インターフェイスの街角 (17)
UNIX で MIDI!

増井俊之

最近は、パーソナル・コンピュータ上で DTM (DeskTop Music) を楽しむ人が多くなり、MIDI ファイルを公開している Web サイトもすこしずつ増えてきているようです。

Macintosh や Windows にくらべると、UNIX で音や画像を扱うためのツールはまだまだ十分とはいえません。しかし、UNIX で MIDI データを扱うためのツールが徐々に出始めており、UNIX マシン上で MIDI による音楽を聴いたり、あるいは楽音によるインターフェイスを利用することが可能になってきました。今回は、UNIX で MIDI を活用する手法について解説します。

音によるインターフェイス

現在の計算機でよく使われる GUI はもっぱら視覚情報に頼っており、人間の五感を活用しているとはとてもいえません。メールの到着やエラーの発生をユーザーに知らせるために音を出す程度のことはよくおこなわれていますが、もっと幅広い用途に音声以外の音情報 (non-speech audio) を活用するための研究が進められています。

たとえば、画面上で各種の機能を表すアイコンと同程度の有用性を実現するために、計算機の状態などを音で表現する ``earcon''という考え方が提案されています [1]。さまざまなエラーを 1 つのビープ音で表現するのではなく、エラーの種類に応じて異なる音を出す earcon を利用すれば、耳でエラーの状態を判別できて便利かもしれません。たとえば、すべてのエラー通知音は ``ド''で始まることにしておき、コンパイル時のエラーの場合はその後ろに ``ミ''を続け、構文エラーであればその後ろに ``ソ''を付け加える、といったふうにすれば、コンパイラが構文エラーを検出した際に ``ドミソ''という音で通知できます。さらに、音質や音の高さが異なる earcon を用意しておき、さまざまな状態やイベントを音だけで知らせることも可能でしょう。たとえば、Emacs ではどのような操作をしても計算機は視覚的にしか応答しません。したがって、操作を誤ったときに計算機の反応を見逃してしまい、あとで青ざめることが (すくなくとも私の場合は) よくあります。しかし、領域を削除したときに削除を表す earcon でフィードバックをおこなうようにしておけば、そのような間違いは減るはずです。

計算機からユーザーへのフィードバックに、視覚的な情報だけでなく音情報も活用しようという試みもあります。GUI では、画面上でマウスを動かすときに場所に応じてマウスカーソルを変化させ、機能などをユーザーに通知する手法が使われています。これらの情報の代わりに楽音を使用して、カーソルを動かすときや、カーソルがウィンドウ枠を越えたときに音を出したり、背景のテクスチャに応じて移動音を変えたり、あるいは現在のマウスカーソルの位置を音で知らせるといった試みもおこなわれています。

音を計算機で利用する場合、これまでは AIFF や WAV などの波形データがよく使われてきました。しかし、多種多様な音を扱うときは MIDI データのほうが効率的です。

ソフトウェア・シンセサイザー

従来、MIDI で音を出すためには、MIDI 音源ハードウェアが使われてきました。しかし、最近はすべての音源処理をソフトウェアでおこなう、いわゆる**ソフトウェア・シンセサイザー**が急速に普及しつつあります。ソフトウェア・シンセサイザーでは、特別なハードウェアがなくても

ソフトウェアをインストールするだけで MIDI 音楽を楽しむことができます。ソフトウェア・シンセサイザーの登場とともに、MIDI 音楽への関心がさらに高まりつつあるようです。

Windows 95/98 では、ヤマハの XG SoftSynthesizer、ローランドのバーチャル・サウンドキャンバス、QuickTime3 などのソフトウェア・シンセサイザーが使えます。ヤマハは、XG SoftSynthesizer を Web ブラウザ経由で使えるようにするプラグイン MIDPLUG for XG を配布しています。これを組み込めば、Web ブラウザで XG SoftSynthesizer の音が聴けるようになります。

ソフトウェア・シンセサイザーなどで MIDI データを扱うときは、通常は ``.mid''という拡張子の付いた **SMF** (Standard MIDI File) 形式のデータを使います。

UNIX 上では、これらのソフトウェア・シンセサイザーや Web ブラウザの MIDI プラグインはまだあまり使われていないようです。しかし、**TiMidity** というソフトウェア・シンセサイザーを利用して、UNIX 上で MIDI 音楽を演奏したり、Netscape Navigator 上で MIDI 音楽を聴くことができます。

TiMidity

TiMidity は、1995 年ごろに Tuukka Toivonen 氏が開発したソフトウェア・シンセサイザーです。残念ながら、Toivonen 氏は TiMidity の開発やサポートを終了してしまいましたが、出雲まさなお氏が大幅に機能を拡張した TiMidity++を配布しています。

TiMidity は、PC 用のサウンドカードとして人気の高かった Gravis[1]の UltraSound（略称 GUS）用にフリーで配布されていた音源ファイル（*.pat ファイル）を音源データとして使います。**wav2pat** というツールで、WAV ファイルから*.pat ファイルを作ることもできます。

TiMidity++は、gtk+や Ncurses を含む各種のグラフィカル・インターフェイス、ノイズフィルタやリバーブ、コーラスなどの各種エフェクト、AIFF などの出力形式での保存、ネットワーク上の SMF や圧縮された SMF の演奏など、さまざまな機能をもっています。もっとも単純な利用形式では、インターフェイスに ``dumb''を使って次のようにコマンド行から MIDI ファイルを指定すれば演奏できます。

1 現在は、ゲーム用ジョイスティックなどを販売しています。

図 1　UMP の表示

図 2　MIDPLUG の表示


```
% timidity -id chou.mid
MIDI file: chou.mid
Playing chou.mid
Format: 1  Tracks: 3  Divisions: 480
Track name: (piano)
Track name: (bass)
（演奏開始……終了）
flush_output ?
Playing time: ~15 seconds
Notes cut: 0
Notes lost totally: 0
%
```

TiMidity の Netscape プラグイン UMP

TiMidity を UNIX 上の Netscape Navigator で使うためのプラグインとして、UMP (UNIX MIDI Plug-in) が公開されています。TiMidity++の配布パッケージにも Netscape 用のプラグイン作成ツールが含まれていますが、FreeBSD や Linux、Solaris、IRIX などでは、こちらを使うほうが簡単です。

HTML に以下のようなタグを記述すると、SMF ファイル chou.mid を演奏する Web ページが作れます。

```
<EMBED SRC="chou.mid" WIDTH=150 HEIGHT=40>
```

UMP を組み込んだ Netscape では図 1 のようなロゴが表示され、chou.mid のデータが演奏されます。

ヤマハの MIDPLUG を組み込んだ Windows 上のブラウザでは、同じ HTML に対して図 2 のようなパネルが表示され、早送り/停止などの制御ができます。

音楽記述言語の利用

SMF 形式の MIDI データがあれば、TiMidity やヤマハ XG などのソフトウェア・シンセサイザーで MIDI

音楽を聴くことができますが、音楽を自作するには MIDI データの作成方法が課題になります。

現在、インターネット上で公開されている MIDI データのほとんどは、パーソナル・コンピュータ用の市販の DTM ソフトウェアで作られているようです。Macintosh では、以前から Performer や Vision などの GUI ベースのソフトウェアがよく使われていましたし、最近は Windows でも動く Logic、Cakewalk、XGworks、ミュージ郎などを利用する人が増えています。

これらの GUI ベースの DTM ソフトウェアが登場する前は、MS-DOS 上などでテキストベースの音楽記述言語[2]が使われていました（いまでもひろく利用されているようです）。残念ながら、UNIX では GUI を用いた DTM ソフトウェアはまだほとんどありませんが[3]、テキストベースの音楽記述言語の多くは UNIX でも利用できます。

テキストベースの音楽記述は、楽譜などとくらべると読みにくいものです。しかし、変数やマクロ定義、制御構造など、プログラミング言語で一般的な機能が使えるものもあり、かえって音データを作成しやすい場合もあります。テキストベースの記述であれば使い慣れたエディタやテキストフィルタをそのまま利用できますし、自動生成も比較的容易です。以下では、UNIX で使える音楽記述言語をいくつか紹介します。

PMML

PMML (Practical Music Macro Language) は、会津大学の西村 憲氏の開発した音楽記述言語です。図 3 は、PMML で記述した「ちょうちょ」です。PMML の構文は C 言語によく似ていて、MIDI 信号をきめ細かく制御するための数多くの機能に加え、マクロ定義、変数、数式、条件分岐、繰返しなど、汎用のプログラミング言語のもつ機能や、乱数生成関数など特殊な効果を出すのに便利な機能も備えています。

SMF ファイルは、以下のように PMML コンパイラ **pmml** を用いて生成します。

```
% ls
chou.pml
% pmml chou.pml
```

2 MML (Music Macro Language) と呼ばれることもあります。

3 NEXTSTEP で動作するツールがありましたが、最近はほとんどみかけません。

図 3　PMML による「ちょうちょ」の記述 (chou.pml)

```
//
// PMMLによる「ちょうちょ」
//
// [G E]  ソとミの同時発声を示す
// ^G     1オクターブ上のソ音
// _G     1オクターブ下
// r      休符
//
tempo(160)
timesig(4,4)
newtrack(piano){
    ch=1 o=4 v=96 prog(1) vol(96)
}
newtrack(bass){
    ch=2 o=3 v=120 prog(33) vol(96)
}
piano {
    for($i,1,2){
        [G E] [E C] [E C] r
        [F D] [D _B] [D _B] r
        C D E F G G G r
    }
}
bass {
    E C C r D _G _G r _A _B C D E E E r
    C _ E F F# G A B G A ^C B G ^C G C r
}
```

```
% ls
chou.mid   chou.pml
%
```

SPICE

後藤浩昭 (GORRY) 氏の開発した SPICE は、音楽記述言語による記述と SMF とを相互に変換するシステムです。SPICE は、もともとパーソナル・コンピュータ用の BASIC で MIDI を扱うために開発されたもので、雑誌などでの公開やデータ交換を容易にするための簡潔な記述に特徴があります。任意の数式を評価したり関数を定義する機能はありませんが、マクロ定義、変数の使用、繰返し制御など、楽曲記述や MIDI 制御のための高度な機能を数多く備えています。図 4 は、SPICE で「ちょうちょ」を記述した例です。

SMF ファイルは、以下のように SPICE コンパイラ **smc** を使って生成します。

```
% ls
chou.smc
% smc chou.smc
% ls
chou.mid   chou.smc
%
```

図 4　SPICE による「ちょうちょ」の記述 (chou.smc)

```
#tempo 150
;
100  @z1   ; MIDIチャンネル1をトラック100に割り当てる
101  @z2   ; MIDIチャンネル2をトラック101に割り当てる
;
100  @000 r  ; トラック100にピアノを使用
101  @032 r  ; トラック101にウッドベースを使用
;
100  v100 r w127 r p64 r M0
101  v127 r w127 r p64 r M0
;
;  |: ..... :| で繰返しを指定
;  g0&eでソとミの同時発声を指定
;  rは休符
;  < と > でオクターブの上下を指定
;
100  |:2 <g0&e e0&c e0&c r f0&d d0&>b <d0&>b r
100  <c d e f g g g r :|
101  e c c r d >g g r a b <c d e e e r
101  c >e f f+ g a b g a <c >b g <c >g c
;
```

t2mf/mf2t

Piet van Oostrum 氏が開発した t2mf と mf2t は、バイナリファイルである SMF とそのテキスト表現とを相互に変換するためのシステムです。PMML や SPICE は、ユーザーが自分でテキストを編集して音楽を作るために工夫された言語であり、マクロや制御構造などのプログラミング言語の機能をもっています。これに対し、t2mf はテキスト形式で表現した MIDI データを SMF のバイナリデータに変換するだけなので、テキストの入力や修正は簡単ではありません。とはいえ、MIDI データを生成するプログラムを作るような場合は、SMF を直接生成するよりも、t2mf 用のテキストを生成してから t2mf で SMF に変換するほうがはるかに扱いやすくなります。

図 5 は、t2mf による「ちょうちょ」の記述例です。PMML や SPICE での記述にくらべてデータ量が増えてしまいますが、MIDI 信号やデータを目で見て確認するときは有効でしょう。

SMF ファイルは、以下のように t2mf を使って生成します。

```
% ls
chou.txt
% t2mf chou.txt > chou.mid
% ls
chou.mid   chou.txt
%
```

図 5　t2mf による「ちょうちょ」の記述 (chou.txt)

```
#
# t2mfによる「ちょうちょ」の記述
#
# MFile フォーマット トラック数 ピッチベンド
MFile 0 1 96
# トラック開始
MTrk
#
0 Meta Text "Choucho"
0 Tempo 400000
#
# チャンネル1にピアノ(0)
# チャンネル2にウッドベース(32)を割り当て
#
0 PrCh ch=1 p=0
0 PrCh ch=2 p=32
#
# ピアノ: ソ = 67
# ベース: ミ = 52
#
0 On ch=1 n=67 v=96
0 On ch=2 n=52 v=96
100 Off ch=1 n=67 v=0
100 On ch=1 n=64 v=96
100 Off ch=2 n=52 v=0
100 On ch=2 n=48 v=96
200 Off ch=1 n=64 v=0
200 On ch=1 n=64 v=96
200 Off ch=2 n=48 v=0
      ……
1300 On ch=2 n=52 v=96
1400 Off ch=1 n=65 v=0
1400 On ch=1 n=67 v=96
1400 Off ch=2 n=50 v=0
1400 On ch=2 n=52 v=96
1600 Meta TrkEnd
TrkEnd
```

汎用言語からの MIDI 制御

PMML や SPICE などの音楽記述言語は、音の生成や効果の付加といった MIDI 特有の機能に、マクロ定義や制御構造といったプログラミング言語の要素を加えたものになっています。

PostScript に代表されるページ記述言語、PMML や SPICE などの音楽記述言語、あるいは TEX などの文書整形言語、Director に代表されるオーサリング言語は、もともとアプリケーションに固有の処理を簡単に記述することを第一に考えられたものですが、複雑な処理をおこなうための汎用プログラミング言語的な拡張が組み込まれています。ところが、その結果として言語ごとに変数定義や制御構造の記述方式が大きく異なってしまい、ユーザーはア

プリケーションの数だけプログラミング言語を憶えなくてはなりません[4]。目的の機能に固有の記述手法と、汎用的なプログラムの表現手法とを分離して扱えればいいのですが、そのための適切な手法はまだ確立されていません。

一方、汎用のプログラミング言語のなかからアプリケーション固有の処理を呼び出す手法は従来からよく使われています。たとえば OpenGL は、特殊なグラフィックス記述言語ではなく、ライブラリとして汎用のプログラミング言語から呼び出す形式なので、複雑なプログラムも作りやすくなっています。最近のインターフェイス作成ツールの多くは、特殊なインターフェイス記述言語を使わずに汎用言語のライブラリとして実現されています。可能であれば、このような実現方式のほうが複雑な処理には適しています。

MIDI による音楽についても、汎用言語からライブラリとして音楽が記述できればいろいろと都合がよいと思います。たとえば、プログラムのなかで前述の earcon を使いたい場合、音楽記述言語を使うよりはライブラリとして呼び出すほうが便利です。

ストトンシステム

私が開発した〝ストトンシステム〟は、Perl で直感的な音楽記述をおこなうシステムです。専用言語で特殊な表現を用いて音列を表現するのではなく、Perl のような汎用プログラミング言語を使います。たとえば、&play(”ドレミファ”) という関数を実行すると〝ドレミファ〟という音が鳴り、&play(”スットントン”) を実行すればそれに似た音のドラムが鳴ります。

ストトンシステムは専用の音楽記述言語を使わず、Perl のライブラリとして MIDI を制御するため、Perl の変数や繰返し制御文などがそのまま演奏に利用できます。さらに、プログラムのエラーメッセージを MIDI で表現するといったことも簡単です。

プログラムの例

ストトンシステムでは、プログラムをそのまま音読できるようにするために〝ド〟や〝レ〟などの階名をそのまま使用します。たとえば、ピアノで〝ドレミファ〟を演奏させるプログラムは次のようになります。

```
#!/usr/local/bin/perl
# sample1
require 'midi.pl';
&prologue;
&setinst(1,'Piano');
&channel(1);
&play("ドレミファ");
&epilogue;
```

ストトンシステムは、t2mf で解釈できるテキストを出力します。これを t2mf の入力に与えれば SMF が生成されます。たとえば、上のサンプル・プログラムと t2mf、TiMidity を組み合わせて、

```
% sample1 | t2mf | timidity -id -
```

のようにすれば音を出すことができます。

〝ドレミファ〟を 10 回繰り返して演奏したい場合には、次のように普通の Perl の制御文が使えます。

```
#!/usr/local/bin/perl
# sample2
require 'midi.pl';
&prologue;
&setinst(1,'Piano');
&channel(1);
for (1..10){
    &play("ドレミファ");
}
&epilogue;
```

「どんぐりころころ」をピアノとベースで演奏するには、並列発音を指示する関数 parbegin と parend を使います。

```
#!/usr/local/bin/perl
# sample3
require 'midi.pl';
&prologue;
&setinst(1,'Piano');
&setinst(2,'Bass');
&channel(2);
&play("__");  # ベースを2オクターブ下げる
&parbegin;
    &channel(1); # ピアノチャンネル
    &serbegin;
        &play("ソッミミファミレド");
        &play("ソッミミレーーッ");
    &serend;
    &channel(2); # ベースチャンネル
    &serbegin;
        &play("ミッドドレレソソミミドドソーーッ");
    &serend;
&parend;
&parbegin;
    &channel(1);
    &play("ミミソソララッラ^ドド_ミミソーッッ");
```

4 TEX で制御構造のあるプログラムを書くのは悪夢としか思えません。

```
    &channel(2);
    &play("ドド ミミファファド ファレレララソーッッ");
&parend;
&epilogue;
```

オクターブの上下は、PMML と同じ表記法を使っています。

parbegin と parend に囲まれた play で指示される音列は同時に発音されます。ただし、serbegin から serend までの部分は逐次的に発音されます。

ストトンシステムは Perl プログラムですから、任意の Perl の式や組込み関数も利用できます。たとえば、次のプログラムはランダム音を 200 回生成します。

```
#!/usr/local/bin/perl
# sample4
require 'midi.pl';
&prologue;
&setinst(1,'Piano');
&channel(1);
for $i (1..200){
    $j = int(rand(7));
    &play($j == 0 ? 'ド' :
          $j == 1 ? 'レ' :
          $j == 2 ? 'ミ' :
          $j == 3 ? 'ファ' :
          $j == 4 ? 'ソ' :
          $j == 5 ? 'ラ' :
          $j == 6 ? 'シ' :
          '');
}
&epilogue;
```

ストトンシステムの実装

ストトンシステムのおもな仕事は、play に渡される階名リストを MIDI 信号の番号に変換して発音時刻順にソートするだけなので、実装は比較的簡単です。play では階名の列を MIDI のイベント列に変換し、その終了後、または parend、serend が呼ばれた時点で、それまでに蓄積した出力イベント列を時刻順にソートして t2mf の形式で出力します。

これらすべての処理は、末尾に示した短いライブラリ **midi.pl** 内の関数だけでおこなえます。

ストトンシステムの応用

ここでは一例として Perl でシステムを記述しましたが、C などのプログラミング言語でも同様に MIDI 音楽を簡単に出力できるので、さまざまな応用方法が考えられます。

- 自動作曲

  BGM 程度なら、いくつかのフレーズを適当に組み合わせて並べるだけで簡単に作れるでしょう。

- earcon の応用

  earcon を用いたプログラムも、次のように手軽に書けるようになります。

```
if((f = fopen(file,"r"))==NULL){
    error("ド ミソミド ");
}
```

- システム状態の通知

  昼休みにチャイムを鳴らしたり、メールが届くと音楽を演奏したり、トラフィックの状態を通奏低音で鳴らしたり、システムのさまざまな状態を MIDI の音で通知できます。

そのほかに、波形ファイルを用意するのが面倒なために音データの使用をためらっていたような応用でも、ストトンシステムなどの簡便な MIDI 音生成システムであれば気軽に活用できるのではないでしょうか。

おわりに

UNIX 上での DTM はまだまだこれからです。しかし、とりあえず MIDI を使って音を出せるようになれば、おもしろい応用方法が出てくるのではないでしょうか。機械の合成音でエラーを通知するようなインターフェイスは不快に感じられることもあるようですが、美しい音で通知すれば違和感が解消されるかもしれません。楽音を活用したインターフェイスの発展に期待したいと思います。

（ますい・としゆき　ソニー CSL）

[参考文献]

[1] Meera M. Blattner, D. A. Sumikawa and R. M. Greenberg, Earcons and icons: Their structure and common design principles, in *Human-Computer Interaction*, Vol.4, No.1, pp.11–44, 1989

訂正とお詫び

前号で紹介した URL に誤植がありました。以下のとおり訂正し、読者ならびに関係各位にお詫びいたします。

- 160 ページ右段

  誤：http://meringue.mapfan.com/...

  正：http://www.mapfan.com/...

関連 URL

TiMidity	http://www.cgs.fi/~tt/
TiMidity++	http://www.goice.co.jp/member/mo/timidity/
TiMidity プラグイン UMP	http://pubweb.bnl.gov/people/hoff/
PMML	http://www-cgl.u-aizu.ac.jp/pmml/index-j.html
SPICE	http://na01.shonan.ne.jp/~gorry/spice/
t2mf、mf2t	http://ring.aist.go.jp/pub/pack/dos/art/music/conv/mf2tsrc.zip
ヤマハ	
XG SoftSynthesizer	http://www.yamaha.co.jp/xg/s-synth/s-synth.html
MIDPLUG for XG	http://www.yamaha.co.jp/xg/midplug/
ローランド	
バーチャル・サウンドキャンバス	http://www.rolandcorp.com/japan/lib/download/VSC88V2T.html
Gravis	http://www.gravis.com/

midi.pl

```
%instno = ( # 楽器名と音源番号の対応
    'Piano', 0, 'Bass', 32, # ........
);
%note = ( # 音名と音高番号の対応
    'ド', 60, 'レ', 62, 'ミ', 64, 'ファ', 65,
    'ソ', 67, 'ラ', 69, 'シ', 71, 'ッ', -1,
);
$notepat = join('|',keys(%note));
$level = 0;          # ネストレベル
$env[$level] = 's'; # par / ser
$channel = 1;
$unitlen = 100;

sub setunitlen{  # 演奏単位時間の設定
    $unitlen = $_[0];
}
sub setinst {
    local($ch,$inst) = @_;
    $inst{$ch} = $instno{$inst};
    &add("0 PrCh ch=$ch p=$inst{$ch}\n",0);
}
sub channel {
    local($ch) = @_;
    $channel = $ch;
}
sub parbegin {
    &begin('p');
}
sub serbegin {
    &begin('s');
}
sub begin {
    $level++;
    $env[$level] = $_[0];
    $out[$level] = '';
    $len[$level] = 0;
}
sub serend {
    &parend;
}
sub parend {
    $level--;
```

```
    &add($out[$level+1],$len[$level+1]);
}
sub add { # 既存データに新しいデータを追加
    local($buf,$len) = @_;
    if($env[$level] eq 's'){ # データを連接
        $out[$level] .= &timeshift($buf,$len[$level]);
        $len[$level] += $len;
    }
    else { # データをマージ
        $out[$level] = &timesort($out[$level].$buf);
        $len[$level] = ($len[$level] > $len ?
                $len[$level] : $len );
    }
}
sub play {
    local($_) = @_;
    local($t,$buf,$note,$notename);
    while($_ ne ''){
        if(s/^_//){
            $offset[$channel] -= 12; next;
        }
        if(s/^\^//){
            $offset[$channel] += 12; next;
        }
        if(s/^--//){
            $t += $unitlen; next;
        }
        if(s/^($notepat)([#b]?)//){
            $notename = $1;
            $sharpflat = $2;
            if($t > 0 && $note > 0){
                $buf .= sprintf("%d Off ch=%d n=%d v=0\n",
                    $t-1,$channel,$note+$offset[$channel]);
            }
            $note = $note{$notename};
            $note += ($sharpflat eq '#' ? 1 :
                  $sharpflat eq 'b' ? -1 : 0);
            if($note > 0){
                $buf .= sprintf("%d On ch=%d n=%d v=96\n",
                $t,$channel,$note+$offset[$channel]);
            }
        }
        else {
            s/^.//;
        }
        $t += $unitlen;
    }
    if($t > 0 && $note > 0){
        $buf .= sprintf("%d Off ch=%d n=%d v=0\n",
            $t-1,$channel,$note+$offset[$channel]);
    }
    &add($buf,$t);
}
sub bytime {
    local($ta,$tb);
    $a =~ /^(\d+)\s/;
    $ta = $1;
    $b =~ /^(\d+)\s/;
    $tb = $1;
    $ta <=> $tb;
}
sub timesort { #イベントを時刻順にソート
    local($s) = @_;
    local(@a);
    @a = split(/\n/,$s);
    @a = sort bytime @a;
    join("\n",@a) . "\n";
}
sub timeshift { #イベント発生時刻をシフト
    local($s,$t) = @_;
    local(@a,$i,$s1,$s2);
    @a = split(/\n/,$s);
    for($i=0;$i<=$#a;$i++){
        $a[$i] =~ /^(\d+)\s+(.*)$/;
        $s1 = $1; $s2 = $2;
        $a[$i] = $s1+$t . " $s2";
    }
    join("\n",@a) . "\n";
}
sub prologue {
    print "MFile 0 1 96\n";
    print "MTrk\n";
    print "0 Tempo 200000\n";
}
sub epilogue {
    print $out[0];
    print "$len[0] Meta TrkEnd\n";
    print "TrkEnd\n";
}
1;
```