構造的音楽記述言語「楽」とそのコンパイラの実現法
—MIDIによる計算機音楽研究用システム—

松　原　康　夫

Specification of a Music Description Language "GAKU" and its Compiler Construction
—A Studying System for Computer Music—

Yasuo Matsubara

A language to describe music data is important for studies of computer music. The language must be easy not only to read or write, but also to process with a computer. Many such languages have been proposed. None of them, however, are widely used. It seems now that more research of such languages is required.

Music can be considered as parallel occurrences of certain sequences. Such sequences may themselves include not only notes but also further parallel occurrences of notes or other sequences.

In this paper, "GAKU", a new language to describe music is introduced. With this language, parallel occurrences of elements can be described by enclosing them with { }, and sequential occurrences of elements can be described by enclosing them with ( ). An element may be a note, a parallel occurrence or a sequential occurrence. Therefore, a piece of music may be described in a structured manner.

In addition, the construction of a compiler to process the language is described. This compiler outputs a code which can be performed using a MIDI instrument.

1．はじめに

現代においては音楽とコンピュータとの関わりは大変深いものとなっており，日常接する音の多くは何らかの形でコンピュータが関わっている。

元来音楽は人間によって作曲され，人間によって演奏されてきた。人間が行なうこのような活動は詳しく見て行くと極めて複雑な内容を含んでいる。

最も簡単と考えられる演奏についても，最初に楽譜を目で捉えるパターン認識の過程を含む。さらにそれを手や息などの動作に変換して実際の音を出して行く。

楽譜で記述されているのは骨組みだけであり，豊かな表情は演奏を行なう人間が付け加えるのである。この過程がどのように行なわれるかについては全く不明と言うしかない。

このような演奏が良くて，あのような演奏は良くないという専門的な方法論が少し存在するのであろう。しかしそれを数学的に定式化することは困難である。

そして人間が演奏するときに具体的にどのような演奏をしているかを捉えることもなかなか困難である。

これに対し，コンピュータを音楽に応用することはいろいろな形で行なわれている。

演奏をコンピュータで自動化することはコンピュータ音楽の中でも最も頻繁に実行されていることである。技術が伴わないために音楽を演奏することができなかった非専門家にも，コンピュータを使うことによって演奏が可能となった。

現代においてコンピュータを使った音楽を大衆化させた主役がMIDIである。MIDIとはMusic Instrument Digital Interfaceの略であり，いろいろなディジタル楽器を結び付けるインターフェースの規約である[13]。

MIDIにはいろいろと不十分な点もあるが少なくとも従来の楽譜で記述できる音楽の演奏には現実的な意味では十分であるし，その普及度から比較的安価にいろいろな実験ができる。

次に作曲の問題を考えてみる。これにもいくつかの観点がある。一つはコンピュータを文房具として使うことによって作曲の中の純粋に機械的な部分を軽減することである。これは作曲家がワープロを使うのと同様である。もう一つはこれまで人間が作った曲を分析することで，あるジャンルの音楽または特定の作曲家の傾向を調べることである。音楽に限らず芸術作品は，必然性と任意性の両面を有する。人間は次に何が来るかが方程式で一意的に決まってしまうものや，逆に何の法則もない乱数等には美を感じないといわれている。丁度その中間に位置するものが芸術作品となる。コンピュータを使って音楽全般または特定のジャンルまたは作曲家について，何が必然で何が任意であるかを探ることも興味深いことである。

さらには，コンピュータを使って従来にはない音楽の可能性を開拓することが考えられる。

一つは従来の楽器では出せなかった音を使って曲を作ることである。単に半音より細かい微分音だけではなく，周波数の帯域にある広がりもつ音をコンピュータで合成する[10]。

また，演奏と人間の関わり方を革新するものもある。例えば人間が描いた絵を自動的に音に変換したり，各種のセンサーからの信号を音に変換するものもある[9]。

計算機音楽の研究にもいろいろな方向がある。例えば曲をデータとして入力し，その構造を分析するが，その方法として統計処理をする情報論的な方法，構造をAI的に分析する方法や，言語理論を応用するなど多岐に渡っている[5]。また，楽器の音を分析したり逆に合成する研究は古くから進められており[2]，分析にはフーリエ変換，線形予測符号化などの方法が使われている。合成には加算合成，減算合成，など多くの方法が考えられており，最近では実際の楽器の発音機構をシミュレートすることも行なわれている。

人間が音楽をどのように認識するかを考える認知科学的な研究もあり[1)8)]，AIや自動作曲などとも関連して今後の重要なテーマとなるであろう。

計算機の高度な技術が要求される分野としては楽譜の自動読み取りや，自動採譜などがある。楽譜は人間には容易に読み取ることができるが，計算機でこれを行なうことは困難なパターン認識の問題となる[6]。自動採譜は単音の場合はある程度実用になるが，複音についてはAIの手法を応用して人間の聞く過程をシミュレートすることなどが試みられている。

計算機で曲を分析するにしても，また演奏

をさせるにしても，コンピュータが認識できる形でデータを記述することが必要になる。

これまでにもいろいろな言語仕様が提案され[3)4)]，実現されている。ASNIによる標準化案も提案されている[4)]が，一般に受け入れられているとは言い難い。音楽記述言語は，まだまだいろいろなことを試行するべき段階にあると考えられる[16)]。

音楽記述言語は人間が理解しやすく書きやすいという条件と同時に，計算機での処理が容易である必要があり，この点で一般のプログラミング言語と共通する。従ってプログラミング言語を処理するコンパイラの技法[14)15)]を如何にして応用するかが，記述の容易性と高い記述能力を同時に実現するための鍵となる。

本論文では，音楽に内在する直列性と並列性を構造的に記述できることと，音符が自然な形で記述できることに重点をおいた音楽記述言語の仕様を提案する。

併せて，この言語を処理するコンパイラを構築するための技法に付いても述べる。

2．MIDIインターフェース

我々はMIDIインフェースを基本とした計算機音楽の研究用システムの構築を目指しているので，MIDIについて述べておこう。

MIDIの規格にはハードウェアに関する規定と，それによって伝送されるデータに関する規定がある。

2．1　ハードウェア

ハードウェアとしては非同期方式のシリアル転送を行なう。転送は8ビット単位で行なわれ，前後にスタートビットとストップビットが付け加えられる。転送速度は31.25 K Baudeなので1バイト送るのに320 μSかかる。電気的には5 mAカレントループであり，論理0は電流が流れている状態とする。5ピンのDINコネクタを使い，レシーバにはフォトカプラを使うなどの規定がある。

2．2　データフォーマット

データは一つまたは複数のバイトからなるメッセージでやり取りする。メッセージは最初にステータスと呼ばれるバイトが存在し，これによってメッセージの種類が定まる。それによって幾つのデータバイトが以下に続くかをも指定する。データバイトは0，1，2バイトのいずれかである。ステータスバイトのMSBは1とし，データバイトのMSBは0とすることで両者は区別される。

メッセージには大きく分けてチャンネルメッセージとシステムメッセージがある。チャンネルメッセージは1～16までのいずれかのチャンネルを指定して送られるものであり，そのチャンネルに該当しない機器はこれを無視する。チャンネルメッセージについてはあとでさらに詳しく述べる。

システムメッセージは特定のチャンネルに限らず，接続された全ての機器を対象とするものであり，コモンメッセージ，リアルタイムメッセージ，エクスクルーシブメッセージがある。

コモンメッセージにはソングポジションポインタ，ソングセレクト，チューンリクエスト，EOXがあり，それぞれ曲のどの位置から開始するか，どの曲を選択するか，音高の調整，エクスクルーシブメッセージの終わりを示すため等に使われる。

リアルタイムメッセージは接続された機器をうまく同期させるために使われる。よく使われるものとして次のようなものがある。

0xFA　スタート
0xFC　ストップ
0xFF　システムリセット

このように，本論文では16進数を表わすのに“0x”を頭に付ける。
リアルタイムメッセージは1バイトのステー

タスだけであり，データバイトを伴わない。

エクスクルーシブメッセージはMIDIの統一規格に乗り切らない，メーカー毎に異なる機器の設定等に利用される。そのためにメーカーはID番号をMIDI規格協議会からもらう必要がある。

データフォーマットは，最初に0xF0が来てエクスクルーシブメッセージの始まりを示す。その直後の1バイトはID番号を表わす。そのあとに任意個数のデータバイトが続き，最後に0xF7（EOX）が来てエクスクルーシブメッセージの終わりを示す。0xF0と0xF7の間のバイトのMSBは全て0とする。

2．3　チャンネルメッセージ

チャンネルメッセージにはボイスメッセージとモードメッセージがある。

先にモードメッセージについて述べる。モードには1～4があり，それぞれOMNIモードのon/offとpoly/Monoモードを指定する。

Mode	Omni	Poly/mono
1	on	Poly
2	on	mono
3	off	Poly
4	off	mono

Omniモードがonと言うことは全てのチャンネル番号のボイスメッセージに反応すると言うことで，チャンネルの概念が普及していない時に電源を入れればとりあえず音が出るようにするために設けられたものである。現代では電源投入時にモード3になる機器が一般的である。

次にボイスメッセージについて述べる。ボイスメッセージは実際に楽器の音をコントロールするものである。ステータスの下位4ビット（Xで表わす）はチャンネル番号から1を引いた数を表わす。以下においてノートナンバーが使われるが，これは中央のCの音を60として半音毎につけられた番号である。つまりハ長調でト音記号の下の下線のドが60であり，その1音上のレは62となる。以下にメッセージの名前，ステータスバイトの値，データバイト数，の順に記し，説明を加える。

(1)ノートオフ　　0x8X　　2

音を止めるものであり，最初のデータバイトでノートナンバーを表わす。2番目のデータバイトはノートオフベロシティと呼ばれるが通常は意味を持たない。

(2)ノートオン　　0x9X　　2

最初のデータバイトはノートナンバーを表わし，2番目のデータバイトはベロシティを表わす。ベロシティは音の強さを表わす。このメッセージは指定された音を指定された強さで発音することを意味する。普通の強さは64で表わされる。ベロシティが0のノートオンはノートオフの代わりに使われる。

ノートナンバーは1～127で表わす。C4と呼ばれる中央のドは60となる。ノートナンバー1は#C－1であり，127はG9である。

(3)ポリフォニックキープレッシャ

0xAX　　2

これは特定のキーを発音した後でその強さを変えるものである。最初のデータバイトはノートナンバーを指定し，2番目のデータバイトはプレッシャを表わす。

(4)コントロールチェンジ

0xBX　　2

最初のデータバイトは制御番号を表わし2番目のデータバイトは制御値を表わす。

制御番号が122から127までは決められた内容がある。

制御番号が122の時は制御値が0ならLocal Control offであり，制御値が127ならLocal Control Onである。Local control offとはシンセサイザ等において，キーボードと音源との結合を切るものである。

制御番号が123の時はAll Notes Offであり，全ての音を停止する。制御値は0とする。

制御番号124から127まではOmni modeと

poly/mono モードの指定に使われる。これ以外の制御番号についてはさらにいくつかのデータバイトを送れるものもある。ピアノのソフトペダルやソステヌート，ダンパー等のペダルや，ポルタメント，メインボリューム等の制御がこれらに含まれる。

(5)プログラムチェンジ

0xCX　1

ここでいうプログラムとはシンセサイザの音色に対応するものである。データバイトはそのプログラムを指定する。

(6)チャンネルプレッシャ

0xDX　1

これは特定の音ではなく，そのチャンネル全体のプレッシャを変えるものである。

(7)ピッチベンダチェンジ

0xEX　2

これは楽器全体の音を上下に修正するものである。2つのデータバイトでその修正量を指定する。0x2000 のときは修正量 0 となる。

3．インターフェースボード

MIDI インターフェースボードをパソコンに接続することにより，MIDI 規格に従う任意の楽器を制御することができる。しかしながら，その制御をすべてパソコンで行なうことはかなり重大な負荷となる。

中でも特に問題になるのがタイミングの制御である。MIDI 信号はどの高さの音をどの強さで出すかを指定することができる。しかしそのタイミングはデータの内容ではなく，その信号が出されるそのタイミングに依存しているのである。このタイミングは少なくともミリセック単位で制御する必要がある。しかしながら市販のパソコン例えば PC 9801シリーズでは秒単位の時間しか知ることができない。

そのため実際には，タイミングの制御等を代行してくれるインターフェースボードを使用することになる。ここではローランド社の MPU-PC98[13] というボードを使うことにする（以下 MPU と呼ぶ)。

3．1 CPU との接続

MPU は PC98シリーズの拡張スロットに挿入する。プログラムからは以下の3つのバイト単位のポートを介してアクセスすることができる。

ステータスポート
データポート
コマンドポート

通常はデータバポートは I/O 空間の 0xE0D0 番地に設定され，ステータスポートとコマンドポートは同じ番地の 0xE0D2 番地に設定される。したがってこれらにアクセスするには C 言語ならば inp 関数や outp 関数を使う。

ステータスポートはデータやコマンドを呼んだり書いたりできるかどうかを示すフラグを保持しているポートで，これは読み取られるだけである。

特にステータスバイトのビット 7 をデータセットレディ DSR といい，これが 0 のときにはデータポートを読み取ることができる。

同様にステータスバイトのビット 6 をデータレシーブレディ DRR といい，これが 0 のときはデータポートにデータを書くことができる。

以上を C の関数としてまとめると以下のようになる。

データを読む関数

```
int getdata ( )
{ int i ;
  while (inp (STATPORT) & 0X80 !=0)
                                  kbhit ( ) ;
  i=inp (DATAPORT) &0xff ;
  return (i)
}
```

データを書く関数

```
void sda (ic)
```

```
int ic ;
{ char ch ;
  ch＝ic ;
  while (inp (STATPORT) & 0x40 !＝0)
                                    kbit ( ) ;
  outp (DATAPORT, ch) ;
}
```

これらを使って，コマンドを MPU に送る関数を書くことができる。コマンドとは，これを MPU に送るとその処理が終了したときに何らかの応答データが返って来るものである。応答データが返されるとき，割り込みをかけるようであるが，これは必ずしも使う必要はない。ここでは割り込みを禁止する関数 _disable と，元に戻す関数 _enable を使う。以下にコマンドを MPU に送って ACK (0xfe) が返るのを待つ関数を示す。

```
void sendcom (ic)
int ic;
{ char ch ;   ch＝ic ;
  printf (“Sendcom %2x”, ch) ;
  while ( (inp (STATPORT) & 0x40) !＝0)
                                    kbhit ( ) ;
  _disable ( ) ;
  outp (COMDPORT, ch);
  ch＝getdata ( ) ;
  if (ch＝＝(char) ACK)
        printf (“Ack Return¥n”) ;
  else printf (“NON Ack Return %x ¥n”, ch) ;
  _enable ( ) ;
}
```

3．2　MPU の機能

MPU ではトラックという概念を用いているが，これは MIDI 規格とは関係のないものである。これは例えばテープレコーダのトラックに似た概念であり，1～8のトラック番号を用いる。最初にアクティブトラックを指定することによって演奏や記録時のトラックを選択することができる。これによってパート別の演奏や全体の演奏などを試すことができる。

パソコン（以下ホストという）と MPU の間のやり取りはバイトデータを基本として行なわれる。

MPU の主な使い方は，演奏と記録である。この他にあるトラックのデータを演奏しながら別のトラックに記録を行なうオーバーダビングも可能である。また，MPU を単なる 31.25 K ボーの UART とすることもできる。ここでは演奏と記録の方法を述べる。

3．3　演奏

MPU を用いた演奏を開始するには演奏開始コマンド 0x0A を MPU に送り，停止するには演奏停止コマンド 0x05 を送る。演奏開始コマンドを送る前に，諸条件をセットするために以下のようないくつかのコマンドを送るのが普通である。

［コマンドバイト（データバイト数）名前］の順で示す。

［0x8E (0)コンダクター off］

［0x8F (0)コンダクター on］

コンダクターは演奏中にテンポ等を変えるために用いるものであり，詳しくは後述する。

［0xEC (1)アクティブトラック on/off］

どのトラックをアクティブとするかをデータバイトで指定する。データバイトの最下位のビットから0，1，…と数えるとビット0，1，2，…はそれぞれトラック1，2，3…に対応する。

［0xB8 (0)クリア・プレイ・カウンタ］

すべてのトラックのプレイ・カウンタをクリアする。演奏開始前に必ず実行する必要がある。

以上のような設定を行なった後に演奏開始コマンドを送ると，MPU はスタート 0xFA を MIDI アウトに出し，CPU に対してトラックデータリクエスト 0xFT をホストに送る。

ここで下位4ビットTはトラック番号－1の値である。

これに対してホストは直ちに，対応するトラックの演奏データを送る必要がある。この演奏データは最初にタイミングバイトが存在し，次に一つのMIDIメッセージが続く。

MPUはそのタイミングバイトの値をそのトラックのプレイ・カウンタにセットし，そのMIDIメッセージをもそのトラックに対応するバッファに記録する。MPUは内部クロックでカウンタをデクリメントし，値が0になるとバッファに記録したMIDIメッセージをMIDIケーブルに送り出す。

このようにして複数のトラックのデータを並行して演奏することができる。

演奏上同時に鳴る音については，実際には一つずつノートオン情報をMPUに送るが，2番目以降のタイミングバイトを0にするので実質的には同時に発音するのである。

また，240クロック以上の待ち時間をとりたいときはトラックデータリクエストに対して1バイトのMPUメッセージであるタイミングオーバーフロー0xF8を送る。するとMPUはこのトラックについて240クロック数えてから次のデータリクエストを出す。ホスト側はこの方法で任意の長さの音符や休符を実現できる。但し電源投入時は4分音符一つの長さは24クロックとなる。

以上のようにして演奏を行ない。あるトラックの演奏データが使い終わったらMPUからの次のトラックデータリクエストに対してホストは一つのタイミングバイトの後に続けてデータエンド0xFCを送る。そのトラックはoffとなり，以後そのトラックデータ・リクエストは出されない。

MPUはすべてのトラックがoffになると，オールエンド0xFCをホストに送る。これに対してホストはストッププレイング0x05をMPUに送って演奏を終了する。MPUはストップ0xFCをMIDIアウトに出す。

3．4　コンダクターとテンポ

前にも触れたように，曲を演奏している間に，適当なタイミングでテンポを変える等の操作をしたい。これを実現するためにコンダクターの機能が使われる。

コンダクターの機能をオンにして演奏を開始すると，MPUはトラックの一つと同様にコンダクタデータリクエストをホストに送る。ホストはこれに対して，タイミングバイトのあとにMPUコマンドをデータとして送らなければならない。

またテンポに関するコマンドとして次のようなものがある。

[0xE0 (1)セット・テンポ]

データバイトでテンポを設定する。

[0xE1 (1)レラティブ・テンポ]

データバイトで，現在のテンポとの相対的なテンポを以下のように設定する。

データバイト	0	0x20	0x40	0x80	0xC0	0 xFF
比率	0	0.5	1	2	3	4

[0xE2 (1)グラデュエーション]

データバイトで，現在のテンポから相対的なテンポに移る速さを決める。0x01が遅く0xFFが速い。0x00は瞬時に変化する。

[0xB1 (1)リセット・レラティブ・テンポ]

相対テンポを1に戻す。

3．5　レコード

演奏とは逆に，MIDI楽器の演奏をしている間にそのデータを記録（レコード）することができる。

そのためには0x22スタート・レコーディングをホストからMPUに送るが，やはりその前にいくつか設定のためのコマンドを送る必要がある。

[0x86 (0)ベンダー off]

[0x87 (0)ベンダー on]

これらによってピッチホイールとコンティニュアス・コントロールメッセージをホストに送るかどうかを決める。

[0xEE(1)
セットチャンネルアクセスセプタンス　1—8]
[0xEF(1)
セットチャンネルアクセスセプタンス　9—16]
それぞれデータバイトのビットに対応するチャンネルのボイスメッセージをホストに送るかどうかを指定する。
[0x85(1)メトロノーム on（4拍子)]
[0x83(0)メトロノーム on（3拍子)]
[0x84(0)メトロノーム off]
メトロノームは記録のときに拍子を表わす音を出すものである。
[0x88(0) MIDI スルー off]
[0x89(0) MIDI スルー on]
例えばピアノを弾いてその演奏データを記録するとき，スルー on になっていると，ピアノから出たボイスメッセージは MPU を経由して再びピアノに入り，同じ音を再びトリガーすることになる。これを防ぐためにはスルー off とする必要がある。
以上のような設定のあとに，スタート・レコーディング 0x22 をホストは MPU に送る。MPU はスタート 0xFA を MIDI アウトに出し，レコードカウンタをリセットしてインクリメントを始める。
MIDI インにメッセージが入ると MPU はタイミングバイトを付けてホストに送る。タイミングが240以上の時は MPU はタイミングオーバーフローを表わす MPU メッセージ 0xF8 をホストに送る。
レコードを終了するには，ホストから MPU にストップ・レコーディング 0x11 を送る。これに対して MPU はストップ 0xFC を MIDI アウトに出し，タイミングバイトとデータエンドマーク 0xFC をホストに送る。

4．　音楽記述言語

音楽のデータを記述するにはいろいろなレベルが考えられる。最も低いレベルでは，演奏される音楽の一つ一つの音の周波数や音色，強さの記述が考えられる。従来最もよく使われてきたのが楽譜のレベルであるが，これより高位のレベルとしては，このような楽譜を作り出すプログラムや自然言語で書いた構想のようなものが考えられる。
ここでは具体的な曲の構造を分析したり，またそれを演奏に結び付けるために便利と考えられる楽譜のレベルでの記述を考える。

4．1　言語の設計目標

音楽データをコンピュータに入力する方法は，おおよそ以下の3つにわけられる。
(1)数値による入力
(2)テキストベースの言語
(3)グラフィックインターフェース
(4)演奏データのリアルタイムレコード
(5)楽譜の自動認識
(1)はもっとも古くから行なわれているが問題はその能率の悪さと読みにくさである。
(4)は楽譜を見て演奏できる人間がいないと不可能となる。最近では(3)の方法が普及してきているが，グラフィックインターフェースの使い勝手の善し悪しの問題があるし，大量のデータを入力するには能率が悪い。(5)は一部実用化されているが信頼性が低い。
以上のような理由からここでは(2)の方法を取り上げる。この方法の利点は少し慣れるだけで比較的容易に入力でき，ある程度解読することができることである。
テキストベースの言語は基本的に直線的なつながりを持つ宿命がある。これに対して楽譜の場合は複数の旋律の流れを並行に記述するために2次元的な広がりを持つものである。従ってこの並行性をテキストベースの言語で如何にして表現するかが問題となる。他にも

いくつか問題点があるので，以下にここで設計する音楽記述言語の設計目標を列挙する。

(1)ある程度音楽の知識のあるものにとって自然な記述であること。

(2)複数の直列な流れが並行したり，直列な流れの中に並列な流れが混入する等の関係が容易にかつ理解しやすく記述できること。

(3)基本的には表現法を除いた，骨格となるデータが入力できるようにするが，演奏上の詳細などを記述できる可能性を持たせる。

以上のような設計目標に基づき，ここでは以下のような言語を設計した。

4．2　音符の表現

最初に音の高さの表現を考える。これについては少し音楽をかじったことのある人間にはA，B，C，D，E，F，Gという表現が自然なものである。つまりハ長調で言うドがCであり，レ，ミ，ファ，ソ，ラ，シがそれぞれD，E，F，G，A，Bに相当するのである。

但しこれでは7音階しか表現できない。さらにオクターブの区別をするためにこれに数字を付け加える。これについてはMIDI規格での習慣があるのでそれに従う。つまりト音記号の第1下線のドをC4とする。このオクターブ上のドをC5とする。この間のレからシまではそれぞれD4，E4，F4，G4，B4とするのである（図4．1）。

派生音については次のようにする。シャープは'＋'を，フラットは'－'を，ナチュラルは'n'を前につけて表わす。

次に長さの表現を考える。最も基本的な長さを4分音符とする。これにスラッシュ／をつけることで半分の長さとする。従って16分音符は二つスラッシュがつく。逆にアステリスク＊をつけることで2倍の長さを表わす。

また付点はピリオド．で表わす。一つで1.5倍の長さを表わし二つで1.75倍の長さを表わす。

休符は4分休符をPで表わし，／，＊，．が使えるものとする。

4．3　直列と並列の表現

音符を小括弧（　）の中にコンマで区切って並べることで直列に演奏することを表わす。また大括弧｛　｝の中に音符をコンマで区切って並べることで並列に演奏することを表わす。

さらにこれらの記述が互いにネスト化することができる。

4．4　例

ここで，いくつかの例によって音楽記述言語の書き方を示す。

最初に図4．2の楽譜の記述を示す。これは単旋律の例である。

```
(G4/,E4/,E4/,P/,F4/,D4/,D4/,P/)
```

では次に二つの旋律を含む例を示そう。バッハの2声のインベンション1番の最初の2小節を示す（図4．3）。但し装飾音は省略してある。

```
{ [右手]
  (P//,C4//,D4//,E4//,
                F4//,D4//,E4//,C4//,
      G4/,C5/,B4/,C5/,
   D5//,G4//,A4//,B4//,
                C5//,A4//,B4//,G4//,
      D5/,G5/,F5/,G5/
    ),
  [左手]
  (P＊,P//,C3//,D3//,E3//,
                F3//,D3//,E3//,C3//,
   G3/,G2/,P,
   P//,G3//,A3//,B3//,
            C4//,A3//,B3//,G3//
  )
}
```

以上において［　］の中は注釈で何を書いても構わない。このように右手パートと左手パートを並行して記述することができる。ま

図４．１

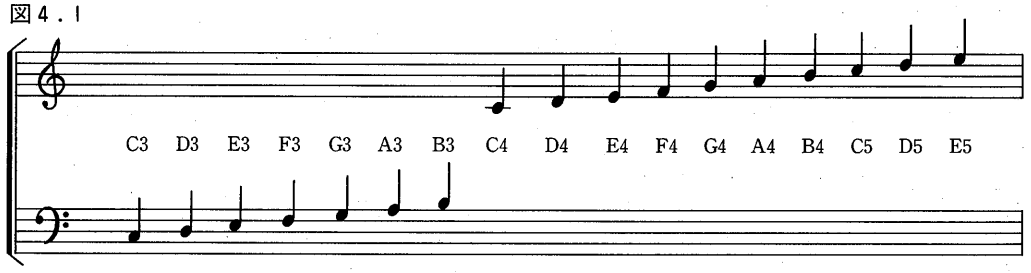


図４．２

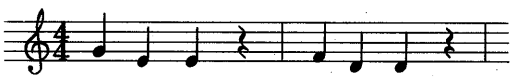


図４．３

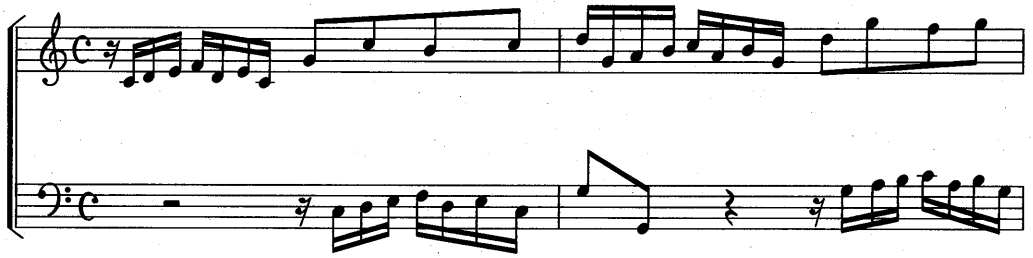


図４．４

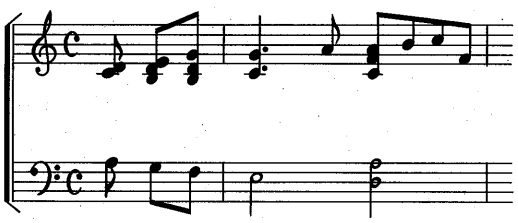
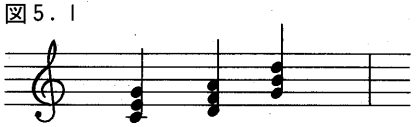


図５．１



た，ここでは二つの小説を一つの直列につなげて書き，それを並列に記述したが，これは別の書き方も可能である。例えば1小節づつ並列にまとめてこれを直列に記述するのである。

```
(  {
    (P//,C4//,D4//,E4//,
                F4//,D4//,E4//,C4//,
       G4/,C5/,B4/,C5/,
    ),
    (P＊P//,C3//,D3//,E3//,
                F3//,D3//,E3//,C3//,
    )
   }
   ,
   {
    (
    D5//,G4//,A4//,B4//,
              C5//,A4//,B4//,G4//,
       D5/,G5/,F5/,G5/,
    )
    ,
    (G3/,G2/,P,
    P//,G3//,A3//,B3//,
            C4//,A3//,B3//,G3//
    ),
   }
)
```

また以上において対応する左括弧と右括弧を同じ列に合わせて書き，これに含まれる内容は少し右にずらして記述している。これは字下げ（indentation）と言ってコンピュータのプログラム言語でよく用いられる手法である。このように書かなくても処理系は正しく解釈するが，人間が理解しやすいように字下げをするのである。

また次のように声部の数が変化するような場合も記述できる（図4．4）。

```
(
  ({D4/,C4/,A3/},{E4/,D4/,B3/,
    G3/},{G4/,D4/,B3/,F3/}
   ),
  { ({G4.,C4.},A4/,
   {A4/,F4/,C4/},B4/,C5/,F4/
    ),
    (E3＊,{A3＊,D3＊},
    )
    }
)
```

4．5　その他の記述

現在ここまで示したように，いわば音楽の骨格の部分を記述できる範囲を処理系として実現した。しかしこれだけではタイやスラー等の記述ができないだけでなく，強弱やテンポの変化などを記述できないため，演奏するためのデータとしては不十分である。

また，調声の記述方法をも付け加えるべきである。これは絶対音で記述するので，なくても記述はできることになる。しかし入力を容易とするためにはあった方がよい。ここではこれらの考えられる記述法について述べておく。

調声はフラットまたはシャープの数を示す。フラットならば負の数でシャープなら正の数でその個数を表わす。

例えばフラットが3個あれば

scale＝－3

のように表わす。

3連符や5連符に付いては直列記述の最初に”：3”や”：5”などのように指示することでその直列記述の中の音符の長さをそれぞれ2/3や2/5とする。例えば次のようにする。

(：3,C5/, D5/,E5/)

もっと一般的な記述を考えると，4分音符の長さはデフォールトでは120であるが，これを一時的に変更する必要が生じるかも知れない。そのために次のようにして指定できるようにする。

quarter＝125

次に，小節間で同じ音が続く場合がある。この場合，一つの直列なつながりとして記述すればよいのであるが，他の事情（理解しやすさなど）からそうしたくないときがある。このような場合，前後の音を止めずに続ける記述法が必要となる。例えばＣ４の音が次の小節のＣ４とつながる場合，これを次のように記述することが考えられる。

((Ｐ＊,Ｐ,{Ｃ４＝,Ｅ４,Ｇ４}),
({＝Ｃ４,Ｆ４,Ａ４},Ｐ,Ｐ＊)
)

音の強さについては，絶対的な強さを指定する場合と，クレッシェンドやデクレッシェンドのように強さが変化する場合がある。また一時的に強い音を出して元の強さに戻る場合等もある。

また，全体の強さを表現する場合と特定の旋律だけの強さを指定する場合とがある。その区別は｛　｝と（　）の包含関係を利用して記述することが考えられる。つまりある括弧の中に入って，強さの指定がなければ，その外部の指定をそのまま使い，もし指定があれば内部での指定を優先する。

具体的に全体の強さを指定するには例えば

velocity＝70

というように０～127の数で指定する。

特定の音の強さを指定するには音符データの前にコロンで区切ってベロシティを書く

120：C3/

クレッシェンドやデクレッシェンドについては４分音符の長さについてどれだけベロシティを変えるかを指定する。例えば４分音符当たり１づつベロシティを増やすのは

cresc＝１

逆に４分音符当たり２づつ減らすのは

cresc＝－２

と書く。

したがってベロシティの変化を停止するには　　cresc＝０

と書けばよい。

これについても括弧毎に異なる指定ができるものとする。

次にテンポの指定であるが，これはパート毎に異なる指定をする訳には行かない。つまりどこに書いてあっても全体に関わる指定と見なす必要がある。これらは全てMPU-PC98の機能を直接利用するものとする。

まず定常的なテンポの指定については

tempo＝100

のように０～255の数で指定する。

さらにアクセルやリタルダンドのために以下の機能を使う。

relative＝０

のように，０,0.5,１,２,３,４のどれかを指定する。これは現在のテンポに対する相対テンポを指定するものである。

gradient＝５

これは現在のテンポから相対テンポに移るときの速さを０～255の数で指定する。

０の時は瞬時に変化するが，それ以外は数値が小さいほどゆっくり変化し，大きいほど早く変化する。

少なくともピアノについては完全な演奏ができるようにしたい。そのため，サステインペダルのon/offを次のように記述する。

s_on　　s_off

最後に次の問題を考える。実際の演奏では，楽譜に音符が書いてあっても，その長さの間ずっと音を出している訳ではない。次の音に移る前にわずかだが音の無い時間がある。これが短ければレガートに聞こえ，長ければマルカートに聞こえる。この音の無い時間をここではギャップと呼ぶことにする。これを表わすために

gap＝12

などと書くことにする。

ただしスタッカートなどのように実際の音が音符より極端に短い場合は音符と休符に分けてコーディングするものとする。

5．オブジェクト・コードと演奏プログラム

5．1　オブジェクト・コード

前章までで提示した音楽記述言語で書かれた音楽を処理して，どのような目的コードを出力するべきかを考える。

ここではこのような音楽記述言語の記述能力を確認するために，実際に演奏できるコードを出力することを考える。このように実際に演奏できることは，実際にコーディングする過程において，デバッグのためにも重要である。つまりコーディングしてはこれをコンパイルし，実際に演奏を行なうことによって正しくコード化できたことを確認することが可能となる。

オブジェクトコードは二つのファイルに出力する。一つは基本的な演奏データであり，もう一つはコンダクターに与えるデータである。そしてソースコードのファイル名は８文字以内の英数字（xxxx とする）に拡張子“sco”をつけたものとする。それをコンパイルして出力するデータは xxxx.int というファイルに出力し，
コンダクタへのデータは xxxx.con というファイルに出力する。

説明の都合上，両方とも十進数を空白または改行で区切ったテキストデータとする（バイナリーファイルにすることは容易である）。各数は０～255の１バイトで表現できる値とする。

基本データのファイルは，タイミングバイトの後に続くコマンドバイトとそれに応じた数のデータバイトからなる。

最も頻繁に使われるのがノート on であり，この後にノートナンバーを表わすバイトと，ベロシティを表わすバイトが続く。音を消すのはベロシティを０にする。

これ以外には，ペダル等のコントロールチェンジ情報がある。これらはタイミングバイトの後に３つのバイトが続く。

タイミングが240以上になるときにはタイミングオーバーフロー 0xF8 つまり248が単独のバイトで現れる。

コンダクターへのデータは基本データとは切り離して考える必要がある。なぜなら，実際の演奏プログラムにおいて，基本データとは別のカウンタによって制御されるからである。タイミングオーバーフローの扱いは基本データと同様である。

5．2　並列と直列の表現

ここで，如何にして基本データにおいて，並列または直列な音が表現されるかを示しておく。

最初に例えば次のような直列な音の列を取り上げる。Ｅ４，Ｄ４，Ｃ４，Ｂ３ただしチャンネルは１とする。するとこれは４分音符の長さを120単位とすると，次のようなデータの列となる。最初にタイミングバイトが０でＥ４の音をノート on（十六進 0x90は十進で144）とする。ノートナンバーは64，ベロシティは60とする。

０，144，64，60，

次に４分音符の長さだけ待ってこの音を消す必要がある。

120，144，64，０，

そして直ちに次のＤ４音を出す。

０，144，62，60，

さらに４分音符の長さだけ待ってこの音を止め，次のＣ４の音を出す。

120，144，62，０，０，144，60，60

４分音符の長さだけ待ってＣ４の音を止め，Ｂ３の音を出す。

120，144，60，０，０，144，59，60

最後にＢ３の音を止めて終了する。

120，144，59，０

以上のようにもともとシーケンシャルな音の流れの表現は自明かも知れない。

では，図５．１のような和音の表現を見る。

最初にＣ４，Ｅ４，Ｇ４ の音を同時に出すために次のようなデータの系列が来る。

0，144，60，60，0，144，64，60，0，144，67，60，

このように実際にはシーケンシャルな列として表現されるが，演奏時に待ち時間０で次々に音が出されるので，人間の耳には問題なく同時に音が出されるように聞こえる。

次に４分音符の長さだけ待って以上の３つの音を消す。

120，144，60，0，0，144，64，0，0，144，67，0，

ここで最初のタイミングバイトは120であるが，次の二つは０である。

そして直ちに次の和音が出される。

0，144，62，60，0，144，65，60，0，144，69，60，

次に120だけ待ってこれらの音を消し，

120，144，62，0，0，144，65，0，0，144，69，0，

次の音を出す。

0，144，67，60，0，144，71，60，0，144，74，60，

最後にこれらの音を止める。

120，144，67，0，0，144，71，0，0，144，74，0，

以上のように，オブジェクトコードに於ける並列性の表現を人間が理解することは困難である。このことから，前章で示したような並列性の表現が有用であることが理解できる。

５．３　演奏プログラム

演奏プログラムは，以上のような二つのデータを読み込んで，演奏を行なう。基本データとコンダクターへのデータとは別の配列に読み込まれる。

最初にテンポをコンピュータのコンソールから読み取り，そのテンポをセットする。

そうして必要な初期操作を行なって演奏を開始する。そのあとはMPUからのトラックデータリクエストに応えて，基本データをMPUに送る。このときコマンドの種類に応じて送るバイト数を決める必要がある。

また，トラックデータリクエストに混在してMPUから出されるコンダクタ・データ・リクエスト0xF9に応えて次のコンダクタ・データを送ってやる。

従って演奏プログラムは，MPUからのデータがトラック・データ・リクエスト（0xF0～0xF7）かコンダクタ・データ・リクエスト（0xF9）かによって異なる処理をする。

６．データ構造と処理

以上で述べたような形式のオブジェクトを生成するために，コンパイラ内部でどのようなデータ構造をどのように使用するか検討する。

６．１　データ構造

オブジェクトに出力するデータの基本単位は，必ずタイミングバイトを伴っている。従ってこのタイミングバイトを伴うデータの集まりを一つのまとまり（チャンク）として扱う必要がある。そしてデータ構造ではこのタイミングデータは必ずしもバイトである必要はない。つまり内部では非常に長いタイミングを表現してよいのである。そしてオブジェクトとして出力するときにバイトとし，バイトで表現できないときにはタイミングオーバーフローを使用するのである。

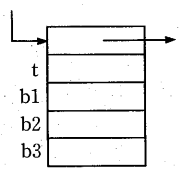
では最大どの程度の長さを表現できればよいであろうか。４分音符が120として，１小節が480となり，長い曲なら数百小節あるのですくなくとも16ビットでは不十分である。ここでは32ビットの長さをもつ，長い整数型を使うことにする。

以上のようないくつかのデータを表わすために次のようなcell型のデータを宣言して使

うことにする。

```
struct cell{
    struct cell *next;
    long int t;
    int b1, b2, b3;
};
```

これを以下のような図で表わす。



ここでは一つのメッセージがタイミングバイトの他には高々3バイトで表現できることを前提としている。

最初にポインタ型があるのは，このようなセル型のデータをポインタで線形につなぐためである。

新しくこのような記憶構造が必要になる度に malloc（ ）関数で主記憶領域を割り付けてもらう。この関数はその先頭番地を返すものである。もし何らかのエラーで割り付けができないときは NULL を返す。

6．2　音符の処理

次に，具体的な音符の処理を考える。例えば C3/. というような音符データがあれば，これは通常三つの cell で表現する。最初の cell はノート on であり，この高さの音を出し始めることを表わす。そしてそのタイミングデータは 0 である。次の cell は逆にこの音を止めるものである。そのタイミングデータは，8分音符の長さからその文脈でのギャップの長さを引いたものである。そして最後の cell はギャップの長さだけ時間を待つダミーデータである。b1が 0 のデータでダミーを表わす。

休符は一つのダミー cell で表現する。

どのような音の系列であっても全てこのような cell を線形につないだリスト構造で表現する。

6．3　直列シーケンスの処理

直列なシーケンスを表わす括弧（ ）のなかに，いくつかの要素があるとする。それは単純な音符か，またはさらに直列なシーケンスかあるいは並列な表現かも知れない。

どのような場合でもそれらの意味は全て cell をつないだ線形リストで表現されているとする。各々の要素のデータ構造が与えられているとき，この直列なシーケンスでは，前の要素を表わすリスト構造の後に次の要素を表わすリスト構造を接続してやる。こうして得られるのが，現在の直列シーケンスの意味を表わすリスト構造である。

このことによって，前のデータが演奏されてから次のデータが演奏されることが保証される。

6．4　並列シーケンスの処理

並列な関係を表わす｛ ｝の中にいくつかの要素があって，それらの意味がそれぞれリスト構造で表現されているとする。

各々の要素は，同様に単純な音符かも知れないし，直列なシーケンスかあるいは並列なシーケンスである。

この場合，単純にリスト構造をつなぐという訳には行かない。それでは直列な演奏を意味するからである。そうではなく，一つ一つの cell のタイミングデータを見て正しい順序に併合する必要がある。

例えば二つのリスト構造 A と B があって，これらを一つのリスト構造 C に併合するものとする。A と B のそれぞれの先頭の cell のタイミングデータがそれぞれ120と90であったとする。このとき先に実行されるべきデータはタイミングデータ90のものである。そこで B の先頭の cell を B から外して C の末尾につなぐ。ここで注意しなければならないのは，このまま次の処理に移るとタイミン

グデータの処理を誤ることである。というのは，最初の状況において，Aの先頭のcellは120単位数えてから実行するべき内容を表わし，Bの先頭のcellは90単位だけ数えてから実行するべき内容を表わしていたのである。いま，Bの先頭にあったcellがCの末尾に移されたので，Aの先頭のcellの内容はさらに120－90＝30単位だけ数えてから実行するべきことになる。このことから，二つのリストを併合するアルゴリズムは以下のようになる。

併合アルゴリズム：

入力：二つの線形リストAとB

出力：一つの線形リストC

初期化：Cを空リストとする

(1) Aが空リストならば⑿へ行く。

(2) Bが空リストならば⒀へ行く。

(3) Aの先頭のcellのタイミングデータをtaに代入。

(4) Bの先頭のcellのタイミングデータをtbに代入。

(5)ta＜tbならば(6)に行き，そうでなければ(9)に行く。

(6) Aの先頭のcellをとってCの末尾につなぐ。

(7) Bの先頭のcellのタイミングデータからtaを引く。

(8)(1)に行く。

(9) Bの先頭のcellをとってCの末尾につなぐ。

⑽ Aの先頭のcellのタイミングデータからtbを引く。

⑾(1)に行く。

⑿ Cの末尾にBを接続して⒁に行く。

⒀ Cの末尾にAを接続して⒁に行く。

⒁終了

以上の併合アルゴリズムはあるキーでソートされた二つのデータを併合するアルゴリズムに似ている。異なるのは，そのキー値がそのまま表現されずに，前のデータとのキー値の差で表わされていることである。そのため，一方を選択すると，他方のタイミングデータを修正する必要が生じるのである。

6．5　コンダクタ・データの分離

前章で述べたように，コンパイル結果は二つのファイルに出力する。一つは基本的な演奏データであり，もう一つはコンダクタ・データである。ここまでは全てのデータを一つの線形リストにまとめてきた。このなかからテンポに関するものだけを別のデータとして出力する必要がある。そのためのアルゴリズムを以下に示す。

分離アルゴリズム：

入力：線形リストC

出力：基本データの線形リストAとコンダクタ・データの線形リストB

初期化：AとBを空リストとする。

変数t＝0とする。

(1) Cの先頭のcellが基本データなら(2)に行く。そうでなければ(8)に行く。

(2) Cの先頭のcellをとって，Aの末尾につなぐ。その際tの値をそのcellのタイミングデータに加える。

(3) t＝0とする。

(4) Cが空なら⒁に行く。

(5) Cの先頭のcellが基本データなら(6)に行き，コンダクタ・データなら(8)に行く。

(6) Cの先頭のcellをとってAの末尾につなぎ，そのタイミングデータをtに加える。

(7)(4)に行く。

(8) Cの先頭のcellをとって，Bの末尾につなぐ。その際tの値をそのcellのタイミングデータに加える。

(9) t＝0とする。

⑽ Cが空なら⒁に行く。

⑾ Cの先頭のcellがコンダクタ・データなら⑿にゆき，基本データなら(2)に行く。

⑿ Cの先頭のcellをとってBの末尾につなぎ，そのタイミングデータをtに加える。

⒀⑽に行く。

⑭終了

変数tは例えばコンダクタデータから基本データに種類が切り替わったときにコンダクタデータであった間の経過時間を積算するものである。これによって基本データだけを分離したときに正しいタイミングデータを計算できる。

6．6　オブジェクトの出力

以上のような方法で形成されたリストから，基本データとコンダクタデータをそれぞれファイルに出力する。そのためにはタイミングデータが240を超える時にはタイミングオーバーフローの処理を行ない，コマンドの内容によって出力するバイト数を制御する必要がある。さらに，休符などのときにダミーデータとb1が0のcellを使っているのでその処理も必要となる。

以下に基本データの出力手順を示す。

基本データの出力手順
入力：基本データのリスト　A
　出力ファイル名の文字列型変数　kihon
出力：基本データのファイル
初期化　t＝0とする
(1)出力ファイルをオープンする。
(2)Aが空なら⑫に行く。
(3)変数tにAの先頭のcellのタイミングデータを加える。
(4)もしb1＝0なら先頭のcellを取り除き，(2)に行く。
(5)もしt≧240なら(6)に行き，そうでなければ(7)に行く。
(6)t＝t－240を行ない，タイミングオーバーフローを表わすデータ248（つまり0xF8）をファイルに書いて(5)に行く。
(7)tの値をファイルに書く。
(8)b1の値をファイルに書く。
(9)b2の値をファイルに書く。
⑩b3の値をファイルに書く。
⑪t＝0として(2)に行く。
⑫ファイルをクローズする。
⑬終了する。

ここでは具体的なステータスとしてノート・onとコントローチェンジしか扱っていない。これらは全て3バイトからなるがこれら以外のステータスを処理するにはその長さに応じた処理が必要となる。

7．字句解析

次章においては，トップダウンの決定的な構文解析を行なう。そのために単なる文字の列から記号つまりシンボルを取り出す機能が必要となる。このような作業を字句解析といい，一つの関数getsym（ ）でこれを行なう。

7．1　字句解析と意味処理

関数getsymは，次の一つの記号を読みとって対応する番号を関数値として返すことが基本的な機能である。それと同時にある程度の意味処理を行なうことが必要となる。

どの程度の意味処理を行なうかはいろいろな選択肢があるが，ここでは例えば一つの音符に対応するリスト構造を構成してその先頭へのポインタを返すことにする。

音符以外の調声の指定やテンポ，強弱の指定などについては，キーワードまたはそれに'＝'をつけたもの，整数などがそれぞれ一つのシンボルとする。

以下に記号とそれを表わす番号の対応表を示す。

表7．1　記号と番号の対応表

番号	記号
1	音符
2	休符
3	(
4	)
5	{
6	}
7	,
8	:
9	整数
10	s_on
11	s_off
12	tempo＝
13	rel＝
14	grad＝
15	scale＝
16	v＝
17	cresc＝
18	gap＝
19	quarter＝
0	記号でないもの

以上の scale＝などにおいて'＝'も記号の一部としている。独立した'＝'は記号としては存在しない。

ここで音符を表わす文字列を構文図で表わす(図７．１)。

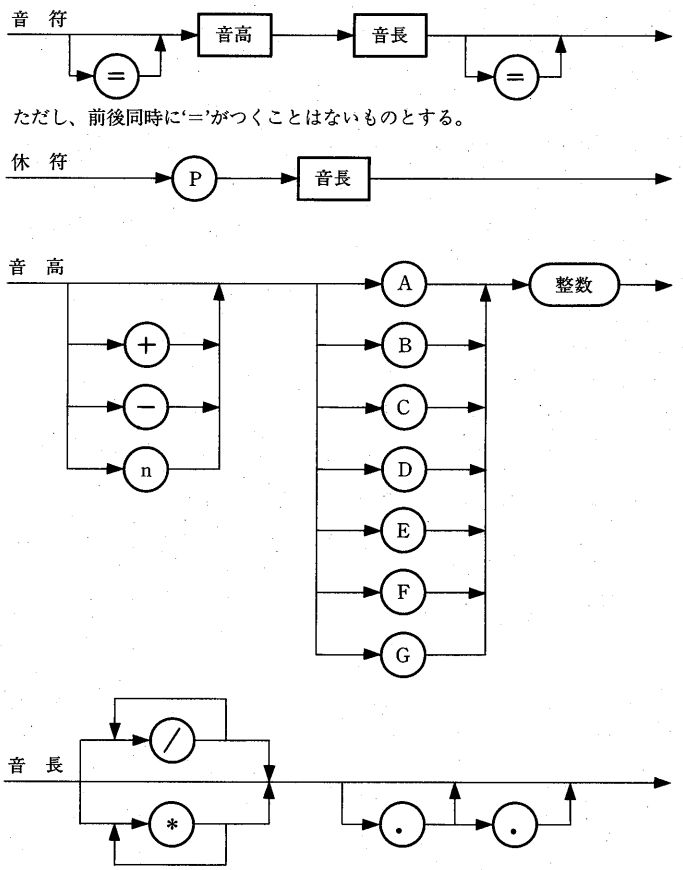
音高は次のように計算する。最初に変数 modi の値を，'＋'，'－'，'n'がある場合にはそれぞれ１，２，３，にセットする。どれもない場合には modi ＝０とする。そして次の文字が入った変数 ch と，引き数で渡された scale の値によって次のように場合分けして h の値を与える。最初に ch の値によって次のように基本的な h の値を与える。

```
if(ch=='A')h=21;
else  if(ch=='B')h=23;
else  if(ch=='C')h=12;
else  if(ch=='D')h=14;
else  if(ch=='E')h=16;
else  if(ch=='F')h=17;
else  if  h=19;
```

次に，modi==１なら h の値を１増やし，modi==２なら h の値を１減ずる。そして modi==０の時には次のようにして h の値を調整する。

```
if(ch==='A'){
        if(scale<=-3)h--;
        else  if(scale>=5)h++;
}
else  if(ch==='B'){
              if(scale<=-1)h--;
```

図７．１　音符と休符の構文図



```
else   if(scale>=7)h++;
  }
  else   if(ch=='C'){
                  if(scale<=-6)h--;
                  else   if(scale>=2)h++;
  }
  else   if(ch=='D'){
                  if(scale<=-4)h--;
                  else   if(scale>=4)h++;
  }
  else   if(ch=='E'){
                  if(scale<=-2)h--;
                  else   if(scale>=6)h++;
  }
  else   if(ch=='F'){
                  if(scale<=-7)h--;
                  else   if(scale>=1)h++;
  }
  else   if(ch=='G'){
                  if(scale<=-5)h--;
                  else   if(scale>=3)h++;
  }
```

modi＝＝3のときにはscaleによる調整を行なわない。

次にオクターブの調整を行なう。整数の値が変数nに入っているとして

```
h=h+n*12;
```

とする。

そしてこの値が1～127の範囲に入っていなければエラーとする。このときhの内容が音の高さを表わす。

音長は次のようにして変数tに求めることができる。

最初にt=120とする。

そして'*'が現れる度に

```
t=t* 2 ;
```

を行ない，'/'が現れる度に

```
t=t/ 2 ;
```

を行なう。

次に'.'が1つだけあれば　t=t*1.5; を2つあれば　t=t*1.75;　を行なう。

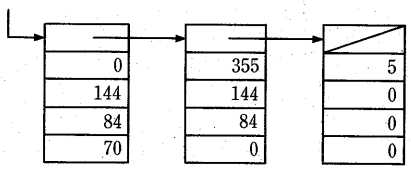
7．2　リスト構造

音符または休符の場合は，リスト構造を構成してその先頭を指すポインタを引き数として返すことにする。ここでいくつかの場合にどのようなリスト構造を返すかを見る。ただしこの文脈でgapの値は5であり，ベロシティは70であると仮定する。

最初に普通の音符

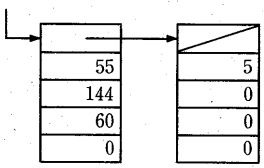
```
+B5*.
```

の場合を見る。この場合，音の高さは前節の計算を行なって　23+60+1＝84　となる。音の長さは　120*2*1.5＝360　となる。このとき次のようなリストが作られる。

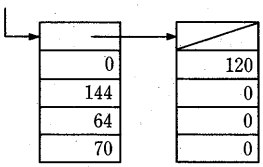


最後のcellはb1＝0としたダミーcellである。その文脈に於けるベロシティを与えるのは構文解析を行なってからとする。

＝C4/の場合は

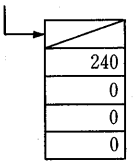


E4＝ の場合



休符についてはやはりダミーを用いる。

P*



以上のようにその文脈におけるベロシティとgapの値を与えるために，getsymはリストへのポインタを入れる変数だけでなく，これらをも引き数としてとる必要がある。

8．構文解析

本章では音楽記述言語「楽」の構文とその処理について述べる。最初に構文図を示す(図8．1)。

この構文図に従って，再帰的下降という方法で構文解析を行なうプログラム[15]を構成する。この方法が適用できるためには，次の一つの記号を見ることによって次に進むべき選択肢が決定できることが必要である。

具体的には，構文図を先頭から辿っていって分岐点に来たときに，次の記号によってどちらに分岐すればよいかが一意的に定まる。例えば音楽の構文図を辿ると直列記述か並列記述のどちらかを選ぶ分岐点に辿りつく。このとき次の記号が'('であれば直列記述を選び，'{'であれば並列記述を選ぶことができる。構文図全体がこのような性質を持っていることは容易に確認できる。

直列記述の中にさらに直列記述や並列記述が可能であり，並列記述の中にも直列記述と並列記述が可能である。したがって全体として再帰的な構造を持つ。

ここで設定項目は，これに対応する具体的なデータをオブジェクトファイルに出力しない。そのかわり他のデータに影響を与える環境変数を設定するものである。これに対し，生成項目は具体的なデータを作りだすものである。

再帰的下降法[15]では，一つの連結した構文図に対応して一つの再帰的な手続きを用意する。

そしてこの手続きは，構文のチェックをするとともに，意味上の処理をも行なう。つまり最終的にオブジェクトコードを生成するための内部的なデータ構造を作り上げて行くのである。

以下において各構文図に対応してどのような手続き（C言語においては関数）が対応するかを見て行く。最初に各構文図に対応する関数名を挙げておく：

構文図	関数
音楽	gaku
直列記述	choku
並列記述	para
生成項目	gen
設定項目	set

8．1　音楽

音楽に対応する関数gakuは，次の記号が'('であればchokuを呼び出して，'{'であればparaを呼び出す。そしてこれらから関数値として返されるリスト構造へのポインタを関数値として返す。

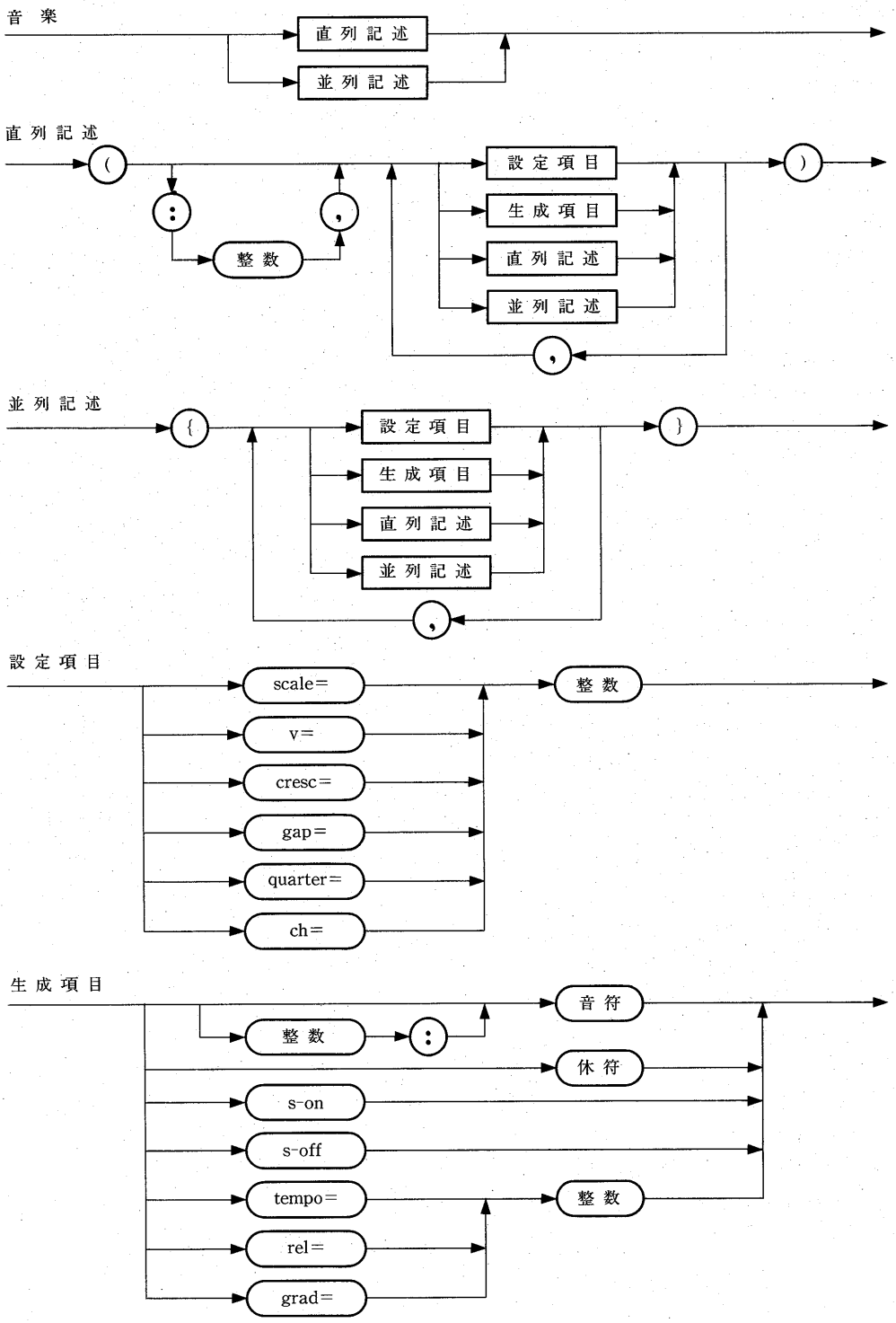
この関数はいろいろなパラメータのデフォルトの値を決める必要がある。そのようなパラメータとしてベロシティと，その増分，調声，ギャップおよび4分音符の長さがある。

ベロシティはその時の音の強さを表現するもので変数velocityで表わす。デフォールト値は64とする。

増分というのはベロシティが4分音符の長さ毎にどれくらいづつ変化するかを指定するもので，デフォールト値は0である。これが正の値のときは音が次第に強くなるクレッシェンドを意味し，マイナスのときは逆のデクレッシェンドを意味する。

調声はシャープまたはフラットが幾つついているかを表わすもので，変数scaleで表現する。デフォールト値は0とする。

図８．１　音楽記述言語の構文図



ギャップは，演奏時に音符の長さだけ全部音を出すのではなく，終わりの方は音が無くなる期間を意味する。この期間の長さを与えるのが変数 gap である。デフォールト値は2とする。

4分音符の長さは変数 quarter で表わす。デフォールトでは120とし，細かい長さの調整が必要なときにこれを変える。また3連音符や5連音符では quarter の値をそれまでの2/3または2/5の値に変える。

これら5つのパラメータはどの関数も保持しなければならない。そのため，構文要素に対応する関数を呼び出すときは必ずこれらの値を渡す必要がある。そしてその関数の内部ではこれらの値を必要に応じて変えることができる。

とくに，字句解析の関数 getsym を呼び出すときには，scale と velocity，gap，quarter の値を渡してやるものとする。これらの5つのパラメータを環境変数と呼ぶ。

8．2　直列記述

これに対応する関数 choku は，次の記号の種類によって4つの異なる処理を行なう。設定項目であれば，環境変数を設定することになる。そのため set にはこれら4つの変数の先頭番地を渡して，内部でこれらの変数の値を変えることができるようにする。

生成項目は具体的なリストを返すものである。gen または，choku，para からは一般にリストの先頭番地が返される。これは場合によっては空リストかも知れない。ここではこれらを，現れた順に直列につなぐ。そして，その長さを計算し，それによって新しい velocity の値を計算する。

最終的に'）'が来て呼び出し元に戻るときにはそのリストの先頭番地を関数値として返すのである。

8．3　並列記述

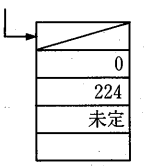
これに対応する関数 para では内部で choku と同様の場合分けが必要である。set については全く同様である。設定項目と直列記述および並列記述については，返されるリストを今までのリストと併合する点が直列記述とは異なる。このため，6．4．で述べたアルゴリズムを用いる。

'｝'が来て，呼び出し元に戻るときにはこのようにして併合したリストを返す。

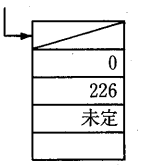
8．4　生成項目

これに対応する関数 gen では，具体的なリスト構造を返す。とはいっても getsym で基本的なリスト構造を構成しているので，実際には構文のチェックが大部分の仕事である。但し，音符の前に音の強さを表わす数値があるときには，その値をリストの velocity の欄に入れる必要がある。また，tempo＝，relative＝，gradient＝などが出てきたときには，getsym から次のような cell が返されるので，そのb2に次の整数によって値を入れる。

tempo＝の場合

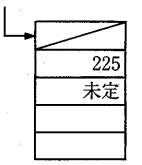


grad＝の場合



この未定の部分に次の整数を入れる。

rel＝の場合



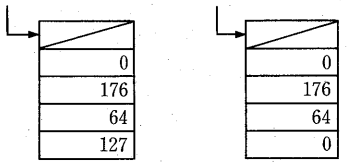
未定の部分に，次の数の値によって次のような値を入れる。

次の数	0	1	2	3	4	5
入れる数	0	64	128	192	255	32

但し，次の数が5のときは相対テンポが0.5

であることを表わす。
s_on と s_offの場合には次のようなリストが getsym からが返されるのでそのまま返す。

s_on　　s_off



8．5　設定項目

関数 set では，環境変数の値を設定する。これらは，この関数を呼び出したところでの環境変数の値を変えるものである。そのため引き数は値呼びではなく番地で渡される。

9．おわりに

本論文ではコンピュータと音楽の関わりについて概観し，MIDI インターフェースを基礎とした音楽研究用システムについて述べた。特に，コンピュータに音楽データを入力するための言語の設計とそのコンパイラの構築法に重点を置いた。

コンピュータと音楽の関わりは現代では非常に深くかつ多岐に渡ったものとなっている。従って計算機音楽に関する研究もいろいろな方向が考えられる。そのような研究に於いては，音楽データを計算機に入力する手段が必要となる。

音楽データを入力する手段にもいろいろの方法が工夫されている。最近ではグラフィック画面上でマウスを用いて楽譜を描いて行くタイプのものがあり，誰にでも容易にデータを入力することができる。しかし，この方法では表現能力や，入力速度などに問題がある。また，イメージスキャナで直接楽譜を読み取る方法では，人間には容易に解釈できるが計算機には解釈が困難な記法が沢山あり，信頼性に欠け，速度の点でも問題がある。

これに対して計算機で効率よく処理できるものとして，従来からテキストベースの音楽記述言語が種々考えられてきたが、2次元的な楽譜の持つ，直列性と並列性を同時にしかも自然な形で表現するものは見られない。

本論文で提案した言語はある程度音楽の知識のあるものには自然な形で音符を記述でき、しかも互いに入り組んでいる並列性と直列性を自然に記述できるものである。この言語は基本的にはピアノで演奏できる程度の音楽を記述するものである。MIDI の能力を十分に発揮するためにも，より以上の記述能力が要求されるが、古典的な曲の構造や演奏のデータを分析することには使用できる。今後とも使用経験を重ねることによって仕様を改良して行く予定である。

引用文献

1）川野　洋：音楽情報の理論，情報処理 Vol. 29, No. 6，特集：計算機と音楽から，pp. 539—548 (1988)
2）中村　勲：計算機の音楽音響学への応用，同上，pp. 549—556
3）田口友康：計算機による音楽演奏，同上 pp. 557—565
4）松島俊明，神前尚生：音楽表記の標準化動向，同上　pp. 566—578
5）坪井邦明：音楽分析への AI 手法の応用，　同上　pp. 579—585
6）大照　完：楽譜の入力方法とその自動認識，同上　pp. 586—592
7）高沢嘉光：計算機による採譜，　同上　pp. 593—607
8）波多野誼余夫　編：音楽と認知，認知科学選書 12，東京大学出版会 (1987)
9）杉山知之：未来の楽器ハイパーインストルメント，Computer Today 1989/11、No. 34　pp. 4—7
10）湯浅譲二：世阿彌による「九位」，同上　pp. 19—24

11) 小谷善行：言語研究からみた音楽の構造，同上，pp. 31—36
12) ローランド：MPU—401，MPU—PC 98テクニカルリファレンスマニュアル
13) MIDI 規格協議会：MIDI— 1．0 規格（日本語版）
14) D. Gries 著，牛島訳：コンパイラ作成の技法，日本コンピュータ協会（1978）
15) N. Wirth 著，片山　訳：アルゴリズム＋データ構造＝PASCAL プログラム，日本コンピュータ協会
16) 片寄晴弘：音楽情報処理，bit，vol. 23，pp. 1443—1448（1991）