \  
bps = 4.
```

which specifies a new output key, right-hand title and bars-per-stave setting. The included file may start with any text before its first “.MS” line; this text is passed straight to the output by the preprocessor.

Output key

The facility to specify the output key independently of the input key takes account of the fact that a natural in one key may become a sharp in another. A piece may be printed in a key other than that in which it is entered by using the “keyout = ...” facility in the heading of the piece. To print a particular piece in two distinct keys, first in the key in which it was input, and then in a second key, store the input

```
..MS  
title = "\fBGlenarry's March\fp";  
key = a; chords; autobeam; bars = 8.  
  
e< d< |  
c "A" a a b< c< | d "E7" b b e< d< |  
c "A" a a b< c< | d< "G" c=< b< a< g= e< d< |  
c "A" a a b< c< | d "D" b b "E7" e< d< |  
c "A" a< c< b "G" g=< b< | a> "A" |  
..ME
```

in a file, then use

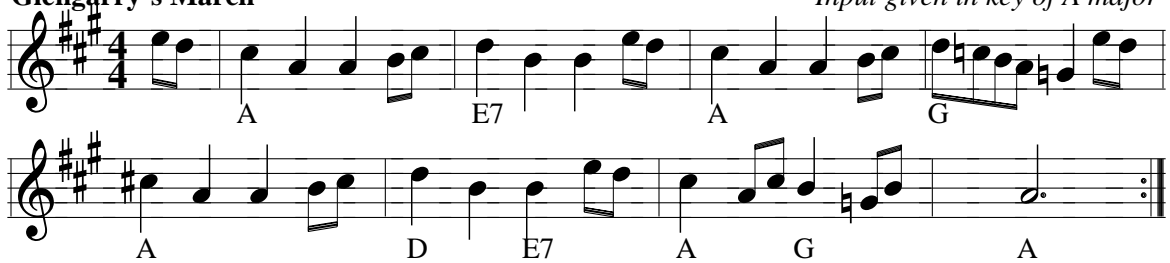
```
..MS < file
```

to produce the first copy, and

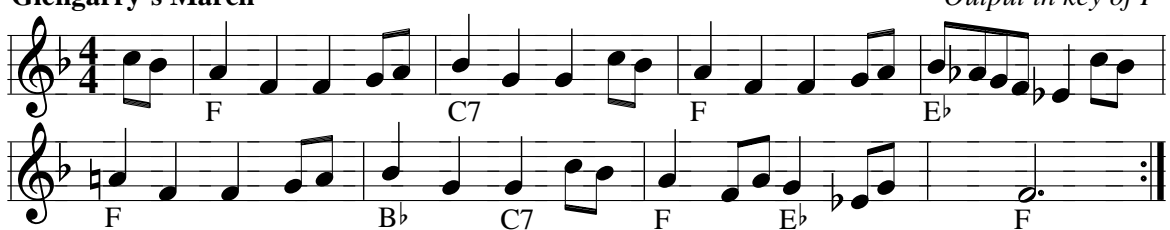
```
..MS < file rtitle = "\ fIOutput in key of F\ fP"; keyout = f.
```

for a second copy. This then appears as

Glengarry's March *Input given in key of A major*



Glengarry's March *Output in key of F*



An alternative technique for transposition is to use a simple

```
key = d;
```

header, and to follow it with a header entry such as

```
transpose = 2;
```

This will cause all output to be printed two sharps up from the output key currently set, following all output key changes during the piece. Negative values, of course, imply less sharps or more flats. This is particularly useful when producing scores for transposing instruments.

In transpositions, single sharps, naturals and flats are printed correctly relative to the new key. However, double sharps and flats (arising from, for example, an