Projektarbeit

MusicTEX-Konverter MID2TEX

Andreas Köhler

Betreuer: Dr.W.Riedel und Dr.J.Anders

12.7.1996

Herzlichen Dank den Projektbetreuern für die freundliche Unterstützung

Inhaltsverzeichnis

1 Einleitung **3**

2 Vorbetrachtungen **4**
2.1 Grundelemente der Notenlehre 4
2.2 Die Fileformate 6
2.2.1 MIDI 6
2.2.2 MusicTEX 8

3 Implementation **12**
3.1 Entwurfsmethodik 12
3.2 Tools zur Entscheidungsfindung 13
3.3 Datenstruktur 13
3.4 Topologie 17
3.5 Wesentliche Funktionen und ihre Wirkungsweise 18
3.5.1 main 18
3.5.2 ins_dyn_event 19
3.5.3 split_notes 19
3.5.4 detect_accords 20
3.5.5 insert_notetimes_and_slurs 20
3.5.6 detect_note_value 20
3.5.7 detect_note_positions_with_signs 21
3.5.8 set_slur_regions 22
3.5.9 beam_detect 22
3.5.10 set_beam_regions 23
3.5.11 slur_geometry 23
3.5.12 detect_rests 24
3.5.13 beam_geometry 24
3.5.14 calculate_rest_positions 25
3.5.15 set_loudness_infos 26
3.5.16 write_texfile 27
3.6 Einflußnahme auf die Funktionalität 27

4 Validierung **29**

5 Analyse **30**
5.1 Speicherplatz und Zeitverhalten 30
5.2 Möglichkeiten und Grenzen 31

A Beispiele **33**
A.1 5.tex - MOZART MIDI-File 5.mid 33
A.2 elise.tex - zweiter Track von elise.mid 34
A.3 morley.tex - generiert aus morley.mid (MOZART) 38

B Programmquelltexte **44**
B.1 makefile 44
B.2 defines.h 44
B.3 dyn.h 45
B.4 liste.h 47
B.5 maintypes.h 47
B.6 midi.h 48
B.7 readmidi.h 50
B.8 texgen.h 51
B.9 dyn.c 52
B.10 liste.c 103
B.11 readmidi.c 105
B.12 texgen.c 127

Kapitel 1

Einleitung

Mittlerweile existiert eine nahezu unüberschaubare Menge von Fileformaten. Obwohl das Schlagwort Multimedia in aller Munde ist, bestimmen doch weitgehend unimediale Formate das Geschehen. So gibt es Filestandards aus den Bereichen Datenbanken, Tabellenkalkulation, Grafik, Video (auch multimedial mit Bild und Ton), Sound, ... Die isolierte Handhabung dieser Formate ist häufig unbefriedigend. Es entsteht der Wunsch nach Überführungstools. Ein Musiker, der spontan am Keyboard nach ansprechenden Melodien sucht, muß anschließend zu Papier und Bleistift greifen und in Form von Noten das Gefundene niederschreiben. Der Gedanke an Automatisierung liegt nahe. Ein Multimediaformat, das Notenbild und Klang vereint, existiert noch nicht. Somit muß der vom Keyboard erzeugte MIDI-Datenstrom in einen Dokumentenstandard konvertiert werden. Hierfür ist MusicTEX besonders gut geeignet. Dieses Makropaket für TEXbzw. LATEXbeinhaltet alle wesentlichen Elemente der Notenlehre. Das ASCII-Format bietet darüber hinaus die Möglichkeit, das automatisch generierte File nachträglich mit vergleichsweise niedrigem Aufwand zu editieren, um eigene Wünsche, wie z.B. spezielle Überschriften, einzubringen. Da MIDI sehr weit verbreitet ist und viele Konvertierungsprogramme MIDI verstehen, ist auch eine Visualisierung anderer Musikformate (RMI, MOD u.a.) über MIDI als Zwischenformat möglich.

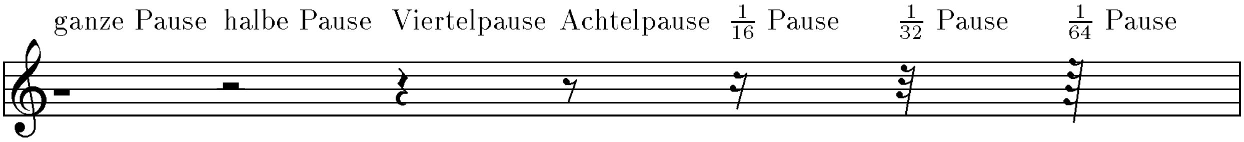
Kapitel 2

Vorbetrachtungen

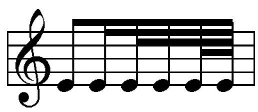
2.1 Grundelemente der Notenlehre

Folgende Notenwerte existieren:

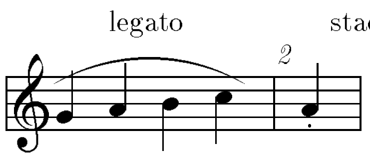
Analog gibt es Pausen bestimmter Zählzeit:



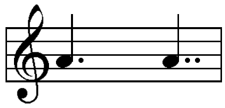
Stehen mehrere Fähnchennoten hintereinander, so können diese durch Balken, englisch: beams, verbunden werden.
Die Anzahl der Balken an einer Note ist identisch mit der Anzahl von Fähnchen, die dieser Note angehangen werden müßten.



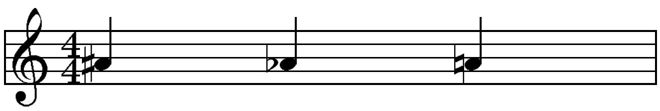
Wird eine Note nur extrem kurz (im Vergleich zu ihrer Zählzeit ca. ein Viertel) angeschlagen, so spricht man von staccato. Im Gegensatz dazu wird der fließende Übergang von Noten als legato bezeichnet.



Steht rechts neben der Note ein Punkt, so verlängert sich die Zählzeit um die Hälfte. Folgen zwei Punkte, so wird die Note um die Hälfte und nochmal um die Hälfte der Hälfte länger gespielt, insgesamt also 1.75 * (die eigentliche Zählzeit). Ein Beispiel hierfür zeigt das Bild:



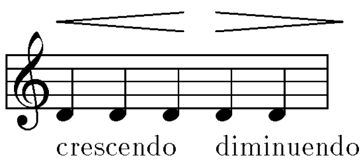
Bezüglich der Notenhöhe gibt es Halb- und Ganztöne. Das Voranstellen eines Kreuzes hebt die Notenhöhe um einen halben Ton an, ein b senkt ihn um einen Halbton. Mit dem Aufhebungszeichen kann eine gültige Halbtonverschiebung zurückgenommen werden. Die an die Tonart gebundenen Halbtöne besitzen globale Gültigkeit, daß heißt, sie gelten am Anfang jedes Taktabschnittes. Innerhalb des Taktabschnittes bewirken Kreuze, b-Zeichen bzw. Aufhebungszeichen die Abweichung von den globalen Einstellungen. Diese Zeichen haben jedoch lokalen Charakter. Sie wirken nur bis zum nächsten Taktstrich (barre).



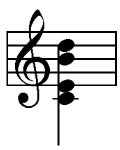
Lautstärken kann man sowohl statisch als auch dynamisch angeben. Die statischen Werte sind:

- fff: fortissimo possibile (so stark wie möglich)
- ff : fortissimo (sehr stark)
- f : forte (stark)
- mf : mezzoforte (mittelstark)
- mp : mezzopiano (mittelstark)
- p : piano (leise)
- pp : pianissimo (sehr leise)
- ppp: pianissimo possibile (so leise wie möglich)

Werden die Töne lauter, so handelt es sich um crescendo, werden sie leiser, um diminuendo.



Der zeitgleiche Zusammenklang mehrerer Töne wird als Akkord bezeichnet und wie eine Note behandelt. So besitz ein Viertelnotenakkord die Zählzeit ein Viertel. Der Akkord gehört zu nur einer Stimme, obwohl er selbst mehrstimmig ist.



Das soll für diesen kleinen Exkurs genügen. Weitergehende Informationen zur Notenlehre enthält [Lit2Schenk].

2.2 Die Fileformate

2.2.1 MIDI

1984 wurde auf Bestreben der beiden großen Keyboardhersteller Roland und Sequential Circuit ein Schnittstellenstandard für Synthesizer verabschiedet. Um weitestgehend systemunabhängig zu sein, wurde außerdem ein neues Dateiformat spezifiziert - das MIDI Fileformat [Lit1Born].

MIDI-Files beginnen mit einem Header-Chunk, dem sich mindestens ein (u.U. viele) Track-Chunk anschließt.

Header-Chunk	
Bytes	Bemerkung
4	ID (MThd)
4	Headerlänge
2	MIDI-Typ
2	Anzahl der Tracks
2	Zeitformat

Gültige MIDI-Typen sind:

- Typ 0: 1 Track
- Typ 1: mehrere unabhängige Tracks, die z.B. unterschiedliche Instrumente repräsentieren, aber gleichzeitig gespielt werden können
- Typ 2: mehrere Tracks, die komplette Abschnitte eines Musikstückes bzw. sogar verschiedene Stücke enthalten

Am üblichsten ist der Typ 1.

Zeitformat:

- Geschwindigkeit in Ticks pro Viertelnote - das höchstwertigste Bit 15 ist mit 0 belegt
- SMPTE-Time Code, der den Bezug der Töne zu Frames ermöglicht (hauptsächlich in der Film- und Videotechnik verwendet) - das höchstwertigste Bit 15 ist gesetzt

Track-Chunk	
Bytes	Bemerkung
4	ID (MTrk)
4	Tracklänge
n	Datenbereich

Der Datenbereich enthält eine Sequenz von MIDI-Events, denen jeweils eine Deltazeit vorangestellt ist. Damit ist der zeitliche Abstand des Events relativ zum vorherigen Event bestimmt. Die Deltazeit besitzt eine variable Länge. Jedes Zeitbyte beinhaltet 7 Datenbits. Das höchstwertigste Bit gibt an, ob weitere Zeitbytes folgen. In diesem Fall ist Bit7 gleich 1, beim letzen Byte 0.

Die gültigen MIDI-Events sind in den folgenden Tabellen dargestellt:

Kanal bezogene Events		
Code	Datenbytes	Bemerkung
0x8_	2	Note auf Kanal _ aus
0x9_	2	Note auf Kanal _ an
0xA_	2	Aftertouch (polyphon)
0xB_	2	Änderung von Steuerfunktionen
0xC_	1	Programmwechsel
0xD_	1	Aftertouch (monophon)
0xE_	2	Änderung der Tonhöhensteuerung

System - Events		
Code	Datenbytes	Bemerkung
0xF0	n	Integration neuer Funktionen - System Exclusive
0xF2	2	Song Pointer
0xF3	1	Song Select
0xF6	0	Tune-Request
0xF7	0	Ende einer System Exclusive Sequenz
0xF8	0	Zeitsynchronisation
0xFA	0	Rücksetzen des Song Position Pointer - Start
0xFB	0	Fortsetzung nach nächsten 0xF8 Event
0xFC	0	Unterbrechung der laufenden Sequenz - Stop
		- der Song Position Pointer bleibt unverändert
0xFE	0	Verbindungscheck zum MIDI-System
0xFF	0	System - Reset

Meta - Events		
Code	Datenbytes	Bemerkung
0xFF0002	1	Numerierung von Pattern oder Tracks
0xFF01	n	Speicherung von Texten
0xFF02	n	Copyright Message
0xFF03	n	Zuordnung von Namen für Sequenzen bzw. Tracks
0xFF04	n	Instrumentennamen
0xFF05	n	Song Text
0xFF06	n	Marker setzen
0xFF07	n	Cue-Point, für Vertonung von Videos und Filmen
0xFF2001	1	Bezug der folgenden Meta-Events zu einem Kanal
0xFF2F00	0	Trackende
0xFF5103	3	Setze Tempo
0xFF5405	5	SMPTE - Offset
0xFF5804	4	Taktart
0xFF5902	2	Tonart
0xFF7F	n	systemspezifische Informationsübertragung

2.2.2 MusicTEX

Analog der Notenlehre ist der Umfang vom Makropaket MusicTEX enorm groß. Deshalb beschränke ich mich im folgenden auf die wesentlichsten der von mir im Programm verwendeten MusicTEX-Konstrukte.

Am Anfang eines jeden MusicTEX-Files müssen einige \input-Anweisungen stehen (musicnft, musictex und wenn vorhanden musicadd). Mit \musicsize=16 kann die Grösse der Notenzeilen festgesetzt werden. Gültig sind die Werte 16 und 20. \facteur-Anweisungen dienen der Angabe von Abständen. Die folgenden Erklärungen sollen anhand von Beispielen vorgenommen werden. Wesentliche Kopf- und Rahmenelemente sind:

TEX-Kommandosequenz	Erklärung
\nbinstruments{2}	Es werden 2 Instrumente genutzt.
\instrumentii{Piano}	Das zweite Instrument ist ein Klavier.
\nbporteesiv=1	Instrument 4 besteht aus einem Staff.
\cleftoksv={6000}	Der unterste Staff von Instrument 5 ist ein Bass-Staff, die folgenden sind Violine-Staffs.
\generalmeter{\meterfrac{2}{4}}	Alle Instrumente mit ihren Staffs stehen im $\frac{2}{4}$ Takt.
\signaturegenerale{0}	Es handelt sich um eine vorzeichenlose Tonart. Sie gilt für alle Instrumente und deren Staffs.
\normal	normale Notengröße
\debutmorceau	Notenbeginn
\finmorceau	Ende des Musikstückes
\suspmorceau	Unterbrechung des Musikstückes
\reprmorceau	Fortsetzung des Musikstückes, das durch \suspmorceau unterbrochen worden ist

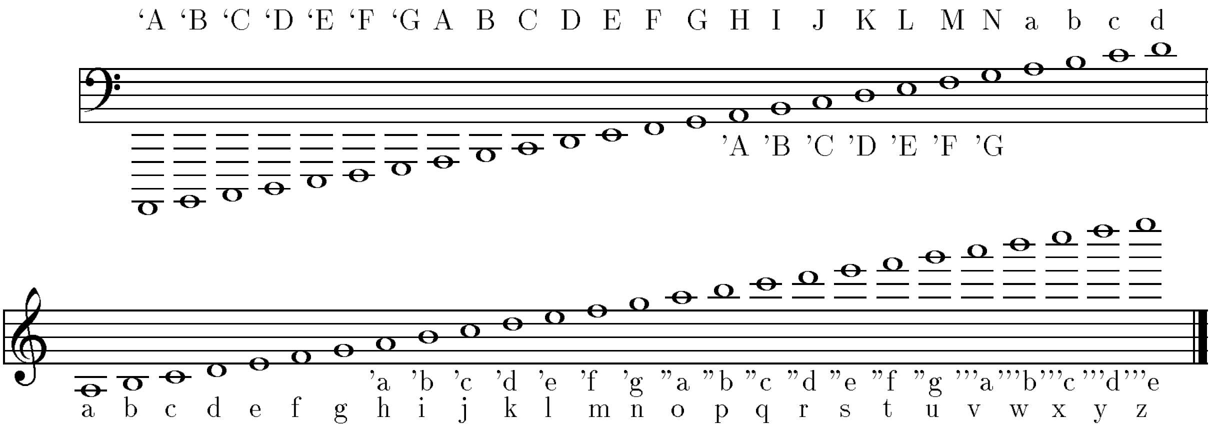
Im folgenden sind Kommandos des Notenteiles aufgelistet:

TEX-Kommandosequenz	Erklärung
\Notes	Beginn einer Folge von Noten
\enotes	Ende einer Folge von Noten
	(gleichzeitige Noten sollten in einer Sequenz zwischen
	\Notes und \enotes stehen!)
\NOTes	analog \Notes, bloß mit größerem Notenabstand
\zcharnote {e}{\qp}	zeichnet auf Position e eine $\frac{1}{4}$ Pause
\barre	erzeugt einen Taktstrich
\wh a	ganze Note auf Position a
\hu b	halbe Note mit Stiel nach oben auf Position b
\hl b	halbe Note mit Stiel nach unten auf Position b
\qu b	Viertelnote mit Stiel nach oben auf Position b
\cu b	Achtelnote mit Stiel nach oben auf Position b
\ccu b	$\frac{1}{16}$ Note auf Position b mit Stiel nach oben
\cccu b	analog $\frac{1}{32}$ Note
\ccccu b	analog $\frac{1}{64}$ Note
\qu {=c}	Viertelnote mit Aufhebungszeichen in Höhe von c
\qu {_c}	Viertelnote mit b als Vorzeichen
\qu {^c}	Viertelnote mit Kreuz als Vorzeichen
\pt h\cl h	auf Position h befindet sich eine punktierte
	$\frac{1}{8}$ Note mit Stiel nach unten.
\ppt h\cl h	analog, bloß doppelt punktiert
\pause	ganze Pause
\hpause	halbe Pause
\qp	Viertelpause
\ds	Achtelpause
\qs	$\frac{1}{16}$ Pause
\hs	$\frac{1}{32}$ Pause
\qqs	$\frac{1}{64}$ Pause
\lst g	Note mit Position g ist Staccatonote.
	Die Punkt wird unter die Note gezeichnet.
\ust g	analog, bloß mit Punkt über der Note

Etwas komplexere Kommandosequenzen:

`\zq {=f}\zq i\qu k`	Akkorde werden angegeben, indem die Noten von der kleinsten bis zur höchsten aufgezählt werden. Sie beginnen mit \zq, falls Viertelnotenköpfe gezeichnet werden sollen, mit \zh, falls leere Notenköpfe (halbe Noten) an den Akkord herangehangen werden sollen. Dann folgt die tiefste Note des Akkords (hier mit Aufhebungszeichen). Dieses Kommando schreibt man auch für die mittleren Noten. Der letzten Note ist \qu vorangestellt. Hiermit wird gesagt, daß der Viertelkopfnotenakkord mit einem Stiel zu versehen ist, der nach oben (upper) geht. Es folgt die Notenposition.
`\itenu4p ... \tten 4`	Legatobogen (hier als tenuto); beginnt mit \iten gefolgt von u (upper) oder l (lower). Als nächstes muß eine Nummer angegeben werden (0 bis 6, bei musicadd bis 9). p ist eine Positionsangabe. Beendet wird legato durch \tten und Angabe der Nummer.
`\ibu0f0\qh0{cge}\tbu0\qh0g`	Es handelt sich um den Code eines einfachen Beams aus Achtelnoten. Er beginnt mit \i (initiate). Es folgen ein bis vier bs, je nach Balkenzahl (bei $\frac{1}{64}$ Noten 4). Als nächstes wird wieder die Orientierung angegeben (hier upper). Es folgt eine eindeutige Nummer. Der Balken bzw. die Balken werden auf Höhe des f gezeichnet. Als letztes Initialisierungselement muß der Anstieg des Beams spezifiziert werden (hier 0). Die am Beam beteiligten Noten (bis auf die letzte) folgen auf \q. Außerdem muß angegeben werden, ob die Note am Beam hängt (h) oder auf ihm steht (b). Im Beispiel sind gleich 3 hängende Noten mit {} zusammengefaßt. Beendet wird ein Beam durch \tb gefolgt von der Orientierung und der Nummer. Es schließt sich die letzte Note an. **Achtung: Es gibt für Legatobögen und Beams bloß einen Nummernraum!** Möchte man nicht alle Balken terminieren, so gibt man z.B. \tbbb an. Dadurch werden der $\frac{1}{64}$ und der $\frac{1}{32}$ Balken beendet. Kommen Balken hinzu, so schreibt man z.B. \nbb (es kommt der $\frac{1}{16}$ Balken hinzu).

Die Notenpositionen sind zum Verständnis eines MusicTEX-Files außerordentlich wichtig. Aus den anschließenden Notenzeilen werden sie ersichtlich.



Diese Darstellung ist der zu [Sw1Taupin] gehörenden Dokumentation entnommen.

Kapitel 3

Implementation

3.1 Entwurfsmethodik

Der Entstehung des MIDI nach TEXKonverters läßt sich kein Entwurfsmodell eindeutig zuordnen. Das Prinzip entsprach im wesentlichen dem top-down Entwurf, daß heißt der schrittweisen Zerlegung des Problems in Teilprobleme. Nach intensiver Studie der Fileformate mußte über die Strukturierung durch Beantwortung der folgenden Fragen entschieden werden:

1. Wie gelangt man vom MIDI-Format zum MusicTEX-Format?
   (a) Welche Informationen sind wesentlich, welche können verworfen werden?
   (b) Welche Abhängigkeiten existieren zwischen bestimmten MIDI-Events und entsprechenden MusicTEX-Kommandos?
   (c) Gibt es eine zu favorisierende Datenstruktur?
   (d) Wie sollte diese beschaffen sein, und welchen Anforderungen muß sie gerecht werden?
   (e) In welcher Reihenfolge sind die Informationen zu bearbeiten?
   (f) Wie ist das Programm funktionell zu strukturieren?
2. Wie kann ich eine spezielle Funktion umsetzen?
   (a) Welchen Algorithmus verwende ich?
   (b) Wo muß ich aus Gründen des Aufwandes abstrahieren?
   (c) Welchem Niveau sollen die errechneten Daten entsprechen?
   (d) Wie ist zwischen Zeit- und Speicherplatzaufwand abzuwägen?
3. Welchen optischen Ansprüchen soll das MusicTEX-Dokument genügen?

Mit der Beantwortung dieser Fragen ergab sich ein Konzept, das im schrittweisen Ausprogrammieren der einzelnen Prozeduren mündete. Die Implementierung der einzelnen Funktionen entsprach am ehesten der „code and fix“ Methode, die zwar verrufen, auf niedrigem Level aber unumgänglich ist. Zur Validierung („fix“) diente die Funktion debug_out mit deren Hilfe der jeweils aktuelle Inhalt der Datenstruktur ausgelesen werden konnte.
Das eigentliche Problem bestand darin, die Informationen des MIDI-Files (a) in die des TEX-Files (b) zu überführen.

P: a $\rightarrow$ b

Die Abbildung P als auch b selbst sind mehrdeutig, da die Überführung nichtdeterministischen Charakter hat. Ziel war, ein übliches und aus Sicht des Nutzers akzeptierbares Ergebnis b zu generieren. Der Umgang mit solch aufgeweichten Begriffen setzt sicher etwas Kreativität voraus.

3.2 Tools zur Entscheidungsfindung

Ausgangspunkt für die Programmierung war das Programm „readmidi" [Sw2Anders] , dessen Funktionalität lediglich durch weitere MIDI-Event Behandlungsroutinen erweitert werden mußte. Tools wie MOZART [Sw3Web] (Windows 3.1) und Rosegarden [Sw4CG] (IRIX) erlauben die Eingabe von Noten und das Ablegen der Daten in MIDI-Files. Damit war es möglich, die Umsetzung bestimmter Notenelemente in MIDI-Events nachzuvollziehen. Die durchaus unterschiedliche Vorgehensweise beider Programme begrenzte den möglichen Spielraum.
Dazu ein Beispiel:

> Rosegarden setzt Noten, die weder legato noch staccato sind, so um, daß sie über ihre volle Zählzeit gespielt werden. MOZART bricht solche Noten eher ab, das heißt, innerhalb einer Melodie endet die Note bevor die sich anschließende beginnt. Bei MOZART werden nur Legatonoten über die volle Zeit gespielt, also erst mit dem Ende ihrer Zählzeit auf einem der 16 physischen MIDI-Kanäle ausgeschaltet.

Da die letztere Vorgehensweise offensichtlich am ehesten legato entspricht, orientiert sich dieses Projekt näher an MOZART als an Rosegarden.

3.3 Datenstruktur

Sequentielles Ein- und sofortiges Wiederauslesen der Daten ist aufgrund der Komplexität der Aufgabe nicht tragbar. Die deshalb erforderliche Datenstruktur muß geeignet sein hinsichtlich:

1. der Speicherung variabel großer Datenmengen
2. der Möglichkeiten für die Verarbeitung der Daten
3. des Ablegens unterschiedlichster Informationen

Da die Tracks (Stimmen) der MIDI-Files vom Typ 1 (üblichster Typ) unabhängig sind, empfiehlt sich eine autonome Bearbeitung. Die zeitliche Ordnung der MIDI-Events innerhalb von Tracks muß erhalten bleiben. Für all diese Restriktionen ist eine einfach verkettete Liste pro Track geeignet. Die interne Repräsentation eines ein-Track MIDI-Files könnte folgendermaßen aussehen:

begin[0] $\rightarrow$ $\boxed{\text{bind}}$ $\rightarrow$ $\boxed{\text{bind}}$ $\rightarrow$ ... $\rightarrow$ $\boxed{\text{bind}}$ $\rightarrow$ nil

begin[1] $\rightarrow$ nil

.

.

.

begin[MAXTRACKS -1] $\rightarrow$ nil

Dabei ist bind folgendermaßen definiert:

```
struct bind {
 char marker;
 unsigned long time;
 struct bind *next;
 void *data;
};
```

Der Datenbereich, auf den data zeigt, ist variabel, durch marker jedoch eindeutig bestimmt. So gibt es:

- Lautstärkeinformationen:

  ```
  struct Loudness { /* marker == ´p´ */
   int statics, dynamics;
  };
  ```

  Eines der beiden Bytes ist immer 0, das jeweils andere ungleich 0. Folgende Tabellen zeigen die möglichen Belegungen und die entsprechende Bedeutung:

statics	
Wert	Bemerkung
1	ppp
2	pp
3	p
4	mp
5	mf
6	f
7	ff
8	fff

dynamics	
Wert	Bemerkung
1	kurzes crescendo
2	mittleres crescendo
3	langes crescendo
4	sehr langes crescendo
5	kurzes diminuendo
6	mittleres diminuendo
7	langes diminuendo
8	sehr langes diminuendo

- Tonartinformationen:

  ```
  struct KeySig { /* marker == ´k´ */
   short sign;
  };
  ```

Die Semantik von sign ist entsprechend der Spezifikation des `MIDI-Meta-Events` „Key Signature“:

- falls (Bit7 == 0), so handelt es sich um eine Kreuztonart, sonst B-Tonart
- die niederwertigsten 4 Bit geben die Anzahl der Vorzeichenstriche an

Die Kodierung wurde aus dem MIDI-Standard übernommen, da sie für mid2tex gut geeignet ist.

- Taktartinformationen:

```
struct TimeSig { /* marker == ´t´ */
 short nom, denom;
};
```

nom gibt den Zähler, denom den Nenner des Taktes an.

- Taktstrichinformationen:
  → besitzen keinen Datenteil (marker == 'b').

- Textinformationen:

```
struct Lyric { /* marker == ´l´ */
 char *text;
};
```

Die Speicherzellen für die Liedtextelemente müssen entsprechend der Textlänge erst allociert werden, da noch keine Textvariable definiert ist.

- Pauseninformationen:

```
struct Rest { /* marker == ´r´ */
 char position;
 int kind;
 char flg;
};
```

Ist flg gesetzt, so wird die Pause aus Platzgründen um 0.56*NOTESKIP nach rechts verschoben. Bit8 von position weist darauf hin, daß die Pause in die Baßregion geschrieben werden muß, Bit7 bedeutet Violineregion (das ist wichtig für Tracks, deren Spektrum über Baß und Violine reicht). Beide Bits zusammen dürfen nie gesetzt sein! Die niederwertigsten 6 Bit definieren den Index im Array „positions“, welches die Codes für alle unterstützten Positionsangaben der Notenelemente enthält. Die Belegung für kind ist im Folgenden dargestellt:

kind	
0	$\frac{1}{64}$ Pause
1	$\frac{1}{32}$ Pause
2	$\frac{1}{16}$ Pause
3	$\frac{1}{8}$ Pause
4	$\frac{1}{4}$ Pause
5	$\frac{1}{2}$ Pause
6	$\frac{1}{1}$ Pause

- Noteninformationen:

```
struct Note { /* marker == ´n´ */
 short octave, pitch, loud;
 unsigned long end;
 int duration;
 short flags;
 int beam_nr, slur_nr;
 char note_pos, slur_pos, beam_pos;
 char treated;
 char next_beamn_kind;
 struct Note *accord_next;
};
```

Die 2Byte-Variablen octave und pitch spezifizieren die zu Grunde liegende Frequenz. loud dient als Maß für die Lautstärke und ist somit Ausgangspunkt für die Berechnung statischer und dynamischer Lautstärkeinformationen. Noten besitzen nicht bloß eine Anfangszeit `((bind *)->time)` , sondern auch ein Ende. Diese Zeit wird im 4Byte Wert end abgelegt. duration gibt an, wie lange der Ton gespielt wird, ist also immer kleiner oder gleich der Differenz aus Endzeit und Anfangszeit. Legatobögen (slurs) und Notenbalken (beams) müssen durch eine Nummer eindeutig identifiziert werden können (beam_nr, slur_nr). Die 3 Positionsangaben note_pos, slur_pos und beam_pos enthalten auf den niederwertigsten 6 Stellen den Index für das Positionsarray (positions). Bei slur_pos und beam_pos zeigt Bit6 den Beginn eines Legatobogens bzw. eines Balkens an. Bit7 weißt auf einen oberen Balken/Legatobogen hin. Ist Bit7 gleich 0, so handelt es sich um einen unteren Balken/Legatobogen. Die beiden höchstwertigsten Bits von note_pos geben das Vorzeichen der Note an (00 keines, 01 Kreuz, 10 B, 11 Aufhebungszeichen). treated dient den bearbeitenden Funktionen, um Noten als bereits behandelt zu markieren. next_beamn_kind gibt bei Noten die „gebeamt“ sind an, wieviel Balken die nächste Note am jeweiligen beam hat. Das ist wichtig für die Ausgabe des MusicTEX-Files. Akkorde zählen wie Einzelnoten. Deshalb ist es unpraktikabel, alle Noten eines Akkords in der einfach verketteten Liste zu belassen, wenn der Akkord erkannt wurde. Die Noten eines Akkords werden über accord_next verpointert. Offen ist noch die Belegung von Flags, die aus Gründen der Komplexität in einer Tabelle dargestellt ist:

flags	
Bits	Bedeutung
0	Note ist punktiert
1	Note ist doppelt punktiert
2	Note ist staccato
3-5	Zählzeit (Bit5Bit4Bit3: 000 $\frac{1}{64}$, 001 $\frac{1}{32}$, 010 $\frac{1}{16}$, 011 $\frac{1}{8}$, 100 $\frac{1}{4}$, 101 $\frac{1}{2}$, 110 $\frac{1}{1}$)
6	letzte Beamnote
7	letzte Slurnote
8	Note muß in die Violineregion
9	Note muß in die Baßregion
10	Fähnchen nach unten (nicht gesetzt $\rightarrow$ nach oben)
11-14	Bit14Bit13Bit12Bit11: Wert des Anstieges eines Beams
15	Orientierung des Beams (1 $\rightarrow$ fallend, 0 $\rightarrow$ steigend)

Über die zeitliche Ordnung hinaus existiert noch eine Ordnung über zeitgleichen Events: Taktstrich (Barre) $\rightarrow$ Tonart $\rightarrow$ Takt $\rightarrow$ Lyrik $\rightarrow$ Lautstärke $\rightarrow$ Pause $\rightarrow$ Note

Beispiel dazu (schematisch dargestellt):

Zeit:	n-3		n		n		n+10		
... $\rightarrow$	$\boxed{\text{marker: ...}}$	$\rightarrow$	$\boxed{\text{marker: 'r'}}$	$\rightarrow$	$\boxed{\text{marker: 'n'}}$	$\rightarrow$	$\boxed{\text{marker: ...}}$	$\rightarrow$	...
	$\downarrow$		$\downarrow$		$\downarrow$		$\downarrow$		
	...		$\boxed{\text{Pause}}$		$\boxed{\text{Note}}$		...		

3.4 Topologie

Es existieren 4 Quelltextmodule:

- **readmidi.c** ... beinhaltet sämtliche Funktionen für das Einlesen des MIDI-Files. In dieses Modul ist außerdem die main-Funktion integriert.

  Zum Lesen des MIDI-Files wird aus main heraus die Funktion read_track entsprechend der Anzahl von Tracks bzw. so oft wie nötig (falls nur ein einziger Track in MusicTEX-Code überführt werden soll) gerufen. read_track liest den Track-Kopf und springt im Anschluß daran die Funktion ReadEvent an, solange bis von dieser ein EndOfTrack-Event erkannt wurde. Die Funktion ReadEvent liest zuerst die dem Ereignis vorangestellte Deltazeit und dann den eigentlichen Eventcode. Entsprechend dieses Codes wird eine Behandlungsroutine angesprungen (z.B. ReadMidiMetaEvent), die dann u.U. wiederum eine von vielen Behandlungsroutinen anspringt (z.B. ReadMidiInstrumentName).

- **liste.c** Da Noten erst nach dem Ausschalten in die einfach verkettete Liste, die den Kern der Datenstruktur bildet, aufgenommen werden können, müssen aktive Noten zwischengespeichert werden. Auch dazu wird eine einfach verkettete Liste verwendet. Funktionen für den Zugriff auf diese temporäre Struktur sind im Modul liste.c zusammengefaßt.

- **dyn.c** ...ist das größte Modul und bildet das eigentliche Kernstück. Alle Funktionen, die die Hauptdatenstruktur bearbeiten, sind hier zu finden. Elemente, die unmittelbar aus bestimmten MIDI-Events resultieren, können mit ins_dyn_event eingefügt werden. debug_out ermöglicht das Auslesen der Liste im Debug-Modus (wird durch Definition von DEBUG aktiviert). Die zentralen Bearbeitungsfunktionen, wie z.B. insert_notetimes_and_slurs, werden unmittelbar durch main gerufen.

- **texgen.c** ...enthält Routinen zum Auslesen des MusicTEX-Files. Kernstück bildet die Funktion write_texfile.

3.5 Wesentliche Funktionen und ihre Wirkungsweise

Im folgenden werden konkrete Stellen im Code durch eindeutige Marken (in Form von Kommentaren) referenziert. Das soll dem besseren Verständnis dienen.

3.5.1 main

Für den Aufruf des Programmes gibt es zwei Möglichkeiten:

```
<programmname> [-<tracknummer>] <midifile> [<texfile>]
<programmname> -c <midifile>
```

-c steht für „count only“ und startet das Programm in einem Modus, welcher lediglich die Anzahl der Tracks ausgibt, jedoch kein TEX-File erzeugt. Falls beim Aufruf 1 eine Tracknummer spezifiziert wird, so wird nur dieser eine Track nach MusicTEX-Code überführt, sonst alle (im Rahmen des Möglichen).
main beginnt mit der Auswertung der Argumente:

mainmark1 Falls in der Kommandozeile kein TEX-Filename angegeben und -c nicht gesetzt wurde, so wird ein Name für das MusicTEX-File benötigt. Dazu wird die MIDI-File-Endung durch eine TEX-Endung substituiert. Falls das MIDI-File mit einem Großbuchstaben beginnt, wird .TEX , sonst .tex angehangen. Der Name wird zunächst auf der globalen Variablen texfilename abgelegt.

mainmark2 Nachdem das MIDI-File geöffnet wurde, kann mit dem Einlesen des Kopfes begonnen werden. Zum Abspeichern der Daten dient der struct head (siehe readmidi.h bzw. midi.h).

mainmark3 Falls das '-c'-Argument gesetzt war, so kann das Programm sofort beendet und die Anzahl der Tracks ausgegeben werden.

mainmark4 Alle Events dürfen nur zu speziellen durch den one64tel-Wert teilbaren Zeiten auftreten. Das wird später gewährleisten, daß die detect_rests-Funktion korrekte Zählzeiten für die Takte erzeugt. Beispiel: Ist eine $\frac{1}{64}$ Note genau 6 Einheiten lang, so dürfen Ereignisse nur zu den Zeitpunkten 0, 6, 12 usw. auftreten. Diese diskreten Zeiten liefert die Funktion get_valid_time, die immer dann gerufen wird, wenn ein Datum mit möglicherweise inkorrekter Anfangszeit in die Liste durch ins_dyn_event aufgenommen werden soll.

mainmark5 Es schließt sich die Initialisierung weiterer Variablen an. nom0 und key0 bringen zum Ausdruck, daß bisher weder ein Takt- noch ein Tonartevent gelesen wurde. Das ist wichtig für MIDI-Files, deren Takt- und Tonartinformation im ersten, einem eigentlich leeren Track, stehen. nom0, denom0 und key0 dienen dazu, die Daten des ersten Tracks aufzunehmen und, falls der nächste Track keine Takt- und Tonartinformation besitzt, die gespeicherten Werte auf diesen Track anzuwenden. Gilt zu diesem Zeitpunkt noch (nom0 == 0) bzw. (key0 == 0), so sind für das jeweilige Event die default-Werte $\frac{4}{4}$ Takt und C-DUR bzw. A-MOLL gültig.

mainmark6 Wurde auf der Kommandozeile kein spezieller Track spezifiziert, muß jeder Track eingelesen und verarbeitet werden. Ist dies nicht der Fall, so wird bis zum angegebenen Track gelesen, und im Anschluß daran werden nur die Daten des einen interessierenden Tracks nach MusicTEX-Code überführt.

mainmark7 Abschließend muß das TEX-File geöffnet, geschrieben und wieder geschlossen werden.

3.5.2 ins_dyn_event

Die Funktion ins_dyn_event dient dem Einfügen von Noten, Taktart, Lyrik, Tonart und Taktstrichen in die dynamische Struktur. Andere Informationen, wie Pausen und Lautstärkeinformationen, können keinem MIDI-Event zugeordnet, sondern müssen durch umfangreiche Berechnungen erkannt werden.

insdmark1 Zuerst muß für das bindende Element Speicherplatz allociert werden. Im Anschluß daran werden marker und time gesetzt.

insdmark2 Je nach Ereignis müssen jetzt (mit Ausnahme der Barre 'b') noch Datenbereiche angefordert und, soweit möglich, belegt werden.

insdmark3 Ist das getan, so muß das Element entsprechend der definierten Ordnung in die Kette eingehangen werden (erst Zeitpunkt suchen, dann Rangfolge zeitgleicher Ereignisse beachten).

3.5.3 split_notes

Da im MIDI-File keine Restriktionen bzgl. der Dauer von Noten existieren, kann es vorkommen, daß manche Noten länger sind als 4*per_quarter (länger als eine ganze Note) oder einen Taktstrich überklingen. Diese Noten müssen in mehrere aufgespalten werden, um die korrekte Arbeitsweise der sich anschließenden Funktionen gewährleisten zu können.

splnmark1 Die Zeigervariablen dynhelp3 und dynhelp2 dienen dazu, die dynamische Struktur zu durchlaufen. Sie werden mit dem entsprechenden Element von begin initialisiert.

splnmark2 dynhelp3 läuft immer bis zum nächsten Taktstrich, dessen Zeit wesentlich ist. dynhelp2 dient zum Prüfen der Noten, was im kritischen Fall ein Aufspliten des einen in zwei Elemente nach sich zieht.

Der Algorithmus der Notenüberprüfung mit eventueller Trennung von Noten könnte sicherlich auch mit relativ geringem Aufwand rekursiv implementiert werden.

3.5.4 detect_accords

deacmark1 Am Anfang wird ein Element des Feldes akt durch die entsprechende Eintragung in begin initialisiert. Durch ständiges Weiterschieben dieses Zeigers wird später nil erreicht und die Funktion abgebrochen.

deacmark2 Für die Berechnung von Akkorden sind bloß Noten von Interesse. Gilt (reg == 0), so wird der Track über zwei Notenzeilen (Baß und Violine) gezeichnet. Es muß ausgeschlossen sein, daß ein Akkord aus Noten unterschiedlicher Bereiche gebildet wird. Deshalb wird attention auf 1 gesetzt, falls die aktuelle Note eine Baßnote ist und (reg == 0) gilt. Damit kann später getestet werden, ob das Anhängen einer Violinennote zulässig ist.

deacmark3 Alle Nachfolger der aktuellen Note, die die gleiche Anfangszeit haben, werden jetzt unter Zuhilfenahme von attention und der Notendauer geprüft. Kriterium für Akkordnoten ist ein gleicher duration-Wert. Beim Erkennen eines Akkordes werden seine Noten über accord_next verpointert. Aufgrund der Vorsortierung von Noten nach der Frequenz ist gewährleistet, daß ((NOTENZEIGER → (note_pos & 0x3f)) kleiner als (NOTENZEIGER → accord_next → (note_pos & 0x3f))) gilt. Die frei werdenden bind-Elemente müssen aus der Liste entfernt und dealloziert werden.

3.5.5 insert_notetimes_and_slurs

Diese Funktion ist sehr einfach gehalten. Per Zeigervariable wird die Liste durchlaufen und für jede erkannte Note die Funktion detect_note_value gerufen.

3.5.6 detect_note_value

dnvmark1 Bei jeder Note wird zunächst das Flag für Legatoende gesetzt. Mittels einer while-Schleife wird der erste Notenwert gesucht, der größer oder gleich der Notendauer der aktuellen Note ist. Dies geschieht unter Zuhilfenahme des Feldes faktor und der Konstante NORMFAKTOR, die eine Normierung auf ganze Zahlenwerte erzwingen. Damit sind die Werte problemlos vergleichbar. Sollte die sich anschließende Warnung ausgegeben werden, so muß das bedenklich stimmen, da zu lange Noten bereits in split_notes beseitigt wurden.

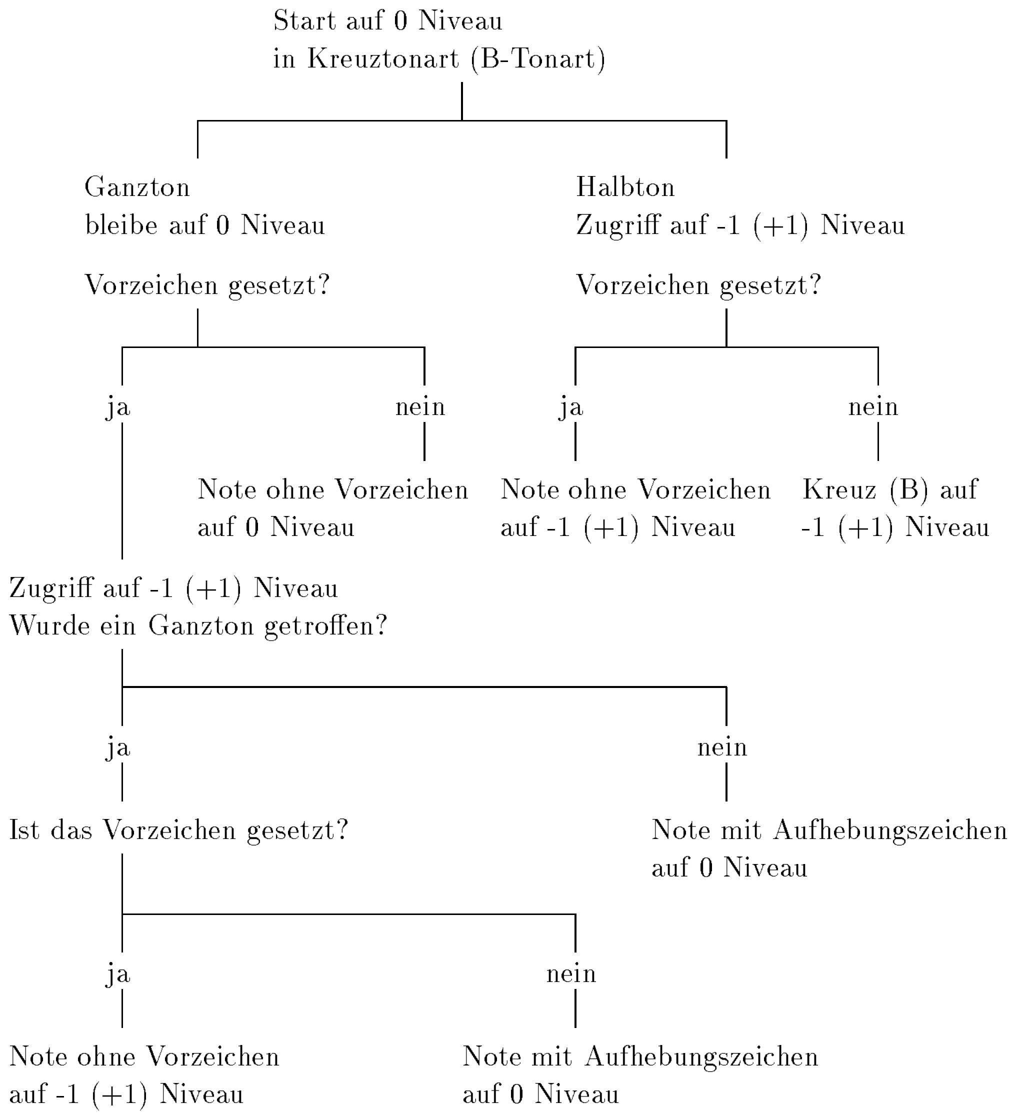
dnvmark2 Wird eine Note über ihre volle Zählzeit gespielt, so kann es sich um legato oder staccato handeln. Die Funktion note_with_time hat zwei Rückgabeparameter, die einer besonderen Erklärung bedürfen, desicion und ok. Desicion ist genau dann 1, wenn es eine Note gibt, die zu dem Zeitpunkt beginnt, an dem die zu untersuchende Note endet. ok gibt an, ob die gefundene Note in der richtigen Region (Violine oder Baß) liegt.

dnvmark3 Falls sich einer Note keine zweite unmittelbar anschließt, die Zählzeit ausreichend kurz ist und bei Multiplikation mit 4 der nächste Taktstrich nicht überschritten wird, so wird die Note als Staccatonote definiert. Diese Vorgehensweise ist sicher etwas starr, entspricht jedoch exakt der Umsetzung von Staccatonote bei MOZART.

dnvmark4 Wurde eine unmittelbar folgende Note innerhalb der richtigen Region gefunden, so handelt es sich um legato. In allen noch nicht betrachteten Fällen wird die Note als „normal“ interpretiert. Bei allen nicht Legatonoten muß am Funktionsende das Legato-Endeflag wieder gelöscht werden.

3.5.7 detect_note_positions_with_signs

Bei Noten muß man unterscheiden zwischen der eigentlichen Frequenz und der Position, auf der sie gezeichnet werden. Die Zuordnung ist mehrdeutig, da durch Voranstellen eines Kreuzes oder eines Bs die der Position zugeordnete Frequenz um einen Halbton nach oben bzw. nach unten verschoben werden kann. Um diese Mehrdeutigkeit einzugrenzen, sind hier Bs innerhalb von Kreuztonarten und Kreuze innerhalb von B-Tonarten verboten. Desweiteren wird stets versucht, zuerst ohne Notenvorzeichen auszukommen. Falls das nicht gelingt, muß ein Kreuz, ein B oder ein Aufhebungszeichen gesetzt werden. Der Entscheidungsbaum ist im Folgenden dargestellt:



Dabei charakterisiert das 0 Niveau die eigentliche Notenfrequenz. -1 (+1) Niveau referenziert die um einen halben Ton tiefer (höher) liegende Frequenz. Die Notenausgabeposition entspricht immer einem Ganzton, der mit Hilfe von pitch_to_pos der eigentlichen Position zugeordnet wird.

dnpsmark1 Ein Feldelement von akt dient wiederum zum Durchlaufen der Liste. cross stellt eine Art Flag für die Kreuztonart dar. Gilt (cross == 0), so handelt es sich um eine B-Tonart. Im Folgenden werden 3 Felder eine besondere Rolle spielen. crosses und bs geben die Reihenfolge der Vorzeichenaktivierung für die Notenpositionen an. So kann bei gegebener Vorzeichenstrichzahl abgelesen werden, auf welchen Positionen Vorzeichen global (entsprechend der Tonart) aktiv sind. Das Feld process enthält Informationen zu Ganz- und Halbtönen (Bit6), globalen Vorzeichen (Bit7) und zu den jeweils aktuellen Vorzeichen (Bit0 für die Oktave 0, ..., Bit5 für die Oktave 5). Indiziert wird process durch den Pitchwert der jeweiligen Note.

dnpsmark2 Innerhalb der folgenden do-while Schleife werden alle Noten eines Akkordes (eventuell auch bloß eine einzige) behandelt. Die Verfahrensweise kann der Baumdarstellung entnommen werden. Das Feld pitchtopos ist für die Errechnung der Notenposition wichtig.

dnpsmark3 Ein Tonartereignis zieht die Veränderung der in process abgelegten Vorzeichenaktivierungen nach sich. Gegebenenfalls muß auch cross modifiziert werden.

dnpsmark4 Taktstriche sind von Interesse, da hier die Gültigkeit der lokalen Vorzeichen endet. Bit7 muß auf die Bits 0 bis 5 für alle Pitchwerte kopiert werden.

3.5.8 set_slur_regions

Diese Funktion dient der Eintragung des Flags für Legatobeginn (0x40 von slur_pos) sowie der Festlegung einer Region. Das ist wichtig, da im Folgenden ausgeschlossen werden muß, daß Legatonoten des Violinenbereiches per Balken (beam) mit Baßnoten verbunden werden bzw. umgekehrt. Beams über mehrere Regionen sind zwar in MusicTEX möglich, ziehen aber ein paar unerwünschte Nebeneffekte nach sich.

3.5.9 beam_detect

bdmark1 Innerhalb einer while Schleife wird die gesamte Liste durchlaufen. Von Interesse sind bloß die Noten, die kleiner oder gleich der $\frac{1}{8}$ Note sind.

bdmark2 Mit dynhelp wird jetzt zum nächsten Taktstrich gesprungen. Die etwas ungewöhnliche Initialisierung von ok durch 8, 9 oder 10 erzwingt in note_with_time die Suche nach Fähnchennoten.

bdmark3 Falls zur Zeit not $\rightarrow$ end eine Fähnchennote gefunden wurde, diese in der richtigen Region ist und kein Taktstrich übersprungen wurde (ist bei den für dieses Projekt verwendeten Taktstrichen nicht möglich), so müssen noch weitere Tests durchgeführt werden. Die Note, an die eine andere „gebeamt" werden soll, muß das Legato-Endeflag gesetzt haben, falls es eine Legatonote ist. Die Note, welche angehangen werden soll, darf nur dann eine Legatonote sein, wenn sie die erste Note des Legato ist. Aufgrund beider Restriktionen kann ausgeschlossen werden, daß Legatobögen durch Balken bzw. durch die Stiele von Balkennoten geschnitten werden. Desweiteren darf die anzuhängende Note nicht bereits einem anderen Balken zugeordnet sein.

bdmark4 Genügt die durch note_with_time gefundene Note nicht den gesamten Anforderungen, so muß weitergesucht werden.

bdmark5 Falls innerhalb der letzten Bearbeitungsphase remsign auf Null gesetzt wurde, so können die Noten (not *) und (not2 *) durch einen Balken verbunden werden. Wurde zu (not *) noch keine Balkennummer vergeben, so wird dies jetzt getan. Dafür stellt global einen garantiert eindeutigen Wert zur Verfügung. Bei (not *) wird gegebenenfalls ein bereits aktiviertes Balkenendeflag gelöscht, bei (not2 *) wird dieses Flag gesetzt. Die Information über die Region muß bis zur letzten Balkennote immer weiter vererbt werden (mit hilfsvariable helpshort).

3.5.10 set_beam_regions

Die Funktion set_beam_regions ist in ihrer Arbeitweise der Funktion set_slur_regions ähnlich.

sbrmark1 Wird ein Balkenbeginn gefunden, (beam_nr && (!treated)), so muß der Beam bis zum Ende (flags & 0x40) gelesen werden. Dabei wird eine durchschnittliche Notenhöhe errechnet und der Balken auf Grundlage dieser einer speziellen Region (Baß oder Violine) zugeordnet, falls das nicht schon geschehen ist.

sbrmark2 Im nächsten Teil muß der entsprechende Listenabschnitt ein zweites mal durchlaufen werden, um die Region in alle Noten des Balkens eintragen zu können. Die hier praktizierte Methode ist sicher etwas unübersichtlich und unnötig zeitaufwendig. Es wäre theoretisch sauberer gewesen, eine Datenstruktur stack (mit den Operationen push, pop, init) zu implementieren. Im ersten Zyklus hätten die am Balken beteiligten Noten in den Keller gelegt werden können. Durch Leeren des stacks wäre es dann möglich gewesen, die Angabe der Region in sämtliche structs zu schreiben. Eine solche Vorgehensweise ist jedoch mit einem höheren Programmieraufwand verbunden.

3.5.11 slur_geometry

sgeomark1 Per while Schleife wird wiederum die gesamte Liste durchlaufen. Wird eine Note gefunden, für die eine Legatonummer, aber noch keine Legato-Positionsangabe existiert, so handelt es sich um die erste Note eines noch nicht bearbeiteten Legatobogens.

sgeomark2 Um zu erfahren, wann der Legatobogen beendet wird, muß jetzt in der Liste vorwärts gelaufen werden. Die Endezeit wird auf der lokalen vorzeichenlosen Variable endtime abgespeichert.

sgeomark3 Die Positionen der von Legatobeginn bis Legatoende liegenden Noten müssen in das Feld used eingetragen werden. Darüber hinaus werden die Positionen der höchsten (maximum) und der tiefsten (minimum) Legatonote abgespeichert.

sgeomark4 Jetzt wird unterhalb von minimum sowie oberhalb von maximum (mit Toleranz 3) je ein freier Bereich gesucht. Ausgewählt und auf sp abgespeichert wird der, der den beiden Randnoten (Legatoanfang und Legatoende) am nächsten liegt.

sgeomark5 In der folgenden while Schleife wird die Legato-Bogenposition in alle am Legatobogen beteiligten Noten eingetragen. Darüber hinaus werden überall dort „lower stems“ gesetzt („upper“ gilt per default), wo ein nach oben gerichteter Notenstiel den Legatobogen schneiden würde.

3.5.12 detect_rests

drmark1 Die gesamte Funktion ist ziemlich kurz und übersichtlich, da die eigentliche Bearbeitung durch deal_with_takt und process_helprest realisiert wird. Mit einer äußeren while Schleife wird wieder die gesamte Liste abgearbeitet. Allerdings springt die Laufvariable dabei von einem Taktabschnitt zum nächsten.

drmark2 Für jeden Taktabschnitt und jede Region (es gibt 2 genau dann wenn (regio == 0)) werden die bearbeitenden Funktionen gerufen. deal_with_takt ordnet die Noten stur in Melodien ein bis alle behandelt sind. Die entstehenden Freiräume zwischen benachbarten Noten werden durch die Funktion ins_helprest in eine temporäre Struktur eingefügt. process_helprest läuft in folgenden 3 Etappen ab:

1. verbinde Pausen wenn möglich (Pause1 (a bis b) und Pause2 (b bis c) können durch Pause3 (a bis c) substituiert werden)
2. teile Pausen entsprechend dem Nenner des Taktes (z.B.: im $\frac{2}{4}$ Takt müssen die Pausen- und Notenwerte $\frac{1}{8}$ Note + $\frac{1}{4}$ Pause + $\frac{1}{8}$ Note geschrieben werden als $\frac{1}{8}$ Note + $\frac{1}{8}$ Pause + $\frac{1}{8}$ Pause + $\frac{1}{8}$ Note, damit die Aufteilung $\frac{1}{4}+\frac{1}{4}$ erhalten bleibt)
3. ordne den Pause Zählzeiten zu (wieder mit NORMFAKTOR und faktors); da Noten nur zu diskreten durch die Dauer einer $\frac{1}{64}$ Note teilbaren Zeiten beginnen (damit auch enden), sind auch die Freiräume exakt durch Pausen ausfüllbar

3.5.13 beam_geometry

In der Funktion beam_geometry geht es darum, jedem Balken eine Position und einen Anstieg zuzuordnen. Da durch beam_detect bzw. set_beam_regions nahezu keine geometrischen Charakteristika des Notenbildes ausgewertet wurden, kann es in Einzelfällen nötig sein, einen bereits erkannten Balken in zwei aufzutrennen.

bgmark1 Das Feld akt dient wiederum dem Durchlaufen der Liste. measurecounter spielt inhaltlich keine Rolle (ist für das Debugging günstig, da ersichtlich wird, in welchem Takt eine besondere Situation aufgetreten ist). dynhelp3 verweist immer auf das erste Ereignis im Takt.

bgmark2 Beim Auftreten eines Taktstriches müssen measurecounter und dynhelp3 aktualisiert werden.

bgmark3 Falls eine Note mit leerer treated Eintragung und gültiger beam_nr (ungleich Null) gefunden wird, so muß dieser Balken bearbeitet werden.

bgmark4 Nach der Initialisierung einiger Variablen (sbpos dient der Speicherung der durch Legatobögen und Balken belegten Positionen) wird vom Taktbeginn (dynhelp3) bis zur aktuellen Balkenanfangsnote (Feldelement von akt) die Liste durchlaufen. Am Schluß (dynhelp2 erreicht dynhelp) stehen im Feld sbpos alle zum aktuellen Zeitpunkt (akt $\rightarrow$ time) belegten (sbposelement ungleich Null) Positionen. Es muß erwähnt werden, daß sowohl bei Legatobögen als auch bei Balken die im MusicTEX-File anzugebende nicht mit der tatsächlichen Position übereinstimmt. Bei Balken ist die Differenz der Höhen besonders groß. Deshalb wird mit der Konstante UPPLOWSHIFT die eine Position durch Addition bzw. Subtraktion in die andere überführt (je nachdem, ob der Balken unterhalb oder oberhalb der Noten liegt).

bgmark5 Da existierende Legatobögen und Balken nicht geschnitten werden dürfen, wird für den neuen Balken eine obere (upp) und eine untere (low) Grenze entsprechend sbpos gesetzt. Das gültige Gebiet wird durch einen mit 1 belegten Bereich in usedpos gekennzeichnet.

bgmark6 Jetzt wird in Richtung der letzten Balkennote vorwärts gelaufen. Entsprechend der belegten Positionen wird der gültige Bereich in usedpos immer mehr eingeschränkt. Wird eine Note des zu behandelnden Balkens erreicht, deren Position ungültig ist (usedpos ist an dieser Stelle 0), so muß der Balken in zwei aufgeteilt werden. Im optimalen Fall bleibt nach dem Erreichen der letzten Balkennote ein zulässiger Bereich übrig, der für die Position des Balkens ausreichend Spielraum bietet. Am Ende dieses Programmabschnittes kennzeichnen die Zeiger dynhelp und not2 die letzte Note des aktuellen Balkens (wurde der ursprüngliche Balken geteilt, so referenzieren dynhelp und not2 die letzte Note des ersten Teilbalkens).

bgmark7 Im Folgenden wird zuerst der Höhenunterschied der beiden Randnoten (Balkenanfang und Balkenende) auf der Variable helpmaxone abgelegt. orientflag weist auf einen steigenden Balken hin ((orientflag == 0) gdw. Balken fallend). Beide Variablen dienen später der Berechnung des Anstieges. Entsprechend der mittleren Höhe der Balkennoten (ergibt sich aus höchster -maxb und tiefster -minb Note) wird jetzt zuerst nach einer oberen oder unteren Position gesucht (tiefe Noten sollten einen oberen, hohe Note einen unteren Balken bekommen). Ein Bereich gilt als ausreichend groß für den Balken, wenn er größer als ($\frac{helpmaxone}{2}$ + BEAMTOLERANCE) ist. Wie der Name bereits sagt, definiert BEAMTOLERACE einen Toleranzwert, der für ausreichend Abstand zwischen dem Balken und anderen Elementen des Notenbildes sorgt. Solange kein genügend großer freier Bereich gefunden wird, muß weitergesucht werden (erst in der anderen Richtung, dann zwischen minb und maxb). Am Ende steht in pos die Position, die in das MusicTEX-File eingetragen werden muß und in realpos die Position, die durch den Balken tatsächlich belegt wird. valid wird auf 0 gesetzt, falls kein ausreichend großer Bereich (größer als MINBEAMBROAD) gefunden wurde. Die Variable maxone enthält die Größe des freien Gebietes.

bgmark8 Ist valid ungleich Null, so kann der Balken gezeichnet werden. Mit Hilfe von maxone, helpmaxone und orientflag wird ein Wert für sl (Anstieg des Beams) errechnet. Falls sich der Balken oberhalb der Noten befindet, muß in pos das höchstwertigste Bit gesetzt werden.

bgmark9 Jetzt müssen sowohl die gefundenen Werte (Position und Anstieg) als auch Beginn- und Endeflag bei den Noten dieses Balkens eingetragen werden.

bgmark10 Wird der Code für (valid == 0) abgearbeitet, so ist der gefundene Bereich nicht ausreichend groß und der Balkeneintrag muß gelöscht werden.

3.5.14 calculate_rest_positions

crpmark1 Die Funktion calculate_rest_positions beginnt mit der Initialisierund der Laufvariable (Feldelement von akt), des Taktzählers measurecounter und der Felder cur_slur und cur_beam, die später die durch Legatobögen und Balken belegten Positionen kennzeichnen. Die Funktion tritt wiederum in eine while Schleife ein (durchlaufen der gesamten Liste). Dabei werden die Elemente Note, Taktstrich und Pause behandelt.

crpmark2 Falls eine Pause gefunden wird, so werden im Anschluß daran alle Elemente der aktuellen Zeit (rtime) überprüft. Die Positionen der aktiven Legatobögen und Balken sowie der zeitgleichen Noten und ihren Stielen werden in das Feld full eingetragen. Die Anzahl der Pausen der Zeit rtime wird in der Variable restcount mitgeführt.

crpmark3 Entsprechend der Region (Violine oder Baß) werden jetzt einige Parameter für die zwei folgenden while-Schleifen gesetzt.

crpmark4 Die erste Schleife durchläuft das Array full etwas unterhalb der Mitte beginnend bis zur höchsten möglichen Position. Der Zyklus bricht ab, wenn ein ausreichend großes freies Gebiet gefunden wurde. Wie groß dieses sein muß, hängt von der Größe der jeweiligen Pause (resthigh) und der Konstante RESTTOLERANCE ab. Desweiteren werden zur richtigen Justierung das Feld halfresthigh und HALFRESTTOLERANCE verwendet. **Vorsicht! Es gilt (halfresthighelement != $\frac{resthighelement}{2}$).**

crpmark5 Die zweite Schleife läuft entgegengesetzt, von etwas oberhalb der Mitte bis zur tiefsten möglichen Position. Wird ein ausreichend großer freier Bereich gefunden, so wird dieser nur dann nach bestpos übernommen, falls die Pause sich in diesem Fall näher an der Mitte befindet als mit der alten bestpos Eintragung.

crpmark6 Wurde bestpos noch nicht belegt, so muß die Pause in den größten zur Verfügung stehenden freien Bereich geschrieben werden. Da dieser aber offensichtlich nicht groß genug ist, wird die Pause später leicht nach rechts verschoben, damit keine Kollisionen auftreten. Auf die Notwendigkeit dieser Rechtsverschiebung weist re $\rightarrow$ flg hin.

crpmark7 Bevor das Feld full um den neuen Pausenbereich korrigiert wird und die nächste zeitgleiche Pause behandelt werden kann (falls es noch eine gibt) muß die Position für halbe und ganze Pausen modifiziert und für alle Pausen eingetragen werden (auf re $\rightarrow$ position).

crpmark8 Die Überprüfung der Positionen der Elemente von rtime reicht nicht aus. Es muß außerdem bekannt sein, welche Bereiche momentan von Legatobögen und Balken belegt werden. Deshalb werden in die Felder cur_slur und cur_beam alle Beams bzw. Slurs eingetragen und nach ihrem Ende wieder entfernt.

3.5.15 set_loudness_infos

Die Funktion set_loudness_infos ruft die Funktion ins_levent und fügt damit Lautstärkeelemente in die dynamische Liste ein. Der zweite Parameter von ins_levent gibt an, ob es sich um eine statische (1) oder dynamische (0) Lautstärkeeintragung handelt.

slimark1 Nach dem Einsprung in set_loudness_infos werden alle nötigen Initialisierungen vorgenommen. staticflag und dynamicflag geben an, ob ein entsprechendes Lautstärkeelement erzeugt werden soll oder nicht (staticflag = 1 bedeuted also, daß zu Beginn jedes Tracks eine statische Lautstärkeeintragung in die Liste vorgenommen werden muß). Die wesentlichste Datenstruktur bildet das Feld loudrecarr, auf dem die letzten 6 Durchschnittswerte (criterium), statischen Werte (range) und Zeiten (time) eingetragen werden.

slimark2 Die Basis für eine Eintragung in loudrecarr bilden jeweils zeitgleiche Noten. Das gesetzte staticflag provoziert die Generierung eines statischen Lautstärkeevents.

slimark3 Das Feld loudrecarr wird so indiziert, daß es einen Ring bildet. Das j-te Element von loudrecarr besitzt genau dann eine Zeiteintragung, wenn der Ring gefüllt ist. Dann können die dynamischen Ereignisse (wie z.B. das Lauterwerden der Music) erkannt und per ins_levent in die Liste eingefügt werden. Pegelt sich die Lautstärke nach einer Schwankung wieder ein, dann wird außerdem das staticflag gesetzt.

slimark4 Behandelte Elemente von loudrecarr müssen gelöscht werden, indem das time-Feld auf 0 gesetzt wird.

3.5.16 write_texfile

Durch die MusicTEX-Syntax ist diese Funktion nahezu selbsterklärend. Da sie jedoch sehr umfangreich ist, sollen im Folgenden wiederum einige markante Punkte herausgegriffen werden.

wtmark1 Im ersten Teil wird durch Zusammenhängen der Strings für Instrumenten- und Trackname eine Notenzeilenbezeichnung gebildet. Tracks ohne Noten bzw. die, welche zuviel sind, werden gelöscht.

wtmark2 Die Stimmen werden umgekehrt, da bei MusicTEX die letzten Notenzeilen zuerst aufgeschrieben werden müssen.

wtmark3 Jetzt kann der Kopf geschrieben werden.

wtmark4 Mit \debutmorceau wird die Notensequenz eingeleitet.

wtmark5 Taktstriche werden als \barre oder \alaligne (erzwingt Zeilenumbruch) geschrieben.

wtmark6 Änderungen bei Takt oder Tonart werden erst nach \changecontext sichtbar.

wtmark7 Innerhalb von Takten können Ton- und Taktart ebenfalls geändert werden (obwohl Taktartänderungen kritisch für die Zählzeit wären).

wtmark8 Zwischen \Notes und \enotes können alle zeitgleichen Lautstärke-, Lyrik-, Pausen- und Notenevents ausgegeben werden.

wtmark9 Die Verarbeitung von Noten ist etwas diffiziler. Zuerst werden vorhandene Legato-Eintragungen geschrieben. Dem schließt sich die Behandlung von Balken an. Es muß unterschieden werden zwischen der Initialisierung von Beams (\i), dem Ende von Beams (\t) und dem Hinzufügen einzelner Teilbalken an bereits vorhandene (\n). Mit dem Aufruf von write_note wird der eigentliche Notencode bzw. ein ganzer Akkord ausgegeben.

wtmark10 Das MusicTEX-File endet mit dem Befehl \finmorceau und einem sich anschließenden \end .

3.6 Einflußnahme auf die Funktionalität

Die Vielfalt der Möglichkeiten für die Argumentangabe beim Programmstart ist etwas dürftig. Trotzdem kann die Funktionsweise des Programmes innerhalb bestimmter Grenzen durch Konstantendefinitionen variiert werden:

INTERFACTEUR Abstand zwischen benachbarten Instrumenten

BOTTOMFACTEUR Abstand zwischen den Notenzeilen

MUSICSIZE Größe der Elemente des Notenbildes

PAGENUMBERS Flag für Seitennummernangabe

MEASURENUMBERS Flag für Taktnummernangabe

LINEBREAKER falls an Taktstrichen festgestellt wird, daß seit dem letzten Zeilenumbruch mindestens LINEBREAKER Zeitpunkte bearbeitet wurden, so wird der nächste Zeilenumbruch erzwungen

INNERBARMAX falls im letzten Takt mindestens INNERBARMAX Zeitpunkte behandelt wurden, so wird ein Zeilenumbruch erzwungen

INNERBARZ werden innerhalb eines Taktes mehr als INNERBARZ Zeitpunkte registriert, so kann mitten im Takt ein Zeilenumbruch herbeigeführt werden (**Achtung! Dadurch werden Noten schwer lesbar!**)

BASSLYRICPOSITION Position von Liedtexten im Baßbereich

VIOLINELYRICPOSITION Position von Liedtexten im Violinenbereich

BASSLOUDNESSPOSITION Position von Lautstärkeinformationen im Baßbereich

VIOLINELOUDNESSPOSITION Position von Lautstärkeinformationen im Violinenbereich

Da Liedtexte und Lautstärkeinformationen üblicherweise auf einer konstanten Position ausgegeben werden, lohnt es sich nicht, Kollisionstests durchzuführen (das endet u.U. bei dem Resultat, daß keine Position frei ist). Allerdings sollte der Nutzer bei unschönen Überlappungen versuchen, die letzten vier der oben aufgeführten Konstanten zu modifizieren.

Kapitel 4

Validierung

Die Korrektheit von Programmen dieser Größe läßt sich nicht mathematisch beweisen. Es kommt vielmehr darauf an zu zeigen, daß die Software bei unterschiedlichsten Eingaben akzeptable Ergebnisse liefert. Viele FTP-Archive beinhalten frei verfügbare MIDI-Files. Desweiteren besteht die Möglichkeit, MIDI-Files selbst zu generieren, entweder mit Tools wie MOZART und Rosegarden oder direkt am Keyboard. Dabei muß der MIDI-OUT Anschluß des Keyboards mit dem MIDI-IN Anschluß der Soundkarte verbunden werden. Außerdem muß beiden Endsystemen mitgeteilt werden, welche Kanäle (0 bis 15) mit welchem Interface (MIDI-IN, MIDI-OUT) gekoppelt werden müssen. Leider konnte im Rahmen dieses Projektes aus Zeitgründen kein derartiger Versuch durchgeführt werden.
Um Fehlerursachen zu finden, wurde der Debugger gdb [Sw6Gnu] verwendet. Durch den Aufruf von gcc [Sw5Gnu] mit der Option -ggdb konnten sowohl übliche debug-Informationen als auch gdb-Erweiterungen in den Code eingebunden werden. Beispiel einer gdb-Sitzung:

```
linhost:/home/koa/project/prog > gdb mid2tex
GDB is free software and you are welcome to distribute copies of it
under certain conditions; type "show copying" to see the conditions.
There is absolutely no warranty for GDB; type "show warranty" for details.
GDB 4.13 (i486-slackware-linux), Copyright 1994 Free Software Foundation, Inc...
(gdb) break readmidi.c:1037
Breakpoint 1 at 0xf324: file readmidi.c, line 1037.
(gdb) run -2 elise.mid
Starting program: /home/koa/project/prog/mid2tex -2 elise.mid

Breakpoint 1, main (argc=3, argv=0xbffff9fc) at readmidi.c:1037
1037            write_texfile();
(gdb) info locals
lenhelp = 30
opthelp = 1 '\001'
(gdb) continue
Continuing.

Program exited normally.
(gdb) quit
linhost:/home/koa/project/prog >
```

Kapitel 5

Analyse

5.1 Speicherplatz und Zeitverhalten

Zwischen Speicherplatz und Zeitverhalten existiert ein direkter Zusammenhang. Versucht man das Programm schneller zu machen, so steigt üblicherweise der Bedarf an der Ressource Speicher bzw. umgekehrt. Der Programmierer strebt dabei (ein sicherlich subjektives) Optimum an (subjektiv darf nicht im nichtdeterministischen Sinn verstanden werden → für jeden Programmierer existiert eine endliche Anzahl von Nebenbedingungen, deren Bedeutung durch ihn eingeschätzt werden muß).

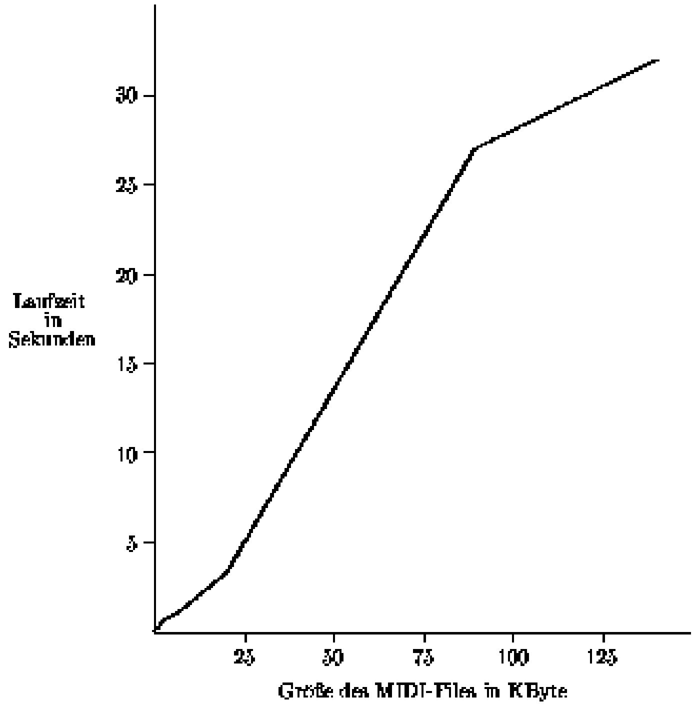
Für hinreichend große MIDI-Files kann der Speicheraufwand durch die Größe der einfach verketteten Liste abgeschätzt werden. Setzt man INTEGER-Werte und Zeiger mit 4Byte an, so ergeben sich für die unterschiedlichen Structs folgende Bedarfszahlen:

Name	Speicheraufwand in Byte
bind	13
Note	33
TimeSig	4
Lyric	4
KeySig	2
Rest	6
Loudness	8

Das (ohne Option durch gcc übersetzte) mid2tex-Programm wird im Folgenden durch konkrete Eingabefiles analysiert (ohne definiertes DEBUG):

Name	Größe des MIDI-Files in KByte	Anzahl von Ereignissen							Speicherbedarf der Liste in KByte	Laufzeit in sek
		'n'	't'	'l'	'k'	'b'	'r'	'p'		
brandbrg.mid	140,5	18451	19	0	19	2496	27169	18	1365,6	32,0
sinfon5.mid	89,4	10833	15	0	15	3749	17509	7683	1017,1	27,0
elise.mid	20,0	1992	5	0	5	628	1641	4	128,1	3,3
frueh.mid	6,4	520	4	0	7	380	467	3	37,1	1,1
mzduet1.mid	2,5	438	2	0	2	130	83	2	23,0	0,7
morley.mid	1,9	305	3	0	3	99	9	3	15,3	0,5
5.mid	0,2	20	1	0	1	4	1	1	1,0	0,1

Akkorde wurden in dieser Aufstellung mit nur einem „struct Note" veranschlagt, d.h. vernachlässigt. Die Programmausführung erfolgte auf einem INTEL80486 mit 8MByte RAM, 256KByte „on board" Cache und gestartetem X. Dennoch stellte der Hauptspeicher keinen ernstzunehmenden Engpaß dar (ohne gestartetes X wurden für „mid2tex brandbrg.mid" ca. 30 sek benötigt). Setzt man für den Code und die restlichen Daten einen Speicherbedarf von 500KByte an und geht man davon aus, daß MIDI-Files 150KByte Größe eigentlich nicht übersteigen, so liegt der Speicherbedarf des Programmes unter 2MByte.
Die folgende Darstellung bezieht sich auf die Laufzeit:



Die Größe des MIDI-Files ist dem Bedarf an Zeit und dem verwendeten Speicher nahezu proportional, bezieht man eine Art offset für Code und statische Daten mit ein. Eine wesentliche Rolle spielt jedoch auch der prozentuale Anteil der für das MusicTEX-File relevanten Events bezogen auf die Gesamtzahl von Events im MIDI-File.

Insgesamt bleib festzuhalten: **Speicherbedarf und Laufzeit steigen linear!**

5.2 Möglichkeiten und Grenzen

TEX-Files sind ASCII-Files und als solche mit zahlreichen Tools (z.B Texteditoren) modifizierbar. Der Aufruf von TEX [Sw7] geschieht durch:

```
tex <texfilename>
```

Dabei wird ein DVI-File generiert, das z.B. durch:

```
dvips [-o <outputfile.ps>] <inputfile.dvi>
```

(dvips [Sw8RES]) in eine Postscript-Datei überführt werden kann. Infolgedessen sind die Noten auf nahezu allen Ausgabegeräten darstellbar.

Zu experimentellen Zwecken kann mit mid2tex die Visualisierung von MIDI-Files erfolgen. Als Ergebnis entsteht jedoch meistens ein Notenbild, das von Musikern (z.B. Pianisten) nur schwer gespielt werden kann. Musikalische Stimmen sind häufig viel zu überladen und zeitgleiche Ereignisse räumlich versetzt (durch mögliche Kollisionen bzw. „spacing“ Noten und Pausen). Dadurch wird das Notenbild schwer überschaubar. Desweiteren werden Noten erzeugt, die per Hand nicht exakt spielbar sind (punktierte $\frac{1}{32}$ und $\frac{1}{64}$ Noten). Die Verfahrensweise für die Erkennung von staccato ist sehr starr. Deshalb treten Staccatonoten nur sehr selten auf (eine Ausnahme bilden MusicTEX-Files, die aus MOZART-MIDI-Files erzeugt wurden).

Anhang A

Beispiele

A.1 5.tex - MOZART MIDI-File 5.mid

A.2 elise.tex - zweiter Track von elise.mid

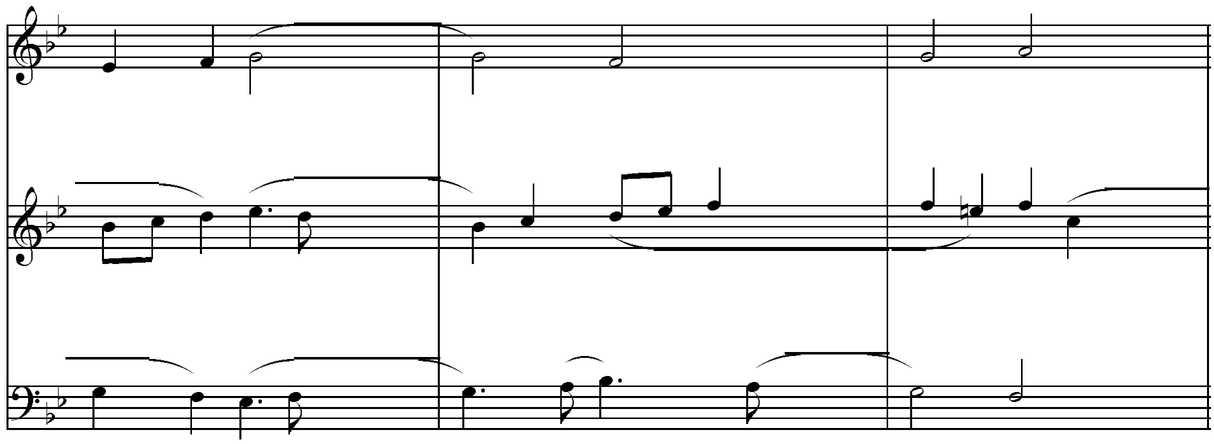
elise.tex

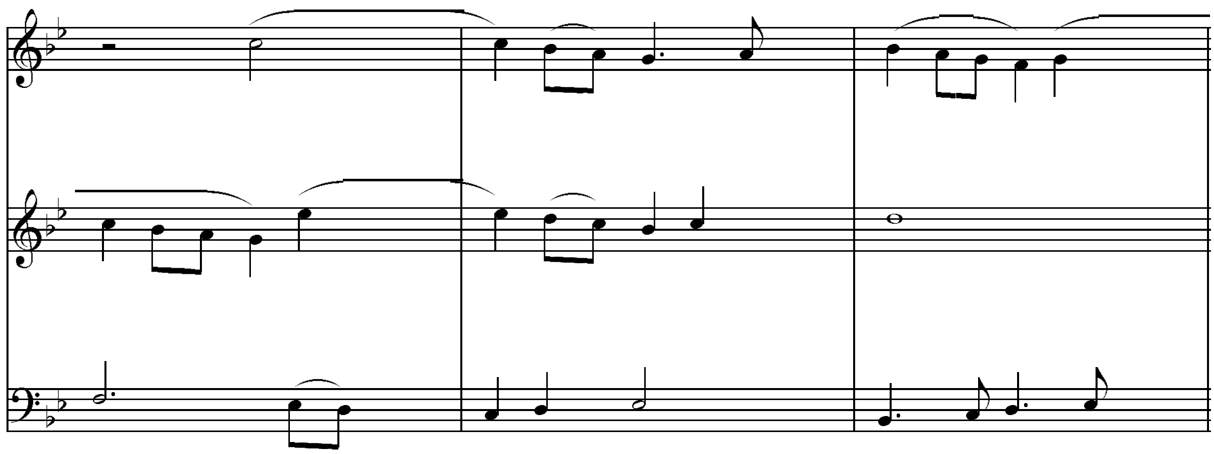
generated by mid2tex

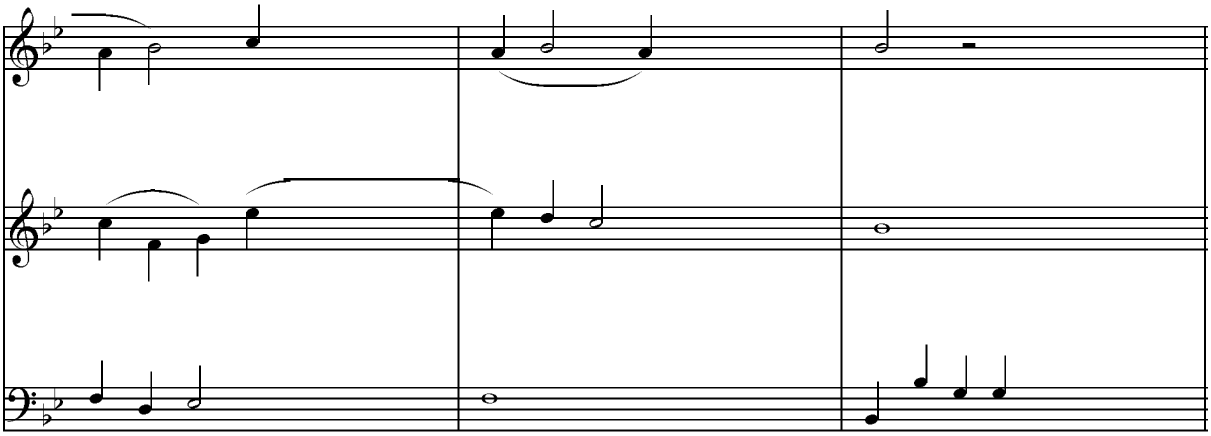
A.3 morley.tex - generiert aus morley.mid (MOZART)

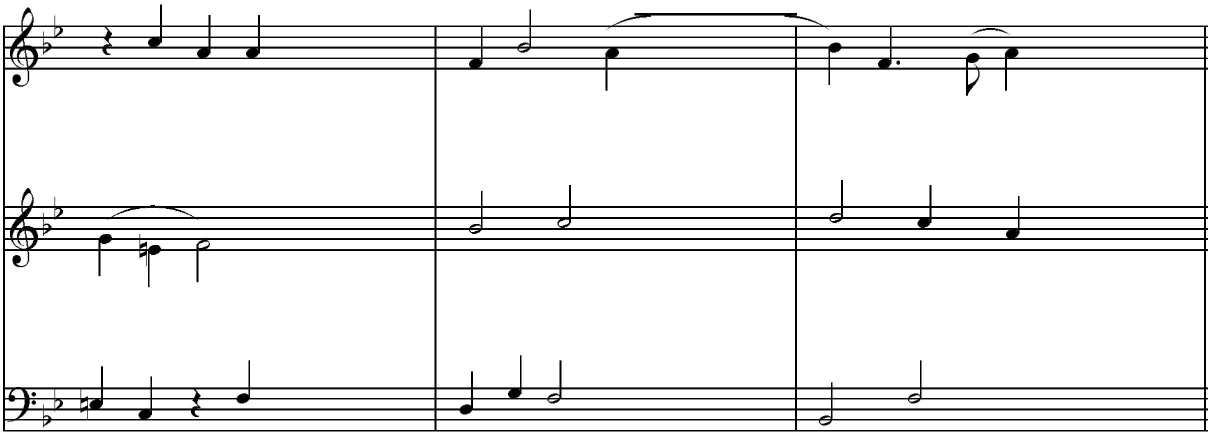
morley.tex

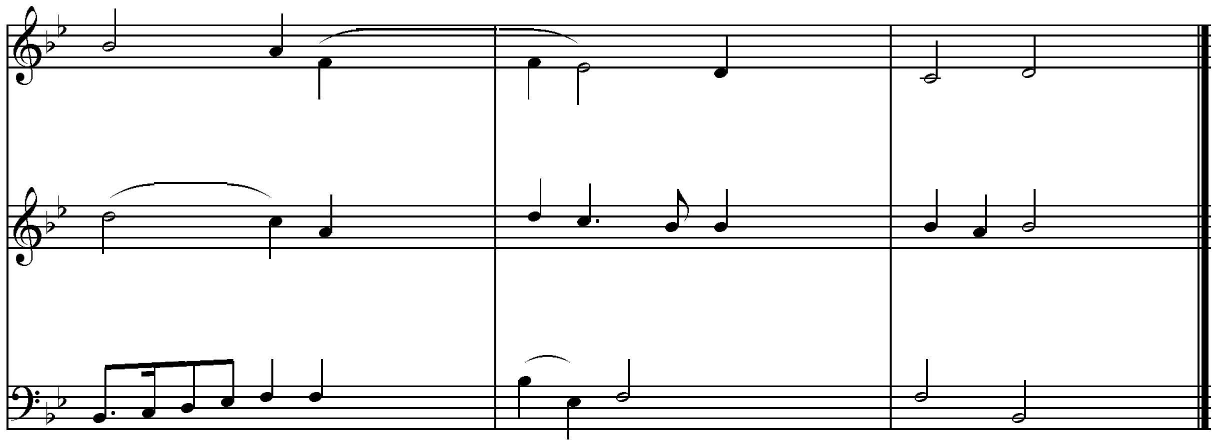
generated by mid2tex











Anhang B

Programmquelltexte

B.1 makefile

```
OBJS = dyn.o liste.o texgen.o readmidi.o
CC = gcc
CFLAGS = -pedantic -ggdb -Wall

readmidi: $(OBJS)
  $(CC) $(CFLAGS) -o mid2tex $(OBJS)

readmidi.o : readmidi.c midi.h maintypes.h defines.h readmidi.h
  $(CC) $(CFLAGS) -c readmidi.c

liste.o : liste.c liste.h
  $(CC) $(CFLAGS) -c liste.c

dyn.o : dyn.c maintypes.h defines.h dyn.h
  $(CC) $(CFLAGS) -c dyn.c

texgen.o : texgen.c maintypes.h defines.h texgen.h
  $(CC) $(CFLAGS) -c texgen.c
```

B.2 defines.h

```
/***************************/
/* defines.h               */
/***************************/

#define MAXTRACKS 30     /* maximum for number of tracks */
/* #define DEBUG 0 */
```

B.3 dyn.h

```
/***************************/
/* dyn.h                   */
/***************************/

static int global;
static unsigned long uslonghelp;

/* the following array was built for detecting durations of notes      */
/* and avoids 1/32.. , 1/64. and 1/64.. notes                           */

static char halfresthigh[7] = { 5, 4, 3, 4, 4, 0, 0 };
static char resthigh[7] = { 11, 9, 7, 5, 7, 3, 3 };

char RESTTOLERANCE = 6;
char HALFRESTTOLERANCE = 2;

char UPPLOWSHIFT = 7;    /* 5 shift the position because of TeX */

static short loudarray[8] = { 10, 25, 40, 55, 75, 95, 115, 300 };

static double faktors[21] =
  { 1, 1, 1, 2, 3, 3, 4, 6, 7, 8, 12, 14,
   16, 24, 28, 32, 48, 56, 64, 96, 112 };

static int NORMFAKTOR = 16;
static short note_codes[21] =
  { 0x0, 0x1, 0x2, 0x8, 0x9, 0xA, 0x10, 0x11, 0x12, 0x18, 0x19, 0x1A,
    0x20, 0x21, 0x22, 0x28, 0x29, 0x2A, 0x30, 0x31, 0x32 };

/* the next arrays are for calculating the position of notes with */
/* their signs */
static int crosses[7] = { 5, 0, 7, 2, 9, 4, 11 };
static int bs[7] = { 11, 4, 9, 2, 7, 0, 5 };
static char process[12] = { 64, 0, 64, 0, 64, 64, 0, 64, 0, 64, 0, 64 };
static char pitch_to_pos[12] = { 0, 0, 1, 0, 2, 3, 0, 4, 0, 5, 0, 6 };
char positions[42] = {
  ´C´, ´D´, ´E´, ´F´, ´G´, ´A´, ´B´,
  ´C´, ´D´, ´E´, ´F´, ´G´, ´H´, ´I´,
  ´J´, ´K´, ´L´, ´M´, ´N´, ´a´, ´b´,
  ´c´, ´d´, ´e´, ´f´, ´g´, ´h´, ´i´,
  ´j´, ´k´, ´l´, ´m´, ´n´, ´o´, ´p´,
  ´q´, ´r´, ´s´, ´t´, ´u´, ´v´, ´w´ };

static char LAMBDA = 9; /* maximal length of normal stems (1/64 note) */
static char BEAMDELTA = 19; /* maximal length of beamed stems */
```

```
static char BEAMTOLERANCE = 7; /* 7 minimal distance between beamed */
                                /* notes and the beam itself */
static char BEAMTOLMAX = 12;    /* 11 maximal distance between beamed notes */
                                /* and the beam itself */
static char SLURWIDE = 4;       /* vertical wide of slurs */
static char MINBEAMBROAD = 5;   /* minimal distance between the highest and */
                                /* the lowest note for middle beams */
static char BASSMID = 15;       /* middle position of bass clef */
static char VIOLINEMID = 27;    /* middle position of violine clef */
static char SLOPEFACTOR = 5;    /* 3 for the slope of beams - smaller */
                                /* SLOPEFACTOR means bigger slopes */

static struct bind *prev[MAXTRACKS], *remember[MAXTRACKS];
static struct bind *bp, *dy, *dy2, *d, *d2, *d3;
static struct bind *dynhelp2, *dynhelp3;
struct bind *begin[MAXTRACKS], *akt[MAXTRACKS], *dynhelp;

static struct Barre {           /* ´b´ */
  short number_of_voices;
  int takt_duration;
} *bar;

struct loudrec {        /* for storing some loudness datas */
  unsigned long time;
  unsigned short criterium;
  char range;
} *loudrecarr[6];

int modulo4pp[6] = { 1, 2, 3, 4, 5, 0 };

struct KeySig *ks;
struct Lyric *ly;
struct TimeSig *ts;
static struct Note *not2, *not3;
struct Note *not;
struct Rest *re;
struct Loudness *lo;

static struct helprest {
  unsigned long begin;
  unsigned long end;
  struct helprest *next;
} *ank, *hr1, *hr2, *hr3, *hres1, *hres2, *hres3;
```

B.4 liste.h

```
/***************************/
/* liste.h                 */
/***************************/

static struct element {
  int oct, pit, laut, chann;
  unsigned long int t;
  struct element *folger;
} *anker;
```

B.5 maintypes.h

```
/***************************/
/* maintypes.h             */
/***************************/

struct bind {
  char marker;  /* detect the kind of element */
  unsigned long time;
  struct bind *next;
  void *data;
};

struct KeySig {          /* marker == ´k´ */
  short sign;
};

struct Lyric {           /* marker == ´l´ */
  char *text;
};

struct TimeSig {         /* marker == ´t´ */
  short nom, denom;
};

struct Note {            /* marker == ´n´ */
  short octave, pitch, loud;
  unsigned long end;
  int duration;
  short flags;
  int beam_nr, slur_nr;
  char note_pos, slur_pos, beam_pos;
  char treated;
  char next_beamn_kind;
  struct Note *accord_next;
```

```
};

struct Rest {            /* marker == ´r´ */
  char position;
  int kind;
  char flg;    /* indicates a useful skip */
};

struct Loudness {        /* marker == ´p´ */
  int statics, dynamics;
};
```

B.6 midi.h

```
/******************************/
/*                            */
/* midi.h                     */
/*                            */
/* Definition der Midi-Events */
/*                            */
/******************************/


/***************************/
/* Channel Voice Messages. */
/***************************/

#define MIDI_NOTE_OFF    (0x80)
#define MIDI_NOTE_ON    (0x90)
#define MIDI_POLY_AFTERTOUCH  (0xA0)
#define  MIDI_CTRL_CHANGE  (0xB0)
#define MIDI_PROG_CHANGE  (0xC0)
#define MIDI_CHNL_AFTERTOUCH  (0xD0)
#define MIDI_PITCH_BEND    (0xE0)

/********************/
/* System Messages. */
/********************/

#define MIDI_SYSTEM_MSG    (0xF0)

#define MIDI_SYSTEM_EXCLUSIVE  (0xF0)
#define MIDI_SONG_POSITION_PTR  (0xF2)
#define MIDI_SONG_SELECT  (0xF3)
#define MIDI_TUNE_REQUEST  (0xF6)
#define MIDI_EOX    (0xF7)
#define MIDI_FILE_META_EVENT    (0xFF)
```

```
/*****************************/
/* System Real-Time Messages. */
/*****************************/

#define MIDI_TIMING_CLOCK  (0xF8)
#define MIDI_START    (0xFA)
#define MIDI_CONTINUE    (0xFB)
#define MIDI_STOP    (0xFC)
#define MIDI_ACTIVE_SENSING  (0xFE)
#define MIDI_SYSTEM_RESET  (0xFF)

/******************************/
/* MIDI file format constants. */
/******************************/

#define MIDI_FILE_HEADER  "MThd"
#define MIDI_TRACK_HEADER  "MTrk"

/*************************/
/* MIDI file Meta-Events. */
/*************************/

#define MIDI_SEQUENCE_NUMBER  (0x00)
#define MIDI_TEXT_EVENT    (0x01)
#define MIDI_CHANNEL_PREFIX  (0x20)
#define MIDI_END_OF_TRACK  (0x2F)
#define MIDI_SET_TEMPO    (0x51)
#define MIDI_SMPTE_OFFSET  (0x54)
#define MIDI_TIME_SIGNATURE  (0x58)
#define MIDI_KEY_SIGNATURE  (0x59)
#define MIDI_SEQUENCER_SPECIFIC  (0x7F)

/*****************************/
/* Specific Text Meta-Events. */
/*****************************/

#define MIDI_COPYRIGHT_NOTICE  (0x02)
#define MIDI_TRACK_NAME    (0x03)
#define MIDI_INSTRUMENT_NAME  (0x04)
#define MIDI_LYRIC    (0x05)
#define MIDI_TEXT_MARKER  (0x06)
#define MIDI_CUE_POINT    (0x07)

#define MAJOR 0
#define MINOR 1
```

```
typedef struct header_struct {
  char MThd[5];
  unsigned int chunk_length;
  short int format, tracks, per_quarter;
} Header;

typedef struct track_str {
  char MTrk[5];
  int chunk_length;
} Track_Header;
```

B.7 readmidi.h

```
/***************************/
/* readmidi.h             */
/***************************/

FILE *fp, *tfp;
static Header head;
static Track_Header THead;

#ifdef DEBUG
static double D_Tempo;
extern void debug_out();
#endif

static double one64tel, sequencecounter, lastsc;
static unsigned long converter, bartime, lastbartime;

static char Running_Event;
static char notes_flag;
static unsigned long zeit;
static int help;
static int taktzeit;
static unsigned long diskrete_tz;
static unsigned long int firstnotetime;
static int firstnotetimeflag;
static int Signature;
static int T_genus;
static int Track_Nr;
static char printonly;                /* for one track processing */
char Staffs[MAXTRACKS][100];          /* names of instruments */
char TrackName[MAXTRACKS][80];        /* names of tracks */
char texfilename[50];      /* name of texfile */
static unsigned char max[MAXTRACKS];
static unsigned char min[MAXTRACKS];
```

```
char region[MAXTRACKS];                /* 0 both, 1 violine, 2 bass */
static char count;                     /* count_only bit */
static short nom0, denom0, key0;       /* for first track datas */

static int valid, valid2;

static int ReadEvent();

extern void ins_dyn_event();
extern void note_insert();
extern int note_delete();
extern int search_note();
extern void dyn_init();
extern void split_notes();
extern void detect_accords();
extern void insert_notetimes_and_slurs();
extern void detect_note_positions_with_signs();
extern void set_slur_regions();
extern void reset_treat_info();
extern void beam_detect();
extern void set_beam_regions();
extern void slur_geometry();
extern void detect_rests();
extern void beam_geometry();
extern void calculate_rest_positions();
extern void set_loudness_infos();
extern void write_texfile();
```

B.8 texgen.h

```
/****************************/
/* texgen.h                 */
/****************************/

char *romans[MAXTRACKS] =
  { "i", "ii", "iii", "iv", "v", "vi", "vii", "viii", "ix",
    "x", "xi", "xii", "xiii", "xiv", "xv", "xvi",
    "xvii", "xviii", "xix", "xx", "xxi", "xxii", "xxiii",
    "xxiv", "xxv", "xxvi", "xxvii", "xxviii", "xxix",
    "xxx" };

/* basically it is possiple to increase this value up to MAXTRACKS if a */
/* new version of tex will support that */
int CURRENTMAXTRACKS = 9; /* because of tex there is no possibility to */
                          /* generate notes of more than 9 tracks */

char *notecode[7] = { "cccc", "ccc", "cc", "c", "q", "h", "wh" };
```

```
char *restcode[7] = { "qqs", "hs", "qs", "ds", "qp", "hpause", "pause" };
char *static_loudness[8] = { "ppp", "pp", "p", "mp", "mf", "f", "ff", "fff" };
char *dynamic_loudness[8] = { "cresc", "Cresc", "CResc", "CREsc", "dimin",
                              "Dimin", "DImin", "DIMin" };

short INTERFACTEUR = 17;        /* space between two instruments */
short BOTTOMFACTEUR = 14;       /* space between the lines */
short MUSICSIZE = 16;           /* size of music may also be set to 20 */
short PAGENUMBERS = 1;          /* print page numbers (not -> 0) */
short MEASURENUMBERS = 0;       /* print measure numbers(not->0;yes->1) */
short LINEBREAKER = 15;         /* force a line breaking after more than */
                                /* LINEBREAKER (f.e.15) notes or rests */
short INNERBARMAX = 8;          /* force a line breaking after a bar longer */
                                /* than INNERBARMAX (f.e.8) */
short INNERBARZ = 18;    /* allow line breaking in the middle of a takt */
                         /* because of many events */

/* the following constants specify the position of their events */
char *BASSLYRICPOSITION = "L";
char *VIOLINELYRICPOSITION = "!g";
char *BASSLOUDNESSPOSITION = "I";
char *VIOLINELOUDNESSPOSITION = "!j";

extern struct bind *begin[MAXTRACKS], *akt[MAXTRACKS], *dynhelp;
extern struct KeySig *ks;
extern struct Lyric *ly;
extern struct TimeSig *ts;
extern struct Note *not;
extern struct Rest *re;
extern struct Loudness *lo;

extern char Staffs[MAXTRACKS][100];
extern char texfilename[50];
extern char region[MAXTRACKS];
extern FILE *tfp;
extern char TrackName[MAXTRACKS][80];
extern char positions[42];
extern char UPPLOWSHIFT;

static char Inst[MAXTRACKS][180];
```

B.9 dyn.c

```
/***************************/
/* dyn.c                   */
/***************************/
```

```
#include <stdio.h>
#include <stdlib.h>
#include "maintypes.h"
#include "defines.h"
#include "dyn.h"

void dyn_init()
{
  int ii;

  global = 1;
  for (ii=0; ii < MAXTRACKS; ii++) {
    begin[ii] = NULL;
    remember[ii] = NULL;
  }
}

void ins_dyn_event(tnr, mark, phr, nn, dom, oct, pit, loud, beg, dur)
  char mark, *phr;
  int tnr, nn, dom, oct, pit, loud, beg, dur;
{
  /* insdmark1 */
  if ((dynhelp = (struct bind *) malloc (sizeof(struct bind))) == NULL) {
    fprintf(stderr, "malloc error\n");
    exit(10);
  }
  dynhelp->marker = mark;
  dynhelp->time = beg;

  /* insdmark2 */
  switch (mark) {
    case ´n´: if ((not = (struct Note *)
                           malloc (sizeof(struct Note))) == NULL) {
        fprintf(stderr, "malloc error\n");
        exit(10);
        }
        not->octave = oct;
        not->pitch = pit;
        not->loud = loud;
        not->duration = dur;
        not->end = 0;
        not->flags = 0;
        if (((not->octave) * 12 + (not->pitch)) < 0x20) not->flags = 0x0200;
        if (((not->octave) * 12 + (not->pitch)) > 0x29) not->flags = 0x0100;
        not->beam_nr = 0;
        not->slur_nr = 0;
        not->note_pos = 0;
```

```
        not->slur_pos = 0;
        not->beam_pos = 0;
        not->treated = 0;
        not->next_beamn_kind = 0;  /* type of next beam note */
        not->accord_next = NULL;
        dynhelp->data = (void *) not;
        break;
    case ´t´:
        if ((ts = (struct TimeSig *)
                  malloc (sizeof(struct TimeSig))) == NULL) {
          fprintf(stderr, "malloc error\n");
          exit(10);
        }
        ts->nom = nn;
        ts->denom = dom;
        dynhelp->data = (void *) ts;
        break;
    case ´l´:
        if ((ly = (struct Lyric *) malloc (sizeof(struct Lyric))) == NULL) {
          fprintf(stderr, "malloc error\n");
          exit(10);
        }
        if ((ly->text = (char *) malloc (strlen(phr)+1)) == NULL) {
          fprintf(stderr, "malloc error\n");
          exit(10);
        }
        strcpy(ly->text, phr);
        dynhelp->data = (void *) ly;
        break;
    case ´k´:
        if ((ks = (struct KeySig *) malloc (sizeof(struct KeySig))) == NULL) {
          fprintf(stderr, "malloc error\n");
          exit(10);
        }
        ks->sign = nn;
        dynhelp->data = (void *) ks;
        break;
    case ´b´:
        if ((bar = (struct Barre *) malloc (sizeof(struct Barre))) == NULL) {
          fprintf(stderr, "malloc error\n");
          exit(10);
        }
        bar->takt_duration = dur;
        bar->number_of_voices = 1;
        dynhelp->data = (void *) bar;
        break;
  }
```

```
  dynhelp->next = NULL;

  /* insdmark3 */
  if (begin[tnr] == NULL) {
    begin[tnr]=dynhelp;
    akt[tnr]=dynhelp;
  }
  else {
    if ((remember[tnr] != NULL) && ((remember[tnr]->time) < beg))
      akt[tnr] = remember[tnr];
    else akt[tnr] = begin[tnr];
    prev[tnr] = NULL;
    while ((akt[tnr] != NULL) && ((akt[tnr]->time) < beg)) {
      prev[tnr] = akt[tnr];
      akt[tnr] = akt[tnr]->next;
    }
    if ((akt[tnr] == NULL) || ((akt[tnr]->time) > beg) || (mark == ´b´)) {
      if (prev[tnr] != NULL) prev[tnr]->next = dynhelp;
      else begin[tnr] = dynhelp;
      dynhelp->next = akt[tnr];
    }
    else {
      /* insert elements with equal time into list in correct permutation */
      if (akt[tnr]->marker == ´n´) not = (struct Note *) akt[tnr]->data;
      while ( (akt[tnr] != NULL) && ((akt[tnr]->time) == beg) &&
        ( ((akt[tnr]->marker) == ´b´) ||
          (((akt[tnr]->marker) == ´k´) && (mark != ´k´)) ||
          (((akt[tnr]->marker) == ´t´) && ((mark == ´l´) || (mark == ´n´))) ||
          (((akt[tnr]->marker) == ´l´) && (mark == ´n´)) ||
          (((akt[tnr]->marker) == ´n´) && (mark == ´n´) &&
           ((((not->octave) * 12) + (not->pitch)) < ((oct * 12) + pit))) ) ) {
        prev[tnr] = akt[tnr];
        akt[tnr] = akt[tnr]->next;
        if ((akt[tnr]) && (akt[tnr]->marker == ´n´))
            not = (struct Note *) akt[tnr]->data;
      }
      if (prev[tnr] != NULL) prev[tnr]->next = dynhelp;
      else begin[tnr] = dynhelp;
      dynhelp->next = akt[tnr];
    }
  }
}

void split_notes(Track, pq)
  int Track;
  short int pq;
```

```
{
  int notend, h;
  unsigned long barrtime;

  /* splnmark1: initiation */
  dynhelp3 = begin[Track];
  dynhelp2 = dynhelp3;
  notend = 1;
  while (notend) {
    /* splnmark2: loop now */
    while ((dynhelp3 != NULL) && ((dynhelp3->marker) != ´b´))
       dynhelp3 = dynhelp3->next;
    if (dynhelp3 == NULL) {
      notend = 0;
    }
    else {
      barrtime = dynhelp3->time;
      dynhelp3 = dynhelp3->next;
    }
    while (dynhelp2 && ((!dynhelp3) || (barrtime > (dynhelp2->time)))) {
            if (dynhelp2->marker == ´n´) {
              not = (struct Note *) dynhelp2->data;
              h = (dynhelp2->time) + (not->duration);

              /* note should be splited because of long time */
              if (((not->duration) > (4 * pq)) && ((!dynhelp3) ||
                   (((dynhelp2->time) + (4 * pq)) < barrtime))) {
                    ins_dyn_event(Track, ´n´, 0, 0, 0, not->octave, not->pitch,
                                  not->loud, (dynhelp2->time) + (4 * pq),
                                  (not->duration) - (4 * pq));
                    (not->duration) = 4 * pq;
              }
              else {
                /* note sounds about next barre */
                if ((dynhelp3) && (h > barrtime)) {
                  (not->duration) = barrtime - (dynhelp2->time);
                  ins_dyn_event(Track, ´n´, 0, 0, 0, not->octave, not->pitch,
            not->loud, barrtime, h - barrtime);
        }
       }
     }
     dynhelp2 = dynhelp2->next;
    }
  }
}

/* the following funktion returns one in case of existing a */
```

```
/* note that begins on time timepoint, zero else */
/* right region was found -> okflag returns 1 else 0 */

short note_with_time(bplocal, timepoint, bpp, okflag)
  struct bind *bplocal, **bpp;
  char *okflag;
  unsigned long timepoint;
{
  struct bind *bhelp;
  char remem;
  struct Note *nhelp;

  /* notes shorter than 1/4 allow for beams */
  if ((*okflag) > 4) remem = 0;
  else remem = 1;

  /* delete bit 4 */
  *okflag &= 0x3;

  while ((bplocal) && (((bplocal->time) < timepoint) ||
                      ((bplocal->marker) != ´n´))) bplocal = bplocal->next;
  if ((bplocal) && ((bplocal->time) == timepoint) &&
      ((bplocal->marker) == ´n´)) {
    if (*okflag) {
      nhelp = (struct Note *) bplocal->data;
      bhelp = bplocal;
      while ( (bhelp) && ((bhelp->time) == timepoint) &&
        (!((!((nhelp->flags) & 0x200)) && (*okflag == 1) &&
         (remem || (!((nhelp->flags) & 0x20))) )) &&
              (!((!((nhelp->flags) & 0x100)) && (*okflag == 2) &&
               (remem || (!((nhelp->flags) & 0x20))) )) ) {
               bhelp = bhelp->next;
        if ((bhelp) && ((bhelp->time) == timepoint))
          nhelp = (struct Note *) bhelp->data;
      }
      if ((bhelp) && ((bhelp->time) == timepoint)) {
        *okflag = 1;
        *bpp = bhelp;
        return(1);
      }
      else {
        *okflag = 0;
        *bpp = bplocal;
        return(1);
      }
    }
    else {
```

```
      if (remem) {
        *okflag = 1;
        *bpp = bplocal;
        return(1);
      }
      else {
        nhelp = (struct Note *) bplocal->data;
        bhelp = bplocal;
        while ((bhelp) && ((bhelp->time) == timepoint) &&
               ((nhelp->flags) & 0x20)) {
          bhelp = bhelp->next;
          if ((bhelp) && ((bhelp->time) == timepoint))
            nhelp = (struct Note *) bhelp->data;
        }
        if ((bhelp) && ((bhelp->time) == timepoint)) {
          *okflag = 1;
          *bpp = bhelp;
          return(1);
        }
        else {
          *okflag = 0;
          *bpp = bplocal;
          return(1);
        }
      }
    }
  }
  else {
    *okflag = 0;
    return(0);
  }
}

short barre_between(bploc, reftime)
  struct bind *bploc;
  unsigned long reftime;
{
  short rem = 0;
  while ((bploc) && ((bploc->time) < reftime)) {
    if ((bploc->marker) == ´b´) rem = 1;
    bploc = bploc->next;
  }
  return(rem);
}

void detect_note_value(reg, tr, notedurat, flagptr, endptr, begint, bindptr,
                       slurnrpt, pq)
```

```
  int notedurat, *slurnrpt;
  short int pq, *flagptr, tr;
  unsigned long *endptr, begint;
  struct bind *bindptr;
  char reg;
{
  int i;
  char ok;  /* allow slur only to be in one region */
  char newslur;  /* for setting the slur-begin flag */
  short desicion;

  /* dnvmark1 */
  *flagptr = *flagptr | 0x80;
  i = 0;
  while ((i<21) && ((NORMFAKTOR * notedurat) > (faktors[i] * pq))) i++;
  if (i==21) {
    fprintf(stderr,"warning: cut very long note\n");
    *flagptr = *flagptr | note_codes[20];
    *endptr = begint + (7 * pq);
  }
  else {
    if ((NORMFAKTOR * notedurat) == (faktors[i] * pq)) {
      /* dnvmark2: input ok forces a special region */
      if (!reg && ((*flagptr) & 0x0100)) ok = 1;
      else  if (!reg && ((*flagptr) & 0x0200)) ok = 2;
        else ok = 0;
      desicion = note_with_time(bindptr, notedurat + begint, &dynhelp, &ok);
      /* dnvmark3 */
      if ((i<15) && (!desicion)) {
        if (barre_between(bindptr, begint + (notedurat * 4))) {
          /* call it a normal note */
          *flagptr = *flagptr | (note_codes[i]);
          *endptr = begint + notedurat;
        }
        else {
          /* staccato detected */
          *flagptr = *flagptr | ((note_codes[i+6]) + 4);
          *endptr = begint + (notedurat * 4);
        }
      }
      else {
        /* dnvmark4 */
        if (dynhelp) not = (struct Note *) dynhelp->data;
        if (((i>=15) && (!desicion)) ||
            ((desicion) && (!ok)) || (not->slur_nr)) {
          /* call it a normal note */
          *flagptr = *flagptr | (note_codes[i]);
```

```
          *endptr = begint + notedurat;
        }
        else {
          /* legato detected */
          *flagptr = *flagptr | (note_codes[i]);
          *flagptr = *flagptr & 0xff7f;
          *endptr = begint + notedurat;
          newslur = 0;
          if (!(*slurnrpt)) {
            newslur = 1;
            *slurnrpt = global++;
          }
          *flagptr = *flagptr | ((not->flags) & 0x0300);
          not->flags = not->flags | ((*flagptr) & 0x0300);
          not->slur_nr = *slurnrpt;
          if (newslur) not->slur_pos |= 0x40;
        }
      }
    }
    else {
      /* normal note detected */
      *flagptr = *flagptr | (note_codes[i]);
      *endptr = begint + ((faktors[i] * pq) / NORMFAKTOR);
    }
  }
  if (!(*slurnrpt)) *flagptr = *flagptr & 0xff7f;
}

void insert_notetimes_and_slurs(reg, Track, pq)
  int Track;
  short int pq;
  char reg;
{
  akt[Track] = begin[Track];
  while (akt[Track]) {
    if ((akt[Track]->marker) == 'n') {
      not = (struct Note *) akt[Track]->data;
      detect_note_value(reg, Track, not->duration, &(not->flags), &(not->end),
                        akt[Track]->time, akt[Track], &(not->slur_nr), pq);
    }
    akt[Track] = akt[Track]->next;
  }
}

void detect_accords(reg, Track)
  int Track;
  char reg;
```

```
{
  char attention;  /* remembers the region */

  /* deacmark1 */
  akt[Track] = begin[Track];

  while (akt[Track]) {
    /* deacmark2 */
    if ((akt[Track]->marker) == 'n') {
      not2 = (struct Note *) akt[Track]->data;
      dynhelp = akt[Track]->next;
      dynhelp2 = akt[Track];
      /* prevent accords over two regions */
      if ((((not2->octave) * 12 + (not2->pitch)) < 0x20) && (!reg))
        attention = 1;
      else attention = 0;
      /* deacmark3 */
      while ((dynhelp) && ((dynhelp->time) == (akt[Track]->time))) {
        not = (struct Note *) dynhelp->data;
        if (((not->duration) == (not2->duration)) &&
            ((!attention) || (((not->octave) * 12 + (not->pitch)) < 0x2a))) {
          while (not2->accord_next) not2 = not2->accord_next;
          not2->accord_next = not;
          dynhelp2->next = dynhelp->next;
          free(dynhelp);
          dynhelp = dynhelp2->next;
        }
        else {
          dynhelp2 = dynhelp;
          dynhelp = dynhelp->next;
        }
      }
    }
    akt[Track] = akt[Track]->next;
  }
}

void beam_detect(reg, Track)
  int Track;
  char reg;
{
  struct bind *found;
  char ok, remsign;
  unsigned long remtime;
  short helpshort;

  /* bdmark1 */
```

```
akt[Track] = begin[Track];
while (akt[Track]) {
  if (akt[Track]->marker == ´n´) {
    not = (struct Note *) akt[Track]->data;
    if (!((not->flags) & 0x20)) {
      /* bdmark2: 1/8 or shorter found */
      dynhelp = akt[Track]->next;
      while ((dynhelp) && ((dynhelp->marker) != ´b´))
        dynhelp = dynhelp->next;
      if (((not->flags) & 0x100) && (!reg)) ok = 9;
      else  if (((not->flags) & 0x200) && (!reg)) ok = 10;
        else ok = 8;
      /* bdmark3 */
      if ((note_with_time(akt[Track], not->end, &found, &ok)) && (found) &&
          (ok) && ((!dynhelp) || ((found->time) < (dynhelp->time)))) {
            not2 = (struct Note *) found->data;
            remsign = 0;
            if (((((not->slur_nr) && (!((not->flags) & 0x80))) ||
                ((not2->slur_nr) && (!((not2->slur_pos) & 0x40)))) &&
                ((not->slur_nr) != (not2->slur_nr))) || (not2->beam_nr)) {
              remtime = found->time;
              found = found->next;
              remsign = 1;
              /* while the restrictions of time and regions are */
              /* not injured try to search a valid beam_note */
              /* bdmark4 */
          while ((found) && ((found->time) == remtime) && (remsign) &&
                 (!(((not->flags) & 0x200) &&
                 (!reg) && ((not2->flags) & 0x100))) ) {
            not2 = (struct Note *) found->data;
            if ((!(not2->beam_nr)) && (not->slur_nr) &&
                (!((not->flags) & 0x80)) &&
                ((not2->slur_nr) == (not->slur_nr)) &&
                (!((not2->flags) & 0x200)))
                     remsign = 0;
            if ((!(not2->beam_nr)) && (!((not->slur_nr) &&
                (!((not->flags) & 0x80)))) && (((not2->slur_nr)  &&
                ((not2->slur_pos) & 0x40)) || (!not2->slur_nr)) &&
                (!((not2->flags) & 0x200)))
                     remsign = 0;
            if (remsign) found = found->next;
          }
        }
        /* bdmark5 */
        if (!remsign) {
              /* beam found */
              if (not->beam_nr) {
```

```
                  not->flags &= 0xffbf;
                  not2->beam_nr = not->beam_nr;
            }
            else {
              not->beam_nr = global;
              not2->beam_nr = global++;
            }
            not2->flags |= 0x40;
            helpshort = not->flags & 0x300;
            not->flags |= (not2->flags) & 0x300;
            not2->flags |= helpshort;
          }
        }
      }
    }
    akt[Track] = akt[Track]->next;
  }
}

void other_niveau(cr, of, pf, osp, psp)
  char cr;
  short of, pf, *osp, *psp;
{
  if (cr) {
    if (pf) {
      *psp = pf - 1;
      *osp = of;
    }
    else {
      *psp = 11;
      *osp = (of >> 1);
    }
  }
  else {
    if (pf != 11) {
      *psp = pf + 1;
      *osp = of;
    }
    else {
      *psp = 0;
      *osp = (of << 1);
    }
  }
}

void detect_note_positions_with_signs(track)
  int track;
```

```
{
  char cross;
  int signhelp;
  int i;
  short o, osecond, psecond, rem_o;
  int p;

  /* dnpsmark1 */
  akt[track] = begin[track];
  cross = 1;

  while (akt[track]) {
    switch (akt[track]->marker) {
      case ´n´: /* dnpsmark2 */
          not = (struct Note *) akt[track]->data;
          do {
            o = not->octave;
            o = (1 << o);
            p = not->pitch;
            if (process[p] & 0x40) {
              /* hit whole tone */
            if (process[p] & o) {
              /* sign active */
              other_niveau(cross, o, p, &osecond, &psecond);
              if (process[psecond] & 0x40) {
                /* hit whole tone */
                if (process[psecond] & osecond) {
                  /* sign active */
                  osecond >>= 1;
                  while (osecond) {
                    osecond >>= 1;
                    not->note_pos += 7;
                  }
                  not->note_pos += pitch_to_pos[psecond];
                }
                else {
                  process[p] -= o;
                  o >>= 1;
                  while (o) {
                    o >>= 1;
                    not->note_pos += 7;
                  }
                  not->note_pos += pitch_to_pos[p];
                  not->note_pos |= 0xc0;
                }
              }
              else {
```

```
          /* hit half tone */
          process[p] -= o;
          o >>= 1;
          while (o) {
            o >>= 1;
            not->note_pos += 7;
          }
          not->note_pos += pitch_to_pos[p];
          not->note_pos |= 0xc0;
        }
      }
      else {
        o >>= 1;
        while (o) {
          o >>= 1;
          not->note_pos += 7;
        }
        not->note_pos += pitch_to_pos[p];
      }
      }
      else {
      /* hit half tone */
      other_niveau(cross, o, p, &osecond, &psecond);
      rem_o = osecond;
      osecond >>= 1;
      while (osecond) {
        osecond >>= 1;
        not->note_pos += 7;
      }
      not->note_pos += pitch_to_pos[psecond];
      if (!(process[psecond] & rem_o)) {
        process[psecond] |= rem_o;
        if (cross) not->note_pos |= 0x40;
        else not->note_pos |= 0x80;
      }
      }
      not = not->accord_next;
    } while (not);
    break;
  case ´k´: /* dnpsmark3 */
    ks = (struct KeySig *) akt[track]->data;
    if ((ks->sign) & 0x80) cross = 0;
    else cross = 1;
    for (i=0; i<12; i++) process[i] &= 0x40;
    signhelp = (ks->sign) & 0x7;
    while (signhelp--)
      if (cross) process[crosses[signhelp]] |= 0xbf;
```

```
            else process[bs[signhelp]] |= 0xbf;
          break;
      case ´b´: /* dnpsmark4 */
          for(i=0; i<12; i++) {
          process[i] &= 0xc0;
          if (process[i] & 0x80) process[i] |= 0x3f;
          }
          break;
      default : break;
    }
    akt[track] = akt[track]->next;
  }
}

void slur_geometry(t)
  int t;
{
  unsigned long endtime;
  char used[42], sp, nothalt;
  int i;
  int minimum, maximum;
  char beginposmin, beginposmax, endposmin, endposmax;

  akt[t] = begin[t];
  while (akt[t]) {
    /* sgeomark1 */
    if ((akt[t]->marker) == ´n´) {
      not = (struct Note *) akt[t]->data;
      if (((not->slur_nr) > 0) && (((not->slur_pos) & 0x3f) == 0)) {
        /* first slur note found */

        /* sgeomark2: next are for setting endtime */
        dynhelp = akt[t];
        not3 = not;
        while (!(((not3->flags) & 0x80) &&
               ((not->slur_nr) == (not3->slur_nr)))) {
          dynhelp = dynhelp->next;
          while ((dynhelp->marker) != ´n´) dynhelp = dynhelp->next;
          not3 = (struct Note *) dynhelp->data;
        }
        endtime = dynhelp->time;

        /* sgeomark3 */
        for (i=0; i<42; i++) used[i] = 0;
        maximum = 0;
        minimum = 41;
        not2 = not;
```

```
dynhelp = akt[t];
beginposmin = (not->note_pos) & 0x3f;
not3 = not2;
while (not3->accord_next) not3 = not3->accord_next;
beginposmax = (not3->note_pos) & 0x3f;
nothalt = 1;
while (nothalt) {
  if ((not->slur_nr) == (not2->slur_nr)) {
    not3 = not2;
    while (not3->accord_next) not3 = not3->accord_next;
    if (minimum > ((not2->note_pos) & 0x3f))
        minimum = (not2->note_pos) & 0x3f;
    if (maximum < ((not3->note_pos) & 0x3f))
        maximum = (not3->note_pos) & 0x3f;
  }
  if (((dynhelp->time) != endtime) &&
      ((dynhelp->time) != akt[t]->time)) {
    not3 = not2;
    while (not3->accord_next) not3 = not3->accord_next;
    for (i=((not2->note_pos) & 0x3f);i<=((not3->note_pos) & 0x3f);i++)
      used[i] = 1;
  }
  if (((not2->flags) & 0x80) && ((not->slur_nr) == (not2->slur_nr)))
      nothalt = 0;
  else {
    dynhelp = dynhelp->next;
    while ((dynhelp->marker) != ´n´) dynhelp = dynhelp->next;
    not2 = (struct Note *) dynhelp->data;
  }
}
/* sgeomark4 */
endposmin = (not2->note_pos) & 0x3f;
not3 = not2;
while (not3->accord_next) not3 = not3->accord_next;
endposmax = (not3->note_pos) & 0x3f;
if (maximum < 38) maximum += 3;
else maximum = 41;
if (minimum > 3) minimum -= 3;
else minimum = 0;
while ((maximum < 41) && (used[maximum])) maximum++;
while ((minimum > 0) && (used[minimum])) minimum--;
if ((((maximum - beginposmax) + (maximum - endposmax)) <=
    ((beginposmin - minimum) + (endposmin - minimum))) &&
     (maximum != 41)) {
  sp = maximum;
  sp |= 0x80;
}
```

```
else sp = minimum;
/* sgeomark5 */
dynhelp = akt[t];
not2 = not;
nothalt = 1;
while (nothalt) {
  if (((not2->flags) & 0x80) && ((not->slur_nr) == (not2->slur_nr)))
     nothalt = 0;
  if ((not2->slur_nr) == (not->slur_nr)) {
    not3 = not2;
    while (not3->accord_next) {
      not3->slur_pos &= 0x40;
      not3->slur_pos |= sp;
      not3 = not3->accord_next;
    }
    not3->slur_pos &= 0x40;
    not3->slur_pos |= sp;

    if (((sp & 0x80) &&
        ((sp & 0x3f) <= (LAMBDA + ((not3->note_pos) & 0x3f))))
        || ((!(sp & 0x80)) &&
         (((sp & 0x3f) + LAMBDA) <= ((not2->note_pos) & 0x3f)))) {
      /* set lower stem */
      not3 = not2;
      while (not3) {
        not3->flags |= 0x400;
        not3 = not3->accord_next;
      }
    }
  }
  else {
    not3 = not2;
    while (not3->accord_next) not3 = not3->accord_next;

    /* if neccesary set lower stems */
    if (((sp & 0x80) &&
        (((sp & 0x3f) + SLURWIDE ) <=
            (LAMBDA + ((not3->note_pos) & 0x3f)))
         && (((sp & 0x3f) + SLURWIDE ) > ((not2->note_pos) & 0x3f)))
        || ((!(sp & 0x80)) &&
         (((sp & 0x3f) - SLURWIDE ) <=
              (LAMBDA + ((not3->note_pos) & 0x3f)))
         && (((sp & 0x3f) - SLURWIDE ) > ((not2->note_pos) & 0x3f)))) {
           not3 = not2;
           while (not3) {
             not3->flags |= 0x400;
             not3 = not3->accord_next;
```

```
                  }
              }
            }

            if (nothalt) {
              dynhelp = dynhelp->next;
              while ((dynhelp->marker) != ´n´) dynhelp = dynhelp->next;
              not2 = (struct Note *) dynhelp->data;
            }
          }
        }
      }
      akt[t] = akt[t]->next;
    }
}

void reset_treat_info(track)
  int track;
{
  akt[track] = begin[track];
  while (akt[track]) {
    if ((akt[track]->marker) == ´n´) {
      not = (struct Note *) akt[track]->data;
      not->treated = 0;
    }
    akt[track] = akt[track]->next;
  }
}

void set_slur_regions(track)
  int track;
{
  short region, count, sum;

  akt[track] = begin[track];
  count = 1;
  sum = 0;
  while (akt[track]) {
    if ((akt[track]->marker) == ´n´) {
      not = (struct Note *) akt[track]->data;
      if ((not->slur_nr) && (!(not->treated))) {
        /* now look forward and treat */
        dynhelp = akt[track];
        not2 = not;
        while (!(((not2->slur_nr) == (not->slur_nr)) &&
               ((not2->flags) & 0x80))) {
          if ((not2->slur_nr) == (not->slur_nr)) {
```

```
            count++;
            sum += (not2->note_pos) & 0x3f;
          }
          dynhelp = dynhelp->next;
          while ((dynhelp->marker) != ´n´) dynhelp = dynhelp->next;
          not2 = (struct Note *) dynhelp->data;
        }
        /* last note of current slur found */
        sum += (not2->note_pos) & 0x3f;
        region = (not2->flags) & 0x300;
        if ((!region) && ((sum / count) > 20)) region = 0x100;
        if (!region) region = 0x200;
        /* set slur-begin flag */
        not->slur_pos |= 0x40;
        dynhelp = akt[track];
        not2 = not;
        while (!(((not2->slur_nr) == (not->slur_nr)) &&
               ((not2->flags) & 0x80))) {
          if ((not2->slur_nr) == (not->slur_nr)) {
            not2->treated = 1;
            not2->flags |= region;
            /* correct the begin flag */
            if (not2 != not) {
              while (not2) {
                not2->slur_pos &= 0xbf;
                not2 = not2->accord_next;
              }
            }
          }
          dynhelp = dynhelp->next;
          while ((dynhelp->marker) != ´n´) dynhelp = dynhelp->next;
          not2 = (struct Note *) dynhelp->data;
        }
        /* set all value to the last note of slur */
        not2->treated = 1;
        not2->flags |= region;
        while (not2) {
          not2->slur_pos &= 0xbf;
          not2 = not2->accord_next;
        }
      }
    }
    akt[track] = akt[track]->next;
  }
}

void set_beam_regions(track)
```

```
  int track;
{
  short region, count, sum;

  akt[track] = begin[track];
  count = 0;
  sum = 0;
  while (akt[track]) {
    if ((akt[track]->marker) == ´n´) {
      /* sbrmark1 */
      not = (struct Note *) akt[track]->data;
      if ((not->beam_nr) && (!(not->treated))) {
        /* now look forward and treat */
        dynhelp = akt[track];
        not2 = not;
        while (!(((not2->beam_nr) == (not->beam_nr)) &&
              ((not2->flags) & 0x40))) {
          if ((not2->beam_nr) == (not->beam_nr)) {
            count++;
            sum += (not2->note_pos) & 0x3f;
          }
          dynhelp = dynhelp->next;
          while ((dynhelp->marker) != ´n´) dynhelp = dynhelp->next;
          not2 = (struct Note *) dynhelp->data;
        }
        /* last note of current beam found */
        region = (not2->flags) & 0x300;
        if (!region) {
          /* last note has no concrete region -> set it! */
          if ((sum / count) > 20) region = 0x100;
          else region = 0x200;
          not2->flags |= region;
        }
        not2->treated = 1;

        /* sbrmark2 */
        /* set the region value to all other notes of current beam */
        dynhelp = akt[track];
        not2 = not;
        while (!(((not2->beam_nr) == (not->beam_nr)) &&
              ((not2->flags) & 0x40))) {
          if ((not2->beam_nr) == (not->beam_nr)) {
            not2->treated = 1;
            not2->flags |= region;
          }
          dynhelp = dynhelp->next;
          while ((dynhelp->marker) != ´n´) dynhelp = dynhelp->next;
```

```
          not2 = (struct Note *) dynhelp->data;
        }
      }
    }
    akt[track] = akt[track]->next;
  }
}

void ins_helprest(start, fine, callnumber)
  unsigned long start, fine;
  unsigned int callnumber;
{
  if ((start >= fine) || ((start + 0xffffff) < fine)) {
    fprintf(stderr, "warning: %d --> extreme rest found: start %lx end %lx\n",
            callnumber, start, fine);
  }

  hres1 = ank;
  hres2 = NULL;
  while ((hres1) && ((hres1->begin) <= start)) {
    hres2 = hres1;
    hres1 = hres1->next;
  }
  if ((hres3 = (struct helprest *) malloc (sizeof(struct helprest))) == NULL) {
    fprintf(stderr, "malloc error\n");
    exit(10);
  }
  hres3->begin = start;
  hres3->end = fine;
  hres3->next = hres1;

  if (hres2) hres2->next = hres3;
  else ank = hres3;
}


void deal_with_takt(tp, bt, attention, fac, reg)
  unsigned long bt;
  int tp;
  char attention, reg;
  short fac;
{
  int changes;
  struct Note *lastSnote;
  char sign, helpsign, nothing;
  unsigned long lastnotetime;
```

```
changes = 1;
nothing = 1;
while (changes) {
  lastnotetime = bt;
  changes = 0;
  d2 = d;
  while ((d2) && ((d2->marker) != ´b´)) {
    if ((d2->marker) == ´n´) {
            not = (struct Note *) d2->data;
            sign = 0;  /* allow processing */
            if ((not->slur_nr) && (!((not->slur_pos) & 0x40)) &&
                (bt != (d2->time))) sign = 1;
            if ((attention) && (!((not->flags) & 0x300))) {
              d3 = d2->next;
              helpsign = 1;
              while (d3 && ((d3->marker) != ´b´) && helpsign) {
                if ((d3->marker) == ´n´) {
                  not2 = (struct Note *) d3->data;
                  if ((not2->flags) & 0x200) {
                    if ((not->end) > (d3->time)) sign = 1;
                    helpsign = 0;
            }
            else d3 = d3->next;
          }
          else d3 = d3->next;
        }
      }
                  if ((!(not->treated)) && ((attention == 0) ||
                      (!((not->flags) & 0x100))) && (!sign)) {
                    if ((d2->time) > lastnotetime) {
                      ins_helprest(lastnotetime, (d2->time), 1);
        }
                    nothing = 0;
                    not->treated = 1;
                    if (attention || (reg == 2)) (not->flags) |= 0x200;
                    else (not->flags) |= 0x100;
                    lastnotetime = not->end;
                    if ((not->slur_nr) && (!((not->flags) & 0x80))) {
                      d2 = d2->next;
          lastSnote = not;
                      helpsign = 1;
                      while (d2 && ((d2->marker) != ´b´) && helpsign) {
                        if ((d2->marker) == ´n´) {
                          not2 = (struct Note *) d2->data;
                          if ((not2->slur_nr) == (not->slur_nr)) {
                            not2->treated = 1;
              lastSnote = not2;
```

```
                              lastnotetime = not2->end;
                        if (attention || (reg == 2)) (not2->flags) |= 0x200;
                        else (not2->flags) |= 0x100;
                        if ((not2->flags) & 0x80) helpsign = 0;
                      }
                    }
                    d2 = d2->next;
                  }
                  if ((!(d2)) && (!((lastSnote->flags) & 0x80)))
                     lastSnote->flags |= 0x80;
                }
                changes++;
                while ((d2) && ((d2->marker) != ´b´) && ((d2->time) < lastnotetime))
                   d2 = d2->next;
              }
              else d2 = d2->next;
            }
            else d2 = d2->next;
          }
          if ((d2) && (changes) && ((d2->time) > lastnotetime)) {
            ins_helprest(lastnotetime, (d2->time), 7);
          }
        }
        if (nothing) {
          if (d2) ins_helprest(bt, (d2->time), 2);
          else {
            uslonghelp = fac * tp;
            uslonghelp += bt;
            ins_helprest(bt, uslonghelp, 3);
          }
        }
      }

      void process_helprest(perq, tr, bt, tp, reg)
        short reg;
        int tp, tr, perq;
        unsigned long bt;
      {
        unsigned long tphelp, dur, i;
        unsigned long b;

        /* debug output */
        hr1=ank;
        while (hr1) {
          hr1 = hr1->next;
        }
```

```
/* first part realizes linking of rests */
hr1 = ank;
while (hr1) {
  hr2 = hr1->next;
  hr3 = hr1;
  while (hr2) {
    while (hr2 && ((hr2->begin) < (hr1->end))) {
      hr3 = hr2;
      hr2 = hr2->next;
    }
    if (hr2 && ((hr1->end) == (hr2->begin))) {
      /* link rests */
      hr1->end = hr2->end;
      hr3->next = hr2->next;
      free(hr2);
      hr2 = hr3->next;
    }
    else hr2 = NULL;
  }
  hr1 = hr1->next;
}

/* second part splits notes depending on tp */
hr1 = ank;
while (hr1) {
  b = bt;
  while (b < (hr1->begin)) b += tp;
  if ((b > (hr1->begin)) && (b < (hr1->end))) {
    ins_helprest(b, (hr1->end), 4);
    hr1->end = b;
  }
  else {
    if (b == (hr1->begin)) {
      tphelp = 4*tp;
      dur = (hr1->end) - (hr1->begin);
      while ((dur < tphelp) && (tphelp > tp)) tphelp = (tphelp >> 1);
      if (dur > tphelp) {
        uslonghelp = (hr1->begin) + tphelp;
        ins_helprest(uslonghelp, (hr1->end), 5);
        hr1->end = uslonghelp;
      }
    }
  }
  hr1 = hr1->next;
}

/* third part calculates rest-values and creates rest-events */
```

```
hr1 = ank;
dy2 = NULL;
dy = begin[tr];
while (hr1) {
  dur = (hr1->end) - (hr1->begin);
  i = 0;
  while ((i<6) && ((NORMFAKTOR * dur) > (faktors[i*3] * perq))) i++;
  if ((re = (struct Rest *) malloc (sizeof(struct Rest))) == NULL) {
    fprintf(stderr, "malloc error\n");
    exit(10);
  }
  re->flg = 0;
  re->position = reg;
  if ((NORMFAKTOR * dur) == (faktors[i*3] * perq)) re->kind = i;
  else {
    if (i) {
      re->kind = i-1;
      uslonghelp = (faktors[((i-1)*3)]) * perq;
      uslonghelp = uslonghelp / 16;
      uslonghelp += (hr1->begin);
      ins_helprest(uslonghelp, (hr1->end), 6);
    }
    else re->kind = 0;
  }
  while (dy && (((dy->time) < (hr1->begin)) ||
    (((dy->time) == (hr1->begin)) && ((dy->marker) != ´n´)))) {
    dy2 = dy;
    dy = dy->next;
  }
  if ((bp = (struct bind *) malloc (sizeof(struct bind))) == NULL) {
    fprintf(stderr, "malloc error\n");
    exit(10);
  }
  bp->marker = ´r´;
  bp->data = (void *) re;
  bp->time = (hr1->begin);
  if (dy2) dy2->next = bp;
  else begin[tr] = bp;
  bp->next = dy;
  dy2 = bp;
  hr1 = hr1->next;
}

/* fourth part deletes the helprest-list */
hr1 = ank;
while (hr1) {
  hr2 = hr1->next;
```

```
    free(hr1);
    hr1 = hr2;
  }
  ank = NULL;
}


void detect_rests(t, regio, pq)
  int t;  /* number of track */
  int pq;  /* per quarter */
  char regio;
{
  unsigned long barrtime;
  int taktpart;  /* logical part of takt, depends on nom */
  short factor;

  /* drmark1 */
  akt[t] = begin[t];
  dynhelp = akt[t];
  ank = NULL;
  factor = 4;  /* default is 4/4 */
  taktpart = pq;  /* default is 4/4 */
  barrtime = 0;  /* time of last barre */
  while (akt[t]) {
    /* jump to next bar */
    dynhelp2 = dynhelp;
    while ((dynhelp2) && ((dynhelp2->marker) != ´b´)) {
      if ((dynhelp2->marker) == ´t´) {
        ts = (struct TimeSig *) dynhelp2->data;
        taktpart = pq * 4 /(ts->denom);
        factor = ts->nom;
      }
      dynhelp2 = dynhelp2->next;
    }
    if (dynhelp) {
      /* drmark2 */
      d = dynhelp;
      deal_with_takt(taktpart, barrtime, (regio == 0), factor, regio);
      if (regio == 1)
        process_helprest(pq, t, barrtime, taktpart, 0x40);
      else
        process_helprest(pq, t, barrtime, taktpart, 0x80);
      if (!regio) {
        deal_with_takt(taktpart, barrtime, 0, factor, 1);
        process_helprest(pq, t, barrtime, taktpart, 0x40);
      }
    }
```

```
    akt[t] = dynhelp2;
    if (akt[t]) {
      dynhelp = akt[t]->next;
      barrtime = akt[t]->time;
    }
  }
}

void beam_geometry(tr)
  int tr;    /* number of track */
{
  char crit, usedpos[42], helpbpos;
  char maxone, helpmaxone, pos, maxb, minb, bestpos, besthigh, bestupperflag;
  int upp, low;
  short slope;
  short measurecounter;  /* for debugging with gdb */
  char orientflag, valid;
  char lowcut, uppcut, realpos, upperflag;
  short sl;
  long sbpos[42];    /* for storing beam and slur numbers */
  unsigned long remtime;
  short i, middle;
  unsigned short counter;
  int loop;
  double arg, helparg;
  struct bind *lastdealt, *last;
  struct Note *notehelp, *prev_note, *helpn, *nohe;

  /* bgmark1 */
  akt[tr] = begin[tr];
  measurecounter = 1;
  dynhelp3 = begin[tr];  /* first event in takt */
  while (akt[tr]) {
    if ((akt[tr]->marker) == 'b') {
      /* bgmark2 */
      dynhelp3 = akt[tr]->next;
      measurecounter++;
    }
    if ((akt[tr]->marker) == 'n') {
      /* bgmark3 */
      not = (struct Note *) akt[tr]->data;
      if ((!(not->treated)) && (not->beam_nr)) {
        /* bgmark4: found begin of beam */
        maxb = (not->note_pos) & 0x3f;
        minb = maxb;
        not->treated = 1;
        not->beam_pos = 0x40;
```

```
/* reset sbpos-array */
for (i=0; i<42; i++) sbpos[i] = 0;
/* read until first beam note and set slur and beam positions */
dynhelp2 = dynhelp3;
dynhelp = akt[tr];
while (dynhelp2 != dynhelp) {
  if ((dynhelp2->marker) == ´n´) {
    not3 = (struct Note *) dynhelp2->data;
    if ((not3->slur_nr) && (!((not3->flags) & 0x80))) {
      for (loop= ((not3->slur_pos) & 0x3f) - 2;
           loop < ((not3->slur_pos) & 0x3f) + 3; loop++)
        if ((loop >= 0) && (loop <=41)) sbpos[loop] = (not3->slur_nr);
    }
    else {
      if (not3->slur_nr) {
        for (loop= ((not3->slur_pos) & 0x3f) - 5;
             loop < ((not3->slur_pos) & 0x3f) + 5; loop++)
          if ((loop >= 0) && (loop <=41) &&
              (sbpos[loop] == (not3->slur_nr))) sbpos[loop] = 0;
      }
    }
    if ((not3->beam_nr) && (not3->treated) &&
        (!((not3->flags) & 0x40))) {
      helpbpos = (not3->beam_pos) & 0x3f;
      if ((not3->beam_pos) & 0x80) helpbpos += UPPLOWSHIFT;
      else {
        if (helpbpos > UPPLOWSHIFT) helpbpos -= UPPLOWSHIFT;
        else helpbpos = 0;
      }
      if (helpbpos > 41) helpbpos = 41;
      for (loop = helpbpos -2; loop < helpbpos + 3; loop++)
        if ((loop >= 0) && (loop <=41)) sbpos[loop] = (not3->beam_nr);
    }
    else   if ((not3->beam_nr) && (not3->treated)) {
        helpbpos = (not3->beam_pos) & 0x3f;
        if ((not3->beam_pos) & 0x80) helpbpos += UPPLOWSHIFT;
        else {
          if (helpbpos > UPPLOWSHIFT) helpbpos -= UPPLOWSHIFT;
          else helpbpos = 0;
        }
        if (helpbpos > 41) helpbpos = 41;
        for (loop = helpbpos -5; loop < helpbpos + 6; loop++)
          if ((loop >= 0) && (loop <=41) &&
              (sbpos[loop] == (not3->beam_nr))) sbpos[loop] = 0;
      }
  }
  dynhelp2 = dynhelp2->next;
```

```
}

/* bgmark5 */
/* now start with processing the beam */
lastdealt = akt[tr];
/* reset the valid area array */
for (i=0; i<42; i++) usedpos [i] = 0;

/* specify the first valid region */
upp = not->note_pos & 0x3f;
while ((upp < 41) && (!(sbpos[upp]))) upp++;
low = not->note_pos & 0x3f;
while ((low > 0) && (!(sbpos[low]))) low--;
for (i=low; i<=upp; i++) usedpos [i] = 1;
usedpos [not->note_pos & 0x3f] = 2;

remtime = dynhelp->time;
counter = 0;

/* bgmark6 */
/* jump to last possible note of current beam */
crit = 1;
last = dynhelp2;
nohe = (struct Note *) dynhelp2->data;
if ((nohe->flags) & 0x40) crit = 0;
while (crit) {
  dynhelp = dynhelp->next;
  if ((dynhelp) && ((dynhelp->time) > remtime) &&
       ((dynhelp->marker == ´n´) || (dynhelp->marker == ´r´))) {
    counter += 2;
    remtime = dynhelp->time;
  }
  if ((dynhelp) && ((dynhelp->marker) == ´n´)) {
    not2 = (struct Note *) dynhelp->data;
    if ((not->beam_nr) == (not2->beam_nr)) {
      if (!usedpos[(not2->note_pos) & 0x3f]) {
        /* have to split beam */
        dynhelp = lastdealt;
        crit = 0;
        if (akt[tr] == lastdealt) {
          not->beam_nr = 0;
          not->beam_pos = 0;
          not2 = not;
        }
        else {
          not3 = (struct Note *) lastdealt->data;
          not3->flags |= 0x40;
```

```
        if (!((not2->flags) & 0x40)) {
          not2->beam_pos |= 0x40;
          global++;
          not2->beam_nr = global;
          lastdealt = lastdealt->next;
          while (lastdealt) {
            if ((lastdealt->marker) == ´n´) {
              notehelp = (struct Note *) lastdealt->data;
              if ((notehelp->beam_nr) == (not->beam_nr)) {
                notehelp->beam_nr = global;
                if ((notehelp->flags) & 0x40) lastdealt = NULL;
              }
            }
            if (lastdealt) lastdealt = lastdealt->next;
          }
        }
        else {
          not2->flags &= 0xffbf;
          not2->beam_nr = 0;
        }
        not2 = not3;
      }
    }
    else {
      /* deal with printable beam note */
      last = dynhelp;
      lastdealt = dynhelp;
      helpn = not2;
      while (helpn->accord_next) helpn = helpn->accord_next;
      if (((helpn->note_pos) & 0x3f) > maxb)
         maxb = (helpn->note_pos) & 0x3f;
      if (((not2->note_pos) & 0x3f) < minb)
         minb = (not2->note_pos) & 0x3f;
      if ((not2->flags) & 0x40) crit = 0;
      not2->treated = 1;
      usedpos[(not2->note_pos) & 0x3f] = 2;
      if (not2->slur_nr) {
        if ((not2->slur_pos) & 0x80)
          for (i=((not2->slur_pos) & 0x3f) - 2; i<42; i++)
             usedpos[i] = 0;
        else
          for (i=((not2->slur_pos) & 0x3f) + 2; i>=0; i--)
             usedpos[i] = 0;
      }
    }
  }
```

```
    else {
      if (not2->slur_nr) {
        upp = 0;
        for (i=(not2->slur_pos & 0x3f); i<42; i++)
          if (usedpos[i]) upp++;
        low = 0;
        for (i=(not2->slur_pos & 0x3f); i>=0; i--)
          if (usedpos[i]) low++;
        if (upp > low)
          for (i=(not2->slur_pos & 0x3f); i>=0; i--) usedpos[i] = 0;
        else
          for (i=(not2->slur_pos & 0x3f); i<42; i++) usedpos[i] = 0;
      }
      if (not2->treated) {
        helpbpos = (not2->beam_pos) & 0x3f;
        if ((not2->beam_pos) & 0x80) helpbpos += UPPLOWSHIFT;
        else  if (helpbpos > UPPLOWSHIFT)
            helpbpos -= UPPLOWSHIFT;
          else helpbpos = 0;
        for (loop=helpbpos-2; loop < (helpbpos + 3); loop++)
          if ((loop >= 0) && (loop <= 41)) usedpos[loop] = 0;
      }
      usedpos[(not2->note_pos) & 0x3f] = 2;
    }
  }
  if (dynhelp == NULL) crit = 0;
}

if ((last == dynhelp2) &&
    ((dynhelp == NULL) || ((nohe->flags) & 0x40))) {
  not->beam_nr = 0;
}
else {
  if ((dynhelp == NULL) && (last != dynhelp2)) {
    not2 = (struct Note *) last->data;
    not2->flags |= 0x40;
    dynhelp = last;
  }
}

/* bgmark7 */
/* now dynhelp and not2 mark last note of current beam */

if (not->beam_nr) {

  /* store the difference between the position of the last note  */
  /* and the position of the first note        */
```

```
if (((not2->note_pos) & 0x3f) >= ((not->note_pos) & 0x3f)) {
  orientflag = 1;
  helpmaxone = (((not2->note_pos) & 0x3f) -
               ((not->note_pos) & 0x3f));
}
else {
  orientflag = 0;
  helpmaxone = (((not->note_pos) & 0x3f) -
               ((not2->note_pos) & 0x3f));
}

/* as a value for a middle position */
middle = maxb + minb;

/* set position */
pos = 0x70;

/* depending on middle search for a nice position */
if ((((not->flags) & 0x100) && (middle > (2 * VIOLINEMID))) ||
    (((not->flags) & 0x200) && (middle > (2 * BASSMID)))) {
  /* try to find a lower beam */
  upperflag = 0;
  i = minb -1;
  maxone = 0;
  if (maxb > BEAMDELTA) lowcut = maxb - BEAMDELTA;
  else lowcut = 0;
  if ((minb > BEAMTOLMAX) && (lowcut < (minb - BEAMTOLMAX)))
     lowcut = minb - BEAMTOLMAX;
  if (lowcut > helpmaxone) lowcut -= helpmaxone;
  else lowcut = 0;
  while ((usedpos[i] == 1) && (i > lowcut)) {
    maxone++;
    i--;
  }

  if (maxone > ((helpmaxone >> 1) + BEAMTOLERANCE)) {
    if (minb > BEAMTOLERANCE) realpos = minb - BEAMTOLERANCE;
    else realpos = 0;
    if (orientflag) {
      if (realpos > (helpmaxone >> 1))
         realpos -= (helpmaxone >> 1);
      else realpos = 0;
    }
    pos = realpos + UPPLOWSHIFT;
    if (pos > 41) pos = 41;
  }
```

```
    else {
      /* try to find an upper beam */
      upperflag = 1;
      bestpos = minb - maxone;
      bestupperflag = 0;
      if (maxone > helpmaxone)
         realpos = bestpos + (helpmaxone >> 2);
      else realpos = bestpos;
      if (realpos > 41) realpos = 41;
      besthigh = maxone;
      i = maxb + 1;
      maxone = 0;
      uppcut = minb + BEAMDELTA;
      if (uppcut > 41) uppcut = 41;
      if (uppcut > (maxb + BEAMTOLMAX)) uppcut = maxb + BEAMTOLMAX;
      uppcut += helpmaxone;
      if (uppcut > 41) uppcut = 41;
      while ((usedpos[i] == 1) && (i < uppcut)) {
        maxone++;
        i++;
      }

      if (maxone > ((helpmaxone >> 1) + BEAMTOLERANCE)) {
        realpos = maxb + BEAMTOLERANCE;
        if (realpos > 41) realpos = 41;
        if (!orientflag) realpos += (helpmaxone >> 1);
        if (realpos > 41) realpos = 41;
        if (realpos > UPPLOWSHIFT) pos = realpos - UPPLOWSHIFT;
        else pos = 0;
      }
      else {
        if (maxone > besthigh) {
          besthigh = maxone;
          bestupperflag = 1;
          bestpos = maxb + maxone;
          if (maxone > helpmaxone) {
            if (bestpos > (helpmaxone >> 2))
              realpos = bestpos - (helpmaxone >> 2);
            else
              realpos = 0;
          }
          else realpos = bestpos;
        }
      }
    }
  }
  else {
```

```
/* try to find an upper beam */
upperflag = 1;
i = maxb + 1;
maxone = 0;
uppcut = minb + BEAMDELTA;
if (uppcut > 41) uppcut = 41;
if (uppcut > (maxb + BEAMTOLMAX)) uppcut = maxb + BEAMTOLMAX;
uppcut += helpmaxone;
if (uppcut > 41) uppcut = 41;
while ((usedpos[i] == 1) && (i < uppcut)) {
  maxone++;
  i++;
}

if (maxone > ((helpmaxone >> 1) + BEAMTOLERANCE)) {
  realpos = maxb + BEAMTOLERANCE;
  if (realpos > 41) realpos = 41;
  if (!orientflag) realpos += (helpmaxone >> 1);;
  if (realpos > 41) realpos = 41;
  if (realpos > UPPLOWSHIFT) pos = realpos - UPPLOWSHIFT;
  else pos = 0;
}
else {
  /* try to find a lower beam */
  upperflag = 0;
  bestpos = maxb + maxone;
  bestupperflag = 1;
  if (maxone > helpmaxone) {
    if (bestpos > (helpmaxone >> 2))
      realpos = bestpos - (helpmaxone >> 2);
    else
      realpos = 0;
  }
  else realpos = bestpos;
  besthigh = maxone;
  i = minb -1;
  maxone = 0;
  if (maxb > BEAMDELTA) lowcut = maxb - BEAMDELTA;
  else lowcut = 0;
  if ((minb > BEAMTOLMAX) && (lowcut < (minb - BEAMTOLMAX)))
    lowcut = minb - BEAMTOLMAX;
  if (lowcut > helpmaxone) lowcut -= helpmaxone;
  else lowcut = 0;
  while ((usedpos[i] == 1) && (i > lowcut)) {
    maxone++;
    i--;
  }
```

```
      if (maxone > ((helpmaxone >> 1) + BEAMTOLERANCE)) {
        if (minb > BEAMTOLERANCE) realpos = minb - BEAMTOLERANCE;
        else realpos = 0;
        if (orientflag) {
          if (realpos > (helpmaxone >> 1))
             realpos -= (helpmaxone >> 1);
          else realpos = 0;
        }
        pos = realpos + UPPLOWSHIFT;
        if (pos > 41) pos = 41;
      }
      else {
        if (maxone > besthigh) {
          besthigh = maxone;
          bestupperflag = 0;
          bestpos = minb - maxone;
          if (maxone > helpmaxone)
             realpos = bestpos + (helpmaxone >> 2);
          else realpos = bestpos;
          if (realpos > 41) realpos = 41;
        }
      }
    }
  }

  /* check middle region for setting the beam position  */
  /* (only if not already found one)      */

  valid = 1;
  if (pos == 0x70) {
    /* search a nice beam position between minb and maxb  */
    /* only when MINBEAMBROAD is valid      */
    if ((maxb - minb) < MINBEAMBROAD) valid = 0;

    if (valid) {
      if ((((not->note_pos) & 0x3f) <= bestpos) && orientflag) {
        if (bestpos > helpmaxone) bestpos -= helpmaxone;
        else bestpos = 0;
      }
      if ((((not->note_pos) & 0x3f) > bestpos) && (!orientflag)) {
        bestpos += helpmaxone;
        if (bestpos > 41) bestpos = 41;
      }
      maxone = 0;
      for (i=maxb; i>=minb; i--) {
        if (usedpos[i] == 1) maxone++;
        else {
```

```
          if (maxone > besthigh) {
            besthigh = maxone;
            bestpos = i;
            if (besthigh > helpmaxone) {
              bestpos += ((besthigh - helpmaxone) >> 1);
              realpos = bestpos + (helpmaxone >> 2);
              if (!orientflag) bestpos += helpmaxone;
              if (bestpos > 41) bestpos = 41;
              if (realpos > 41) realpos = 41;
            }
            else {
              realpos = bestpos + (besthigh >> 2);
              if (!orientflag) bestpos += besthigh;
            }
            if (((not->note_pos) & 0x3f) <= realpos)
              bestupperflag = 1;
            else bestupperflag = 0;
          }
          maxone = 0;
        }
      }
      pos = bestpos;
      if (bestupperflag) {
        if (pos > UPPLOWSHIFT) pos -= UPPLOWSHIFT;
        else pos = 0;
        upperflag = 1;
      }
      else {
        upperflag = 0;
        pos += UPPLOWSHIFT;
        if (pos > 41) pos = 41;
      }
      maxone = besthigh;
    }
  }

  /* bgmark8 */
  if (valid) {
    if (!counter) counter = 1;
    if (maxone >= helpmaxone) arg = helpmaxone;
    else arg = maxone;

    if (orientflag) slope = 1;
    else slope = -1;

    helparg = SLOPEFACTOR * counter;
    arg /= helparg;
```

```
/* the following lines substitute a function atan */
if (arg > 0.94) slope *= 9;
else  if (arg > 0.77) slope *= 8;
  else  if (arg > 0.64) slope *= 7;
    else  if (arg > 0.52) slope *= 6;
      else  if (arg > 0.41) slope *= 5;
        else  if (arg > 0.31) slope *= 4;
          else  if (arg > 0.21) slope *= 3;
            else  if (arg > 0.12) slope *= 2;
              else  if (arg < 0.04) slope = 0;

/* modify position */
if (pos > 41) pos = 41;
else {
  if (pos & 0x80) pos = 0;
}

/* setting of the upp/low-flag */
if (upperflag) pos |= 0x80;

/* generate a flag-mask for slope */
if (slope < 0) {
  sl = 0;
  sl |= 0x8000;
  slope = slope * (-1);
}
else sl = 0;

slope <<= 11;

/* bgmark9 */
/* set the values to all notes of current beam */
sl |= slope;
dynhelp = akt[tr];
crit = 1;
prev_note = NULL;
while (crit) {
  if ((dynhelp->marker) == ´n´) {
    not3 = (struct Note *) dynhelp->data;
    if ((not->beam_nr) == (not3->beam_nr)) {
      if (prev_note) {
        switch ((not3->flags) & 0x38) {
          case 0x0 : (prev_note->next_beamn_kind)++;
          case 0x8 : (prev_note->next_beamn_kind)++;
          case 0x10: (prev_note->next_beamn_kind)++;
          case 0x18: (prev_note->next_beamn_kind)++;
```

```
                  break;
              default  : break;
            }
          }
          prev_note = not3;
          not3->beam_pos |= pos;
          not3->flags |= sl;
          if ((not3->flags) & 0x40) {
            not3->next_beamn_kind = 0;
            crit = 0;
          }
        }
        else {
          if ((!(not3->slur_nr)) &&
              ((!(not3->beam_nr) || (!(not3->treated)))) &&
              (((not3->flags) & 0x38) != 0x30)) {
            helpn = not3;
            while (helpn->accord_next) helpn = helpn->accord_next;

            /* set lower stems if nessesary */
            if ((realpos > ((not3->note_pos) & 0x3f)) &&
             ((LAMBDA+((helpn->note_pos) & 0x3f)+2) >= realpos)) {
              helpn = not3;
              while (helpn) {
                helpn->flags |= 0x400;
                helpn = helpn->accord_next;
              }
            }
          }
        }
      }
      if (crit) dynhelp = dynhelp->next;
    }
  }
  else {
    /* bgmark10 */
    /* sorry, this beam is unprintable  */
    /* erase it        */
    dynhelp = akt[tr]->next;
    crit = 1;
    while (crit && dynhelp) {
      if ((dynhelp->marker) == ´n´) {
        not3 = (struct Note *) dynhelp->data;
        if ((not3->beam_nr) == (not->beam_nr)) {
          not3->beam_nr = 0;
          not3->beam_pos = 0;
          if ((not3->flags) & 0x40) crit = 0;
```

```
                not3->flags &= 0x7bf;
              }
            }
            if (crit) dynhelp = dynhelp->next;
          }
          not->beam_nr = 0;
          not->beam_pos = 0;
          not->flags &= 0x7bf;
        }
      }
    }
  }
  akt[tr] = akt[tr]->next;
 }
}

void calculate_rest_positions(tr)
  int tr;
{
  char full[42];        /* used positions */
  char bestpos;        /* for storing the best position */
  char highcount;        /* counts free positions */
  char to, mid, from1, to1, from2, to2;  /* variables for looping */
  int i;          /* for looping and indexing */
  long cur_slur[42], cur_beam[42];  /* positions used for slurs and beams */
  unsigned long rtime;      /* store time information */
  char goodpos, goodhigh;      /* temporal best */
  char minbeam, maxbeam;      /* to specify the region from beamed note to */
                              /* the beam itself */
  short restcount;      /* counts the rests of time rtime */
  char topos;        /* store the slope */
  char upperto;        /* setting a loopto value */
  short slope;        /* setting topos */
  short measurecounter;      /* is useful for gdb to stop at a given measure */

  /* crpmark1 */
  measurecounter = 1;
  akt[tr] = begin[tr];
  for (i=0; i<42; i++) {
    cur_slur[i] = 0;
    cur_beam[i] = 0;
  }
  while (akt[tr]) {
    if ((akt[tr]->marker) == 'b') measurecounter++;
    if ((akt[tr]->marker) == 'r') {
      /* crpmark2 */
      restcount = 1;
```

```
rtime = akt[tr]->time;
for (i=0; i<42; i++) {
  if (cur_slur[i] || cur_beam[i]) full[i] = 1;
  else full[i] = 0;
}
dynhelp = akt[tr];  /* store the begin of rest pointer */
dynhelp3 = akt[tr];  /* indicates later the last rest of rtime */
akt[tr] = akt[tr]->next;
while ((akt[tr]) && (rtime == (akt[tr]->time))) {
  /* deal with rests and notes of time rtime */
  if ((akt[tr]->marker) == ´n´) {
    not = (struct Note *) akt[tr]->data;
    not2 = not;
    while (not2->accord_next) not2 = not2->accord_next;

    if (!(not->beam_nr)) {
      if (((not->flags) & 0x30) == 0x30) {
        /* whole note without any stem */
        full[(not->note_pos) & 0x3f] = 1;
      }
      else {
        if ((not->flags) & 0x400) {
          /* lower stem */
          if (((not->note_pos) & 0x3f) > LAMBDA)
            to = ((not->note_pos) & 0x3f) - LAMBDA;
          else to = 0;
          for (i=((not2->note_pos) & 0x3f); i>=to; i-- ) full[i] = 1;
        }
        else {
          /* upper stem */
          to = ((not2->note_pos) & 0x3f) + LAMBDA;
          if (to > 41) to = 41;
          for (i=((not->note_pos) & 0x3f); i<=to; i++ ) full[i] = 1;
        }
      }
    }
    else {
      /* deal with beamed note */
      to = not->beam_pos & 0x3f;
      if ((not->beam_pos) & 0x80) to += UPPLOWSHIFT;
      else  if (to > UPPLOWSHIFT) to -= UPPLOWSHIFT;
        else to = 0;
      if (to > 41) to = 41;
      slope = (not->flags) & 0x7800;
      topos = (slope >> 11) & 0xf;
      if ((not->flags) & 0x8000) {
        /* falling beam */
```

```
          if (to > topos) minbeam = to - topos;
          else minbeam = 0;
          maxbeam = to;
        }
        else {
          /* raising beam */
          minbeam = to;
          maxbeam = to + topos;
          if (maxbeam > 41) maxbeam = 41;
        }
        if (((not2->note_pos) & 0x3f) > maxbeam)
           maxbeam = ((not2->note_pos) & 0x3f);
        if (((not->note_pos) & 0x3f) < minbeam)
           minbeam = ((not->note_pos) & 0x3f);
        for (i=minbeam; i<=maxbeam; i++) full[i] = 1;
      }

      if ((not->slur_nr) && ((not->slur_pos) & 0x40)) {
        /* slur begins here */
        if (not->slur_pos & 0x80) {
          /* upper slur */
          to = SLURWIDE + ((not->slur_pos) & 0x3f);
          if (to > 41) to = 41;
          for (i=((not->slur_pos) & 0x3f); i<=to; i++) full[i] = 1;
        }
        else {
          /* lower slur */
          if (((not->slur_pos) & 0x3f) > SLURWIDE)
            to = ((not->slur_pos) & 0x3f) - SLURWIDE;
          else to = 0;
          for (i=to; i<=((not->slur_pos) & 0x3f); i++) full[i] = 1;
        }
      }
    }
    else {
      if ((akt[tr]->marker) == 'r') {
        restcount++;
        dynhelp3 = akt[tr];
      }
      else {
        /* after rests can only be notes of the same time */
        fprintf(stderr, "unknown element found\n");
        exit(10);
      }
    }
    akt[tr] = akt[tr]->next;
  }
```

```
do {
  /* crpmark3 */
  re = (struct Rest *) dynhelp->data;
  if ((re->position) & 0x80) {
    /* it´s a bass rest */
    mid = BASSMID;
    from1 = mid - halfresthigh[re->kind] -1;
    to1 = VIOLINEMID - 2;
    from2 = mid + halfresthigh[re->kind] + 6;
    to2 = 0;
  }

  if ((re->position) & 0x40) {
    /* it´s a violine rest */
    mid = VIOLINEMID;
    from1 = mid - halfresthigh[re->kind] -1;
    to1 = 41;
    from2 = mid + halfresthigh[re->kind] + 6;
    to2 = BASSMID + 2;
  }

  /* crpmark4 */
  highcount = 0;
  goodhigh = 0;
  bestpos = 60;
  i = from1;
  while (i<to1) {
    if (!full[i]) {
      highcount++;
      if (highcount >= (resthigh[re->kind] + RESTTOLERANCE)) {
        bestpos = i - resthigh[re->kind] - HALFRESTTOLERANCE;
        i = 40;
      }
    }
    else {
      if (highcount > goodhigh) {
        goodhigh = highcount;
        goodpos = i - highcount;
      }
      highcount = 0;
    }
    i++;
  }

  /* rests (except of pause and hpause) can use */
  /* an small upper area because there is enough space */
```

```
highcount = 0;
if ((bestpos == 60) && ((re->kind) < 5)) {
  while (!full[i]) {
    i--;
    highcount++;
  }
  if (highcount > 2) bestpos = i + 2;
}

/* crpmark5 */
highcount = 0;
i = from2;
while (i>to2) {
  if (!full[i]) {
    highcount++;
    if ((highcount > (resthigh[re->kind] + RESTTOLERANCE)) &&
        ((abs(bestpos + (halfresthigh[re->kind]) - mid)) >
         (abs((i + HALFRESTTOLERANCE) +
               (halfresthigh[re->kind]) - mid)))) {
           bestpos = i + HALFRESTTOLERANCE;
           i = 1;
    }
  }
  else {
    if (highcount > goodhigh) {
      goodhigh = highcount;
      goodpos = i;
    }
    highcount = 0;
  }
  i--;
}

/* crpmark6 */
if (bestpos == 60) {
  bestpos = goodpos;
  re->flg = 1;
}

/* crpmark7 */
/* half and whole rests needs an odd position */
if ((re->kind) > 4) bestpos |= 0x1;
re->position |= bestpos;

upperto = bestpos + 2 + (resthigh[re->kind]);
if (upperto > 41) upperto = 41;
for (i=bestpos; i<=upperto; i++) full[i] = 1;
```

```
      restcount--;
      if (restcount) {
        dynhelp = dynhelp->next;
        if ((dynhelp->marker) != ´r´) {
          fprintf(stderr, "error: deal non rest like rest\n");
          exit(10);
        }
      }
    } while (restcount);
    akt[tr] = dynhelp3;
  }
  if ((akt[tr]->marker) == ´n´) {
    /* crpmark8 */
    not3 = (struct Note *) akt[tr]->data;
    if ((not3->beam_nr) &&
        (((not3->beam_pos) & 0x40) || ((not3->flags) & 0x40))) {
      /* begin or end of beam */
      to = not3->beam_pos & 0x3f;
      if ((not3->beam_pos) & 0x80) to += UPPLOWSHIFT;
      else    if (to > UPPLOWSHIFT) to -= UPPLOWSHIFT;
        else to = 0;
      if (to > 41) to = 41;
      slope = (not3->flags) & 0x7800;
      topos = (slope >> 11) & 0xf;
      if ((not3->flags) & 0x8000) {
        /* falling beam */
        if (to > topos) topos = to - topos;
        else topos = 0;
        if (topos > 2) topos -= 2;
        else topos = 0;
        to += 2;
        if (to > 41) to = 41;
        if ((not3->beam_pos) & 0x40)
          for (i=topos; i<=to; i++) cur_beam[i] = not3->beam_nr;
        else {
          for (i=topos; i<=to; i++)
            if (cur_beam[i] == not3->beam_nr) cur_beam[i] = 0;
        }
      }
      else {
        /* raising beam */
        topos += to;
        if (topos > 41) topos = 41;
        if (to > 2) to -= 2;
        else to = 0;
        topos += 2;
        if (topos > 41) topos = 41;
```

```
          if ((not3->beam_pos) & 0x40)
            for (i=to; i<=topos; i++) cur_beam[i] = not3->beam_nr;
          else {
            for (i=to; i<=topos; i++)
              if (cur_beam[i] == not3->beam_nr) cur_beam[i] = 0;
          }
        }
      }
      if ((not3->slur_nr) &&
          (((not3->slur_pos) & 0x40) || ((not3->flags) & 0x80))) {
        /* begin or end of slur */
        if (not3->slur_pos & 0x80) {
          /* upper slur */
          to = SLURWIDE + (not3->slur_pos & 0x3f);
          if (to > 41) to = 41;
          if ((not3->slur_pos) & 0x40)
            for (i=((not3->slur_pos) & 0x3f); i!=to; i++)
              cur_slur[i] = not3->slur_nr;
          else {
            for (i=((not3->slur_pos) & 0x3f); i!=to; i++)
              if (cur_slur[i] == not3->slur_nr) cur_slur[i] = 0;
          }
        }
        else {
          /* lower slur */
          if (((not3->slur_pos) & 0x3f) > SLURWIDE)
            to = (not3->slur_pos & 0x3f) - SLURWIDE;
          else to = 0;
          if ((not3->slur_pos) & 0x40)
            for (i=to; i!=((not3->slur_pos) & 0x3f); i++)
              cur_slur[i] = not3->slur_nr;
          else {
            for (i=to; i!=((not3->slur_pos) & 0x3f); i++)
              if (cur_slur[i] == not3->slur_nr) cur_slur[i] = 0;
          }
        }
      }
    }
    akt[tr] = akt[tr]->next;
  }
}

void ins_levent(tn, type, ltime, value)
  int tn;
  char type;
  unsigned long ltime;
  char value;
```

```
{
  /* this function requires the correct setting of dy to the begin of track */
  /* and of dy2 to NULL                */

  while (dy && (((dy->time) < ltime) ||
         (((dy->time) == ltime) && ((dy->marker) != ´n´) &&
           ((dy->marker) != ´r´)))) {
    dy2 = dy;
    dy = dy->next;
  }

  if ((dynhelp3 = (struct bind *) malloc (sizeof(struct bind))) == NULL) {
    fprintf(stderr, "malloc error\n");
    exit(10);
  }
  if (dy2) dy2->next = dynhelp3;
  else begin[tn] = dynhelp3;
  dynhelp3->next = dy;
  dynhelp3->time = ltime;
  dynhelp3->marker = ´p´;

  if ((lo = (struct Loudness *) malloc (sizeof(struct Loudness))) == NULL) {
    fprintf(stderr, "malloc error\n");
    exit(10);
  }
  dynhelp3->data = (void *) lo;
  if (type) {
    lo->dynamics = 0;
    lo->statics = value;
  }
  else {
    lo->dynamics = value;
    lo->statics = 0;
  }
}


void set_loudness_infos(tnr)
  int tnr;
{
  int i, j, k, l, m, n, sum, count, criterium, loop, loopto;
  unsigned long time, stime;
  char staticflag, dynamicflag, dval, sval;

  /* slimark1 */
  akt[tnr] = begin[tnr];
  dy = begin[tnr];
```

```
dy2 = NULL;
staticflag = 1;
dynamicflag = 0;
j = 0;
for (i=0; i<6; i++) {
  if ((loudrecarr[i] = (struct loudrec *)
                         malloc (sizeof(struct loudrec))) == NULL) {
    fprintf(stderr, "malloc error\n");
    exit(10);
  }
  loudrecarr[i]->time = 0;
}
/* slimark2 */
while (akt[tnr]) {
  if ((akt[tnr]->marker) == ´n´) {
    time = akt[tnr]->time;
    dynhelp = akt[tnr];
    sum = 0;
    count = 0;
    while (dynhelp && (time == (dynhelp->time))) {
      not = (struct Note *) dynhelp->data;
      sum += not->loud;
      count++;
      dynhelp = dynhelp->next;
    }
    criterium = sum / count;
    i = 0;
    while ((i < 7) && (loudarray[i] < criterium)) i++;
    if (staticflag) {
      ins_levent(tnr, 1, time, i + 1);
      staticflag = 0;
    }
    /* fill the record */
    loudrecarr[j]->time = time;
    loudrecarr[j]->criterium = criterium;
    loudrecarr[j]->range = i;

    /* slimark3 */
    j = modulo4pp[j];

    if (loudrecarr[j]->time) {
      k = modulo4pp[j];
      l = modulo4pp[k];
      m = modulo4pp[l];
      n = modulo4pp[m];
      i = modulo4pp[n];
```

```
if ((loudrecarr[j]->criterium) < (loudrecarr[k]->criterium)) {
  if ((loudrecarr[k]->criterium) < (loudrecarr[l]->criterium)) {
    if ((loudrecarr[l]->criterium) < (loudrecarr[m]->criterium)) {
      if ((loudrecarr[m]->criterium)<(loudrecarr[n]->criterium)) {
       dval = 3;
       loopto = n;
       if ((loudrecarr[n]->criterium) ==
           (loudrecarr[i]->criterium)) {
         staticflag = 1;
         stime = loudrecarr[n]->time;
         sval = loudrecarr[n]->range;
       }
      }
      else {
        dval = 2;
        loopto = m;
        if ((loudrecarr[m]->criterium) ==
            (loudrecarr[n]->criterium)) {
          staticflag = 1;
          stime = loudrecarr[m]->time;
          sval = loudrecarr[m]->range;
        }
      }
    }
    else {
      dval = 1;
      loopto = l;
      if ((loudrecarr[l]->criterium) ==
          (loudrecarr[m]->criterium)) {
        staticflag = 1;
        stime = loudrecarr[l]->time;
        sval = loudrecarr[l]->range;
      }
    }
  }
  else {
    dval = 0;
    loopto = k;
    if ((loudrecarr[k]->criterium) == (loudrecarr[l]->criterium)) {
      staticflag = 1;
      stime = loudrecarr[k]->time;
      sval = loudrecarr[k]->range;
    }
  }
  dynamicflag = 1;
}
else {
```

```
if ((loudrecarr[j]->criterium) > (loudrecarr[k]->criterium)) {
  if ((loudrecarr[k]->criterium) > (loudrecarr[l]->criterium)) {
    if ((loudrecarr[l]->criterium) > (loudrecarr[m]->criterium)) {
      if ((loudrecarr[m]->criterium) > (loudrecarr[n]->criterium)) {
       dval = 7;
       loopto = n;
       if ((loudrecarr[n]->criterium) ==
           (loudrecarr[i]->criterium)) {
         staticflag = 1;
         stime = loudrecarr[n]->time;
         sval = loudrecarr[n]->range;
       }
      }
      else {
        dval = 6;
        loopto = m;
        if ((loudrecarr[m]->criterium) ==
            (loudrecarr[n]->criterium)) {
          staticflag = 1;
          stime = loudrecarr[m]->time;
          sval = loudrecarr[m]->range;
        }
      }
    }
    else {
      dval = 5;
      loopto = l;
      if ((loudrecarr[l]->criterium) == (loudrecarr[m]->criterium)) {
        staticflag = 1;
        stime = loudrecarr[l]->time;
        sval = loudrecarr[l]->range;
      }
    }
  }
  else {
    dval = 4;
    loopto = k;
    if ((loudrecarr[k]->criterium) == (loudrecarr[l]->criterium)) {
      staticflag = 1;
      stime = loudrecarr[k]->time;
      sval = loudrecarr[k]->range;
    }
  }
  dynamicflag = 1;
}
else {  /* loudness is equal */
  loop = j;
```

```
            while ((loop != i) &&
                   ((loudrecarr[loop]->criterium) ==
                    (loudrecarr[modulo4pp[loop]]->criterium))) {
              loop = modulo4pp[loop];
            }
            loopto = loop;
          }
        }

        if (dynamicflag) {
          ins_levent(tnr, 0, loudrecarr[j]->time, dval + 1);
          dynamicflag = 0;
        }
        if (staticflag) {
          loopto = modulo4pp[loopto];
          ins_levent(tnr, 1, stime, sval + 1);
          staticflag = 0;
        }
        /* slimark4 */
        loop = j;
        while (loop != loopto) {
          loudrecarr[loop]->time = 0;
          loop = modulo4pp[loop];
        }
      }
      akt[tnr] = dynhelp;
    }
    else akt[tnr] = akt[tnr]->next;
  }
}

void debug_out()
{
  int tnr, barrcounter;

  printf("\nListing the whole dynamic structure:\n");
  for (tnr = 0; tnr < MAXTRACKS; tnr++) {
    dynhelp = begin[tnr];
    barrcounter = 0;
    while (dynhelp != NULL) {
      printf("tnr: %i begin: %lx marker %c ",
             tnr, dynhelp->time, dynhelp->marker);
      switch (dynhelp->marker) {
        case ´n´: not = (struct Note *) dynhelp->data;
            printf("\tduration: %x oct: %i pit: %i\n",
                   not->duration, not->octave, not->pitch);
            printf("\tend: %lx loudness: %x flags: %x \n",
```

```
              not->end, not->loud, not->flags);
       if (not->beam_nr) {
         printf("\t");
         if ((not->flags) & 0x40) printf("end\t");
         if ((not->beam_pos) & 0x40) printf("begin\t");
         printf("beam_number: %d beam_pos: %c orientation: ",
                not->beam_nr, positions[(not->beam_pos) & 0x3f]);
         if ((not->beam_pos) & 0x80) printf("upper\n");
         else printf("lower\n");
       }
       if (not->accord_next) printf("\taccorded: yes\n");
       else printf("\taccorded: no\n");
       if (not->slur_nr) {
         printf("\t");
         if ((not->flags) & 0x80) printf("end\t");
         if ((not->slur_pos) & 0x40) printf("begin\t");
         printf("slur_number: %d slur_pos: %c orientation: ",
                not->slur_nr, positions[(not->slur_pos) & 0x3f]);
         if ((not->slur_pos) & 0x80) printf("upper\n");
         else printf("lower\n");
       }
       printf("\tNotenausgabe: ");
       if (((not->note_pos) & 0xc0) == 0xc0) printf("Aufhebung ");
       else if (((not->note_pos) & 0xc0) == 0x80) printf("b ");
            else if (((not->note_pos) & 0xc0) == 0x40) printf("Kreuz ");
       if (((not->note_pos) & 0x3f) < 5) printf("´");
       printf("%c\n", positions[(not->note_pos) & 0x3f]);
       break;
   case ´t´: ts = (struct TimeSig *) dynhelp->data;
       printf("nn: %i dom: %i\n",ts->nom, ts->denom);
       break;
   case ´l´: ly = (struct Lyric *) dynhelp->data;
       printf("text: %s\n",ly->text);
       break;
   case ´k´: ks = (struct KeySig *) dynhelp->data;
       printf("sign: %x\n",ks->sign);
       break;
   case ´b´: printf("end_of_measure: %d\n", ++barrcounter);
       break;
   case ´r´: re = (struct Rest *) dynhelp->data;
       printf("kind: %x position: %x\n",re->kind, re->position);
       break;
   case ´p´: lo = (struct Loudness *) dynhelp->data;
       if (lo->statics) printf("static with value: %x\n", lo->statics);
       else printf("dynamic with value: %x\n", lo->dynamics);
       break;
   default : printf("unknown marker\n");
```

```
          break;
      }
      dynhelp = dynhelp->next;
    }
  }
}
```

B.10 liste.c

```
/**************************/
/* liste.c                */
/**************************/

#include <stdio.h>
#include <stdlib.h>
#include "liste.h"

void note_insert(oct, pit, laut, t, chann)
  int oct, pit, laut, chann;
  unsigned long int t;
{
  struct element *ptr, *ende;

  if ((ptr = (struct element *) malloc (sizeof(struct element))) == NULL) {
    fprintf(stderr, "malloc error\n");
    exit(10);
  }

  ptr->oct = oct;
  ptr->pit = pit;
  ptr->laut = laut;
  ptr->t   = t;
  ptr->chann = chann;
  ptr->folger = NULL;
  if (anker) {
    for (ende = anker; ende->folger; ende = ende->folger);
    ende->folger = ptr;
    return;
  }
  anker = ptr;

}

int note_delete(oct, pit, laut, zeit, beg, ch)
  int oct, pit, ch, *laut;
  unsigned long int zeit, *beg;
{
```

```
  struct element *ptr1, *ptr2;
  unsigned long int delta;

  if ((ptr1 = anker) == NULL) {
    fprintf(stderr, "oct %d, pit %d not found\n", oct, pit);
    exit(10);
  }
  if ((anker->oct == oct) && (anker->pit == pit) && (anker->chann == ch)) {
    *laut = anker->laut;
    *beg = anker->t;
    if ((delta = zeit - anker->t) <= 0) {
      fprintf(stderr, "note_loeschen: delta = %ld\n", delta);
      exit(10);
    }
    anker = anker->folger;
    free(ptr1);
    return(delta);
  }
  for(ptr2 = ptr1, ptr1 = ptr1->folger;
      ptr1; ptr2 = ptr1, ptr1 = ptr1->folger) {
    if ((ptr1->oct == oct) && (ptr1->pit == pit)) {
      *beg = ptr1->t;
      *laut = ptr1->laut;
      if ((delta = zeit - ptr1->t) <= 0) {
        fprintf(stderr, "note_loeschen: delta = %ld\n", delta);
        exit(10);
      }
      ptr2->folger = ptr1->folger;
      free(ptr1);
      return(delta);
    }
  }
  fprintf(stderr, "oct %d, pit %d not found\n", oct, pit);
  exit(10);
}


int search_note(oct, pit, laut, zeit, beg, ch)
  int *oct, *pit, *laut, *ch;
  unsigned long int zeit, *beg;
{
  struct element *ptr;
  int delta;

  if (!anker) return (-1);

  ptr = anker;
```

```
  *oct = anker->oct;
  *pit = anker->pit;
  *laut = anker->laut;
  *beg = anker->t;
  *ch = anker->chann;
  if ((delta = zeit - anker->t) <= 0) {
    fprintf(stderr, "note_loeschen: delta = %d\n", delta);
    exit(10);
  }
  anker = anker->folger;
  free (ptr);
  return(delta);
}
```

B.11 readmidi.c

```
/***************************/
/* readmidi.c              */
/***************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "midi.h"
#include "maintypes.h"
#include "defines.h"
#include "readmidi.h"

#ifdef DEBUG
static void ausgabe()
{
  printf("MThd = %s, chunk_length = %d\n", head.MThd , head.chunk_length);
  printf("format = %d, tracks = %d, per_quarter = %x\n",
    head.format, head.tracks, head.per_quarter);

  printf("Mtrk = %s, Track_chunk_length = %d\n",
    THead.MTrk, THead.chunk_length);
}
#endif

static unsigned long get_valid_time(ti)
  unsigned long ti;
{
  if (ti < bartime) {
    if (ti < lastbartime) {
      converter = 0;
      sequencecounter = 0;
```

```
    }
    else {
      converter = lastbartime;
      sequencecounter = (double) lastbartime;
    }
  }
  else {
    converter = bartime;
    sequencecounter = (double) bartime;
  }
  while (ti > converter) {
    lastsc = sequencecounter;
    sequencecounter += one64tel;
    converter = (unsigned long) sequencecounter;
  }
  if (ti < converter) ti = (unsigned long) lastsc;
  return(ti);
}

static char midi_getc(fpointer)
  FILE *fpointer;
{
  if (THead.chunk_length-- < 0) {
    fprintf(stderr, "midi_getc: Length Underflow\n");
    exit(10);
  }
  return(getc(fpointer));
}

static void midi_ungetc(c, fpointer)
  FILE *fpointer;
  char c;
{
  THead.chunk_length++;
  ungetc(c, fpointer);
}

static unsigned int ReadVarLen()
{
  unsigned int value;
  unsigned char c;

  if ((value = midi_getc(fp)) & 0x80) {
    value  &= 0x7f;
    do {
      value = (value << 7) + ((c = midi_getc(fp)) & 0x7f);
    }
```

```
    while (c & 0x80);
  }

  return(value);
}


static void ReadMidiTrackName(delta)
  int delta;
{
  int len, i;

  len = ReadVarLen();
  for (i = 0; i < len; i++) {
    if (i < 80 - 1) TrackName[Track_Nr][i] = midi_getc(fp);
    else midi_getc(fp);
  }
  if (i<80) TrackName[Track_Nr][i] = ´\0´;
  else TrackName[Track_Nr][79] = ´\0´;

#ifdef DEBUG
  printf("MIDI_TRACK_NAME (%s) at %d\n", TrackName[Track_Nr], delta);
#endif

}

static void ReadMidiSequSpec(delta)
  int delta;
{
  int len, i;
  char Str[80];
#ifdef DEBUG
  int j;
#endif

  len = ReadVarLen();
  for (i = 0; i < len; i++) {
    if (i < 80 - 1) Str[i] = midi_getc(fp);
  }
  Str[i] = ´\0´;
#ifdef DEBUG
  printf("MIDI_SEQUENCER_SPECIFIC at %d:\n", delta);
  for (j = 0; j < i; j++)
    printf("\t0x%x\n", Str[j]);
#endif
}
```

```
#ifdef DEBUG
static char *p2str(oct, pit)
  int oct, pit;
{
  static char Str[10];
  static char *p_table[12] = {"C", "Cis", "D", "Dis",
           "E",  "F",  "Fis", "G",
           "Gis", "A", "B",  "H"};

  sprintf(Str, "%s%d", p_table[pit], oct);
  return(Str);
}
#endif

static void ReadMidiToneVal(oct, pit, laut)
  int *oct, *pit, *laut;
{
  int w;
  unsigned char inthelp;

  w = midi_getc(fp);
  *laut = midi_getc(fp);

  /* allow only octaves 0 - 5 */
  if ((w>24) && (w<95)) {
    *oct = (w / 12) - 2;
    *pit = (w % 12);
  }
  else {
    *oct = 6;
    *pit = 0;
  }

  inthelp = 12 * (*oct) + (*pit);
  if (max[Track_Nr] < inthelp) max[Track_Nr] = inthelp;
  if (min[Track_Nr] > inthelp) min[Track_Nr] = inthelp;
}

static void erzeuge_note(octave, pitch, laut, begin, dauer, channel)
  int octave, pitch, laut, dauer, channel;
  unsigned long int begin;
{
  if (firstnotetimeflag == -1) {
    firstnotetime = begin;
    firstnotetimeflag = 0;
  }
  /* if there were no TimeSig */
```

```
  if ((valid == 0) && (begin > firstnotetime)) {
    valid++;
    if ((!printonly) || (Track_Nr == (printonly -1))) {
      if ((!notes_flag) && (nom0) && (Track_Nr))
        ins_dyn_event(Track_Nr, ´t´, 0, nom0, denom0, 0, 0, 0, 0, 0);
      else
        ins_dyn_event(Track_Nr, ´t´, 0, 4, 4, 0, 0, 0, 0, 0);
    }
  }

  /* if there were no KeySig */
  if ((valid2 == 0) && (begin > firstnotetime)) {
    valid2++;
    if ((!printonly) || (Track_Nr == (printonly -1))) {
      if ((!notes_flag) && (key0) && (Track_Nr))
        ins_dyn_event(Track_Nr, ´k´, 0, key0, 0, 0, 0, 0, 0, 0);
      else
        ins_dyn_event(Track_Nr, ´k´, 0, 0, 0, 0, 0, 0, 0, 0);
    }
  }

#ifdef DEBUG
  fprintf(stdout, "NOTE into dyn_event_list inserted\n");
  fprintf(stdout, "oct: %d\t pit: %d\t laut: %x\t beg: %lx\t durat: %x\n",
    octave, pitch, laut, begin, dauer);
#endif
  /* notes_flag shows that notes of first track were read  */
  /*              */
  /* because f.e. rosegarden writes empty first track   */
  /* with takt and key info for all other tracks    */

  if (Track_Nr == 0) notes_flag = 1;

  if (((!printonly) || (Track_Nr == (printonly -1))) && (octave != 6))
    ins_dyn_event(Track_Nr, ´n´, 0, 0, 0, octave, pitch, laut,
                  get_valid_time(begin), dauer);
}

static void ReadMidiNoteOff(channel, delta)
  int channel, delta;
{
  int dauer, octave, pitch, laut, la;
  unsigned long int begin;

  ReadMidiToneVal(&octave, &pitch, &laut);

#ifdef DEBUG
```

```
  printf("MIDI_NOTE_OFF at %d(%ld): Channel: %d, pitch = %s, laut = 0x%x\n",
    delta, zeit, channel, p2str(octave, pitch), laut);
#endif
  dauer = note_delete(octave, pitch, &la, zeit, &begin, channel);
#ifdef DEBUG
  printf("Generate_Note with begin = %ld, absolute time = %ld,
      duration = %d\n", begin, zeit, dauer);
#endif
  erzeuge_note(octave, pitch, la, begin, dauer, channel);
}

static void ReadMidiNoteOn(channel, delta)
  int channel, delta;
{
  int dauer, octave, pitch, laut, la;
  unsigned long int begin;

  ReadMidiToneVal(&octave, &pitch, &laut);

  if (laut) {
    note_insert(octave, pitch, laut, zeit, channel);
  }
  else {
    dauer = note_delete(octave, pitch, &la, zeit, &begin, channel);
#ifdef DEBUG
    printf("Generate_Note with begin = %ld, absolute time = %ld,
           duration = %d\n", begin, zeit, dauer);
#endif
    erzeuge_note(octave, pitch, la, begin, dauer, channel);
  }
#ifdef DEBUG
  printf("MIDI_NOTE_ON at %d(%ld): Channel: %d, pitch = %s, laut = 0x%x\n",
    delta, zeit, channel, p2str(octave, pitch), laut);
#endif
}

static void ReadMidiPolyAftertouch(channel, delta)
  int channel, delta;
{
  char dum1, dum2;

  dum1 = midi_getc(fp); dum2 = midi_getc(fp);
#ifdef DEBUG
  printf("MIDI_POLY_AFTERTOUCH: 0x%x, 0x%x, 0x%x\n", channel, dum1, dum2);
#endif
}
```

```
static void ReadMidiCtrlChange(channel, delta)
  int channel, delta;
{
  char dum1, dum2;

  dum1 = midi_getc(fp); dum2 = midi_getc(fp);
#ifdef DEBUG
  printf("MIDI_CTRL_CHANGE: 0x%x, 0x%x, 0x%x\n", channel, dum1, dum2);
#endif
}

static void ReadMidiProgChange(channel, delta)
  int channel, delta;
{
  char dum1;

  dum1 = midi_getc(fp);
#ifdef DEBUG
  printf("MIDI_PROG_CHANGE: 0x%x, 0x%x\n", channel, dum1);
#endif
}

static void ReadMidiChnlAftertouch(channel, delta)
  int channel, delta;
{
  char dum1;

  dum1 = midi_getc(fp);
#ifdef DEBUG
  printf("MIDI_CHNL_AFTERTOUCH: 0x%x, 0x%x\n", channel, dum1);
#endif
}

static void ReadMidiPitchBend(channel, delta)
  int channel, delta;
{
  char dum1, dum2;

  dum1 = midi_getc(fp); dum2 = midi_getc(fp);
#ifdef DEBUG
  printf("MIDI_PITCH_BEND: 0x%x, 0x%x, 0x%x\n", channel, dum1, dum2);
#endif
}

static void ReadMidiSongPositionPtr(delta)
  int delta;
{
```

```
  char dum1, dum2;

  dum1 = midi_getc(fp); dum2 = midi_getc(fp);
#ifdef DEBUG
  printf("MIDI_SONG_POSITION_PTR: 0x%x, 0x%x\n", dum1, dum2);
#endif
}

static void ReadMidiSongSelect(delta)
  int delta;
{
  char dum1;

  dum1 = midi_getc(fp);
#ifdef DEBUG
  printf("MIDI_SONG_SELECT: 0x%x\n", dum1);
#endif
}

static void ReadMidiSystemExclusive()
{
  int dum;
  unsigned int value;

  dum = ReadVarLen();
  while ((value = midi_getc(fp)) != MIDI_EOX);
#ifdef DEBUG
  printf("MIDI_SYSTEM_EXCLUSIVE\n");
#endif
}

static void ReadMidiSetTempo(delta)
  int delta;
{
  unsigned int len, tempo;

  len = midi_getc(fp);
  if (len == 0x03) {
    tempo = 0;
    while(len--) {
      tempo *= 0x100;
      tempo += (unsigned int) (0xff & midi_getc(fp));
    }
#ifdef DEBUG
    D_Tempo = (double) tempo / 1000000.0;
    printf("MIDI_SET_TEMPO at %d: Tempo = %d(%f)\n", delta, tempo, D_Tempo);
#endif
```

```
  }
  else { /* system reset detect */
    midi_ungetc(len, fp);
    midi_ungetc(0x51, fp);
  }
}

static void ReadMidiTimeSig(delta)
  int delta;
{
  int dummy, nn, dd, cc, bb, dominator;

  dummy = midi_getc(fp);
  if (dummy == 0x04) {
    valid = 1;  /* means: TimeSig detected */
    nn = midi_getc(fp);
    dd = midi_getc(fp);
    cc = midi_getc(fp);
    bb = midi_getc(fp);
    dominator = (1 << dd);
    help = taktzeit;
    taktzeit = head.per_quarter * nn * 4 / dominator;
    diskrete_tz = diskrete_tz + taktzeit - help;
    if ((!Track_Nr) && (!nom0)) {
      nom0 = nn;
      denom0 = dominator;
    }
    if ((!printonly) || (Track_Nr == (printonly -1)))
      ins_dyn_event(Track_Nr, ´t´, 0, nn, dominator,
                    0, 0, 0, get_valid_time(zeit), 0);

#ifdef DEBUG
 printf("Takt: %x\n",taktzeit);
 printf("MIDI_TIME_SIGNATURE at %d: Takt:%d/%d\nMIDI-clocks per Metronom: %d,",
   delta, nn, dominator, cc);
 printf("%d 32nd per MIDI-Quarter\n", bb);
#endif
  }
  else { /* system reset detect */
    midi_ungetc(dummy, fp);
    midi_ungetc(0x58, fp);
  }
}

static void ReadMidiTextEvent(delta)
  int delta;
{
```

```
  unsigned int len;
  int value;

  len = ReadVarLen();
  while (len--) value = midi_getc(fp);
#ifdef DEBUG
  printf("MIDI_TEXT_EVENT\n");
#endif
}

static void ReadMidiCopyrightNotice(delta)
  int delta;
{
  unsigned int len;
  int value;

  len = ReadVarLen();
  while (len--) value = midi_getc(fp);
#ifdef DEBUG
  printf("MIDI_COPYRIGHT_NOTICE\n");
#endif
}

static void ReadMidiInstrumentName(delta)
  int delta;
{
  unsigned int len;
  int value=0;

  len = ReadVarLen();
  while (len--) Staffs[Track_Nr][value++] = midi_getc(fp);
  Staffs[Track_Nr][value] = 0;
#ifdef DEBUG
  printf("MIDI_INSTRUMENT_NAME: %s\n",Staffs[Track_Nr]);
#endif
}

static void ReadMidiLyrik(delta)
  int delta;
{
  unsigned int len;
  char *ptr, *ptrhelp;

  len = ReadVarLen();
  if ((ptr = (char *) malloc (len+1)) == NULL) {
    fprintf(stderr, "mallocC error\n");
    exit(10);
```

```
  }
  ptrhelp = ptr;
  while (len--) {
    *ptrhelp = midi_getc(fp);
    ptrhelp++;
  }
  *ptrhelp = 0;
#ifdef DEBUG
  printf("MIDI_LYRIC\n");
#endif
  if ((!printonly) || (Track_Nr == (printonly -1)))
    ins_dyn_event(Track_Nr, ´l´, ptr, 0, 0, 0, 0, 0, get_valid_time(zeit), 0);
}

static void ReadMidiTextMarker(delta)
  int delta;
{
  unsigned int len;
  int value;

  len = ReadVarLen();
  while (len--) value = midi_getc(fp);
#ifdef DEBUG
  printf("MIDI_TEXT_MARKER\n");
#endif
}

static void ReadMidiCuePoint(delta)
  int delta;
{
  unsigned int len;
  int value;

  len = ReadVarLen();
  while (len--) value = midi_getc(fp);
#ifdef DEBUG
  printf("MIDI_CUE_POINT\n");
#endif
}

static void ReadMidiChannelPrefix(delta)
  int delta;
{
  int dum1, dum2;
  dum1 = midi_getc(fp);
  if (dum1 == 0x01) {
    dum2 = midi_getc(fp);
```

```
#ifdef DEBUG
  printf("MIDI_CHANNEL_PREFIX\n");
#endif
  }
  else { /* system reset detect */
    midi_ungetc(dum1, fp);
    midi_ungetc(0x20, fp);
  }
}

static void ReadMidiSmpteOffset(delta)
  int delta;
{
  int dum;

  if (delta != 0) { /* system reset detect */
    midi_ungetc(0x54, fp);
  }
  else {
    dum = midi_getc(fp);
    if (dum == 0x05) {
      midi_getc(fp); midi_getc(fp); midi_getc(fp);
      midi_getc(fp); midi_getc(fp);
#ifdef DEBUG
  printf("MIDI_SMPTE_OFFSET\n");
#endif
    }
    else { /* system reset detect */
      midi_ungetc(dum, fp);
      midi_ungetc(0x54, fp);
    }
  }
}

static void ReadMidiSequenceNumber(MetaEvent, delta)
  int delta;
  unsigned char MetaEvent;
{
  int value;

  value = midi_getc(fp);
  if (value != 0x02) { /* system reset detect */
    midi_ungetc(value, fp);
    midi_ungetc(0x00, fp);
  }
  else {
    value = midi_getc(fp); value = midi_getc(fp);
```

```
    value = midi_getc(fp); value = midi_getc(fp);
#ifdef DEBUG
    printf("MIDI_SEQUENCE_NUMBER\n");
#endif
  }
}

static void ReadMidiKeySig(delta)
  int delta;
{
  int dum, sign;

  dum = midi_getc(fp);
  if (dum != 0x02) { /* system reset detect */
    midi_ungetc(dum, fp);
    midi_ungetc(0x59, fp);
  }
  else {
    valid2 = 1; /* KeySig detected */
    sign = midi_getc(fp);
    /* manipulate sign */
    if ((sign & 0x80) && (sign & 0x08)) {
      sign = sign ^ 0xf;
      sign++;
    }
    if ((!Track_Nr) && (!key0)) key0 = sign;
    if ((!printonly) || (Track_Nr == (printonly -1)))
      ins_dyn_event(Track_Nr, ´k´, 0, sign, 0, 0, 0, 0,
                    get_valid_time(zeit), 0);
    T_genus = (0xff & sign);
    Signature = (0xff & midi_getc(fp));
#ifdef DEBUG
    printf("MIDI_KEY_SIGNATURE at %d: T_genus = ", delta);
    switch (T_genus)  {
      case MINOR: printf("MINOR"); break;
      case MAJOR: printf("MAJOR"); break;
      default: printf("unknown"); break;
    }
    printf(" Signature = %d\n", Signature);
#endif
  }
}

static void ReadMidiEndOfTrack(delta)
  int delta;
{
  int dum, dauer, octave, pitch, laut, channel;
```

```
  unsigned long int begin;

  dum = midi_getc(fp);
  if (dum == 0) {
    dum = midi_getc(fp);  /* read - deltatime */
    while (dum == 0) {  /* falls nach EOTrack noch Folgeelement */
      midi_ungetc(dum, fp);
      ReadEvent();
      dum = midi_getc(fp);
    }
    midi_ungetc(dum, fp);
    while ((dauer = search_note(&octave, &pitch, &laut,
                                zeit, &begin, &channel)) != -1) {
#ifdef DEBUG
      printf("erzeuge_note mit begin = %ld, zeit = %ld, dauer = %d\n",
        begin, zeit, dauer);
#endif
      erzeuge_note(octave, pitch, laut, begin, dauer, channel);
    }

    if (THead.chunk_length) {
      fprintf(stderr,
       "Warning: ReadMidiEndOfTrack: Sync_Error THead.chunk_length = %d\n",
      THead.chunk_length);
      while (THead.chunk_length >= 0) midi_getc(fp);
    }
#ifdef DEBUG
    printf("MIDI_END_OF_TRACK at %d\n", delta);
#endif
  }
  else { /* system reset detect */
    midi_ungetc(dum, fp);
    midi_ungetc(0x2f, fp);
  }
}

static int ReadMidiMetaEvent(delta)
  int delta;
{
  unsigned char MetaEvent;
  int not_eot = 1;

  switch(MetaEvent = midi_getc(fp)) {
    case MIDI_SEQUENCE_NUMBER: ReadMidiSequenceNumber(MetaEvent,delta); break;
    case MIDI_TEXT_EVENT: ReadMidiTextEvent(delta); break;
    case MIDI_TRACK_NAME: ReadMidiTrackName(delta); break;
    case MIDI_SET_TEMPO: ReadMidiSetTempo(delta); break;
```

```
    case MIDI_TIME_SIGNATURE: ReadMidiTimeSig(delta); break;
    case MIDI_END_OF_TRACK: ReadMidiEndOfTrack(delta); not_eot = 0; break;
    case MIDI_KEY_SIGNATURE: ReadMidiKeySig(delta); break;
    case MIDI_SEQUENCER_SPECIFIC: ReadMidiSequSpec(delta); break;
    case MIDI_CHANNEL_PREFIX: ReadMidiChannelPrefix(delta); break;
    case MIDI_SMPTE_OFFSET: ReadMidiSmpteOffset(delta); break;
    case MIDI_COPYRIGHT_NOTICE: ReadMidiCopyrightNotice(delta); break;
    case MIDI_INSTRUMENT_NAME: ReadMidiInstrumentName(delta); break;
    case MIDI_LYRIC: ReadMidiLyrik(delta); break;
    case MIDI_TEXT_MARKER: ReadMidiTextMarker(delta); break;
    case MIDI_CUE_POINT: ReadMidiCuePoint(delta); break;
    default: /* system reset detect */
      midi_ungetc(MetaEvent, fp); break;
  }
  return(not_eot);
}

static int ReadEvent()
{
  unsigned char Event_Type;
  int delta, not_eot = 1;

  delta = ReadVarLen();
  zeit += delta;

  while (zeit >= diskrete_tz) {
    if ((!printonly) || (Track_Nr == (printonly -1))) {
      lastbartime = bartime;
      bartime = diskrete_tz;
      ins_dyn_event(Track_Nr, ´b´, 0, 0, 0, 0, 0, 0, diskrete_tz, taktzeit);
    }
    diskrete_tz += taktzeit;
  }

  Event_Type = midi_getc(fp);
r_event:
  if ((0xf0 & Event_Type) == MIDI_NOTE_OFF) {
    ReadMidiNoteOff((Event_Type & 0xf), delta);
  }
  else if ((0xf0  & Event_Type) ==  MIDI_NOTE_ON) {
    ReadMidiNoteOn((Event_Type & 0xf), delta);
  }
  else if ((0xf0  & Event_Type) ==  MIDI_POLY_AFTERTOUCH) {
    ReadMidiPolyAftertouch((Event_Type & 0xf), delta);
  }
  else if ((0xf0  & Event_Type) ==  MIDI_CTRL_CHANGE) {
    ReadMidiCtrlChange((Event_Type  & 0xf), delta);
```

```
  }
  else if ((0xf0  & Event_Type) ==  MIDI_PROG_CHANGE) {
    ReadMidiProgChange((Event_Type  & 0xf), delta);
  }
  else if ((0xf0  & Event_Type) ==  MIDI_CHNL_AFTERTOUCH) {
    ReadMidiChnlAftertouch((Event_Type & 0xf), delta);
  }
  else if ((0xf0  & Event_Type) ==  MIDI_PITCH_BEND) {
    ReadMidiPitchBend((Event_Type & 0xf), delta);
  }
  else if ((0xf0  & Event_Type) ==  MIDI_SYSTEM_MSG) {
    if (Event_Type == MIDI_FILE_META_EVENT) {
      not_eot = ReadMidiMetaEvent(delta);
    }
    else {
      /* treatment of all other 0xfx-events */
      switch (0xf & Event_Type) {
        case MIDI_SONG_POSITION_PTR: /* 0xf2 */
          ReadMidiSongPositionPtr(delta);
          break;
        case MIDI_SONG_SELECT: /* 0xf3 */
          ReadMidiSongSelect(delta);
          break;
        case MIDI_SYSTEM_EXCLUSIVE: /* 0xf0 */
          ReadMidiSystemExclusive();
          break;
        case MIDI_TUNE_REQUEST: /* 0xf6 */
        case MIDI_TIMING_CLOCK: /* 0xf8 */
        case MIDI_START: /* 0xfa */
        case MIDI_CONTINUE: /* 0xfb */
        case MIDI_STOP: /* 0xfc */
        case MIDI_ACTIVE_SENSING: /* 0xfe */
          midi_getc(fp);
          break;
        default:
          fprintf(stderr, "Event: 0x%x unknown\n", Event_Type);
          fprintf(stderr, "Fileoffset: %lu\n", ftell(fp));
          exit(10);
      }
    }
  }
  else if (Running_Event) {
#ifdef DEBUG
    printf("Running Event\n");
#endif
    midi_ungetc(Event_Type, fp);
```

```
    Event_Type = Running_Event;
    goto r_event;
  }
  else {
    fprintf(stderr, "Event: 0x%x unknown\n", Event_Type);
    fprintf(stderr, "Fileoffset: %lu\n", ftell(fp));
    exit(10);
  }
  Running_Event = Event_Type;
  return(not_eot);
}


static void read_track()
{
  zeit = 0;
  THead.MTrk[0] = getc(fp); THead.MTrk[1] = getc(fp);
  THead.MTrk[2] = getc(fp); THead.MTrk[3] = getc(fp);
  THead.MTrk[4] = ´\0´;
  if (strcmp(THead.MTrk, MIDI_TRACK_HEADER)) {
    fprintf(stderr, "read_track: Sync_error(%s)\n", THead.MTrk);
    exit(10);
  }
  THead.chunk_length  = 0x1000000 * getc(fp);
  THead.chunk_length += 0x10000 * getc(fp);
  THead.chunk_length += 0x100 * getc(fp);
  THead.chunk_length += getc(fp);
#ifdef DEBUG
  ausgabe();
#endif
  Running_Event = 0;
  while (ReadEvent());
}

static void usage(conststrg)
  char *conststrg;
{
  fprintf(stderr,"usage:\t%s [-<nr>] <midifile> [<texfile>]\n", conststrg);
  fprintf(stderr,"\t%s -c <midifile>\n", conststrg);
  exit(10);
}

void main(argc, argv)
  int argc;
  char *argv[];
{
  int lenhelp;
```

```
char opthelp;

if (argc < 2) usage(argv[0]);

if ((argc == 3) && (!strcmp(argv[1],"-c"))) {
  if ((fp = fopen(argv[2], "r")) == NULL) {
    fprintf(stderr, "cannot open %s\n", argv[1]);
    exit(10);
  }
  count = 1;
}
else {
  count = 0;
  if (*argv[1] == '-') {
    if ((argc == 2) || (*argv[2] == '-') || (argc > 4)) usage(argv[0]);
    opthelp = 1;
    printonly = (char) atoi((argv[1])+1);
    if (!printonly) printonly = 20;
  }
  else {
    if (argc > 3) usage(argv[0]);
    printonly = 0;  /* print every track */
    opthelp = 0;
  }

  if ((fp = fopen(argv[1+opthelp], "r")) == NULL) {
    fprintf(stderr, "cannot open %s\n", argv[1+opthelp]);
    exit(10);
  }

  /* fill the array for the name of tex file */
  if (argc == (3+opthelp)) {
    if (!strcmp(argv[1+opthelp], argv[2+opthelp])) {
      fprintf(stderr, "cannot process one file to itself\n");
      exit(10);
    }
    strcpy(texfilename, argv[2+opthelp]);
  }
  else {  /* mainmark1: try to select a nice name for texfile */
    strcpy(texfilename, argv[1+opthelp]);
    lenhelp = strlen(texfilename) - 4;

    /* delete midi file extension */
    if ((!strcmp(&(texfilename[lenhelp]),".mid")) ||
        (!strcmp(&(texfilename[lenhelp]),".MID")))
      texfilename[lenhelp] = 0;
    else {
```

```
      if ((!strcmp(&(texfilename[lenhelp+1]),".mf")) ||
          (!strcmp(&(texfilename[lenhelp+1]),".MF")))
        texfilename[lenhelp+1] = 0;
    }
    if ((texfilename[0] > 64) && (texfilename[0] < 91))
      strcat(texfilename, ".TEX");
    else
      strcat(texfilename, ".tex");
  }

}

/* mainmark2 */
head.MThd[0] = getc(fp); head.MThd[1] = getc(fp);
head.MThd[2] = getc(fp); head.MThd[3] = getc(fp); head.MThd[4] = ´\0´;
if (strcmp(head.MThd, MIDI_FILE_HEADER)) {
  fprintf(stderr, "specified file probably not a midi-file or corrupted\n");
  exit(10);
}

head.chunk_length = 0x1000000 * getc(fp);
head.chunk_length += 0x10000 * getc(fp);
head.chunk_length += 0x100 * getc(fp);
head.chunk_length += getc(fp);

if (head.chunk_length != 6) {
  fprintf(stderr, "wrong header chunk length\n");
  exit(10);
}

head.format = 0x100 * getc(fp); head.format += getc(fp);
head.tracks = 0x100 * getc(fp); head.tracks += getc(fp);

/* mainmark3 */
if (count) {
  printf("%d tracks detected\n", head.tracks);
  exit(0);
}

if (printonly && (printonly > head.tracks)) {
  printf("can´t build tex-file of specified track \n");
  exit(0);
}

if ((!printonly) && ((head.tracks) > 9))
  fprintf(stderr,
     "Warning: printing out so many tracks together may look very ugly\n");
```

```
  head.per_quarter = 0x100 * (0xff & getc(fp));
  head.per_quarter += (0xff & getc(fp));
  if (head.format != 1) {
    fprintf(stderr, "Format %d not supported\n", head.format);
    exit(10);
  }

  if (head.per_quarter & 0x8000) {
    fprintf(stderr, "SMPTE-Time Code not supported\n");
    exit(10);
  }

  /* mainmark4 */
  one64tel = (double) head.per_quarter / 16;

  /* mainmark5 */
  nom0 = 0;
  key0 = 0;
  notes_flag = 0;
  for (lenhelp = 0; lenhelp < MAXTRACKS; lenhelp++) {
    Staffs[lenhelp][0]=0;
    max[lenhelp]=0;
    min[lenhelp]=255;
  }
  taktzeit = 4 * head.per_quarter;
  dyn_init();

  /* mainmark6 */
  if (!printonly)
    for (Track_Nr = 0; Track_Nr < head.tracks; Track_Nr++) {
      valid = 0;      /* no TimeSig */
      valid2 = 0;  /* no KeySig  */

      /* following variables allow event begin only at fixed time-points */
      lastbartime = 0;
      bartime = 0;

      firstnotetimeflag = -1;
      diskrete_tz = taktzeit;
      read_track();
      if (min[Track_Nr] > 0x1f) region[Track_Nr] = 1;
      else  if (max[Track_Nr] < 0x2a) region[Track_Nr] = 2;
        else region[Track_Nr] = 0;
#ifdef DEBUG
      printf("region: %x min: %x max: %x\n",
             region[Track_Nr], min[Track_Nr], max[Track_Nr]);
```

```
      printf("call now split_notes()\n");
#endif
      split_notes(Track_Nr, head.per_quarter);
#ifdef DEBUG
      printf("call now detect_accords()\n");
#endif
      detect_accords(region[Track_Nr], Track_Nr);
#ifdef DEBUG
      printf("call now insert_notetimes_and_slurs()\n");
#endif
      insert_notetimes_and_slurs(region[Track_Nr], Track_Nr, head.per_quarter);
#ifdef DEBUG
      printf("call now detect_note_positions_with_signs()\n");
#endif
      detect_note_positions_with_signs(Track_Nr);
#ifdef DEBUG
      printf("call now set_slur_regions()\n");
#endif
      set_slur_regions(Track_Nr);
#ifdef DEBUG
      printf("call now reset_treat_info()\n");
#endif
      reset_treat_info(Track_Nr);
#ifdef DEBUG
      printf("call now beam_detect()\n");
#endif
      beam_detect(region[Track_Nr], Track_Nr);
#ifdef DEBUG
      printf("call now set_beam_regions()\n");
#endif
      set_beam_regions(Track_Nr);
#ifdef DEBUG
      printf("call now reset_treat_info()\n");
#endif
      reset_treat_info(Track_Nr);
#ifdef DEBUG
      printf("call now slur_geometry()\n");
#endif
      slur_geometry(Track_Nr);
#ifdef DEBUG
      printf("call now detect_rests()\n");
#endif
      detect_rests(Track_Nr, region[Track_Nr], head.per_quarter);
#ifdef DEBUG
      printf("call now reset_treat_info()\n");
#endif
      reset_treat_info(Track_Nr);
```

```
#ifdef DEBUG
      printf("call now beam_geometry()\n");
#endif
      beam_geometry(Track_Nr);
#ifdef DEBUG
      printf("call now reset_treat_info()\n");
#endif
      reset_treat_info(Track_Nr);
#ifdef DEBUG
      printf("call now calculate_rest_positions()\n");
#endif
      calculate_rest_positions(Track_Nr);
#ifdef DEBUG
      printf("call now set_loudness_infos()\n");
#endif
      set_loudness_infos(Track_Nr);
    }
  else {
    for (Track_Nr = 0; Track_Nr < printonly; Track_Nr++) {
      valid = 0;      /* no TimeSig */
      valid2 = 0;     /* no KeySig  */

      /* following variables allow event begin only at fixed time-points */
      lastbartime = 0;
      bartime = 0;

      firstnotetimeflag = -1;
      diskrete_tz = taktzeit;
      read_track();
    }
    if ((printonly == 1) && (notes_flag == 0)) {
      fprintf(stderr, "sorry, first track is empty\n");
    }
    else {
      Track_Nr--;
      if (min[Track_Nr] > 0x1f) region[Track_Nr] = 1;
      else    if (max[Track_Nr] < 0x2a) region[Track_Nr] = 2;
        else region[Track_Nr] = 0;
#ifdef DEBUG
      printf("region: %x min: %x max: %x\n",
             region[Track_Nr], min[Track_Nr], max[Track_Nr]);
#endif
      split_notes(Track_Nr, head.per_quarter);
      detect_accords(region[Track_Nr], Track_Nr);
      insert_notetimes_and_slurs(region[Track_Nr], Track_Nr, head.per_quarter);
      detect_note_positions_with_signs(Track_Nr);
      set_slur_regions(Track_Nr);
```

```
      reset_treat_info(Track_Nr);
      beam_detect(region[Track_Nr], Track_Nr);
      set_beam_regions(Track_Nr);
      reset_treat_info(Track_Nr);
      slur_geometry(Track_Nr);
      detect_rests(Track_Nr, region[Track_Nr], head.per_quarter);
      reset_treat_info(Track_Nr);
      beam_geometry(Track_Nr);
      reset_treat_info(Track_Nr);
      calculate_rest_positions(Track_Nr);
      set_loudness_infos(Track_Nr);
    }
  }

#ifdef DEBUG
  debug_out();
#endif
  /* mainmark7 */
  if ((tfp = fopen(texfilename, "w")) ==  NULL) {
          fprintf(stderr, "cannot open %s\n", texfilename);
          exit(10);
        }

#ifdef DEBUG
  printf("writing texfile\n");
#endif
  write_texfile();
  fclose(fp); /* close MIDI-file */
  fclose(tfp); /* close TEX-file */
}
```

B.12 texgen.c

```
/***************************/
/* texgen.c                */
/***************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "maintypes.h"
#include "defines.h"
#include "texgen.h"

void write_note(bindvar, beamstr)
  struct bind *bindvar;
  char beamstr[10];
```

```
{
  struct Note *notestart, *noteend;
  char accord_kind;
  char extremeposflag;    /* shows that a position is lower than A */
  short accord_type;
  char lastpos;
  char notepos[6];
  char note[9];
  char pflag, ppflag;    /* for storing information about points */

  pflag = 0;
  ppflag = 0;
  notestart = (struct Note *) bindvar->data;
  noteend = notestart;
  if ((noteend->flags) & 0x2) ppflag = 1;
  if ((noteend->flags) & 0x1) pflag = 1;
  accord_type = 0;
  lastpos = ((noteend->note_pos) & 0x3f);
  if (notestart->accord_next) {
    accord_type = (notestart->flags) & 0x38;
    if (((notestart->flags) & 0x38) >= 0x28) accord_kind = 'h';
    else accord_kind = 'q';
    while (noteend->accord_next) {
      extremeposflag = 0;
      if (((noteend->note_pos) & 0x3f) < 5) {
        fprintf( tfp, "{\\transpose=-7\\relax ");
        extremeposflag = 1;
      }
      sprintf( notepos, "%c", positions[(noteend->note_pos) & 0x3f]);
      if (ppflag) fprintf( tfp, "\\ppt %s", notepos);
      else  if (pflag) fprintf( tfp, "\\pt %s", notepos);
      if ((noteend->flags) & 0x4)
              if ((noteend->flags) & 0x400) fprintf( tfp, "\\upz %s", notepos);
              else fprintf( tfp, "\\lpz %s", notepos);
      if (((noteend->note_pos) & 0xc0) == 0) sprintf( note, "%s", notepos);
      else  if (((noteend->note_pos) & 0xc0) == 0x40)
          sprintf( note, "{^%c}", positions[(noteend->note_pos) & 0x3f]);
        else  if (((noteend->note_pos) & 0xc0) == 0x80)
            sprintf( note, "{_%c}", positions[(noteend->note_pos) & 0x3f]);
          else
            sprintf( note, "{=%c}", positions[(noteend->note_pos) & 0x3f]);
      if ((lastpos + 1) == ((noteend->accord_next->note_pos) & 0x3f)) {
        if (note[1] == '^') fprintf( tfp, "\\lsh %s", notepos);
        if (note[1] == '_') fprintf( tfp, "\\lfl %s", notepos);
        if (note[1] == '=') fprintf( tfp, "\\lna %s", notepos);
        if (((!((notestart->beam_nr) && (beamstr))) &&
             (!((notestart->flags) & 0x400)))
```

```
          || (((notestart->beam_nr) && (beamstr)) &&
            ((*(beamstr + 1)) == ´h´))) {
          /* upper stem */
          fprintf( tfp, "\\r");
        }
        else {
          /* lower stem */
          fprintf( tfp, "\\l");
        }
        fprintf( tfp, "%c %s", accord_kind, notepos);
      }
      else {
        lastpos = (noteend->accord_next->note_pos) & 0x3f;
        fprintf( tfp, "\\z%c %s", accord_kind, note);
      }
      noteend = noteend->accord_next;
      if (extremeposflag) fprintf( tfp, "}");
    }
  }

  noteend->flags |= accord_type;
  extremeposflag = 0;
  if (((noteend->note_pos) & 0x3f) < 5) {
    fprintf( tfp, "{\\transpose=-7\\relax ");
    extremeposflag = 1;
  }
  sprintf( notepos, "%c", positions[(noteend->note_pos) & 0x3f]);
  if (ppflag) fprintf( tfp, "\\ppt %s", notepos);
  else if (pflag) fprintf( tfp, "\\pt %s", notepos);
  if ((noteend->flags) & 0x4)
          if ((noteend->flags) & 0x400) fprintf( tfp, "\\upz %s", notepos);
          else fprintf( tfp, "\\lpz %s", notepos);
  if (((noteend->note_pos) & 0xc0) == 0) sprintf( note, " %s", notepos);
  else    if (((noteend->note_pos) & 0xc0) == 0x40)
      sprintf( note, "{^%c}", positions[(noteend->note_pos) & 0x3f]);
    else    if (((noteend->note_pos) & 0xc0) == 0x80)
        sprintf( note, "{_%c}", positions[(noteend->note_pos) & 0x3f]);
      else
        sprintf( note, "{=%c}", positions[(noteend->note_pos) & 0x3f]);

  if ((notestart->beam_nr) && (beamstr)) {
    fprintf(tfp, "\\%s%s", beamstr, note);
  }
  else {
    fprintf( tfp, "\\%s", notecode[((noteend->flags) & 0x38) >> 3]);
    if (!(((notestart->flags) & 0x38) == 0x30)) {
      if ((notestart->flags) & 0x400) fprintf( tfp, "l");
```

```
      else fprintf( tfp, "u");
    }
    fprintf( tfp, "%s", note);
  }
  if (extremeposflag) fprintf( tfp, "}");
}


/* put the times in relation to previous barre */
void prepare_times()
{
  unsigned char i;
  unsigned long barrtime;

  for (i=0;i<MAXTRACKS;i++) {
    akt[i] = begin[i];
    barrtime = 0;
    while (akt[i]) {
      if ((akt[i]->marker) == ´b´) {
        barrtime = akt[i]->time;
        akt[i]->time = 0;
      }
      else akt[i]->time -= barrtime;
      akt[i] = akt[i]->next;
    }
  }
}

void delete_track(tr)
  int tr;
{
  while (begin[tr]) {
    akt[tr] = begin[tr]->next;
    free(begin[tr]->data);
    free(begin[tr]);
    begin[tr] = akt[tr];
  }
}


void write_texfile()
{
  char flag, notready, readyflag, ksflag, tsflag, whilecrit, nothing;
  int i, j, index;
  int looping;
  struct bind *beghelp[MAXTRACKS];
  char reghelp[MAXTRACKS];
```

```
char store;
int startwith, zval;
short events, innerbar;
unsigned int measurecounter;
unsigned int mintime;
unsigned short l, k;
int delflg;    /* shows deleting of tracks */
short firstnom, kscontent, firstdenom;
unsigned long slur[10], beam[10][3];
char beamstring[10], slur_and_beam[10], todelete[10];
char helpInst[MAXTRACKS][180];

prepare_times();
notready = 0;
firstnom = 0;
kscontent = 0;
ksflag = 0;
i = 0;
/* wtmark1 */
while ((notready < CURRENTMAXTRACKS) && (i<MAXTRACKS)) {
  /* generate a string for printing a name */
  Inst[i][0] = 0;
  strcat(Inst[i], Staffs[i]);
  if (Inst[i][0]) strcat(Inst[i], " ");
  strcat(Inst[i], TrackName[i]);

  akt[i] = begin[i];
  flag = 0;
  while ((akt[i]) && (!(flag && firstnom && ksflag))) {
    if ((!firstnom) && ((akt[i]->marker) == ´t´)) {
      ts = (struct TimeSig *) akt[i]->data;
      firstnom = ts->nom;
      firstdenom = ts->denom;
    }
    if ((akt[i]->marker) == ´n´) flag = 1;
    if ((!ksflag) && ((akt[i]->marker) == ´k´)) {
      ks = (struct KeySig *) akt[i]->data;
      kscontent = ks->sign;
      ksflag = 1;
    }
    akt[i] = akt[i]->next;
  }
  if ((!flag) && (begin[i])) delete_track(i);
  if (begin[i]) notready++;
  i++;
}
```

```
/* cut some tracks if neccesary because tex only support up to 9 */
delflg = 0;
while (i<MAXTRACKS) {
  if (begin[i]) {
    delete_track(i);
    delflg++;
  }
  i++;
}
if (delflg == 1) fprintf(stderr, "sorry, had to cut one track\n");
if (delflg > 1) fprintf(stderr, "sorry, had to cut %d tracks\n", delflg);

/* wtmark2 */
if (notready) {
  /* turn over the voices */
  for (i=0; i<MAXTRACKS; i++) {
    beghelp[i] = NULL;
    reghelp[i] = 0;
    helpInst[i][0] = 0;
  }
  startwith = 0;
  for (looping=(MAXTRACKS - 1); looping>=0; looping--) {
    if (begin[looping]) {
      zval = 0;
      while (zval < 180) {
        helpInst[startwith][zval] = Inst[looping][zval];
        zval++;
      }
      reghelp[startwith] = region[looping];
      beghelp[startwith++] = begin[looping];
    }
  }
  for (i=0; i<MAXTRACKS; i++) {
    zval = 0;
    while (zval < 180) {
      Inst[i][zval] = helpInst[i][zval];
      zval++;
    }
    begin[i] = beghelp[i];
    region[i] = reghelp[i];
  }

  /* wtmark3 */
  fprintf(tfp, "\\input musicnft\n");
  fprintf(tfp, "\\input musictex\n");
  fprintf(tfp, "\\input musicadd\n");
  fprintf(tfp, "\\hsize 160mm\n");
```

```
fprintf(tfp, "\\vsize 240mm\n");
if (PAGENUMBERS) fprintf(tfp, "%c ", ´\045´);
fprintf(tfp, "\\nopagenumbers\n");
if (!MEASURENUMBERS) fprintf(tfp, "\\def\\freqbarno{9999}%c\n", ´\045´);
fprintf(tfp, "\\musicsize=%d\n", MUSICSIZE);
fprintf(tfp, "\\def\\nbinstruments{%u}\n", notready);
fprintf(tfp, "\\def\\interfacteur{%d}\n", INTERFACTEUR);
fprintf(tfp, "\\def\\bottomfacteur{%d}\n", BOTTOMFACTEUR);
for (i=0; i<notready; i++) {
  fprintf(tfp, "\\def\\instrument%s{%s}%c\n", romans[i], Inst[i], ´\045´);
  if (region[i]) {
    fprintf(tfp, "\\nbportees%s=%u\n", romans[i], 1);
    fprintf(tfp, "\\cleftoks%s={%u000}%c\n",
            romans[i], (region[i] & 0x2) * 3, ´\045´);
  }
  else {
    fprintf(tfp, "\\nbportees%s=%u\n", romans[i], 2);
    fprintf(tfp, "\\cleftoks%s={%u000}%c\n", romans[i], 6, ´\045´);
  }
}
if (firstnom) fprintf(tfp, "\\generalmeter{\\meterfrac{%u}{%u}}\n",
                      firstnom, firstdenom);
else {
  fprintf(tfp, "\\generalmeter{\\meterfrac{4}{4}}\n");
  firstnom = 4;
  firstdenom = 4;
}

if (ksflag) {
  fprintf(tfp, "\\signaturegenerale{");
  if (kscontent & 0x80) fprintf(tfp, "-");
  fprintf(tfp, "%u}\n", kscontent & 0xf);
}
else fprintf(tfp, "\\signaturegenerale{0}\n");

/* print the name of file in a tex like way */
fprintf(tfp, "\\centerline{\\enorme ");
index = 0;
while (texfilename[index]) {
  switch (texfilename[index]) {
    case ´\134´: fprintf(tfp, "\\%c", ´\134´); break;
    case ´\046´: fprintf(tfp, "\\%c", ´\046´); break;
    case ´\045´: fprintf(tfp, "\\%c", ´\045´); break;
    case ´\043´: fprintf(tfp, "\\%c", ´\043´); break;
    case ´\137´: fprintf(tfp, "\\%c", ´\137´); break;
    case ´\173´: break;
    case ´\175´: break;
```

```
      case ´\044´: break;
      case ´\136´: break;
      case ´\176´: break;
      default : fprintf(tfp, "%c", texfilename[index]); break;
    }
    index++;
  }
  fprintf(tfp, "}\n");

  fprintf(tfp, "\\medskip\n");
  fprintf(tfp, "\\rightline{generated by mid2tex}\n");
  fprintf(tfp, "\\bigskip\n");
  fprintf(tfp, "\\normal\n");

  /* set pointers to start */
  for (i=0;i<notready;i++) {
    akt[i] = begin[i];
    while (akt[i] && (((akt[i]->marker) == ´k´) ||
           ((akt[i]->marker) == ´t´))) {
      if ((akt[i]->marker) == ´t´) {
        ts = (struct TimeSig *) akt[i]->data;
        if ((firstnom != ts->nom) || (firstdenom != ts->denom)) {
          fprintf(tfp, "\\metertoks%s={{\\meterfrac{%u}{%u}}",
                  romans[i], ts->nom, ts->denom);
          if (region[i] == 0)
            fprintf(tfp, "{\\meterfrac{%u}{%u}}", ts->nom, ts->denom);
          fprintf(tfp, "}%c\n", ´\045´);
        }
      }
      if ((akt[i]->marker) == ´k´) {
        ks = (struct KeySig *) akt[i]->data;
        if (kscontent != ks->sign) {
          fprintf(tfp, "\\sign%s=", romans[i]);
          if ((ks->sign) & 0x80) fprintf(tfp, "-");
          fprintf(tfp, "%u\n", (ks->sign) & 0xf);
        }
      }
      akt[i] = akt[i]->next;
    }
  }

  /* wtmark4 */
  fprintf(tfp, "\\debutmorceau\n");
  measurecounter = 1;
  fprintf(tfp, "%c measure %u\n", ´\045´, measurecounter);
  for (j=0; j<10; j++) {
    slur_and_beam[j] = 0;
```

```
    todelete[j] = 0;
    slur[j] = 0;
    beam[j][0] = 0;
  }

  readyflag = 1;
  events = 0;
  innerbar = 0;
  while (readyflag) {
    mintime = 0xfffffff;
    for (i=0; i<notready; i++)
      if ((akt[i]) && ((akt[i]->marker) != ´b´) &&
          ((akt[i]->time) < mintime))
        mintime = akt[i]->time;
    if (mintime == 0xfffffff) {
      /* ready or barres */
      readyflag = 0;
      for (i=0; i<notready; i++) if ((akt[i]) && (akt[i]->next))
         readyflag = 1;
      if (readyflag) {
        /* wtmark5: time for barre */
        flag = 0;
        for (i=0; i<notready; i++) {
          if (akt[i]) akt[i] = akt[i]->next;
          if ((akt[i]) && (((akt[i]->marker) == ´t´) ||
               ((akt[i]->marker) == ´k´))) flag = 1;
        }
        fprintf(tfp, "%c measure %u\n", ´\045´, ++measurecounter);
        if (!flag) {
          if ((events > LINEBREAKER) || (innerbar > INNERBARMAX)) {
            fprintf(tfp, "\\alaligne\n");
            events = 0;
          }
          else {
            fprintf(tfp, "\\barre\n");
            events += 3;
          }
          innerbar = 0;
        }
        else {
          for (i=0;i<notready;i++) {
            /* wtmark6 */
            while (((akt[i]->marker) == ´k´) || ((akt[i]->marker) == ´t´)) {
              if ((akt[i]->marker) == ´t´) {
                ts = (struct TimeSig *) akt[i]->data;
                fprintf(tfp, "\\metertoks%s={{\\meterfrac{%u}{%u}}",
                        romans[i], ts->nom, ts->denom);
```

```
              if (region[i] == 0)
                fprintf(tfp, "{\\meterfrac{%u}{%u}}", ts->nom, ts->denom);
              fprintf(tfp, "}%c\n", ´\045´);
            }
            else {
              ks = (struct KeySig *) akt[i]->data;
              fprintf(tfp, "\\signaturegenerale{%u}\n", (ks->sign) & 0x8f);
            }
            akt[i] = akt[i]->next;
          }
        }
        fprintf(tfp, "\\changecontext\n");
      }
    }
  }
  else {
    /* process all events of time mintime */
    events ++;
    innerbar++;
    if (innerbar > INNERBARZ) {
      /* allow line breaking in the middle of a takt */
      fprintf(tfp, "\\zbarre\n");
      innerbar = 0;
    }
    ksflag = 0;
    tsflag = 0;
    for (i=0; i<notready; i++) {
      dynhelp = akt[i];

      /* wtmark7 */
      /* process at first TimeSigns and KeySigns */
      while (dynhelp && ((dynhelp->time) == mintime) &&
          (((dynhelp->marker) == ´k´) || ((dynhelp->marker) == ´t´))) {
        switch (dynhelp->marker) {
          case ´k´:  ksflag = 1;
              ks = (struct KeySig *) dynhelp->data;
              fprintf(tfp, "\\sign%s=", romans[i]);
              if ((ks->sign) & 0x80) fprintf(tfp, "-");
              fprintf(tfp, "%u\n", (ks->sign) & 0xf);
              break;
          case ´t´:  tsflag = 1;
              ts = (struct TimeSig *) akt[i]->data;
              fprintf(tfp, "\\metertoks%s={{\\meterfrac{%u}{%u}}",
                      romans[i], ts->nom, ts->denom);
              if (region[i] == 0)
                fprintf(tfp, "{\\meterfrac{%u}{%u}}", ts->nom, ts->denom);
              fprintf(tfp, "}%c\n", ´\045´);
```

```
          break;
        default:  fprintf (stderr, "warning: unknown element1\n");
          break;
      }
      dynhelp = dynhelp->next;
    }
    akt[i] = dynhelp;
  }
  if (ksflag || tsflag) fprintf(tfp, "\\changecontext\n");

  /* wtmark8 */
  fprintf(tfp, "\\Notes");
  for (i=0; i<notready; i++) {
    if (i > 0) fprintf(tfp, " & ");   /* put a ´&´ between the satffs */
    while ((akt[i]) && ((akt[i]->time) == mintime) &&
        ((akt[i]->marker) != ´r´) && ((akt[i]->marker) != ´n´)) {
      switch (akt[i]->marker) {
        case ´l´:  ly = (struct Lyric *) akt[i]->data;
            if (!(region[i] & 0x1)) fprintf(tfp, "\\zcharnote {%s} {%s}",
                                      VIOLINELYRICPOSITION, ly->text);
            else fprintf(tfp, "\\zcharnote {%s} {%s}",
                         BASSLYRICPOSITION, ly->text);
            break;
        case ´p´:  lo = (struct Loudness *) akt[i]->data;
            if (!(region[i] & 0x1))
              fprintf(tfp, "\\zcharnote {%s} ", VIOLINELOUDNESSPOSITION);
            else fprintf(tfp, "\\zcharnote {%s} ", BASSLOUDNESSPOSITION);
            if (lo->statics) {
              /* deal with statics */
              fprintf(tfp, "{%s}", static_loudness[(lo->statics) - 1]);
            }
            else {
             /* deal with dynamics */
             fprintf(tfp, "{\\%s}", dynamic_loudness[(lo->dynamics)-1]);
            }
            break;
        default:        fprintf (stderr, "warning: unknown element2\n");
            break;
      }
      akt[i] = akt[i]->next;
    }

    dynhelp = akt[i];
    whilecrit = region[i];
    for (j=0; j<10; j++) todelete[j] = 0;
    while (whilecrit < 5) {
      akt[i] = dynhelp;
```

```
if (whilecrit == 4) fprintf(tfp, " | ");
while ((akt[i]) && ((akt[i]->time) == mintime) &&
          ((akt[i]->marker) != ´b´)) {
  switch (akt[i]->marker) {
    case ´r´:  re = (struct Rest *) akt[i]->data;
        if (((whilecrit == 0) && (re->position & 0x40)) ||
          ((whilecrit == 4) && (re->position & 0x80))) break;
        if (((re->position) & 0x3f) < 5)
           fprintf(tfp, "{\\transpose=-7\\relax ");
        if ((re->kind) < 5) {
          fprintf(tfp, "\\zcharnote {%c}{",
                  positions[(re->position) & 0x3f]);
          if (re->flg) fprintf(tfp, "\\off{0.56\\noteskip}");
          fprintf(tfp, "\\%s}", restcode[re->kind]);
        }
        else {
          fprintf(tfp, "\\ex%s ", restcode[re->kind]);
          fprintf(tfp, "%c", positions[(re->position) & 0x3f]);
        }
        if (((re->position) & 0x3f) < 5) fprintf(tfp, "}");
        break;
    case ´n´:  /* wtmark9 */
        not = (struct Note *) akt[i]->data;
        if (((whilecrit == 0) && ((not->flags) & 0x100)) ||
          ((whilecrit == 4) && ((not->flags) & 0x200))) break;
        if (not->slur_nr) {
          /* search slur number in array */
          j = 0;
          nothing = 10;
          while ((j < 10) && ((slur[j]) != (not->slur_nr))) {
            /* number 9 is not allowed for slurs */
            if ((!slur_and_beam[j]) && (j != 9)) nothing = j;
            j++;
          }
          if ((j < 10) || ((j == 10) && (nothing < 10) &&
              (!((not->flags) & 0x80)))) {
            /* valid slur */
            if (j == 10) {
              /* begin of slur */
              /* write slur as tie because of a better */
              /* appearance */
              j = nothing;
              slur_and_beam[j] = 1;
              slur[j] = not->slur_nr;
              if (((not->slur_pos) & 0x3f) < 5)
                 fprintf(tfp, "{\\transpose=-7\\relax ");
              fprintf(tfp, "\\ITen");
```

```
          if ((not->slur_pos) & 0x80) fprintf(tfp, "u");
          else fprintf(tfp, "l");
          fprintf(tfp, "%d%c",
             j, positions[(not->slur_pos) & 0x3f]);
          if (((not->slur_pos) & 0x3f) < 5) fprintf(tfp, "}");
        }
        else {
          /* end of slur */
          if ((not->flags) & 0x80) {
            slur[j] = 0;
            todelete[j] = 1;
            fprintf(tfp, "\\tten%d", j);
          }
        }
      }
    }
    if (not->beam_nr) {
      /* search beam number in array */
      j = 0;
      nothing = 10;
      while ((j < 10) && ((beam[j][0]) != (not->beam_nr))) {
        if (!slur_and_beam[j]) nothing = j;
        j++;
      }
      if ((j < 10) || ((j == 10) && (nothing < 10) &&
              (!((not->flags) & 0x40)))) {
        /* valid beam */
        if (j == 10) {
          /* begin of beam */
          j = nothing;
          slur_and_beam[j] = 1;
          beam[j][0] = not->beam_nr;
          k = 4;
          l = (not->flags) & 0x38;
          l = (l >> 3);
          if (((not->beam_pos) & 0x3f) < 5)
            fprintf( tfp, "{\\transpose=-7\\relax ");
          fprintf(tfp, "\\i");
          beam[j][1] = 0;
          while (k > l) {
            fprintf(tfp, "b");
            k--;
            (beam[j][1])++;
          }

          if ((not->beam_pos) & 0x80) {
            beam[j][2] = 1;
```

```
          fprintf(tfp, "u");
        }
        else {
          beam[j][2] = 0;
          fprintf(tfp, "l");
        }
        fprintf(tfp, "%d", j);
        if (((not->beam_pos) & 0x3f) > 41)
          printf("errerr %d\n", store);
        fprintf(tfp, "%c",
                positions[(not->beam_pos) & 0x3f]);

        /* output slope */
        l = (not->flags) & 0x7800;
        l = (l >> 11);
        if ((not->flags) & 0x8000) fprintf(tfp, "{-%d}", l);
        else fprintf(tfp, "%d", l);

        if (((not->beam_pos) & 0x3f) < 5) fprintf(tfp, "}");
        if ((4 - k) > (not->next_beamn_kind)) {
          k = 5 - (not->next_beamn_kind);
          beam[j][1] = k - 1;
          fprintf(tfp, "\\rlap{\\qsk\\t");
          while (k--) fprintf(tfp, "b");
          if (beam[j][2]) fprintf(tfp, "u");
          else fprintf(tfp, "l");
          fprintf(tfp, "%d}", j);
        }
      }
      else {
        if ((not->flags) & 0x40) {
          /* end of beam */
          fprintf(tfp, "\\t");
          k = 4;
          l = (not->flags) & 0x38;
          l = (l >> 3);
          while (k > l) {
            fprintf(tfp, "b");
            k--;
          }
          if (beam[j][2]) fprintf(tfp, "u");
          else fprintf(tfp, "l");
          todelete[j] = 1;
          fprintf(tfp, "%d", j);
          if (k != 3) {
            fprintf(tfp, "\\tb");
            if (beam[j][2]) fprintf(tfp, "u");
```

```
      else fprintf(tfp, "l");
      fprintf(tfp, "%d", j);
    }
    beam[j][0] = 0;
    beam[j][1] = 0;
    beam[j][2] = 0;
  }
  else {
    /* middle of beam */
    l = (not->flags) & 0x38;
    l = (l >> 3);
    if ((beam[j][1] < (4 - l)) &&
         ((not->next_beamn_kind) <= (beam[j][1]))) {
      fprintf(tfp, "\\t");
      k = 4 - l;
      while (k--) fprintf(tfp, "b");
      if (beam[j][2]) fprintf(tfp, "u");
      else fprintf(tfp, "l");
      fprintf(tfp, "%d", j);

      fprintf(tfp, "\\t");
      k = (not->next_beamn_kind) + 1;
      while (k--) fprintf(tfp, "b");
      if (beam[j][2]) fprintf(tfp, "u");
      else fprintf(tfp, "l");
      fprintf(tfp, "%d", j);
    }
    else {
      /* not->next_beamn_kind > beam[j][1] */
      /* means: next note more beamed than the */
      /* minimum of previous and aktual */
      if (beam[j][1] < (4 - l)) {
        if ((4 - l) > (not->next_beamn_kind))
            k = (not->next_beamn_kind);
        else k = (4 - l);
        fprintf(tfp, "\\n");
        while (k--) {
          fprintf(tfp, "b");
        }
        if (beam[j][2]) fprintf(tfp, "u");
        else fprintf(tfp, "l");
        fprintf(tfp, "%d", j);

        if ((4 - l) > (not->next_beamn_kind)) {
          fprintf(tfp, "\\rlap{\\qsk\\t");
          k = (4 - l);
          while (k--) fprintf(tfp, "b");
```

```
                          if (beam[j][2]) fprintf(tfp, "u");
                          else fprintf(tfp, "l");
                          fprintf(tfp, "%d}", j);
                        }
                      }
                      else {
                        /* beam[j][1] == (4 - 1) */
                        if ((4 - 1) > (not->next_beamn_kind)) {
                          fprintf(tfp, "\\t");
                          k = (not->next_beamn_kind) + 1;
                          while (k--) fprintf(tfp, "b");
                          if (beam[j][2]) fprintf(tfp, "u");
                          else fprintf(tfp, "l");
                          fprintf(tfp, "%d", j);
                        }
                      }
                    }
                    if ((4 -1) > (not->next_beamn_kind))
                      beam[j][1] = (not->next_beamn_kind);
                    else beam[j][1] = (4 -1);
                  }
                }

                if (((((not->note_pos) & 0x3f) <
                      (((not->beam_pos) & 0x3f) + UPPLOWSHIFT)) &&
                        ((not->beam_pos) & 0x80)) ||
                  ((((not->note_pos) & 0x3f) <
                      (((not->beam_pos) & 0x3f) - UPPLOWSHIFT)) &&
                        (!((not->beam_pos) & 0x80))))
                   sprintf(beamstring, "qh%d", j);
                else sprintf(beamstring, "qb%d", j);
                write_note(akt[i], beamstring);
              }
              else {
                not->beam_nr = 0;
                write_note(akt[i], "");
              }
            }
            else write_note(akt[i], "");
            break;
        default:  fprintf (stderr, "warning: unknown element3\n");
            break;
      }
      akt[i] = akt[i]->next;
    }
    whilecrit = (whilecrit * 5) + 4;
  }
```

```
          for (j=0; j<10; j++) if (todelete[j]) slur_and_beam[j] = 0;
        }
        fprintf(tfp, "\\enotes\n");
      }
    }
    /* wtmark10 */
    fprintf(tfp, "\\finmorceau\n\\end\n");
  }
}
```

Literaturverzeichnis

[Lit1Born] Günter Born. Noch mehr Dateiformate. -Bonn; Paris; Reading, Mass.[u.a.] : Addison–Wesley, 1995, ISBN 3-89319-757-5.

[Lit2Schenk] Paul Schenk. Allgemeine Musiklehre. -Leipzig: VEB Friedrich Hofmeister Musikverlag, 1956, Lizenz-Nr.484-250/D19/64.

[Sw1Taupin] Daniel Taupin. MusicTEX– Using TEXto write polyphonic or instrumental music. Laboratoire de Physique des Solides, batiment 510, Centre Universitaire, F-91405 ORSAY Cedex, October 1995.

[Sw2Anders] Dr.Jörg Anders. Readmidi - ein Tool zum Auslesen eines MIDI-Files in /dev/audio -kompatibler Form. TU-Chemnitz/Zwickau.

[Sw3Web] David Webber. MOZART, Version 1.4.3 Copyright ©1994-1996, 484 Warrington Rd.,Culcheth, Warrington, WA35RA, UK.

[Sw4CG] Chris Cannam, Andy Green. Rosegarden, Copyright ©1994, University of Bath, UK.

[Sw5Gnu] GNU C-Compiler gcc, Version 2.6, Free Software Foundation, Inc. 675 Mass.Ave, Cambridge, MA 02139, USA.

[Sw6Gnu] GNU Debugger gdb, Version 2, June 1991, Copyright ©1989,1991, Free Software Foundation, Inc. 675 Mass.Ave, Cambridge, MA 02139, USA.

[Sw7] TEX, Version 3.1415N.

[Sw8RES] DVIPS, Version 5.58a, Copyright 1986,1994, Radical Eye Software.