理工学部 数物・電子情報系学科 情報工学 EP
秋学期講義「画像・音声情報処理」
担当：長尾 智晴
総合研究棟 S401 室（内線 4131）
nagao@ynu.ac.jp

WAVE（ウェイブ／ウェブ）ファイルの取り扱い

音声ファイルの中で，**拡張子 wav** で表される **WAVE（ウェイブあるいはウェブ）ファイル**がある．WAVE ファイルは Windows OS の PC 上の代表的な音声ファイルフォーマットである．ここでは，この WAVE ファイルの構造と，自作の C 言語のプログラムによる取り扱いの基礎について解説する．

1．WAVE ファイルの構造

■ WAVE ファイルのデータ構造

- WAVE ファイルは**表1**に示すように RIFF（Resource Interchange File Format）chunk（チャンク）で構成され，内部には fmt チャンク（フォーマット情報）と data チャンク（音声データ）が含まれている．

表1 WAVE ファイルのブロックによるデータ構造

長さ［byte］		内 容	意 味	チャンク
4	8	文字列“RIFF”	チャンク ID	RIFF チャンク
4		これ以降 RIFF チャンクサイズ（ファイルサイズ－8）	RIFF チャンクサイズ	
4		文字列“WAVE”	RIFF の種類	
4		文字列“fmt”＋英数空白	チャンク ID（フォーマット定義）	fmt チャンク
4		バイト数	fmt チャンクサイズ　↓16 進表記 リニア PCM なら 16 (10 00 00 00)	
2	16	フォーマットタイプ（詳細省略）	リニア PCM なら 1 (01, 00)	
2		チャンネル数	モノラル 1 (01 00), ステレオ 2 (02 00)	
4		サンプリングレート[Hz]	44.1kHz なら 44100 (44 AC 00 00)	
4		データ速度[byte/sec]	44.1kHz 16bit ステレオなら 44100×2×2＝176400 (10 B1 02 00)	
2		ブロックサイズ [byte/sample×チャンネル数]	16 bit ステレオなら 2×2＝4 (04 00)	
2		サンプルあたりのビット数 [bit/sample]	WAVE フォーマットでは 8 or 16 bit 16bit なら 16 (10 00)	
2		拡張部分のサイズ	リニア PCM の場合は存在しない	
n		拡張部分	リニア PCM の場合は存在しない	
4		文字列“data”	データチャンク（波形データ）ID	data チャンク
4		波形データのサイズ[byte]	データサイズ (=size)	
size		波形データ	8bit or 16bit PCM データが時間順に記録されている（ステレオの場合は LRLR….）． 8bit：unsigned (0～255，無音は 128) 16bit:signed (-32768～+32767，無音は 0)	

※ "fmt"，"data"以外のタグとしては，"fact"：全サンプル数，"LIST"：コメントがある．
※ 数値データは Intel バイトオーダ（最下位 byte が最初，最上位 byte が最後の逆順）

1．WAVE ファイルに対する処理のプログラミング

■ WAVE ファイル用簡易ヘッダファイル（本講義第 11 回関連ファイルのひとつ）

- WAVE ファイルを読み込む簡易なヘッダとして，次に示す wavelib.h を作成した．このヘッダをインクルードすることで，WAVE ファイルを読み込んだり，書き込んだりすることができる．

wave ファイルを取り扱うためのヘッダファイル（wavelib.h） （第 11 回関連ファイル）

```
/* wave ファイル用簡易ヘッダファイル wavelib.h */
#include<stdlib.h>
#include<string.h>

/* 型の定義 */
/* RIFF チャンク */
struct RIFF {
        char ID[5];             /* RIFF チャンク ID("RIFF")              */
        int SIZE;               /* RIFF チャンクサイズ[byte]             */
        char TYPE[5];           /* RIFF の種類("WAVE")                   */
};
/* fmt チャンク */
struct fmt {
        char ID[5];             /* fmt チャンク ID("fmt ")               */
        int SIZE;               /* fmt チャンクサイズ[byte]              */
        int TYPE;               /* フォーマットタイプ                    */
        int Channel;            /* チャンネル数                          */
        int SamplesPerSec;      /* サンプリングレート[Hz]                */
        int BytesPerSec;        /* データ速度[byte/sec]                  */
        int BlockSize;          /* ブロックサイズ                        */
                                /* [byte/sample x channel]               */
        int BitsPerSample;      /* サンプルあたりのビット数              */
                                /* [bit/sample]                          */
};
/* data チャンク */
struct data {
        char ID[5];             /* data チャンク ID("data")              */
        int size_of_sounds;     /* data チャンクサイズ[byte]             */
        unsigned char *sounds;  /* 波形データへのポインタ                */
};

/* 関数のプロトタイプ宣言 */
/* WAVE データのファイル入力 */
void load_wave_data( struct RIFF *RIFF1, struct fmt *fmt1,
                     struct data *data1, char name[] );
/* WAVE データのファイル出力 */
void save_wave_data( struct RIFF *RIFF1, struct fmt *fmt1,
                     struct data *data1, char name[] );
/* 以下は load/save_wave_data 内で利用されている関数 */
void read_char( FILE *fp, int n, char c[] );
void read_int( FILE *fp, int n, int *number );
void write_char( FILE *fp, int n, char c[] );
void write_int( FILE *fp, int n, int number );
void error_process( char message[] );
void error_process2( char c1[], char c2[] );

/* 以下は関数の本体 */
void load_wave_data( struct RIFF *RIFF1, struct fmt *fmt1,
                     struct data *data1, char name[] )
/* ファイルから WAVE データを読み込む.    */
/* name[] が "" ならファイル名を入力.  */
{
        char fname[256];    /* ファイル名用の文字配列 */
        FILE *fp;           /* ファイルポインタ */
        unsigned char buffer[256];   /* 文字列用作業変数 */
        int i;   /* 作業変数 */

        /* 入力ファイルのオープン */
        if ( name[0]=='¥0' ){
  printf("読み込む WAVE ファイルの名前 (拡張子は wav) : ");
                scanf("%s",fname);
        } else strcpy( fname, name );
        if ( (fp = fopen( fname, "rb" ))==NULL )
        error_process("ファイルがオープンできませんでした. ");
        /* データの読み込み */
        /* ========== RIFF 1 ========== */
        read_char( fp, 4, RIFF1->ID );   /* ID */
           error_process2( RIFF1->ID, "RIFF" ); /* Error Process */
        read_int( fp, 4, &RIFF1->SIZE );   /* SIZE */
        read_char( fp, 4, RIFF1->TYPE );   /* Type */
      error_process2( RIFF1->TYPE, "WAVE" );   /* Error Process */
        /* ========== fmt 1 ========== */
        read_char( fp, 4, fmt1->ID );   /* ID */
           error_process2( fmt1->ID, "fmt " );   /* Error Process */
        read_int( fp, 4, &fmt1->SIZE );   /* SIZE */
        read_int( fp, 2, &fmt1->TYPE );   /* TYPE */
        read_int( fp, 2, &fmt1->Channel );   /* Channel */
        read_int( fp, 4, &fmt1->SamplesPerSec ); /* SamplesPerSec */
        read_int( fp, 4, &fmt1->BytesPerSec );   /* BytesPerSec */
        read_int( fp, 2, &fmt1->BlockSize );   /* BlockSize */
        read_int( fp, 2, &fmt1->BitsPerSample );   /* BitsPerSample */
        /* ========== data 1 ========== */
        read_char( fp, 4, data1->ID );   /* ID */
           error_process2( data1->ID, "data" );   /* Error Process */
        read_int( fp, 4, &data1->size_of_sounds ); /* SIZE */
     data1->sounds = (unsigned char *)malloc( data1->size_of_sounds );
        if ( data1->sounds == NULL ){
printf("メモリが確保できません. プログラムを終了します. ¥n");
                exit(1);
        } else {
                for(i=0;i<data1->size_of_sounds;i++){
                        *(data1->sounds + i) = fgetc( fp );
                }
        }
        fclose(fp);
}

void save_wave_data( struct RIFF *RIFF1, struct fmt *fmt1,
                     struct data *data1, char name[] )
/* WAVE データをファイルに書き込む.       */
/* name[] が "" ならファイル名を入力. */
{
        char fname[256];   /* ファイル名用の文字配列 */
        FILE *fp;   /* ファイルポインタ */
        int i,number;   /* 作業変数 */
        unsigned char value;   /* 作業変数 */

        /* 出力ファイルのオープン */
        if ( name[0]=='¥0' ){
     printf("出力する WAVE ファイルの名前 (拡張子は wav) : ");
                scanf("%s",fname);
        } else strcpy( fname, name );
        if ( (fp = fopen( fname, "wb" ))==NULL )
           error_process("ファイルがオープンできませんでした. ");
        /* データの書き込み */
        /* ========== RIFF 1 ========== */
        write_char( fp, 4, RIFF1->ID );       /* ID     */
        write_int(    fp, 4, RIFF1->SIZE );   /* SIZE */
        write_char( fp, 4, RIFF1->TYPE );    /* TYPE */
        /* ========== fmt 1 ========== */
        write_char( fp, 4, fmt1->ID );   /* ID */
        write_int(    fp, 4, fmt1->SIZE );   /* SIZE */
        write_int(    fp, 2, fmt1->TYPE );   /* TYPE */
        write_int(    fp, 2, fmt1->Channel );   /* Channel */
        write_int(    fp, 4, fmt1->SamplesPerSec ); /* SamplesPerSec */
        write_int(    fp, 4, fmt1->BytesPerSec );   /* BytesPerSec */
        write_int(    fp, 2, fmt1->BlockSize );   /* BlockSize */
        write_int(    fp, 2, fmt1->BitsPerSample ); /* BitsPerSample */
        /* ========== data 1 ========== */
        write_char( fp, 4, data1->ID );   /* ID */
        write_int(    fp, 4, data1->size_of_sounds );   /* SIZE */
        for(i=0;i<data1->size_of_sounds;i++){
                fputc( *(data1->sounds + i), fp );
        }
        fclose(fp);
}

void read_char( FILE *fp, int n, char c[] )
/* n バイトの文字列をファイルから読み込む */
{
        int i;

        for(i=0;i<n;i++)
                c[i] = fgetc( fp );
        c[n]='¥0';   /* 文字列の最後を表す NULL 記号 */
}
void read_int( FILE *fp, int n, int *number )
/* n バイトの int をファイルから読み込む */
/* Intel バイトオーダ（byte 逆順）          */
{
        int i,j,num;

        *number = 0;
        for(i=0;i<n;i++){
                num = fgetc( fp );
                for(j=0;j<i;j++)
                        num = num * 256;
                *number = *number + num;
        }
}

void write_char( FILE *fp, int n, char c[] )
/* n バイトの文字列をファイルへ書き込む */
{
        int i;

        for(i=0;i<n;i++)      fputc( c[i], fp );
}

void write_int( FILE *fp, int n, int number )
/* n バイトで number のデータをファイルへ書き込む（逆順） */
{
        int i;
```

アドレス data1.souds から，size_of_sounds[byte] の大きさで波形データが保存されることに注意．

```
    for(i=0;i<n;i++){
        fputc( number % 256, fp );
        number = number / 256;
    }
}

void error_process( char message[] )
/* エラー処理 */
{
    if ( message[0]=='¥0' )
        printf("予想しないエラーが発生しました．¥n");
    else
        printf("%s¥n",message);
    printf("プログラムを終了します．¥n");
    exit(1);   /* 異常終了 */
}
```

```
void error_process2( char c1[], char c2[] )
/* エラー処理２ */
/* c1 と c2 が異なっていたら強制終了する */
{
    if ( strcmp( c1, c2 ) != 0 ){
        printf("ID が異なるファイルです．¥n");
        printf("プログラムを異常終了します．¥n");
        exit(1);   /* 異常終了 */
    }
}
```

■ wavelib.h の利用方法

- wavelib.h をインクルードすることによって次に示す wave **ファイル用の入出力関数**を利用することができるようになる．その際，RIFF チャンク，fmt チャンク，data チャンクに対応する次の**構造体**を宣言して利用している（wavelib.h 中を参照）．

```
/* RIFF チャンク */
struct RIFF {
    char ID[5];              /* RIFF チャンク ID("RIFF")          */
    int SIZE;                /* RIFF チャンクサイズ[byte]         */
    char TYPE[5];            /* RIFF の種類("WAVE")               */
};
/* fmt チャンク */
struct fmt {
    char ID[5];              /* fmt チャンク ID("fmt ")           */
    int SIZE;                /* fmt チャンクサイズ[byte]          */
    int TYPE;                /* フォーマットタイプ                */
    int Channel;             /* チャンネル数                      */
    int SamplesPerSec;       /* サンプリングレート[Hz]            */
    int BytesPerSec;         /* データ速度[byte/sec]              */
    int BlockSize;           /* ブロックサイズ                    */
                             /* [byte/sample x channel]           */
    int BitsPerSample;       /* サンプルあたりのビット数          */
                             /* [bit/sample]                      */
};
/* data チャンク */
struct data {
    char ID[5];              /* data チャンク ID("data")          */
    int size_of_sounds;      /* data チャンクサイズ[byte]         */
    unsigned char *sounds;   /* 波形データへのポインタ            */
};
```

RIFF チャンク

fmt チャンク

data チャンク

音声データの先頭アドレス

- void **load_wave_data**(struct RIFF ***RIFF1**, struct fmt ***fmt1**, struct data ***data1**, char **name[]**)

  【機能】wave ファイル（拡張子.wav）を読み込んで所定の**構造体変数**中に代入する．

  【利用方法】所定の構造体変数を宣言してから次に示すようにこの関数を呼ぶ．

```
struct RIFF RIFF1;     /* RIFF チャンク用構造体変数 */
struct fmt  fmt1;      /* fmt チャンク用構造体変数 */
struct data data1;     /* data チャンク用構造体変数 */

load_wave_data( &RIFF1, &fmt1, &data1, "" );
```

- void **save_wave_data**(struct RIFF ***RIFF1**, struct fmt ***fmt1**, struct data ***data1**, char **name[]**)

  【機能】所定の構造体変数中の音声データをファイル（拡張子.wav）へ出力する．

  【利用方法】所定の構造体変数を宣言してから次に示すようにこの関数を呼ぶ．

```
struct RIFF RIFF1;     /* RIFF チャンク用構造体変数 */
struct fmt  fmt1;      /* fmt チャンク用構造体変数 */
struct data data1;     /* data チャンク用構造体変数 */

save_wave_data( &RIFF1, &fmt1, &data1, "" );
```

- これらの関数でのデータのやりとりは **call by reference** と呼ばれるものであり，構造体変数の先頭アドレスを渡すことで，その変数に値を代入したり，参照したりするものである．プログラムを理解するために，C 言語における**ポインタ**，**構造体**，**構造体変数**，call by reference のあたりをよく復習しておいてほしい．

■ (wavelib.h を用いた)サンプルプログラム１：dispwave.c（第 11 回関連ファイル）

- 指定した wave ファイルの各チャンク内のデータを読み込んで画面に表示するプログラム dispwave.c を右に示す．
- 4 行目で wavelib.h をインクルードしている．
- このプログラムは波形データではなく，属性データを画面に表示するものである．
- oha.wav（女性音声「おはようございます」）を読み込ませたときの画面表示を下に示す．

```
C:¥ip>dispwave
===== wave ファイルの内容を表示するプログラム ====
読み込む WAVEファイルの名前（拡張子はwav）: oha.wav

***** WAVEデータの情報 *****
===== RIFF チャンク =====
  RIFFチャンクID      = RIFF
  RIFFチャンクサイズ  = 20068 [byte]
  RIFFチャンクタイプ  = WAVE
===== fmt  チャンク =====
  fmtチャンクID       = fmt
  fmtチャンクサイズ   = 16 [byte]
  fmtチャンクタイプ   = 1 (= monoral)
  チャンネル数        = 1
  サンプリングレート  = 22050 [Hz]
  データ速度          = 22050 [bytes/sec]
  ブロックサイズ      = 1 [byte/sample x channel]
  サンプルあたりのビット数 = 8 [bit/sample]
===== data チャンク =====
  dataチャンクID      = data
  dataチャンクサイズ  = 20032[byte]
  以下は波形データ

C:¥ip>
```

```c
/* このプログラムの名前：dispwave.c     */
/* WAVE ファイルの情報を示すプログラム */
#include<stdio.h>
#include"wavelib.h"

int main(void)
{
    struct RIFF RIFF1;     /* RIFF チャンク用構造体変数 */
    struct fmt  fmt1;      /* fmt  チャンク用構造体変数 */
    struct data data1;     /* data チャンク用構造体変数 */

    printf("===== wave ファイルの内容を表示するプログラム ====¥n");

    /* wave ファイルの読み込み */
    load_wave_data( &RIFF1, &fmt1, &data1, "" );

    /* wave ファイルの内容の表示 */
    printf("¥n***** WAVE データの情報 *****¥n");
    printf("===== RIFF チャンク =====¥n");
    printf("   RIFF チャンク ID        = %s¥n", RIFF1.ID );
    printf("   RIFF チャンクサイズ  = %d [byte]¥n", RIFF1.SIZE );
    printf("   RIFF チャンクタイプ  = %s¥n", RIFF1.TYPE );
    printf("===== fmt  チャンク =====¥n");
    printf("   fmt チャンク ID         = %s¥n", fmt1.ID );
    printf("   fmt チャンクサイズ   = %d [byte]¥n", fmt1.SIZE );
    printf("   fmt チャンクタイプ   = %d", fmt1.TYPE );
        if ( fmt1.TYPE == 1 ) printf(" (= monoral)¥n");
            else printf(" (= stereo)¥n");
    printf("   チャンネル数            = %d¥n", fmt1.Channel );
    printf("   サンプリングレート  = %d [Hz]¥n", fmt1.SamplesPerSec );
    printf("   データ速度              = %d [bytes/sec]¥n", fmt1.BytesPerSec );
    printf("   ブロックサイズ          = %d [byte/sample x channel]¥n", fmt1.BlockSize );
    printf("   サンプルあたりのビット数 = %d [bit/sample]¥n", fmt1.BitsPerSample );
    printf("===== data チャンク =====¥n");
    printf("   data チャンク ID        = %s¥n", data1.ID );
    printf("   data チャンクサイズ  = %d[byte]¥n", data1.size_of_sounds );
    printf("   以下は波形データ¥n");

    return 0;
}
```

■ (wavelib.h を用いた)サンプルプログラム２：copywave.c（第 11 回関連ファイル）

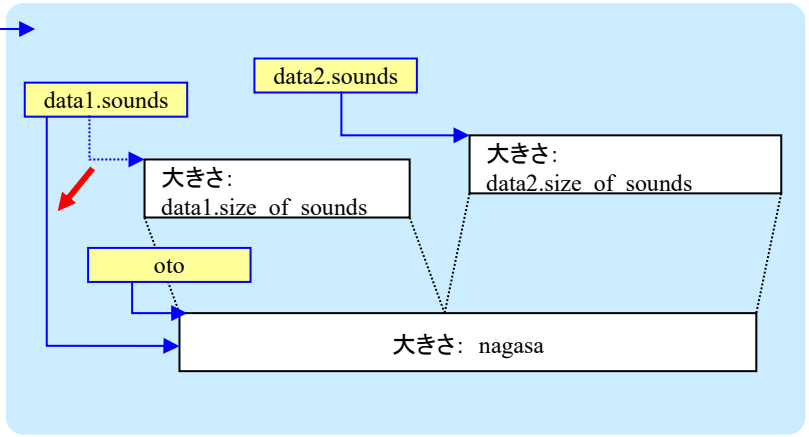
- wavelib.h を用いて wave ファイルをコピーするプログラム copy.c を右に示す．非常にシンプルであることがわかる．
- 音声データは，wavelib.h 中の構造体の宣言からわかるように，構造体変数 data1 の 3 番目のメンバーである **data1.sounds**（unsigned char を指すポインタ）が指すメモリ上のアドレスから，**data1.size_of_sounds**[byte]の大きさで保存されている（wavelib.h 参照）．なお，ここではそれを意識しなくても音声データをコピーすることができる．
- このプログラムを実行して oha.wav を読み込み，a.wav で保存して確認すると，同じ音声になっていることがわかる．

```c
/* このプログラムの名前：copywave.c     */
/* WAVE ファイルをコピーするプログラム */
#include<stdio.h>
#include"wavelib.h"

int main(void)
{
    struct RIFF RIFF1;     /* RIFF チャンク用構造体変数 */
    struct fmt  fmt1;      /* fmt  チャンク用構造体変数 */
    struct data data1;     /* data チャンク用構造体変数 */

    printf("===== wave ファイルをコピーするプログラム ====¥n");

    /* wave ファイルの読み込み */
    load_wave_data( &RIFF1, &fmt1, &data1, "" );

    /* wave ファイルの書き込み */
    save_wave_data( &RIFF1, &fmt1, &data1, "" );

    return 0;
}
```

■ (wavelib.h を用いた)サンプルプログラム３ : concatwave.c (第 11 回関連ファイル)

- wavelib.h を用いて2つの wave ファイルを結合した音声データを作って wave ファイルとして保存するプログラムを右に示す.
- 音声データ1を RIFF1, fmt1, data1, 音声データ 2 を RIFF2, fmt2, data2 にそれぞれ読み込んでいる.
- 続いて, メモリ上に両方の音声データの大きさ(それぞれ data1.size_of_sounds, data2.size_of_sounds)を足した大きさ(=nagasa) [byte]を標準関数 malloc によって確保している(先頭アドレスは oto(音)である).
- 次に, アドレス oto から 1 バイトずつ順に, 1つ目の音声データをコピーしている. また, 1つ目の音声データの最後に続いて2つ目の音声データをコピーしている.
- 続いて結合した音声の大きさと先頭アドレスを, それぞれ1つ目の音声データの大きさ(data1.size_of_sounds)と先頭アドレス(data1.sounds)に代入している. これにより RIFF チャンク全体も大きくなるため, 1 つめの音声の RIFF チャンクのサイズ(RIFF1.SIZE)を更新している.
- この様子をメモリ上で示すと右下の図のようになる.
- 最後に, 音声データ1(結合した音声)をファイルに出力している.
- このプログラムでは結合する2つの音声の fmt チャンク属性値(モノラル/ステレオ, サンプリングレート, データ速度など)が同じであるかどうかの判定は行っていない. このため, **これらの属性が同じ2つの音声ファイルを用いる必要がある**. その意味において本プログラムはあくまでも簡易なテスト用プログラムである点に注意してほしい.
- このプログラムを実行し, oha.wav(「おはようございます」)と ganba.wav(「がんばってね！」)を結合して b.wav で出力すると,「おはようございます. がんばってね！」になる.

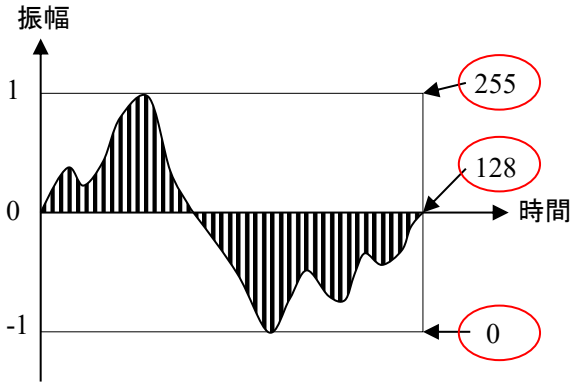
```c
/* このプログラムの名前 : concatwave.c      */
/* ２つの音声データを連結するプログラム */
#include<stdio.h>
#include"wavelib.h"

int main(void)
{
    struct RIFF RIFF1, RIFF2;   /* RIFF チャンク用構造体変数 */
    struct fmt  fmt1, fmt2;     /* fmt チャンク用構造体変数 */
    struct data data1, data2;   /* data チャンク用構造体変数 */
    int i,j;   /* 制御変数 */
    int nagasa;   /* 連結後の音声の長さ */
    unsigned char *oto;   /* 連結後の音声を指すポインタ */

    printf("=====  ２つの音声データを連結する  ====¥n");

    /* wave ファイルの読み込み */
    printf(" １つ目の音声データを読み込みます. ¥n");
    load_wave_data( &RIFF1, &fmt1, &data1, "" );
    printf(" ２つ目の音声データを読み込みます. ¥n");
    load_wave_data( &RIFF2, &fmt2, &data2, "" );

    /* 音声を連結する */
    printf("¥n １つ目の音声の後に２つ目の音声を連結します. ¥n¥n");
    /* 連結した音声データをメモリ上に作成する */
    nagasa = data1.size_of_sounds + data2.size_of_sounds;
    oto = (unsigned char *)malloc( nagasa );
    if ( oto == NULL ){
        printf("メモリが確保できません. プログラムを終了します. ¥n");
        exit(1);
    } else {
        /* １つ目の音声のコピー */
        for(i=0;i<data1.size_of_sounds;i++){
            *(oto + i) = *( data1.sounds + i );
        }
        /* ２つ目の音声のコピー */
        for(i=0;i<data2.size_of_sounds;i++){
            *(oto + data1.size_of_sounds + i) = *( data2.sounds + i );
        }
    }
    /* 音声１にコピー */
    data1.size_of_sounds = nagasa;   /* 音声データサイズを修正 */
    data1.sounds = oto;   /* 音声のコピー */
    RIFF1.SIZE += data2.size_of_sounds; /* RIFF チャンクサイズの修正 */

    /* wave ファイルの保存 */
    printf("連結した音声を保存します. ¥n");
    save_wave_data( &RIFF1, &fmt1, &data1, "" );

    return 0;
}
```



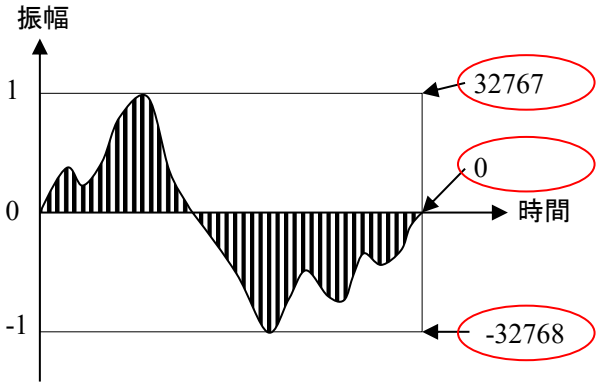
メモリ内での音声データ領域の確保とポインタの付け替え(data1.sounds)の様子

補足説明 （音声データについての補足）

- 音声データは最初のアドレス（音声データ 1 は data1.sounds が指すアドレス）から順に時間軸方向に保存されている．ある時刻のデータ（振幅）を何バイトで表すかは，fmt チャンク属性中の“サンプルあたりのビット数”[bit/sample]（wave の場合は 8bit または 16bit））で表されている．
- fmt チャンク属性中の“チャンネル数”（=1 ならモノラル，=2 ならステレオ）が 2 のときは，データは時間順に LRLRLRLR…と並ぶことになる．
- “サンプルあたりのビット数”が 8bit(1byte)のときと 16bit(2byte)のときの波形との対応を次に示す．



8bit (1byte)のとき



16bit (2byte)のとき

MEMO

本資料で説明したヘッダファイル，サンプルプログラム，サンプルデータは第 11 回講義関連ファイルとしてアップロードしました．各自で動作を確認して下さい．


Who are you?