◇研究報告

音声データベースをパーソナル・コンピュータで活用する試み

情報学部
前　田　英　明

1．まえがき

Texas Instrument社とMITは、DARPA(Dept. of Advanced Research Project Agency) の財政支援の下に420人が参加し、4200文を読み上げたTIMIT(音声)データベースを作成した。この時に得た情報をもともとにし、これに改良を加えたデータベースをNIST Speech Disc 1-1.1 (NIST# PB91-505065)として、一般にリリーズした。

その後、音声データベースが米国のペンシルバニア大学のLDC (linguistic Data Consortium http://www.ldc.upenn.edu) から配布されている.。わが国でも日本情報処理開発協会から研究用連続音声データベースが、また技術研究組合　新情報処理開発機構 (RWCP) からもRWCP音声対話データベースがCD-ROMで配布された。

サウンド・ボードやCD-ROMドライブが搭載されたパーソナル・コンピュータが普及した今日では、インターネットでラジオ放送が行われている。また、音声を使った語学の教材が多数出現している。しかし、先の音声データベースは、パーソナル・コンピュータで使われているサウンド・データとは異なった記録形式なるので、そのままでは使用することが出来ない。

筆者らは、ペンシルバニア大学のLDCから配布されたCALL HOME,CALL FRIENDという日本語音声データベースを、またRWCPから配布された音声対話データベースをパーソナル・コンピュータで利用することを試みた。本稿では、その方法について報告することにしたい。

2．入手した音声データ・ベースのデータ記録方法について

LDCから配布された音声データベース、RWCPから配布されたデータベースはそれぞれ特徴がある。LDCからのCALL HOME日本語版 (LDC96S53) は在米の日本人が米国から日本の家族と電話で対話している状況が録音されている。同様に、CALL FRIEND日本語版 (LDC96S37) は在米の日本人の友人同士が電話で対話している様子が録音されている。いずれのデータベースとも電話を介して対話が行われているために、データは8000回／秒でサンプルされている。サンプルされたデータは米国や日本の電話で使われているmu-law方式で圧縮され、１サンプルあたり８ビットで記録されている。mu-law方式のデータはこれまでAU、SND,AIFF形式と呼ばれ、ワークステションで使われてきた。Javaプログラミングで AU形式のサウンド　データを取り扱うために、Java Media Foundationが用意された。CD-ROMにはケンブリッジ大学のTony Robinsonが考案した圧縮方式で記録されているので、このデータは添付されているShortenとい

うソフトウエアを使用して解凍する必要がある。RWCPから配布されているデータベースは、2者の対話がDAT（Digital Audio Tape）にディジタル録音されたあと、16KHzのサンプリング・レートでダウンサンプルされている。サンプルされたデータは16ビット・リニアで記録されている。すなわち、16ビットの圧縮されていないデータであり、16ビットのうち、上の8ビットが先に、下の8ビットが後にくる。

3．使用した機材

プログラムの開発には次の機材を使用した。

CPU	Pentium Pro(200MHz)
メモリー	64MB
HDD	2 GB
OS	Windows NT　サービスパック　3を使用
ソフトウエア	Visual C++ Version 5

変換したデータのチェックには上記マシンの他に、上記のシステムとネットワークで接続しているSun Microsystems社のSPARC 5、SGI社のIndy、NeXT社のNeXT Cubeなどを使用した。

4．ファイル　ヘッダーについて

サウンドファイル、イメージファイルなどのファイルでは、実際のデータの前に、このデータがどのような形式のデータであるかを示すヘッダーが入っている。

このプロジェクトの対象となるファイルは次の2つのファイルである。

1）AU形式　　ワークステション
2）WAVE形式 Windows マシン

これらのファイル形式のヘッダー情報について説明しておこう。

4．1 AU形式の場合

長さ	バイト位置	内容
4	0から3まで	マジック・ナンバー　.sndの4文字が入っている。
4	4から7まで	ヘッダーの長さ　通常は28　この値は注1の値の長さでかわる
4	8から11まで	サウンド・データの長さ（バイト数）
4	12から15まで	サウンドの形式
		値が1のとき　8ビットのmu-lawデータ
		値が2のとき　8ビットのリニア・データ
		値が3のとき　16ビットのリニア・データ
		値が4のとき　24ビットのリニア・データ
		値が5のとき　32ビットのリニア・データ
		値が27のとき　8ビットのA-lawデータ（欧州の圧縮方式）
4	16から19まで	サンプリング・レート

4	20から23まで	チャンネル数
4	24から27まで	文字による説明（最低4文字）注１の値

このあとにサウンド・データが入る

4．2 WAVE形式の場合

長さ	バイト位置	内容
4	０から３	チャンク・タイプを文字で示す　RIFFの４文字が入る
4	４から７	ファイルの長さ-4（ここからデータの最後までのビット数）
4	８から11	フォームの名前　WAVEの4文字
4	12から15	チャンクのタイプ　fmtという文字と空白文字の4文字
4	16から19	チャンク・データの長さ　通常は16
16	20から35	チャンク・データ
内訳		
2	20,21	値が１のとき　　PCMデータ（リニア・データ）
	値が257のとき	IBM　mu-law
	値が258のとき	IBM　A-law
	値が259のとき	IBM　AVC　ADPCM
2	22, 23	チャネル数
4	２から27	サンプリング・レート　回／秒
4	28から31	１秒当たりのバイト数（プログラムでバッファーの長さを決めるときの参考として）
2	32, 33	ブロック・サイズ １サンプルしたときに、両チャネルのデータを記憶するために必要なバイト数（通常は2＊2バイト）
2	34, 35	サンプル当たりの有効ビット数（通常は16ビット）
チャンク終了		
4	36から39	チャンク・タイプ　dataの４文字
4	40, 41	実際のサウンド・データの長さ（バイト数）
	42から	実際のサウンド・データが入る

5．LDCより配布された音声データベースを変換する場合

CALL HOME,CALL FRIENDはいずれも圧縮されているので、付属しているshortenを使って解凍しなければならない。
データベースにある各ファイルは先頭1024バイトに「このファイルの記録に関する情報」が入っている。
しかし、この情報は変換には必要ないので、解凍時に取り除く。

解凍に当たっては次のコマンドを使用した。

shorten －x －d 1024 解凍するファイルの名前.sph　出力するファイルの名前

註：shortenで圧縮したファイルにはsphという拡張子が付いている。
この作業によって、ヘッダーがないmu-law形式の８ビットデータのファイルが作られる。

このファイルをAU形式のファイルに変換するためには、ファイルの先頭にヘッダーを付けなければならない。
ヘッダーを作成するために付１に示すmkhdr.cプログラムを作成した。

次にWAVE形式のファイルに変換する方法について述べる。
Javaで音声データを取り扱う方法を調べている過程で、Javaに関するFAQを集めたThe Java FAQが出版された。この本のQ11.9で音声データの変換を行うプログラムSOX(http://www.spies.com/sox)の存在を知った。
このサイトからWindows用のSOXを入手し、これを活用した。
1998年の夏には、SOXを入手するのが大変であったが、今日ではLinuxシステムにSOXが標準ソフトウエアとして含まれている。

AU形式のファイルが出来ている場合には、次のコマンドでWAVE形式のファイルを作成することができる。

sox －t ul －r 8012 file-name.au －s －w filename.wav

註：ulはmu-lawの略である。

６．RWCPより入手した音声データベースを変換する場合

RWCPより入手した音声データベースは16ビットのリニア形式で記録されている。データベースに入っているそれぞれのファイルは次の方法で変換する。

６.１ AU形式のファイルに変換する場合

この場合には、先のSOXを使用して、WAVEからAUファイルに変換する方法もある。ファイルヘッダーの項の説明からわかるように、サウンド形式の値を３にすることで、AUファイルでも16ビットのリニア・データを取り扱うことができる。
前述の方法で、AUヘッダーを作成し、これをオリジナル　データの前に付ければ良い。

６.２ WAVE形式のファイルに変換する場合

AU形式のファイルがすでに作られている場合には、mu-lawのファイルを変換したときと同様にSOXを使って、WAV形式のファイルに変換することができる。
しかし、ここでは直接変換する方法を説明する。
DATで作成した16ビットデータの並び方はビッグ・エンディアン、すなわち、16ビットの中にある上の８ビットが先にくる。

図1　Sound Forge XP4.5の表示

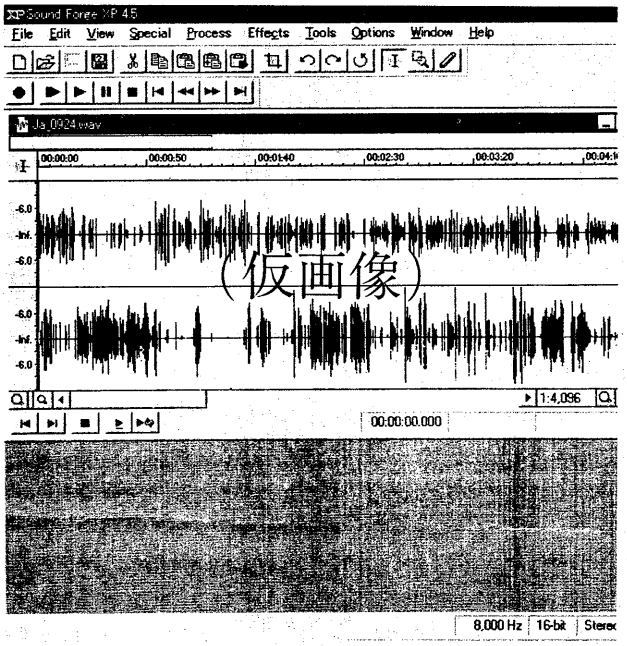
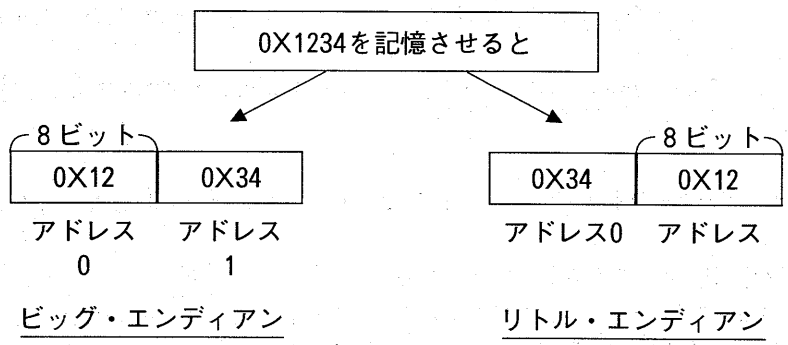


図2　ビッグ・エンディアンとリトル・エンディアンの違い



これに対して、Windows マシンで使われているPentium CPUはリットル・エンデイアン方式、つまり下の８ビットが先にくる。　両者の間には、同じ16ビットデータであっても、並び方に違いがある。このために、データをそのまま使用することが出来ない。
この変換をおこなうために、付２に示すreverse.cというプログラムを作成した。
これにあわせて、付３に示すWAVE形式のヘッダーを作成するwavhdr.c というプログラムも作成した。

7．あとがき

変換した音声データは次の方法で再生した。

1）Windowsのアクセサリーの一部としてあるエンターテイメントの中に、CDプレイヤーとサウンドレコーダーが入っている。
原則的にはどちらのソフトウエアでも再生することができるが、会話の音声データの場合にはデータ量が多いので、ＣＤプレイヤーでは再生が出来なかった。

2）変換に使ったコンピュータの他に、別のWindowsマシンを持っている。
このシステムは、Creative Design社のSound Blaster AWE 64 Goldというサウンドカードを実装している。
このボードに付属しているWAVE Studioというソフトウエアを使用すると、再生は勿論のこと、データの波形を見ることができる。

3）このプロジェクトで変換プログラムを作成し終わった頃、 レコーディング技術者向けの雑誌でSound Foundry社からSound Forge 4.5というソフトウエアが販売されていることを知った。このソフトウエアは音楽編集用であるが、多機能である。データの変換にしても、数十種類の変換が出来る。しかし、多機能だけあってかなりの金額がはる。
用途を音声データの編集や変換に限定すれば、機能は限定されるものの、廉価なSound Forge XP 4.5を使用することができる。
（最近のWindows系のマルチメディアを取り扱うソフトウエアの中には、Sound Forge XP 4.5を添付していることがある。）
このソフトウエアはグラフィック・インターフェースを採用しているので、大変使いやすい。サウンドデータをMP3ファイルに変更する場合には、オリジナルデータはサンプリング・レートが16KHｚ以上で記録されている必要があった。今回作成したAU形式のファイルは８KHzであったから、これを16KHzにアップ・サンプリングする際にSound Forge XPを使用した。
次の図２はSound Forge XP 4.5で２チャネルの音声データを読み込んだときの様子を示している。

サウンドデータの変換を行っている過程でいろいろなことを経験した。 たとえば

1） データの変換を間違えると、雑音になる。

2） 電話の会話を録音したデータであるから、1チャネルデータ、16KHｚのサンプリングレートのデータと誤解してデータの変換を行った。ところがこのデータは８KHｚの２チャネルで録音したデータであった。数字の上では釣り合うので、再生した音は当然のことながらモノラルであった。内容は理解できたが、音声はエコーがかかったような、リンキングがあるような、明瞭度がぐっとかける珍しい音であった。

このような原因を調べるときに先のグラフ表示は大変有効な手段であろう。

バイト単位で、データの上下を変換するプログラムの内容それ自体は単純であるが、ここに大きな落とし穴がある。

音声データの場合には１つのファイルの容量が100から200Mバイトの大きさになるから処理を１億回から２億回位繰り返して行わなければならない。プログラムの作り方によって、１回の処理の中に無駄な部分があると、これが増大され、データの変換時間に大きく影響を与える。何も考えずに作成した場合（novice program）とじっくり検討して作成した場合(expert program)とでは変換時間にどの位の差が有るだろうか。少なくとも５倍くらいの差が出てくることが確認されている。

いま、このプロジェクトに端を発して、「ソフトウエアのパフォーマンス」というテーマに取り組み始めた。

そればかりではなくPentium MMXプロセッサーからSIMD (single Instruction Multi Data)というスーパー・コンピュータ並の機能が組み込まれていることを知った。
この機能を使用するとマルチメディア関係のデータを処理する場合に、速度が飛躍的に改善される。
註：通常のコンピュータはSISD、即ち１つの命令で１つのデータしか処理をしない。
しかし、SIMD方式であると１つの命令で複数の命令を処理することができる。

.SPH 形式のファイルを .WAVE 形式のファイルに変換する手順

例 test1.sphを test1.wav に変換する。

１）SHORTEN プログラムを使用して、音声ファイルを解凍する。

```
shorten -d 1024 test1.sph  temp
```

このコマンドによって、解凍されたデータが temp ファイルに書かれる。
temp の代わりにどのような名前を使用してもよい。

２）mkauhdrを使って、.AU 形式のファイルで必要であるヘッダー情報を作成する。

```
mkauhdr
```

このプログラムでは、次の情報の入力が求められる。

ファイル名　test1 のように拡張子を除いたつずり
ファイルの長さ

これに、よって、 ファイル名.auhdrという名前のファイルが作られる。
この場合には、 test1.auhdr が作られる。

「注意」

同じファイル名で作業を繰り返して行うときには

まえもって、 .auhdr ファイル、この場合には test1.auhdr ファイルを消しておく必要がある。

最初に、

```
attrib test1.auhdr
```

を実行する。

多分、このファイルに AやRの属性が設定されている。 これを次のコマンドで解く

```
attrib －A －R test1.auhdr
```

その後次のコマンドでファイルを削除する。

```
del test1.auhdr
```

3）.AU形式のファイルを作成する。

次のコマンドを使って、先に作成したファイルを合成し、これを1つのファイルにする。

```
copy ／b test1.auhdr＋temp  test1.au
```

このファイルは、SUN,NeXTやSGI Indyでプレイバックすることができる。

SUNでプレイバックする場合には、次のコマンドを使用する。

```
audioplay test1.au
```

SGI Indyではファイルをダブル・クリックする。

4）.AUファイルを.WAVEファイルに変換する。

この場合にはSOX(Sound eXchange) を使用する。

```
sox test1.au －s －w  test1.wav
```

5）プレイバック

メディアプレーヤを使用する。

「スタート」->「アクセサリー」->「マルチメディア」->「メディアプレーヤ」を選択する。

「注意」

「サウンドレコーダ」を使うこともできるが、ファイルが大きい場合には

（1） メモリー不足 が生ずる。
（2）ファイルの読み込みに時間がかかる。

などの理由から、メディアプレーヤを使用するのが良い。

その他

audump　　このプログラムは、 .AU 形式のファイルのヘッダー情報を表示する。

wavdump　　このプログラムは、 .WAVE形式のファイルのヘッダー情報を表示する。

filedump　このプログラムは、コンマンド・ラインで指定したファイルの内容を
16進数と文字で表示する。
このプログラムを使って、ファイルの形式を知ることもできる。

付1　AUヘッダーを作成するプログラム

```
#include <stdio.h>
#include <fcntl.h>
#include <string.h>

#define SIZE 40
#define MULAW   0x01
#define MONO    0x01
#define STEREO 0x02
#define SAMPLING 8000
#define DEBUG FALSE

char buff[SIZE];
char fileName[31];
```

```
char hdrbuff[1024];
char hdrptr[20];

long atol (char*);
long getDataLength(char*,int);
void fill(long);
void getHeaderData(char *);

#if DEBUG == TRUE
void output(void);
#endif

void mkfile(char *);

void main (argc,argv)
int argc;
char *argv[];
/*
  *argv[0] = "mkhdr";
  *argv[1] = file name;
  *argv[2] = dir name;
*/

{
int i;
long leng;
long rate;
  strcpy(fileName,argv[1]);
  getHeaderData(argv[1]);

  for(i=0; i<SIZE; i++) buff[i] = 0;

  buff[ 0] = '.';
  buff[ 1] = 's';
  buff[ 2] = 'n';
  buff[ 3] = 'd';

  /* location of 1st data */
  buff[ 6] = SIZE / 256;
  buff[ 7] = SIZE % 256;
```

```
  /* data length */
  /* buff[ 8] .. buff[11] */
    leng = getDataLength("sample__count",12);
   printf("サンプル数は %ld¥n", leng);
   getchar();
   leng = leng +leng;
  fill(leng);

  /* recording mode = mu-law */
  buff[15] = MULAW;

  /* sampling rate */
  /* buff[16] .. buff[19] */
  rate = getDataLength("sample__rate",11);
  for (i= 19; i> =16; i--) {
    buff[i] = rate % 256;
    rate / = 256;
  }

  /* no. of channels = 2, stereo */
  buff[23] = getDataLength("channel__count",13) ;

  /* file name */
  for (i=0; i<16 && fileName[i]; i++)
    buff[24+i] = fileName[i];
  /*
  output();
  */

  mkfile( fileName);
}

long atol ( char *arg)
{
  char ch;
  long temp = 0;
  while ((ch=*arg++)) {
    if (ch == ',' | ' ') continue;
    temp = temp * 10 + (ch - 0x30);
```

```
  }
  return temp;
}

long getDataLength(char *pattern, int size) {
  int i;
  int value;
  // printf("Pattern %s size= %d¥n",pattern,size);
  for (i= 0; i< 1024; i++) {
    if (strncmp(&hdrbuff[i],pattern,size) == 0) {
      printf("HIT! %d¥n",i);break;
    }
  }
  value = 0;
  if (i< 1024) {
    printf("LOCATION %d¥n",i);
    for (i += size; i< 1024; i++) {
      printf("LOC = %d DATA = %c¥n",i,hdrbuff[i]);
      if (hdrbuff[i] == '¥n') break;

      if (hdrbuff[i] >= '0' && hdrbuff[i] <= '9'){
        value = value * 10 + hdrbuff[i] - '0';
      }
    }
  }
  return value;
}

void fill(long value)
{
  int i=11;
  while (value && i >= 8) {
    buff[i--] = value % 256;
    value /= 256;
  }
}


void output() {
  int i;
```

```
  for(i=0; i<SIZE; i++)
    printf("%02x ",buff[i]);
  printf("¥n");
}


void mkfile(char *name)
{
  int fd;
  char hdrName[31];
  strcpy(hdrName,name);
  strcat(hdrName,".AUHDR");

  if ((fd=open(hdrName,O_CREAT+O_WRONLY)) == -1) {
    printf("ヘッダー・ファイルを作成することができませんでした。¥n");
    exit(1);
  }
  write(fd,buff,SIZE);
  close(fd);
}

void getHeaderData(char *arg) {
char fname[51];
FILE * fd;
int bytes;
int i = 0;
int j;
int ptr = 0;

  strcpy(fname,arg);
  strcat(fname,".SPH");

  if (( fd = fopen (fname,"r")) == NULL) {
      printf("ディレクトリー・ファイル %s が存在しません¥n",fname);
      exit(3);
  }
  fscanf(fd,"%s",hdrbuff);
  bytes = strlen(hdrbuff);
  printf("%s",hdrbuff);
  hdrptr[i] = ptr;
```

```
  while (strncmp(hdrbuff,"end_head",8) != 0) {
    ptr += bytes+1;
    i ++;
    fgets(hdrbuff,180,fd);
  bytes = strlen(hdrbuff);
  printf("%s",hdrbuff);
  hdrptr[i] = ptr;
  getchar();
  }


  fclose(fd);
  getchar();
}
```

付2　上下のバイトを入れ替えるプログラム

```
#include < stdio.h >
#include < fcntl.h >
#include < stdlib.h >
#nclude < sys/timeb.h >
#include < sys/ types.h >

#define SIZE 32768

short reverse(short vala) {
  return ((unsigned) vala >> 8 ) | (vala << 8 ) &0xff;
}

char buffer[ SIZE ];
time_t startTime, endTime;
struct _timeb startt,endt;

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

  FILE *fd1, *fd2;
```

```
  int totalBytes;
  int bytes;
  int i;
  char reply;
  int temp,temp1; int cy;

  if ( argc != 3 ) {
    printf("¥n使用方法が間違っています。¥n" );
    printf( "BITREVERSE  変換元のファイル名  変換先のファイル名¥n¥n" );
    exit( 1 );
  }

  printf("確認します。¥n");
  printf("入力ファイルの名前は    %s¥n",argv[1]);
  printf("出力ファイルの名前は    %s¥n",argv[2]);
  printf("間違いがない場合は Y を、間違の場合には N を入力してください¥n");
reply = getchar();
  if (reply != 'Y' && reply != 'y') {
    printf("Yあるいはy以外の文字が入力されたので、作業を中止します。¥n");
exit ( 2 );
  }

  if (( fd1 = fopen( argv[1], "r+b" )) == NULL ) {
    printf( "入力ファイル %s はありませんでした¥n", argv[1] );
    exit ( 3 );
  }

  if (( fd2 = fopen( argv[2], "w+b" )) == NULL ) {
    printf( "出力ファイル %s  は作成できませんでした¥n", argv[2] );
    exit ( 4 );
  }

  totalBytes = 0;
  printf("1回の読み込みの大きさ %d バイト¥n",SIZE*2);
  time( &startTime); __ftime( &startt);
  while (( bytes = fread(( char * ) &buffer, 1, sizeof( buffer ),fd1 )) > 0) {
    totalBytes += bytes;
      // count = bytes >> 1;
      //printf( "読み込みバイト数  今回%dバイト 全体で%d バイト¥n", bytes, totalBy
tes );
```

```
    for ( i = 0; i < bytes; i+=2) {
      char temp;
      temp = buffer[i];
      buffer[i] = buffer[i+1];
      buffer[i+1] = temp;
    }
    fwrite( &buffer, 1, bytes, fd2);
  }
  time(&endTime); _ftime(&endt);
  printf("%ld バイト変換しました¥n",totalBytes);
  printf("Start %lu Sec %u Msec¥n",startTime, startt.millitm);
  printf("End   %lu Sec %u Msec¥n",endTime, endt.millitm);
    cy = 0;
    temp = endt.millitm - startt.millitm;
    if (temp <0) { cy = 1; temp += 1000;}
    temp1 = endTime - startTime - cy;
    printf("Elapsed time %u Sec %u Msec¥n",temp1,temp);

  fclose( fd1 );
  fclose( fd2 );
  printf("作業は終了しました¥n");
}
```

付3　WAVEヘッダーを作成するプログラム

```
#include <stdio.h>
#include <fcntl.h>
#include <string.h>

#define PCM   0x01
#define MULAW 0x101
#define ALAW  0x102
#define ADPCM 0x103

#define MONO   0x01
#define STEREO 0x02

struct {
```

```
    char id[4];
    long fileSize;
    char formName[4];
    char chunkType1[4];
    //
    long chunkLength;
    short  samplingType;
    short  numberOfChannels;

    long samplingRate;
    long numberOfBytes;

    short  blockAlign;
    short  numberOfBits;
    //
    char chunkType2[4];
    long dataSize;
    } wavHeader;

long atol(char *);
void mkhdr(void);

void main (argc,argv)
int argc;
char *argv[];
{
long para1;
   strncpy(wavHeader.id, "RIFF",4);
   strncpy(wavHeader.formName,"WAVE",4);
   strncpy(wavHeader.chunkType1,"fmt ",4);
   strncpy(wavHeader.chunkType2,"data",4);
   wavHeader.chunkLength = 16;
   wavHeader.samplingType = MULAW;
   wavHeader.numberOfChannels = MONO;
   wavHeader.blockAlign = 1;
   wavHeader.numberOfBits = 8;
   para1 = atol(argv[1]);
   wavHeader.dataSize = para1;
   wavHeader.fileSize = para1 + 36;
   wavHeader.samplingRate = 16000;
```

```
  wavHeader.numberOfBytes = wavHeader.samplingRate;
  mkhdr();
  printf("SIZE OF INT => %d¥n",sizeof(int));
  printf("SIZE OF LONG => %d¥n",sizeof(long));
  printf("%d¥n",sizeof(short));
}

long atol (char *p) {
long value = 0;
  while (*p) {
    if (*p == ',') continue;
    value = value * 10 + (*p - 0x30);
    p++;
  }
  return value;
}

void mkhdr(void)
{
  int fd;
  if ((fd = open("WAVHDR",O__CREAT+O__WRONLY)) == -1) {
    printf("Can't create¥n");
    exit (1);
  }
  write(fd,&wavHeader,sizeof(wavHeader));
  close(fd);
}
```

〈参 考 文 献〉

1） 音声データの圧縮に関して

a) Gibson, Berger, Lookabaugh, Linderbergh and Baker著
Digital Compression for Multimedia
PRINCIPLES & STANDARDS 1998年
発行所 Morgan Kaufman Publisher

2） ファイルの形式に関して

a) Tim kientzle著
INTERNET FILE FORMATS 1995年
発行所 The Coriolis Group, Inc

a) 鈴木直美著

標準ファイル　フォーマット辞典　1998年
発行所（株)インプレス
a）NeXT　Computer　Inc.,著
NeXTSTEP　GENERAL　REFERENCE　Volume 2　1992年
発行所 Addison Wesely Publishers
3）JAVAに関しては
a）Kanervacho著
The　Java　FAQ　1997年
発行所 Addison　Wesely
b）Suliva, Winzeler, Deagen and Brown著
Programming with the Java Media Framework　1998年
発行所 John Wiley
4）サウンド・データを作成する場合の参考書として次の本が参考になる。
Ron　Simpson著
Cutting　Edge　Web　Audio
発行所 Prentice Hall International
邦訳は　アクロバイト監訳
Webページ　サウンドテクニック
サウンドを使った魅力的なホームページの作り方　1998年
発行所 株式会社プレンティスホール出版
5）Linuxでサウンド・データを含めてマルチメディアを取り取り扱う場合は次の本が参考になる。
Tranter著 山形浩生訳
Linux マルチメディアガイド　1997年
発行所オライリー・ジャパン
6）C,C++プログラミング
出稿後に入手した。この本にはC,C++プログラミングだけでなく,サウンド・データを処理するために必要な情報が書かれている。
Tim Kientzle
A Programmer's GUIDE TO SOUNDS　1998年
発行所 Addison Wesley