2019

上級プログラミング

第三回レポート課題

2019/06/08

US162039　梶田悠

目次

レポートの課題内容

2 つの入力音声ファイルを結合し、出力音声ファイルに出力するプログラムを作成しなさい。具体的には、-P で指定された音声ファイルの開始サンプル数から終了サンプル数を outputdata の speechdata にコピーし、-F で指定された音声ファイルの開始サンプル数から終了サンプル数を outputdata の speechdata の続きにコピーした音声データを作成し、-O で指定された音声ファイルに出力するようにしなさい。レポート本体、作成したプログラムおよび「report3 -P nitech_jp_atr503_m001_a01.raw 0

19410 -F nitech_jp_atr503_m001_a04.raw 52063 70466 -O output.raw」を実行し、プロンプトで「sox -t raw -r 16000 -e signed-integer -b 16 -c 1 output.raw output.wav」を実行した結果得られる output.wav を担当教員に提出しなさい。

補足

コードは github にアップロードしてあるので、レポートのコードが読みづらい場合は適宜参照して下さい。

コード

ソースコード全文を以下に掲載する。可読性の為にコメントを意図的に削除している部分があります。

コード 1　Data.h

```
#include <iostream>
#include <fstream>
#include <cstdlib>
using namespace std;

class Data {
    string speechfilename; //データを格納した元の入力ファイル名
    short* speechdata; //データの先頭ポインタ
    int samplesize; //データ数
public:
    Data() { speechfilename = ""; speechdata = NULL; samplesize = 0; }
    void readrawfile(const string&);
    void writerawfile(const string&) const;
    void memoryalloc(const int&);
    void copydata(const Data&, const int&, const int&);
    void catdata(const Data&, const int&, const int&);
};
```

コード 2　Data.cpp

```
#include "Data.h"

//filename からデータを読み speechdata に格納
void Data::readrawfile(const string& filename) {
    speechfilename = filename;
    ifstream fin(filename.c_str(), ios::binary); //バイナリ形式でfilenameを開く
    if (!fin) {                //エラー処理
        cerr << "エラー：ファイルを開けません" << filename << "¥n";
        exit(EXIT_FAILURE);
    }
    fin.seekg(0, ios_base::end);
    int samplesuu = fin.tellg() / sizeof(short);
    memoryalloc(samplesuu);
    fin.seekg(0, ios_base::beg);
    fin.read((char*)speechdata, sizeof(short) * samplesuu);
}

//speechdataの格納データをfilenameにバイナリ形式で書き込む
void Data::writerawfile(const string& filename) const {
    ofstream fout(filename.c_str(), ios::binary);
    if (!fout) {
        cerr << "エラー：ファイルを開けません" << filename << "¥n";
        exit(EXIT_FAILURE);
    }

    fout.write((char*)speechdata, sizeof(short) * samplesize);
}

//speechdata のメモリを、short で memorysize 分確保する
void Data::memoryalloc(const int& memorysize) {
    delete[] this->speechdata;
    this->speechdata = new short[memorysize];
#ifdef _DEBUG
    cout << "allocated memorysize : " << memorysize << endl;
#endif // _DEBUG

}
//第 1 引数で指定された Data クラスのオブジェクト copymoto のデータメンバ
speechdata[start]から
//speechdata[end-1]までを、呼び出し元の speechdata[0]から speechdata[end-start-1]へコピーする。
void Data::copydata(const Data& copymoto, const int& start, const int& end) {
    this->samplesize = 0;
    for (int i = start; i < end; i++) {
        this->speechdata[samplesize] = copymoto.speechdata[i];
        this->samplesize++;
```

```
    }
}

//第 1 引数で指定された Data クラスのオブジェクト catmoto のデータメンバ
speechdata[start]から
//speechdata[end-1]までを、speechdata[samplesize]以降に追加する。
void Data::catdata(const Data& catmoto, const int& start, const int& end) {

#ifdef _DEBUG
    cout << "-----cat-----" << endl;
    cout << "copied size : " << samplesize << endl;
    cout << "cat    size : " << end - start << endl;
    cout << "total  size : " << samplesize + end - start << endl;
    cout << "-----/cat-----" << endl;
#endif // _DEBUG
    //後半のデータを詰める
    for (int i = start; i < end; i++) {
        this->speechdata[samplesize] = catmoto.speechdata[i];
        this->samplesize++;
    }
}
```

コード 3 jpro_report03.cpp

```
#include "Data.h"
#include <vector>

//第二引数の配列から'-'で始まる要素を探し、該当要素のインデックスを第三引数に詰める。該当要
素の個数を返す。
void searchOptionArg(const int& size, char* argv[], vector<int>& indices);

int main(int argc, char* argv[]) {
    char* inputfilename1 = NULL; //接続前部の音声ファイル名
    char* inputfilename2 = NULL; //接続後部の音声ファイル名
    char* outputfilename = NULL; //出力ファイル名
    int startsamplesuu1 = 0; //接続前部を利用する際の開始音声サンプル数
    int endsamplesuu1 = 0; //接続前部を利用する際の終了音声サンプル数(この値は含まず)
    int startsamplesuu2 = 0; //接続後部を利用する際の開始音声サンプル数
    int endsamplesuu2 = 0; //接続後部を利用する際の終了音声サンプル数(この値は含まず)
    string filename; //string に変換するための一時ファイル
    Data inputdata1; //接続前部の音声ファイルのデータの格納場所
    Data inputdata2; //接続後部の音声ファイルのデータの格納場所

    //コマンドライン引数の数が不適切な場合は異常終了
    if (argc != 11) {
        cerr << "Error : Please set args" << endl;
        exit(EXIT_FAILURE);
```

```
    }

    //
    const int requiredOptionNum = 3;
    vector<int> optionIndices;
    searchOptionArg(argc, argv, optionIndices);
#ifdef _DEBUG
    cout << "option Num : " << optionIndices.size() << endl;;
#endif // DEBUG

    //オプションが足らない場合は異常終了
    if (optionIndices.size() != requiredOptionNum) {
        cerr << "Error : Need -P & -F & -O options" << endl;
        exit(EXIT_FAILURE);
    }

    //コマンドライン引き数の各値を変数に格納
    for (int i = 0; i < requiredOptionNum; i++) {
        int index = optionIndices.at(i);
        switch (argv[index][1])
        {
        case 'P':
            inputfilename1 = argv[index + 1];
            startsamplesuu1 = atoi(argv[index + 2]);
            endsamplesuu1 = atoi(argv[index + 3]);
            break;
        case 'F':
            inputfilename2 = argv[index + 1];
            startsamplesuu2 = atoi(argv[index + 2]);
            endsamplesuu2 = atoi(argv[index + 3]);
            break;
        case 'O':
            outputfilename = argv[index + 1];
            break;
        default:
            //不適切なオプションがあれば異常終了
            cerr << "Error : Found incorrect option = " << argv[index][1] << endl;
            exit(EXIT_FAILURE);
            break;
        }
    }

#ifdef _DEBUG
    cout << "inputfilename1  : " << inputfilename1  << endl;
    cout << "startsamplesuu1 : " << startsamplesuu1 << endl;
    cout << "endsamplesuu1   : " << endsamplesuu1   << endl;

    cout << "inputfilename2  : " << inputfilename2  << endl;
```

```
    cout << "startsamplesuu2 : " << startsamplesuu2 << endl;
    cout << "endsamplesuu2   : " << endsamplesuu2   << endl;

    cout << "outputfilename  : " << outputfilename  << endl;
#endif // DEBUG


    //1 つ目の入力ファイルの読み込み
    filename = inputfilename1;
    inputdata1.readrawfile(filename);
    //2 つ目の入力ファイルの読み込み
    filename = inputfilename2;
    inputdata2.readrawfile(filename);

    //音声データの接続
    Data outputdata; //出力ファイルのデータの格納場所
    if ((inputfilename1 != NULL) && (startsamplesuu1 != endsamplesuu1) &&
(inputfilename2 != NULL) &&
        (startsamplesuu2 != endsamplesuu2)) {
        //出力データを格納するためのメモリを確保する
        outputdata.memoryalloc(endsamplesuu1 - startsamplesuu1 + endsamplesuu2 -
startsamplesuu2);
        //inputdata1 の speechdata の 0 から 19409 までを、
        //outputdata の speechdata の 0 から 19409 にコピーする。
        outputdata.copydata(inputdata1, startsamplesuu1, endsamplesuu1);
        //inputdata2 の speechdata の 52063 から 70466 までを、
        //outputdata の speechdata の最後(=この場合 19410)から 18404 個コピーする。
        outputdata.catdata(inputdata2, startsamplesuu2, endsamplesuu2);
    }
    else {
        cerr << "音声を接続する前部・後部の音声ファイル名および開始サンプル数・終了サンプ
ル数を指定してください¥n";
        exit(EXIT_FAILURE);
    }

    //変更したデータをファイルに保存
    if (outputfilename != NULL) {
        filename = outputfilename;
        outputdata.writerawfile(filename);
    }
    else {
        cerr << "outputfilename を指定してください¥n";
        exit(EXIT_FAILURE);
    }
    return 0;
}
```

```
void searchOptionArg(const int& size, char* argv[], vector<int>& indices)
{
    for (int i = 0; i < size; i++) {
        if (argv[i][0] == '-') {
            indices.push_back(i);
        };
    }
}
```

実行結果

上記のプログラムを実行して得られた output.raw に対して、指定された sox コマンドを実行して生成された output.wav ファイルを添付しているのでそちらを適宜参照して下さい。

考察

不適切なオプションで実行された場合について

作成したプログラムは、-F,-P,-O の 3 つのオプションと共に実行される想定である。コマンドライン引数の配列(char* argv[])から”-”が先頭にある要素を検索し、インデックスを vector<int>型の optionIndices 変数に格納してから各要素が F,P,O の内どのオプションを指定しているかで処理を分岐させた。

このオプションの抽出の処理を実装し始めた当初は、vector<int>ではなく固定長配列を用いてインデックスを記録しようとした。この場合、当然要素数を固定しようとするわけだが想定通り 3 つのオプションが入力される事を前提にするのなら要素数は 3 で問題ない。しかし実際的には間違いで 4 つ以上のオプションを入力してしまう可

能性も考えられる。その場合は間違いを指摘し正しい入力を促す警告を使用者に提示するべきと考え可変長な配列を用意して要素数を判別する事で間違いがある場合はその旨を出力するようにした。その結果 vector<int>を使用した。

可変長であれば vector 以外にも、例えば int*型の変数を用意し、要素が増える度にインクリメントしメモリ上に連続させて記録させる方法も考えられた。以前別のプログラムでこの考えで実装した際に期待通りの挙動をしなかったことがあった。この原因はポインタによるメモリ上に連続するデータ構造によって不適切なメモリに値を書き込んでしまっていた事だと結論付けた。この時は要素数が大きかった為このようなバグが起きた可能性が高いと考えられるが、こういったバグが発生する可能性を考慮して vector によるデータ構造を採用した。

デバックの効率化

作成したプログラムの各工程で適切に処理が実行されているかを確認する為に #define _DEBUG マクロによって_DEBUG が定義されている場合は処理の途中経過をコンソールに出力するようにした。

例えばコマンドライン引数の分解においては、指定したオプションの数が過不足ないか、引数で指定した値を適切な変数に代入できているかを確認できる様にした。

また、データの抽出・結合ではデータサイズを 1 つ分多くメモリを確保してしまうなどのミスが予想されるので確保したメモリサイズと、結合時に既に抽出したデータサイズ、結合するデータのサイズを出力するようにして適切に処理出来ているか確認できるようにした。

実際のコンソールの出力結果は例えば以下の様になる。

```
option Num : 3
inputfilename1  : a01.raw
startsamplesuu1 : 0
endsamplesuu1   : 19410
inputfilename2  : a04.raw
startsamplesuu2 : 52063
endsamplesuu2   : 70466
outputfilename  : output.raw
allocated memorysize : 60700
allocated memorysize : 70467
allocated memorysize : 37813
-----cat-----
copied size : 19410
cat    size : 18403
total  size : 37813
-----/cat-----
```