The Automatic Vasospasm Detection Application

Generated by Doxygen 1.8.8

Wed Apr 20 2016 00:56:10

Contents

Chapter 1

Main Page

Introduction

The Automatic Vasospasm Detection Application (or Algorithm, depending on the usage), AVDA, is an application to objectively detect the presence of vasospasms based on comparisons of parameters extracted from transcranial doppler audio.

Setup

AVDA is intended to be compiled on machines running Linux, though it could likely be adapter for other environments. It must be downloaded from GitHub.com and compiled locally. To do this, navigate to the directory in which AVDA should be placed, then execute the following commands

```
git clone https://github.com/sawbg/avda
cd avda
make
```

Sucessfully cloning, compilation, and execution of AVDA requires up-to-date versions of the following executables:

- git
- make
- gcc (4.9)
- arecord

FAQ

- **Why was this project developed?** This project was developed as a course project by two gradute students at the University of Alabama at Birmingham School of Engineering, Nicholas Nolan and Andrew Wisner.
- **Is AVDA an active project?** Though it is not planned to develop AVDA further in the near future, it is hoped that the algorithm discovered and implemented can be used and built upon by researchers to fully automate the detection of vasospasms.
- **AVDA is returning unusually low or high parameters. Why might this be?** In development, this occured when the mic-in volume was set too high. It is likely in this senario that clipping is happening or that the signal (or a strong enough signal) has no been received.
- **How will AVDA be affected by the machine uprising?** The University supercomputer, Cheaha, has assured us that AVDA will not be needed after the uprising occures.
- **What about more specific questions?** Questions relating to AVDA not covered in this FAQ may be sent to the AVDA team via awisner94@gmail.com.

License

Chapter 2

Bug List

File fileio.hpp

file is overly complicated and much more bug-prone

File main.cpp

extra newline character inserted into stdin buffer after PatientName() is run

Chapter 3

Namespace Index

3.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 avda Namespace Reference

Enumerations

- enum Side { Side::Left, Side::Right }

Functions

- std::string PatientName ()
- std::map< Side, DataParams > ReadParams (auto filename)
- void WriteParams (std::map< Side, DataParams > myMap, auto filename)
- DataParams process (float32 *data, uint32 size, float32 samplingRate)
- void absolute (float32 *data, uint32 size)
- float32 average (float32 *data, uint32 size)
- DataParams average (DataParams *params, uint8 size)
- void decibels (float32 *data, uint32 size)
- void diff (float32 *data, uint32 size)
- void fft (cfloat32 *data, uint32 size)
- void mag (cfloat32 *orig, float32 *newmags, uint32 size)
- Maximum max (float32 *data, uint32 size)
- void smooth (float32 *data, uint32 size, uint16 order)
- void play (auto filename)

Variables

- const std::string CSV_HEADER = "Time,Side,Frequency,Noise Level"
- const std::string PATIENT_PATH = "/home/pi/patients/"

6.1.1 Detailed Description

This namespace contains all code related to this project.

6.1.2 Enumeration Type Documentation

6.1.2.1 enum avda::Side [strong]

Side of the head to which a recording pertains.

Enumerator

Left

Right

Definition at line 121 of file definitions.hpp.

```
{ Left, Right };
```

6.1.3 Function Documentation

6.1.3.1 void avda::absolute (float32 * *data*, uint32 *size*)

Ensures all elements in an array are positive. Note that this function replaces array elements if necessary. It does not populate a new array.

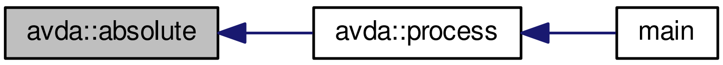
Parameters

data	the array whose elements must all be positive
size	the number of elements in the data array

Definition at line 123 of file sigmath.hpp.

```
                                               {
            for(uint32 i = 0; i < size; i++) {
                data[i] = fabsf(data[i]);
            }
        }
```

Here is the caller graph for this function:



6.1.3.2 float32 avda::average (float32 * *data*, uint32 *size*)

Takes the average of all elements in an array

Parameters

data	the array from which to compute the average
size	the number of elements in the data array

Returns

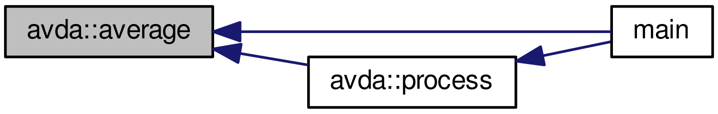
the computed average

Definition at line 129 of file sigmath.hpp.

```
                                                  {
            float32 ave;

            for(uint32 i = 0; i < size; i++) {
```

```
            ave += data[i];
        }

        ave = ave / size;
        return ave;
    }
```

Here is the caller graph for this function:



6.1.3.3 DataParams avda::average (DataParams ∗ *params,* uint8 *size*)

Finds the averages of the elements of an array of DataParams.

Parameters

params	the DataParams array
size	the number of elements in the DataParams array

Returns

a DataParams structure containing the average values of the structure's elements in the params array

Definition at line 140 of file sigmath.hpp.

```
                                                    {
        DataParams ave;

        for(uint8 i = 0; i < size; i++) {
            //freq is an attribute. this is how to add structure attributes
            ave.freq += params[i].freq;
            ave.noise += params[i].noise;
        }

        ave.freq /= size;
        ave.noise /= size;
        return ave;
    }
```

6.1.3.4 void avda::decibels (float32 ∗ *data,* uint32 *size*)

Converts an array of floats to "power decibels", i.e., x[n] = 20∗log10(x[n]). The decibel values are written to the same array that contained the values to be converted. In other words, this function should perform an in-place, element-wise conversion.

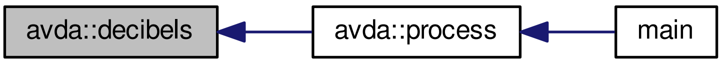
Parameters

data	the array of values to be converted as well as the location where the converted values will be written
size	the number of elements in the data array

Definition at line 154 of file sigmath.hpp.

```
                                             {
        for(uint32 i = 0; i < size; i++) {
            data[i] = 20 * log10(data[i]);
        }
    }
```

Here is the caller graph for this function:



6.1.3.5 void avda::diff (float32 ∗ *data,* uint32 *size*)

Computes the left-handed first derivative of a discrete signal. The first element will be 0.

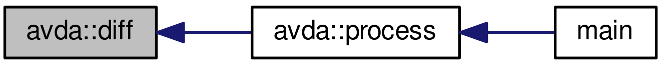
Parameters

data	an array containing the discrete signal data
size	the number of elements in data

Definition at line 160 of file sigmath.hpp.

```
                                         {
        float32 temp[size];
        temp[0] = 0;

        for(uint32 i = 1; i < size; i++) {
            temp[i] = data[i] - data[i-1];
        }

        for(uint32 i = 0; i < size; i++) {
            data[i] = temp[i];
        }
    }
```

Here is the caller graph for this function:



6.1.3.6 void avda::fft (cfloat32 * *data*, uint32 *size*)

Replaces the values of an array of cfloat32's with the array's DFT using a decimation-in-frequency algorithm.

This code is based on code from http://rosettacode.org/wiki/Fast_Fourier_transform#C.↩
2B.2B.

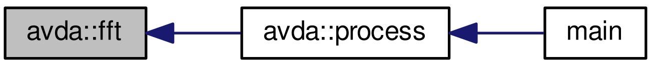
Parameters

data	the array whose values should be replaced with its DFT
size	the number of elements in the data array

Definition at line 173 of file sigmath.hpp.

```
00173                                           {
00174         // DFT
00175         uint32 k = size;
00176         uint32 n;
00177         float32 thetaT = M_PI / size;
00178         cfloat32 phiT(cos(thetaT), sin(thetaT));
00179         cfloat32 T;
00180 
00181         while(k > 1) {
00182             n = k;
00183             k >>= 1;
00184             phiT = phiT * phiT;
00185             T = 1.0L;
00186 
00187             for(uint32 l = 0; l < k; l++) {
00188                 for(uint32 a = l; a < size; a += n) {
00189                     uint32 b = a + k;
00190                     cfloat32 t = data[a] - data[b];
00191                     data[a] += data[b];
00192                     data[b] = t * T;
00193                 }
00194 
00195                 T *= phiT;
00196             }
00197         }
00198 
00199         // Decimate
00200         uint32 m = (uint32)log2(size);
00201 
00202         for(uint32 a = 0; a < size; a++) {
00203             uint32 b = a;
00204 
00205             // Reverse bits
00206             b = (((b & 0xaaaaaaaa) >> 1) | ((b & 0x55555555) << 1));
00207             b = (((b & 0xcccccccc) >> 2) | ((b & 0x33333333) << 2));
00208             b = (((b & 0xf0f0f0f0) >> 4) | ((b & 0x0f0f0f0f) << 4));
00209             b = (((b & 0xff00ff00) >> 8) | ((b & 0x00ff00ff) << 8));
00210             b = ((b >> 16) | (b << 16)) >> (32 - m);
00211 
00212             if (b > a)
00213             {
00214                 cfloat32 t = data[a];
00215                 data[a] = data[b];
00216                 data[b] = t;
00217             }
00218         }
00219     }
```

Here is the caller graph for this function:



6.1.3.7 void avda::mag (cfloat32 ∗ *orig*, float32 ∗ *newmags*, uint32 *size*)

Computes the magitude of an array of complex numbers.

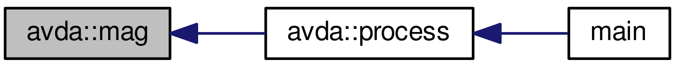
Parameters

orig	the array of complex numbers
newmags	an array to which the magitudes are to be written
size	the number of elements in orig and newmags

Definition at line 221 of file sigmath.hpp.

```
                                                  {
        //loop to run throught the length of array orig
        for(uint32 n = 0; n < size; n++) {
            /*
             * abs should calculate the magnitude of complex array elements.
             * saves to new array
             */
            newmags[n] = std::abs(orig[n]);
        }
    }
```

Here is the caller graph for this function:



6.1.3.8 Maximum avda::max (float32 ∗ *data,* uint32 *size*)

Finds the maximum value in an array.

Parameters

data	the array whose maximum value is to be found
uint32	size the number of elements in the data array

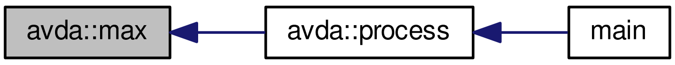
Returns

the maximum value and its index in a Maximum structure

Definition at line 232 of file sigmath.hpp.

```
                                    {
        Maximum m;

        //loop to run through the length of array data
        for (uint32 i = 0; i < size; i++) {
            /*
             * when value at data[i] is above max.value,
             * sets max.value equal to data[i] and max.index equal to i
             */
            if (data[i] > m.value) {
                m.value = data[i];
                m.index = i;
            }
        }

        return m;
    }
```

Here is the caller graph for this function:



6.1.3.9 std::string avda::PatientName ( )

Prompts a user to enter a first, middle, and last name for a patients and creates a file (if necessary) in which all of a patient's data can be saved. A newly created file will contain the CSV header for the file's data.

Must warn a user if the patient folder does not already exist in order to prevent missaving data.

Returns

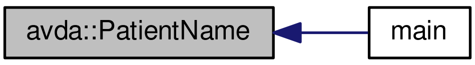
the file under which all patient data is saved

Definition at line 43 of file fileio.hpp.

```
                                {
        std::string fname = "";
        std::string mname = "";
        std::string lname = "";
        std::string patfil = "";
        std::string patientname = "";
        uint32 track1 = 0;
        uint32 track2 = 0;
        uint32 track3 = 0;

        do {
            std::cout << "Please enter the patients name." << std::endl;
            std::cout << "First name: ";
            std::cin >> fname;
            std::cout << "Middle name: ";
            std::cin >> mname;
            std::cout << "Last name: ";
            std::cin >> lname;

            // creates new std::string with path to patient file
            patientname = PATIENT_PATH + lname + ", " + fname
                + " " + mname + ".csv";

            // prints out patientname. shows user the path to the patient file
            //std::cout << patientname << std::endl << std::endl;
            std::ifstream file(patientname.c_str());

            if (file.good()) {
                track1 = 1;
            }

            /*
             * Compares patientname to existing files and lets user know
             * if the file does not exist.
             */
            else if (!file.good()) {
                /*
                 * Do while statement to continue asking user about the file
                 * if their input is not acceptable
                 */
                do {
                    std::cout << "Patient file does not exist, would you like "
                        "to create file or re-enter their name?" << std::endl;
                    std::cout << "  *Type 'create' and press enter key "
                        "to create the patient file." << std::endl;
                    std::cout << "  *Type 'reenter' and press enter key "
                        "to re-enter the patients name." << std::endl;
                    std::cout << std::endl;
```

```
                        std::cin >> patfil;

                        /*
                         * patfil equals create, track1 and 2 will increase
                         * escaping both do while loops
                         */
                        if(patfil == "create") {
                            std::ofstream createfile(patientname.c_str());
                            track1 = 1;
                            track2 = 1;
                            track3 = 1;
                            createfile << CSV_HEADER << std::endl;
                            createfile.flush();
                            createfile.close();
                        }

                        /*
                         *patfil equals renter, track1 will remain zero allowing
                         *user to reenter the patient name.
                         */
                        else if(patfil == "reenter") {
                            track1 = 0;
                            track2 = 1;
                        }

                        /*
                         *The users input was neither create or reenter. User
                         *must enter patient name again.
                         */
                        else {
                            std::cout << std::endl;
                            std::cout << "Your input is not acceptable." << std::endl;
                            std::cout << std::endl;
                        }
                    }while(track2 == 0);
                }
            } while (track1 == 0);

            return patientname; //returns the path to the patient file
        }
```

Here is the caller graph for this function:



6.1.3.10 void avda::play (auto *filename*)

Plays a WAVE file in a loop in a non-blocking manner.

Parameters

filename	the absolute or relative path to the WAVE file

Definition at line 20 of file sound.hpp.

```
                                 {

        }
```

6.1.3.11 DataParams avda::process (float32 ∗ *data,* uint32 *size,* float32 *samplingRate*)

Analyzes a single recording to determine the drop-off frequency and average noiseband noise power.

It should be noted that is algorithm is considered the intellectual property of Andrew Wisner and Nicholas Nolan. The "algorithm" is defined as the use of 1) the frequency drop-off and/or 2) a noise value from the frequency band above the drop-off frequency in order to diagnose (with or without other factors and parameters) the presence of a avdaspasm in a patient. By faculty members and/or students in the UAB ECE department using this algorithm, they agree that the presentation of their code or project that uses this algorithm by anyone directly or indirectly related to the code or project, whether verbally or in writing, will reference the development of the initial algorithm by Andrew Wisner and Nicholas Nolan. Furthermore, a failure to meet this stipulation will warrant appropriate action by Andrew Wisner and/or Nicholas Nolan. It should be understood that the purpose of this stipulation is not to protect priopietary rights; rather, it is to help ensure that the intellectual property of the aforementioned is protected and is neither misrepresented nor claimed implicitly or explicitly by another individual.

Parameters

data	array containing float32 samples of audio
size	number of samples in each recording. MUST be a power of two.
samplingRate	the sampling frequency in Hz or Samples/second

Returns

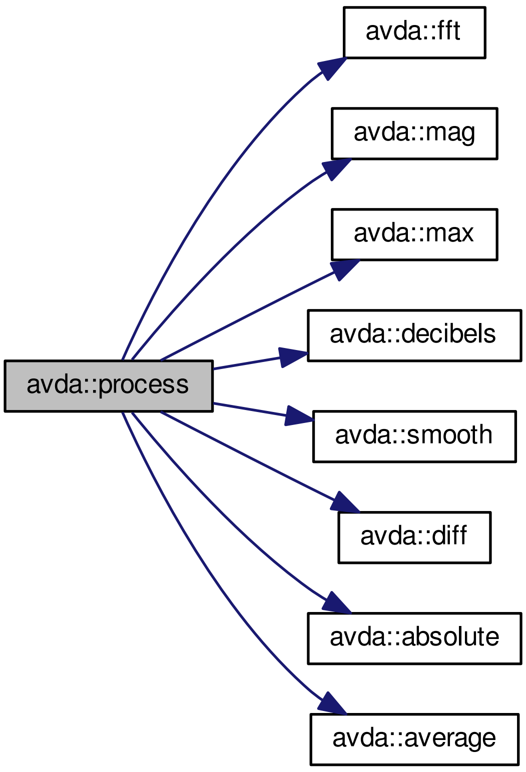
cut-off frequency (Hz) and average noiseband noise power in decibels

Definition at line 48 of file process.hpp.

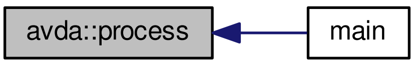
```
00048                                                                         {
00049         if((size & (size - 1) != 0) || size < 2) {
00050             throw std::invalid_argument(
00051                     "The number of samples is not a power of two!");
00052         }
00053 
00054         // declare function-scoped variables
00055         uint32 freqSize = size / 2;
00056         cfloat32* cdata = (cfloat32*)std::malloc(size * sizeof(
      cfloat32));
00057         float32* fdata = (float32*)std::malloc(freqSize * sizeof(
      float32));
00058         float32* origdata = (float32*)std::malloc(freqSize * sizeof(
      float32));
00059 
00060         // convert data to complex numbers for fft()
00061         for(uint32 i = 0; i < size; i++) {
00062             cdata[i] = data[i];
00063         }
00064 
00065         // find frequency spectrum in relative decibels
00066         fft(cdata, size);
00067         mag(cdata, fdata, freqSize);
00068         Maximum maximum = max(fdata, freqSize);
00069 
00070         for(uint32 i = 0; i < freqSize; i++) {
00071             fdata[i] /= maximum.value;
00072         }
00073 
00074         decibels(fdata, freqSize);
00075 
00076         for(uint32 i = 0; i < freqSize; i++) {
00077             origdata[i] = fdata[i];
00078         }
00079 
00080         /*
00081          * Run spectrum values through moving-average filter to smooth the
00082          * curve and make it easier to determine the derivative.
00083          */
00084         smooth(fdata, freqSize, 20);
00085 
00086         /*
00087          * Find the derivative of the smoothed spectrum. Bote that both this
00088          * filter and the previous are necessary to the algorithm.
00089          */
00090         diff(fdata, freqSize);
00091         smooth(fdata, freqSize, 100);
00092         absolute(fdata, freqSize);
00093 
00094         // find the parameters of this specific recording
00095         uint16 offset = 1000;
00096         absolute(&fdata[offset], freqSize - offset);
00097         maximum = max(&fdata[offset], freqSize - offset);
00098         uint32 index = maximum.index + offset;
```

```

        DataParams params;
        params.freq = index * (float)SAMPLE_FREQ / freqSize / 2;
        params.noise = average(&origdata[index + offset],
                freqSize - offset - index);

        free(cdata);
        free(fdata);

        return params;

    }
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.1.3.12 std::map<Side, DataParams> avda::ReadParams (auto *filename*)

Reads the previously computated parameters found in the specified file.

Parameters

filename	the absolute or relative path to the file containing the patient data to read

Returns

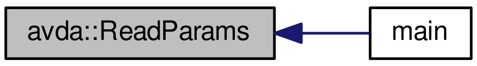
the patient parameters read for each side

Definition at line 141 of file fileio.hpp.

```
00141                                                         {
00142         std::map<Side, DataParams> myMap;
00143         DataParams leftparams;
00144         DataParams rightparams;
00145 
00146         std::ifstream file(filename.c_str());
00147         std::string leftline;
00148         std::string rightline;
00149         std::string leftsearch = "Left";
00150         std::string rightsearch = "Right";
00151         std::string paramstring;
00152         std::string lfreqstr;
00153         std::string lnoisestr;
00154         std::string rfreqstr;
00155         std::string rnoisestr;
00156         uint32 lcnt = 0;
00157         uint32 rcnt = 0;
00158         float32 lfreqval;
00159         float32 lnoiseval;
00160         float32 rfreqval;
00161         float32 rnoiseval;
00162 
00163         /*
00164          * if statement which uses ifstream function to open patient file
00165          * filename)
00166          */
00167         if(file.is_open()) {
00168             /*
00169              * While statement to find the first Left line and save to
00170              *leftline as string.
00171              */
00172             while (getline(file, leftline)) {
00173                 if(leftline.find(leftsearch, 0) != std::string::npos) {
00174                     break;
00175                 }
00176 
00177             }
00178 
00179             /*
00180              * While statement to find first right line and save to rightline
00181              * as string.
00182              */
00183             while (getline(file,rightline)) {
00184                 if(rightline.find(rightsearch, 0) != std::string::npos) {
00185                     break;
00186                 }
00187             }
00188 
00189             // Code to break leftline and rightline into its parts
00190             std::stringstream lss(leftline);
00191             std::stringstream rss(rightline);
00192 
00193             while(getline(lss,paramstring, ',')) {
00194                 lcnt++;
00195 
00196                 if(lcnt == 3) {
00197                     lfreqstr = paramstring;
00198                 }
00199 
00200                 else if(lcnt == 4) {
00201                     lnoisestr = paramstring;
00202                 }
00203             }
00204 
00205             while(getline(rss,paramstring, ',')) {
00206                 rcnt++;
00207 
00208                 if(rcnt == 3) {
00209                     rfreqstr = paramstring;
00210                 }
00211 
00212                 else if(rcnt == 4) {
00213                     rnoisestr = paramstring;
```

```
                }
            }

            /*
             * Statement to convert lfreq, lnoise, rfreq, and rnoise from
             * strings to floats.
             * */
            lfreqval = atof(lfreqstr.c_str());
            lnoiseval = atof(lnoisestr.c_str());
            rfreqval = atof(rfreqstr.c_str());
            rnoiseval = atof(rnoisestr.c_str());

            file.close();
        }

        else {
            throw std::runtime_error("The patient file could not be opened.");
        }

        leftparams.freq = lfreqval;
        leftparams.noise = lnoiseval;
        rightparams.freq = rfreqval;
        rightparams.noise = rnoiseval;

        myMap[Side::Left] = leftparams;
        myMap[Side::Right] = rightparams;

        return myMap;
    }
```

Here is the caller graph for this function:



6.1.3.13 void avda::smooth (float32 ∗ *data,* uint32 *size,* uint16 *order*)

Applies an nth-order moving-average filter to a discrete signal.

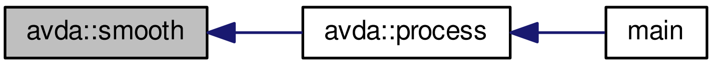
Parameters

data	the array containing the signal to which the filter should be applied
size	the number of elements in the data array
order	the order of the filter

Definition at line 250 of file sigmath.hpp.

```
                                                        {
        float32 coeff = 1 / (float32)order;
        float32 temp[size];

        for(uint32 i = 0; i < size; i++) {
            temp[i] = 0;

            for(uint16 j = 0; j < order && j <= i; j++) {
                temp[i] += data[i - j];
            }

            temp[i] *= coeff;
        }

        for(uint32 i = 0; i < size; i++) {
            data[i] = temp[i];
        }
    }
```

Here is the caller graph for this function:



6.1.3.14 void avda::WriteParams (std::map< Side, DataParams > *myMap*, auto *filename*)

Writes (appends) the passed parameters to the specified file.

Parameters

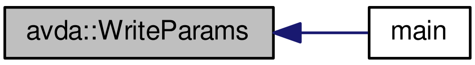
myMap	contains the parameters to be written

the patient CSV file's filename

Definition at line 251 of file fileio.hpp.

```
                                                                  {
        char temp[80];
        std::ofstream file(filename.c_str(),
                std::ofstream::out | std::ofstream::app);

        //Gives pointer measurementtime a data type of time_t.
        time_t measurementtime;
        time(&measurementtime); //Gets the current time.
        strftime(temp, 80, "%c", localtime(&measurementtime));
        std::string fTime = std::string(temp);

        //if statement to print the Left side parameters to the patient file.
        if(file.is_open()) {
            file << fTime + "," + "Left" + ","
                + std::to_string(myMap[Side::Left].freq)
                + ", " + std::to_string(myMap[Side::Left].noise) << std::endl;
        }

        //if statement to print the Right side parameters to the patient file.
        if(file.is_open()) {
            file << fTime + "," + "Right" + ","
                + std::to_string(myMap[Side::Right].freq)
                + ", " + std::to_string(myMap[Side::Right].noise) << std::endl;
        }

        else {
            std::cout << "Patient file can not be opened!" << std::endl;
        }

        file.close();
    }
```

Here is the caller graph for this function:



6.1.4 Variable Documentation

6.1.4.1 const std::string avda::CSV_HEADER = "Time,Side,Frequency,Noise Level"

First line of CSV data file, which declares columns.

Definition at line 25 of file fileio.hpp.

6.1.4.2 const std::string avda::PATIENT_PATH = "/home/pi/patients/"

Absolute path to the folder containing the patients' data

Definition at line 30 of file fileio.hpp.

Chapter 7

Class Documentation

7.1 DataParams Struct Reference

```
#include <definitions.hpp>
```

Public Attributes

- float32 freq = 0
- float32 noise = 0

7.1.1 Detailed Description

Calculated results from processing the audio recordings.

Definition at line 97 of file definitions.hpp.

7.1.2 Member Data Documentation

7.1.2.1 float32 DataParams::freq = 0

Definition at line 98 of file definitions.hpp.

7.1.2.2 float32 DataParams::noise = 0

Definition at line 99 of file definitions.hpp.

The documentation for this struct was generated from the following file:

- src/definitions.hpp

7.2 Maximum Struct Reference

```
#include <definitions.hpp>
```

Public Attributes

- float32 value = 0
- uint32 index = 0

7.2.1 Detailed Description

Maximum value found in an array and the value's index in that array.

Definition at line 106 of file definitions.hpp.

7.2.2 Member Data Documentation

7.2.2.1 uint32 Maximum::index = 0

Definition at line 108 of file definitions.hpp.

7.2.2.2 float32 Maximum::value = 0

Definition at line 107 of file definitions.hpp.

The documentation for this struct was generated from the following file:

- src/definitions.hpp

Chapter 8

File Documentation

8.1 etc/doxygen.config File Reference

Contains Doxygen configuration settings.

8.1.1 Detailed Description

Contains Doxygen configuration settings.

Author

Samnuel Andrew Wisner, awisner94@gmail.com

Definition in file doxygen.config.

8.2 doxygen.config

```
PROJECT_NAME = "The Automatic Vasospasm Detection Application"

INPUT = src/ etc/doxygen.config makefile README.md
OUTPUT_DIRECTORY = doc/

GENERATE_HTML = YES
GENERATE_RTF = YES
GENERATE_LATEX = YES
GENERATE_MAN = YES
GENERATE_XML = NO
GENERATE_DOCBOOK = NO

USE_PDF_LATEX = YES
USE_PDF_HYPERLINKS = YES

RECURSIVE = YES
SOURCE_BROWSER = YES
SOURCE_TOOLTIPS = YES
EXTRACT_ALL = YES
DISABLE_INDEX = NO
GENERATE_TREEVIEW = YES
SEARCHENGINE = YES
SERVER_BASED_SEARCH = NO
USE_MDFILE_AS_MAINPAGE = README.md

LATEX_SOURCE_CODE = YES
STRIP_CODE_COMMENTS = YES
INLINE_SOURCES = YES

HAVE_DOT = YES
CALL_GRAPH = YES
CALLER_GRAPH = YES
```

8.3 makefile File Reference

Contains recipes for building the test applications, the main application, and the documentation.

8.3.1 Detailed Description

Contains recipes for building the test applications, the main application, and the documentation.

Author

Samuel Andrew Wisner, awisner94@gmail.com

Definition in file makefile.

8.4 makefile

```
GCC = g++ -g -std=gnu++14

avda:
   $(GCC) src/main.cpp -o bin/avda

count:
   grep -r "src/" -e "Samuel Andrew Wisner" -l | xargs wc -l

docs:
   rm -r doc/
   doxygen etc/doxygen.config
   cd doc/latex; make pdf;
   git reset
   git add doc/.
   git commit -m "Updated documentation."
   git push

fileio-test:
   $(GCC) src/fileio_test.cpp -o bin/fileiotest

patient-name-test:
   $(GCC) src/patient_name_test.cpp -o bin/patnametest

process-test:
   $(GCC) src/process_test.cpp -o bin/proctest

read-params-test:
   $(GCC) src/read_params_test.cpp -o bin/rptest

stdin-clear-test:
   $(GCC) src/stdin_clear_test.cpp -o bin/cleartest
```

8.5 README.md File Reference

Contains the readme text as markdown, which also doubles as the main page.

8.5.1 Detailed Description

Contains the readme text as markdown, which also doubles as the main page.

Author

Samuel Andrew Wisner, awisner94@gmail.com

Definition in file README.md.

8.6 README.md

```
# vasospasm-detector

## Introduction
The Automatic Vasospasm Detection Application (or Algorithm, depending on the
usage), AVDA, is an application to objectively detect the presence of vasospasms
based on comparisons of parameters extracted from transcranial doppler audio.

## Setup
AVDA is intended to be compiled on machines running Linux, though it could
likely be adapter for other environments. It must be downloaded from GitHub.com
and compiled locally. To do this, navigate to the directory in which AVDA should
be placed, then execute the following commands

   git clone https://github.com/sawbg/avda
   cd avda
   make

Sucessfully cloning, compilation, and execution of AVDA requires up-to-date
versions of the following executables:

* git
* make
* gcc (4.9)
* arecord

## FAQ

* **Why was this project developed?** This project was developed as a course
project by two gradute students at the University of Alabama at Birmingham
School of Engineering, Nicholas Nolan and Andrew Wisner.

* **Is AVDA an active project?** Though it is not planned to develop AVDA
further in the near future, it is hoped that the algorithm discovered and
implemented can be used and built upon by researchers to fully automate the
detection of vasospasms.

* **AVDA is returning unusually low or high parameters. Why might this be?** In
  development, this occured when the mic-in volume was set too high. It is
likely in this senario that clipping is happening or that the signal (or a
strong enough signal) has no been received.

* **How will AVDA be affected by the machine uprising?** The University
  supercomputer, Cheaha, has assured us that AVDA will not be needed after the
uprising occures.

* **What about more specific questions?** Questions relating to AVDA not
covered in this FAQ may be sent to the AVDA team via awisner94@gmail.com.

## License

                    GNU GENERAL PUBLIC LICENSE
                       Version 3, 29 June 2007

 Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

                            Preamble

  The GNU General Public License is a free, copyleft license for
software and other kinds of works.

  The licenses for most software and other practical works are designed
to take away your freedom to share and change the works.  By contrast,
the GNU General Public License is intended to guarantee your freedom to
share and change all versions of a program--to make sure it remains free
software for all its users.  We, the Free Software Foundation, use the
GNU General Public License for most of our software; it applies also to
any other work released this way by its authors.  You can apply it to
your programs, too.

  When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
them if you wish), that you receive source code or can get it if you
want it, that you can change the software or use pieces of it in new
free programs, and that you know you can do these things.

  To protect your rights, we need to prevent others from denying you
these rights or asking you to surrender the rights.  Therefore, you have
certain responsibilities if you distribute copies of the software, or if
you modify it: responsibilities to respect the freedom of others.

  For example, if you distribute copies of such a program, whether
```

```
gratis or for a fee, you must pass on to the recipients the same
freedoms that you received.  You must make sure that they, too, receive
or can get the source code.  And you must show them these terms so they
know their rights.

  Developers that use the GNU GPL protect your rights with two steps:
(1) assert copyright on the software, and (2) offer you this License
giving you legal permission to copy, distribute and/or modify it.

  For the developers' and authors' protection, the GPL clearly explains
that there is no warranty for this free software.  For both users' and
authors' sake, the GPL requires that modified versions be marked as
changed, so that their problems will not be attributed erroneously to
authors of previous versions.

  Some devices are designed to deny users access to install or run
modified versions of the software inside them, although the manufacturer
can do so.  This is fundamentally incompatible with the aim of
protecting users' freedom to change the software.  The systematic
pattern of such abuse occurs in the area of products for individuals to
use, which is precisely where it is most unacceptable.  Therefore, we
have designed this version of the GPL to prohibit the practice for those
products.  If such problems arise substantially in other domains, we
stand ready to extend this provision to those domains in future versions
of the GPL, as needed to protect the freedom of users.

  Finally, every program is threatened constantly by software patents.
States should not allow patents to restrict development and use of
software on general-purpose computers, but in those that do, we wish to
avoid the special danger that patents applied to a free program could
make it effectively proprietary.  To prevent this, the GPL assures that
patents cannot be used to render the program non-free.

  The precise terms and conditions for copying, distribution and
modification follow.

                       TERMS AND CONDITIONS

  0. Definitions.

  "This License" refers to version 3 of the GNU General Public License.

  "Copyright" also means copyright-like laws that apply to other kinds of
works, such as semiconductor masks.

  "The Program" refers to any copyrightable work licensed under this
License.  Each licensee is addressed as "you".  "Licensees" and
"recipients" may be individuals or organizations.

  To "modify" a work means to copy from or adapt all or part of the work
in a fashion requiring copyright permission, other than the making of an
exact copy.  The resulting work is called a "modified version" of the
earlier work or a work "based on" the earlier work.

  A "covered work" means either the unmodified Program or a work based
on the Program.

  To "propagate" a work means to do anything with it that, without
permission, would make you directly or secondarily liable for
infringement under applicable copyright law, except executing it on a
computer or modifying a private copy.  Propagation includes copying,
distribution (with or without modification), making available to the
public, and in some countries other activities as well.

  To "convey" a work means any kind of propagation that enables other
parties to make or receive copies.  Mere interaction with a user through
a computer network, with no transfer of a copy, is not conveying.

  An interactive user interface displays "Appropriate Legal Notices"
to the extent that it includes a convenient and prominently visible
feature that (1) displays an appropriate copyright notice, and (2)
tells the user that there is no warranty for the work (except to the
extent that warranties are provided), that licensees may convey the
work under this License, and how to view a copy of this License.  If
the interface presents a list of user commands or options, such as a
menu, a prominent item in the list meets this criterion.

  1. Source Code.

  The "source code" for a work means the preferred form of the work
for making modifications to it.  "Object code" means any non-source
form of a work.

  A "Standard Interface" means an interface that either is an official
standard defined by a recognized standards body, or, in the case of
interfaces specified for a particular programming language, one that
is widely used among developers working in that language.
```

```

  The "System Libraries" of an executable work include anything, other
than the work as a whole, that (a) is included in the normal form of
packaging a Major Component, but which is not part of that Major
Component, and (b) serves only to enable use of the work with that
Major Component, or to implement a Standard Interface for which an
implementation is available to the public in source code form.  A
"Major Component", in this context, means a major essential component
(kernel, window system, and so on) of the specific operating system
(if any) on which the executable work runs, or a compiler used to
produce the work, or an object code interpreter used to run it.

  The "Corresponding Source" for a work in object code form means all
the source code needed to generate, install, and (for an executable
work) run the object code and to modify the work, including scripts to
control those activities.  However, it does not include the work's
System Libraries, or general-purpose tools or generally available free
programs which are used unmodified in performing those activities but
which are not part of the work.  For example, Corresponding Source
includes interface definition files associated with source files for
the work, and the source code for shared libraries and dynamically
linked subprograms that the work is specifically designed to require,
such as by intimate data communication or control flow between those
subprograms and other parts of the work.

  The Corresponding Source need not include anything that users
can regenerate automatically from other parts of the Corresponding
Source.

  The Corresponding Source for a work in source code form is that
same work.

  2. Basic Permissions.

  All rights granted under this License are granted for the term of
copyright on the Program, and are irrevocable provided the stated
conditions are met.  This License explicitly affirms your unlimited
permission to run the unmodified Program.  The output from running a
covered work is covered by this License only if the output, given its
content, constitutes a covered work.  This License acknowledges your
rights of fair use or other equivalent, as provided by copyright law.

  You may make, run and propagate covered works that you do not
convey, without conditions so long as your license otherwise remains
in force.  You may convey covered works to others for the sole purpose
of having them make modifications exclusively for you, or provide you
with facilities for running those works, provided that you comply with
the terms of this License in conveying all material for which you do
not control copyright.  Those thus making or running the covered works
for you must do so exclusively on your behalf, under your direction
and control, on terms that prohibit them from making any copies of
your copyrighted material outside their relationship with you.

  Conveying under any other circumstances is permitted solely under
the conditions stated below.  Sublicensing is not allowed; section 10
makes it unnecessary.

  3. Protecting Users' Legal Rights From Anti-Circumvention Law.

  No covered work shall be deemed part of an effective technological
measure under any applicable law fulfilling obligations under article
11 of the WIPO copyright treaty adopted on 20 December 1996, or
similar laws prohibiting or restricting circumvention of such
measures.

  When you convey a covered work, you waive any legal power to forbid
circumvention of technological measures to the extent such circumvention
is effected by exercising rights under this License with respect to
the covered work, and you disclaim any intention to limit operation or
modification of the work as a means of enforcing, against the work's
users, your or third parties' legal rights to forbid circumvention of
technological measures.

  4. Conveying Verbatim Copies.

  You may convey verbatim copies of the Program's source code as you
receive it, in any medium, provided that you conspicuously and
appropriately publish on each copy an appropriate copyright notice;
keep intact all notices stating that this License and any
non-permissive terms added in accord with section 7 apply to the code;
keep intact all notices of the absence of any warranty; and give all
recipients a copy of this License along with the Program.

  You may charge any price or no price for each copy that you convey,
and you may offer support or warranty protection for a fee.

  5. Conveying Modified Source Versions.
```

```

  A "User Product" is either (1) a "consumer product", which means any
tangible personal property which is normally used for personal, family,
or household purposes, or (2) anything designed or sold for incorporation
into a dwelling.  In determining whether a product is a consumer product,
doubtful cases shall be resolved in favor of coverage.  For a particular
product received by a particular user, "normally used" refers to a
typical or common use of that class of product, regardless of the status
of the particular user or of the way in which the particular user
actually uses, or expects or is expected to use, the product.  A product
is a consumer product regardless of whether the product has substantial
commercial, industrial or non-consumer uses, unless such uses represent
the only significant mode of use of the product.

  "Installation Information" for a User Product means any methods,
procedures, authorization keys, or other information required to install
and execute modified versions of a covered work in that User Product from
a modified version of its Corresponding Source.  The information must
suffice to ensure that the continued functioning of the modified object
code is in no case prevented or interfered with solely because
modification has been made.

  If you convey an object code work under this section in, or with, or
specifically for use in, a User Product, and the conveying occurs as
part of a transaction in which the right of possession and use of the
User Product is transferred to the recipient in perpetuity or for a
fixed term (regardless of how the transaction is characterized), the
Corresponding Source conveyed under this section must be accompanied
by the Installation Information.  But this requirement does not apply
if neither you nor any third party retains the ability to install
modified object code on the User Product (for example, the work has
been installed in ROM).

  The requirement to provide Installation Information does not include a
requirement to continue to provide support service, warranty, or updates
for a work that has been modified or installed by the recipient, or for
the User Product in which it has been modified or installed.  Access to a
network may be denied when the modification itself materially and
adversely affects the operation of the network or violates the rules and
protocols for communication across the network.

  Corresponding Source conveyed, and Installation Information provided,
in accord with this section must be in a format that is publicly
documented (and with an implementation available to the public in
source code form), and must require no special password or key for
unpacking, reading or copying.

  7. Additional Terms.

  "Additional permissions" are terms that supplement the terms of this
License by making exceptions from one or more of its conditions.
Additional permissions that are applicable to the entire Program shall
be treated as though they were included in this License, to the extent
that they are valid under applicable law.  If additional permissions
apply only to part of the Program, that part may be used separately
under those permissions, but the entire Program remains governed by
this License without regard to the additional permissions.

  When you convey a copy of a covered work, you may at your option
remove any additional permissions from that copy, or from any part of
it.  (Additional permissions may be written to require their own
removal in certain cases when you modify the work.)  You may place
additional permissions on material, added by you to a covered work,
for which you have or can give appropriate copyright permission.

  Notwithstanding any other provision of this License, for material you
add to a covered work, you may (if authorized by the copyright holders of
that material) supplement the terms of this License with terms:

    a) Disclaiming warranty or limiting liability differently from the
    terms of sections 15 and 16 of this License; or

    b) Requiring preservation of specified reasonable legal notices or
    author attributions in that material or in the Appropriate Legal
    Notices displayed by works containing it; or

    c) Prohibiting misrepresentation of the origin of that material, or
    requiring that modified versions of such material be marked in
    reasonable ways as different from the original version; or

    d) Limiting the use for publicity purposes of names of licensors or
    authors of the material; or

    e) Declining to grant rights under trademark law for use of some
    trade names, trademarks, or service marks; or

    f) Requiring indemnification of licensors and authors of that
    material by anyone who conveys the material (or modified versions of
    it) with contractual assumptions of liability to the recipient, for
    any liability that these contractual assumptions directly impose on
    those licensors and authors.

  All other non-permissive additional terms are considered "further
restrictions" within the meaning of section 10.  If the Program as you
received it, or any part of it, contains a notice stating that it is
governed by this License along with a term that is a further
restriction, you may remove that term.  If a license document contains
a further restriction but permits relicensing or conveying under this
License, you may add to a covered work material governed by the terms
of that license document, provided that the further restriction does
not survive such relicensing or conveying.

  If you add terms to a covered work in accord with this section, you
must place, in the relevant source files, a statement of the
additional terms that apply to those files, or a notice indicating
where to find the applicable terms.

  Additional terms, permissive or non-permissive, may be stated in the
form of a separately written license, or stated as exceptions;
the above requirements apply either way.

  8. Termination.

  You may not propagate or modify a covered work except as expressly
provided under this License.  Any attempt otherwise to propagate or
modify it is void, and will automatically terminate your rights under
this License (including any patent licenses granted under the third
paragraph of section 11).

  However, if you cease all violation of this License, then your
license from a particular copyright holder is reinstated (a)
provisionally, unless and until the copyright holder explicitly and
finally terminates your license, and (b) permanently, if the copyright
holder fails to notify you of the violation by some reasonable means
prior to 60 days after the cessation.

  Moreover, your license from a particular copyright holder is
reinstated permanently if the copyright holder notifies you of the
violation by some reasonable means, this is the first time you have
received notice of violation of this License (for any work) from that
copyright holder, and you cure the violation prior to 30 days after
your receipt of the notice.

  Termination of your rights under this section does not terminate the
licenses of parties who have received copies or rights from you under
this License.  If your rights have been terminated and not permanently
reinstated, you do not qualify to receive new licenses for the same
material under section 10.

  9. Acceptance Not Required for Having Copies.

  You are not required to accept this License in order to receive or
run a copy of the Program.  Ancillary propagation of a covered work
occurring solely as a consequence of using peer-to-peer transmission
to receive a copy likewise does not require acceptance.  However,
nothing other than this License grants you permission to propagate or
modify any covered work.  These actions infringe copyright if you do
not accept this License.  Therefore, by modifying or propagating a
covered work, you indicate your acceptance of this License to do so.

  10. Automatic Licensing of Downstream Recipients.

  Each time you convey a covered work, the recipient automatically
receives a license from the original licensors, to run, modify and
propagate that work, subject to this License.  You are not responsible
for enforcing compliance by third parties with this License.

  An "entity transaction" is a transaction transferring control of an
organization, or substantially all assets of one, or subdividing an
organization, or merging organizations.  If propagation of a covered
work results from an entity transaction, each party to that
transaction who receives a copy of the work also receives whatever
licenses to the work the party's predecessor in interest had or could
give under the previous paragraph, plus a right to possession of the
Corresponding Source of the work from the predecessor in interest, if
the predecessor has it or can get it with reasonable efforts.

  You may not impose any further restrictions on the exercise of the
rights granted or affirmed under this License.  For example, you may
not impose a license fee, royalty, or other charge for exercise of
rights granted under this License, and you may not initiate litigation
(including a cross-claim or counterclaim in a lawsuit) alleging that
any patent claim is infringed by making, using, selling, offering for
sale, or importing the Program or any portion of it.
```

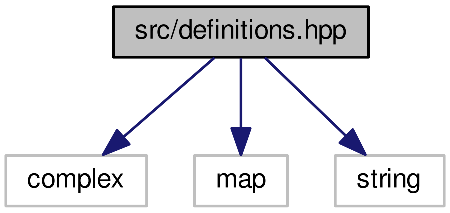
```
combined work, and to convey the resulting work.  The terms of this
License will continue to apply to the part which is the covered work,
but the special requirements of the GNU Affero General Public License,
section 13, concerning interaction through a network will apply to the
combination as such.

  14. Revised Versions of this License.

  The Free Software Foundation may publish revised and/or new versions of
the GNU General Public License from time to time.  Such new versions will
be similar in spirit to the present version, but may differ in detail to
address new problems or concerns.

  Each version is given a distinguishing version number.  If the
Program specifies that a certain numbered version of the GNU General
Public License "or any later version" applies to it, you have the
option of following the terms and conditions either of that numbered
version or of any later version published by the Free Software
Foundation.  If the Program does not specify a version number of the
GNU General Public License, you may choose any version ever published
by the Free Software Foundation.

  If the Program specifies that a proxy can decide which future
versions of the GNU General Public License can be used, that proxy's
public statement of acceptance of a version permanently authorizes you
to choose that version for the Program.

  Later license versions may give you additional or different
permissions.  However, no additional obligations are imposed on any
author or copyright holder as a result of your choosing to follow a
later version.

  15. Disclaimer of Warranty.

  THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY
APPLICABLE LAW.  EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT
HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY
OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO,
THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE.  THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM
IS WITH YOU.  SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF
ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

  16. Limitation of Liability.

  IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS
THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY
GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE
USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF
DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD
PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS),
EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF
SUCH DAMAGES.

  17. Interpretation of Sections 15 and 16.

  If the disclaimer of warranty and limitation of liability provided
above cannot be given local legal effect according to their terms,
reviewing courts shall apply local law that most closely approximates
an absolute waiver of all civil liability in connection with the
Program, unless a warranty or assumption of liability accompanies a
copy of the Program in return for a fee.

                     END OF TERMS AND CONDITIONS

            How to Apply These Terms to Your New Programs

  If you develop a new program, and you want it to be of the greatest
possible use to the public, the best way to achieve this is to make it
free software which everyone can redistribute and change under these terms.

  To do so, attach the following notices to the program.  It is safest
to attach them to the start of each source file to most effectively
state the exclusion of warranty; and each file should have at least
the "copyright" line and a pointer to where the full notice is found.

    {one line to give the program's name and a brief idea of what it does.}
    Copyright (C) {year}  {name of author}

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

  If the program does terminal interaction, make it output a short
notice like this when it starts in an interactive mode:

    {project}  Copyright (C) {year}  {fullname}
    This program comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate
parts of the General Public License.  Of course, your program's commands
might be different; for a GUI interface, you would use an "about box".

  You should also get your employer (if you work as a programmer) or school,
if any, to sign a "copyright disclaimer" for the program, if necessary.
For more information on this, and how to apply and follow the GNU GPL, see
<http://www.gnu.org/licenses/>.

  The GNU General Public License does not permit incorporating your program
into proprietary programs.  If your program is a subroutine library, you
may consider it more useful to permit linking proprietary applications with
the library.  If this is what you want to do, use the GNU Lesser General
Public License instead of this License.  But first, please read
<http://www.gnu.org/philosophy/why-not-lgpl.html>.

```

8.7 src/definitions.hpp File Reference

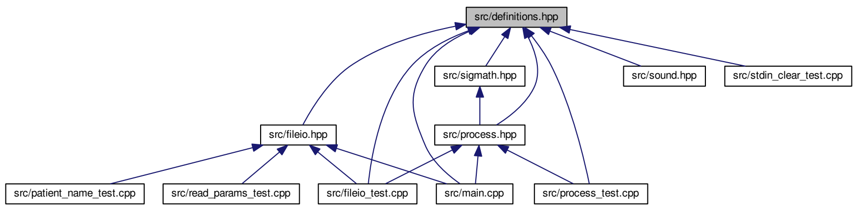
Contains declarations of system-independant (universal size) integers and float types, shortened type names for some commonly used types, and enumerations.

```
#include <complex>
#include <map>
#include <string>
```

Include dependency graph for definitions.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct DataParams
- struct Maximum

Namespaces

- avda

Macros

- #define ENUM signed char

Typedefs

- typedef unsigned char byte
- typedef unsigned char uint8
- typedef signed char sint8
- typedef unsigned short uint16
- typedef signed short sint16
- typedef unsigned int uint32
- typedef signed int sint32
- typedef unsigned long long uint64
- typedef signed long long sint64
- typedef float float32
- typedef double float64
- typedef std::complex< float32 > cfloat32

Enumerations

- enum avda::Side { avda::Side::Left, avda::Side::Right }

Variables

- const uint16 DET_THRESH = 5000
- const uint8 DURATION = 6
- const sint8 ERROR = -1
- const uint16 MAX_DROP_FREQ = 7000
- const uint8 REC_COUNT = 6

- const uint32 SAMPLE_COUNT = 131072
- const uint16 SAMPLE_FREQ = 24000
- const std::string TEMP_FILE = ".temp"
- const uint32 BUFFER_SIZE = SAMPLE_COUNT ∗ sizeof(float32)

8.7.1 Detailed Description

Contains declarations of system-independant (universal size) integers and float types, shortened type names for some commonly used types, and enumerations.

Author

Samuel Andrew Wisner, awisner94@gmail.com

Definition in file definitions.hpp.

8.7.2 Macro Definition Documentation

8.7.2.1 #define ENUM signed char

Definition at line 16 of file definitions.hpp.

8.7.3 Typedef Documentation

8.7.3.1 typedef unsigned char byte

Definition at line 20 of file definitions.hpp.

8.7.3.2 typedef std::complex<float32> cfloat32

Complex float32's.

Definition at line 92 of file definitions.hpp.

8.7.3.3 typedef float float32

Definition at line 33 of file definitions.hpp.

8.7.3.4 typedef double float64

Definition at line 34 of file definitions.hpp.

8.7.3.5 typedef signed short sint16

Definition at line 25 of file definitions.hpp.

8.7.3.6 typedef signed int sint32

Definition at line 28 of file definitions.hpp.

8.7.3.7 typedef signed long long sint64

Definition at line 31 of file definitions.hpp.

8.7.3.8 typedef signed char sint8

Definition at line 22 of file definitions.hpp.

8.7.3.9 typedef unsigned short uint16

Definition at line 24 of file definitions.hpp.

8.7.3.10 typedef unsigned int uint32

Definition at line 27 of file definitions.hpp.

8.7.3.11 typedef unsigned long long uint64

Definition at line 30 of file definitions.hpp.

8.7.3.12 typedef unsigned char uint8

Definition at line 21 of file definitions.hpp.

8.7.4 Variable Documentation

8.7.4.1 const uint32 BUFFER_SIZE = SAMPLE_COUNT ∗ sizeof(float32)

Size of the sample buffer.

Definition at line 84 of file definitions.hpp.

8.7.4.2 const uint16 DET_THRESH = 5000

Threshold for the differential-parameters product to be considered indicative of a vasospasm.

Definition at line 43 of file definitions.hpp.

8.7.4.3 const uint8 DURATION = 6

Duration of recording in seconds.

Definition at line 48 of file definitions.hpp.

8.7.4.4 const sint8 ERROR = -1

Error integer returned when the program must exit with an error.

Definition at line 53 of file definitions.hpp.

8.7.4.5 const uint16 MAX_DROP_FREQ = 7000

Maximum drop-off frequency considered valid.

Definition at line 58 of file definitions.hpp.

8.7.4.6 const uint8 REC_COUNT = 6

Number of recordings (both left and right) to make.

Definition at line 63 of file definitions.hpp.

8.7.4.7 const uint32 SAMPLE_COUNT = 131072

Number of samples to use in processing the recordings. Must be a power of two. SAMPLE_COUNT / SAMPLE_↩ FREQ < DURATION must be true.

Definition at line 69 of file definitions.hpp.

8.7.4.8 const uint16 SAMPLE_FREQ = 24000

Recording sampling rate in Hz (NOT kHz).

Definition at line 74 of file definitions.hpp.

8.7.4.9 const std::string TEMP_FILE = ".temp"

Filename of the temporary recording file.

Definition at line 79 of file definitions.hpp.

8.8 definitions.hpp

```
00001
00009 #ifndef definitions_H
00010 #define definitions_H
00011
00012 #include <complex>
00013 #include <map>
00014 #include <string>
00015
00016 #define ENUM signed char
00017
00018 // Type definitions
00019
00020 typedef unsigned char byte;
00021 typedef unsigned char uint8;
00022 typedef signed char sint8;
00023
00024 typedef unsigned short uint16;
00025 typedef signed short sint16;
00026
00027 typedef unsigned int uint32;
00028 typedef signed int sint32;
00029
00030 typedef unsigned long long uint64;
00031 typedef signed long long sint64;
00032
00033 typedef float float32;
00034 typedef double float64;
00035
00036
00037 // Constants
00038
00043 const uint16 DET_THRESH = 5000;
00044
00048 const uint8 DURATION = 6;
00049
```

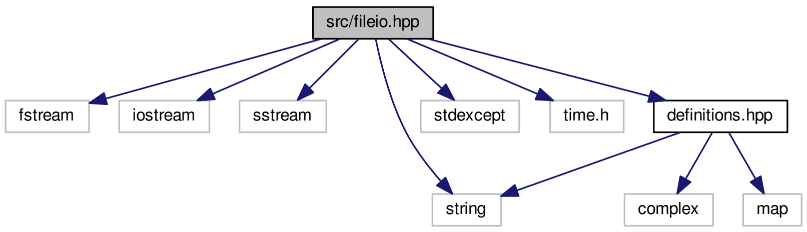
```
const sint8 ERROR = -1;

const uint16 MAX_DROP_FREQ = 7000;

const uint8 REC_COUNT = 6;

const uint32 SAMPLE_COUNT = 131072;//262144;

const uint16 SAMPLE_FREQ = 24000;

const std::string TEMP_FILE = ".temp";

const uint32 BUFFER_SIZE = SAMPLE_COUNT * sizeof(float32);


// Objective/structural type definitions

typedef std::complex<float32> cfloat32;

typedef struct {
    float32 freq = 0;
    float32 noise = 0;
} DataParams;

typedef struct {
    float32 value = 0;
    uint32 index = 0;
} Maximum;


// Enumerations

namespace avda {
    enum class Side { Left, Right };
}


// Doxygen documentation for other files.

#endif
```

8.9 src/fileio.hpp File Reference

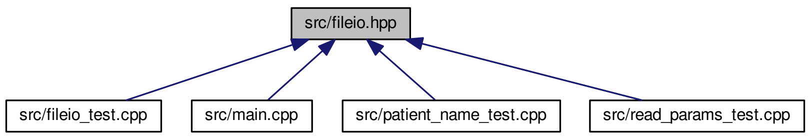
contains functions related to the file I/O use in this program

```
#include <fstream>
#include <iostream>
#include <sstream>
#include <string>
#include <stdexcept>
#include <time.h>
#include "definitions.hpp"
```

Include dependency graph for fileio.hpp:



This graph shows which files directly or indirectly include this file:



Namespaces

- avda

Functions

- std::string avda::PatientName ()
- std::map< Side, DataParams > avda::ReadParams (auto filename)
- void avda::WriteParams (std::map< Side, DataParams > myMap, auto filename)

Variables

- const std::string avda::CSV_HEADER = "Time,Side,Frequency,Noise Level"
- const std::string avda::PATIENT_PATH = "/home/pi/patients/"

8.9.1 Detailed Description

contains functions related to the file I/O use in this program

Author

Samuel Andrew Wisner, awisner94@gmail.com
Nicholas K. Nolan

Bug file is overly complicated and much more bug-prone

Definition in file fileio.hpp.

8.10 fileio.hpp

```
00001
00009 #ifndef fileio_H
00010 #define fileio_H
00011
00012 #include <fstream>
00013 #include <iostream>
00014 #include <sstream>
00015 #include <string>
00016 #include <stdexcept>
00017 #include <time.h>
00018
00019 #include "definitions.hpp"
00020
00021 namespace avda {
00025     const std::string CSV_HEADER = "Time,Side,Frequency,Noise Level";
00026
```

```
const std::string PATIENT_PATH = "/home/pi/patients/";

std::string PatientName() {
    std::string fname = "";
    std::string mname = "";
    std::string lname = "";
    std::string patfil = "";
    std::string patientname = "";
    uint32 track1 = 0;
    uint32 track2 = 0;
    uint32 track3 = 0;

    do {
        std::cout << "Please enter the patients name." << std::endl;
        std::cout << "First name: ";
        std::cin >> fname;
        std::cout << "Middle name: ";
        std::cin >> mname;
        std::cout << "Last name: ";
        std::cin >> lname;

        // creates new std::string with path to patient file
        patientname = PATIENT_PATH + lname + ", " + fname
            + " " + mname + ".csv";

        // prints out patientname. shows user the path to the patient file
        //std::cout << patientname << std::endl << std::endl;
        std::ifstream file(patientname.c_str());

        if (file.good()) {
            track1 = 1;
        }

        /*
         * Compares patientname to existing files and lets user know
         * if the file does not exist.
         */
        else if (!file.good()) {
            /*
             * Do while statement to continue asking user about the file
             * if their input is not acceptable
             */
            do {
                std::cout << "Patient file does not exist, would you like "
                    "to create file or re-enter their name?" << std::endl;
                std::cout << "  *Type 'create' and press enter key "
                    "to create the patient file." << std::endl;
                std::cout << "  *Type 'reenter' and press enter key "
                    "to re-enter the patients name." << std::endl;
                std::cout << std::endl;
                std::cin >> patfil;

                /*
                 * patfil equals create, track1 and 2 will increase
                 * escaping both do while loops
                 */
                if(patfil == "create") {
                    std::ofstream createfile(patientname.c_str());
                    track1 = 1;
                    track2 = 1;
                    track3 = 1;
                    createfile << CSV_HEADER << std::endl;
                    createfile.flush();
                    createfile.close();
                }

                /*
                 *patfil equals renter, track1 will remain zero allowing
                 *user to reenter the patient name.
                 */
                else if(patfil == "reenter") {
                    track1 = 0;
                    track2 = 1;
                }

                /*
                 *The users input was neither create or reenter. User
                 *must enter patient name again.
                 */
                else {
                    std::cout << std::endl;
                    std::cout << "Your input is not acceptable." << std::endl;
                    std::cout << std::endl;
                }
            }while(track2 == 0);
        }
    } while (track1 == 0);
```

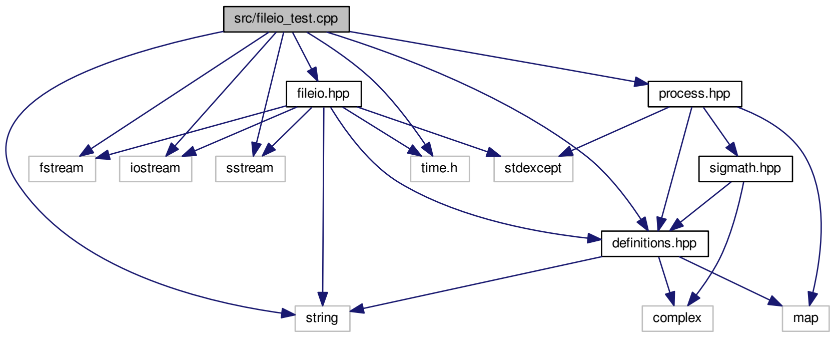
```

        return patientname; //returns the path to the patient file
    }

    std::map<Side, DataParams> ReadParams(auto filename) {
        std::map<Side, DataParams> myMap;
        DataParams leftparams;
        DataParams rightparams;

        std::ifstream file(filename.c_str());
        std::string leftline;
        std::string rightline;
        std::string leftsearch = "Left";
        std::string rightsearch = "Right";
        std::string paramstring;
        std::string lfreqstr;
        std::string lnoisestr;
        std::string rfreqstr;
        std::string rnoisestr;
        uint32 lcnt = 0;
        uint32 rcnt = 0;
        float32 lfreqval;
        float32 lnoiseval;
        float32 rfreqval;
        float32 rnoiseval;

        /*
         * if statement which uses ifstream function to open patient file
         * filename)
         */
        if(file.is_open()) {
            /*
             * While statement to find the first Left line and save to
             *leftline as string.
             */
            while (getline(file, leftline)) {
                if(leftline.find(leftsearch, 0) != std::string::npos) {
                    break;
                }

            }

            /*
             * While statement to find first right line and save to rightline
             * as string.
             */
            while (getline(file,rightline)) {
                if(rightline.find(rightsearch, 0) != std::string::npos) {
                    break;
                }
            }

            // Code to break leftline and rightline into its parts
            std::stringstream lss(leftline);
            std::stringstream rss(rightline);

            while(getline(lss,paramstring, ',')) {
                lcnt++;

                if(lcnt == 3) {
                    lfreqstr = paramstring;
                }

                else if(lcnt == 4) {
                    lnoisestr = paramstring;
                }
            }

            while(getline(rss,paramstring, ',')) {
                rcnt++;

                if(rcnt == 3) {
                    rfreqstr = paramstring;
                }

                else if(rcnt == 4) {
                    rnoisestr = paramstring;
                }
            }

            /*
             * Statement to convert lfreq, lnoise, rfreq, and rnoise from
             * strings to floats.
             * */
            lfreqval = atof(lfreqstr.c_str());
            lnoiseval = atof(lnoisestr.c_str());
            rfreqval = atof(rfreqstr.c_str());
            rnoiseval = atof(rnoisestr.c_str());

            file.close();
        }

        else {
            throw std::runtime_error("The patient file could not be opened.");
        }

        leftparams.freq = lfreqval;
        leftparams.noise = lnoiseval;
        rightparams.freq = rfreqval;
        rightparams.noise = rnoiseval;

        myMap[Side::Left] = leftparams;
        myMap[Side::Right] = rightparams;

        return myMap;
    }

    void WriteParams(std::map<Side, DataParams> myMap, auto filename) {
        char temp[80];
        std::ofstream file(filename.c_str(),
                std::ofstream::out | std::ofstream::app);

        //Gives pointer measurementtime a data type of time_t.
        time_t measurementtime;
        time(&measurementtime); //Gets the current time.
        strftime(temp, 80, "%c", localtime(&measurementtime));
        std::string fTime = std::string(temp);

        //if statement to print the Left side parameters to the patient file.
        if(file.is_open()) {
            file << fTime + "," + "Left" + ","
                + std::to_string(myMap[Side::Left].freq)
                + ", " + std::to_string(myMap[Side::Left].noise) << std::endl;
        }

        //if statement to print the Right side parameters to the patient file.
        if(file.is_open()) {
            file << fTime + "," + "Right" + ","
                + std::to_string(myMap[Side::Right].freq)
                + ", " + std::to_string(myMap[Side::Right].noise) << std::endl;
        }

        else {
            std::cout << "Patient file can not be opened!" << std::endl;
        }

        file.close();
    }
}

#endif
```

8.11 src/fileio_test.cpp File Reference

Contains program that tests the functions in fileio.hpp.

```
#include <fstream>
#include <iostream>
#include <sstream>
#include <string>
#include <time.h>
#include "definitions.hpp"
#include "fileio.hpp"
#include "process.hpp"
```

Include dependency graph for fileio_test.cpp:



Functions

- int main ()

8.11.1 Detailed Description

Contains program that tests the functions in fileio.hpp.

Author

Samuel Andrew Wisner
Nicholas K. Nolan

Definition in file fileio_test.cpp.

8.11.2 Function Documentation

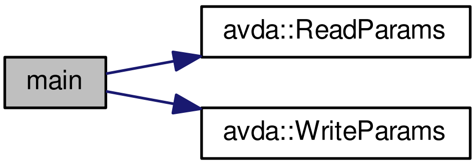
8.11.2.1 int main ( )

Tests the functions in fileio.hpp.

Definition at line 24 of file fileio_test.cpp.

```
00024             {
00025     string path = PATIENT_PATH + "wizmack, sammy andy.csv";
00026     map<Side, DataParams> laMap = ReadParams(path);
00027     cout <<  laMap[Side::Right].freq << endl;
00028     cout << laMap[Side::Right].noise << endl;
00029 
00030     WriteParams(laMap, path);
00031 }
```

Here is the call graph for this function:



8.12 fileio_test.cpp

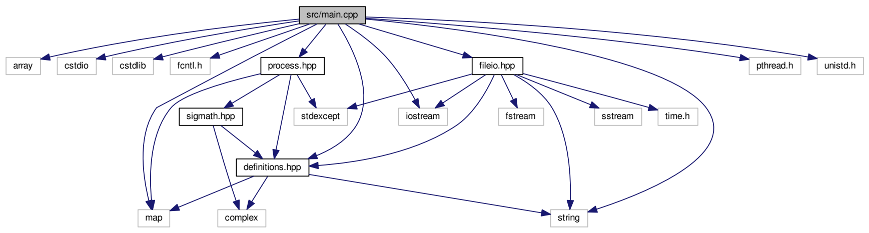
```
#include <fstream>
#include <iostream>
#include <sstream>
#include <string>
#include <time.h>

#include "definitions.hpp"
#include "fileio.hpp"
#include "process.hpp"

using namespace std;
using namespace avda;

int main() {
    string path = PATIENT_PATH + "wizmack, sammy andy.csv";
    map<Side, DataParams> laMap = ReadParams(path);
    cout <<  laMap[Side::Right].freq << endl;
    cout << laMap[Side::Right].noise << endl;

    WriteParams(laMap, path);
}
```

8.13 src/main.cpp File Reference

Contains the main program.

```
#include <array>
#include <cstdio>
#include <cstdlib>
#include <fcntl.h>
#include <iostream>
#include <map>
#include <pthread.h>
#include <string>
#include <unistd.h>
#include "definitions.hpp"
#include "fileio.hpp"
#include "process.hpp"
```

Include dependency graph for main.cpp:



Functions

- int main (int argc, char ∗∗argv)

8.13.1 Detailed Description

Contains the main program.

Author

Samuel Andrew Wisner, awisner94@gmail.com
Nicholas K. Nolan

Bug extra newline character inserted into stdin buffer after PatientName() is run

Definition in file main.cpp.

8.13.2 Function Documentation

8.13.2.1 int main (int *argc,* char ∗∗ *argv*)

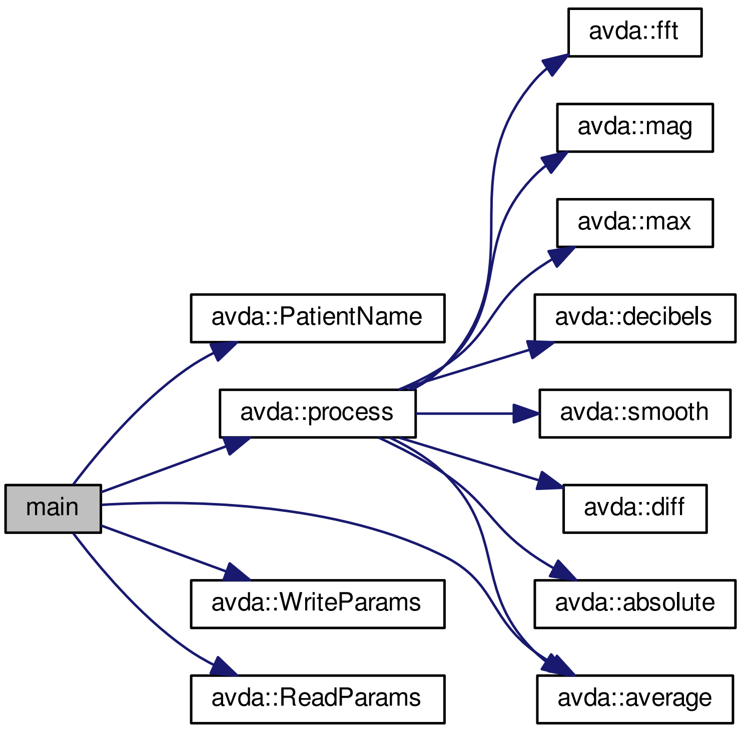
The main program for this project. It will detect avdaspasms over a period of days.

Definition at line 31 of file main.cpp.

```
00031                                    {
00032     // Recorded audio buffer
00033     float32* buffer = (float32*)std::malloc(BUFFER_SIZE);
00034     bool cont = true;   // whether to continue in the recording loop
00035     DataParams params[REC_COUNT];   // holds DataParam's from recordings
00036     string filename = PatientName();   // generate name for patient's file
00037     map<Side, DataParams> results;   // parameters by side
00038 
00039     // arecord command
00040     const string recCommand = string("arecord -t raw -d ")
00041         + to_string(DURATION) + string(" -D plughw:1,0 -f FLOAT -q -r ")
00042         + to_string(SAMPLE_FREQ) + string(" ") + TEMP_FILE;
00043 
00044     // Recording
00045     while(cont) {
00046         for(uint8 i = 0; i < REC_COUNT; i++) {
00047             // prompt
00048             cout << "Press [ ENTER ] to begin analysis for the "
00049                 << (i < REC_COUNT / 2 ? "left" : "right") << " side, depth #"
00050                 << (((i >= REC_COUNT / 2) ? (i - REC_COUNT / 2) : i) + 1)
00051                 << " ";
00052             getchar();  // wait for ENTER to be pressed
00053             cout << "Analyzing..." << endl;
00054 
00055             system(recCommand.c_str());
00056             usleep(DURATION*1000000 + 1500000);   // sleep DURATION + 1.5 seconds
```

```

            int file = open(TEMP_FILE.c_str(), O_RDONLY);  // open temp file
            int retRead = read(file, buffer, BUFFER_SIZE);  // copy to buffer
            close(file);  // close temp file
            remove(TEMP_FILE.c_str());  // delete temp file

            // if something goes wrong reading the temp file, program exits
            if(file < 0 || retRead < BUFFER_SIZE) {
                cerr << "An error occurred reading the doppler audio! "
                    "The program will now exit." << endl;
                return ERROR;
            }

            // process and store parameters
            params[i] = process(buffer, SAMPLE_COUNT,
SAMPLE_FREQ);
            cout << "The analysis is complete." << endl << endl;
        }

        // calculate averaged parameters
        results[Side::Left] = average(params, REC_COUNT / 2);
        results[Side::Right] = average(&params[REC_COUNT / 2], REC_COUNT / 2);

        cout << "Analysis is complete." << endl << endl;

        // print averaged side analysis
        for(int i = 0; i < 2; i++) {
            Side side = (Side)i;
            cout << (side == Side::Left ? "[LEFT]" : "[RIGHT]") << endl;
            cout << "Drop-off frequency: " << (uint16)(results[side].freq + 0.5)
                << " Hz" << endl;
            cout << "Average relative noiseband power: "
                << (sint16)(results[side].noise - 0.5) << " dB" << endl <<endl;
        }

        cont = results[Side::Left].freq > MAX_DROP_FREQ
            || results[Side::Right].freq > MAX_DROP_FREQ;

        if(cont) {
            cout << "An error in aquisition of the doppler audio has occurred! "
                "Ensure the connection from the doppler machine to this device "
                "is secure and the connection uninterruptable." << endl << endl;
        }
    }

    free(buffer);  // free buffer to prevent memory leak
    WriteParams(results, filename);

    // examine likelihood of avdaspasm
    try {
        map<Side, DataParams> baseParams = ReadParams(filename);
        map<Side, bool> comparison;

        for(uint8 i = 0; i < 2; i++) {
            Side side = (Side)i;
            float comp = (results[side].freq - baseParams[side].freq)
                * (baseParams[side].noise - results[side].noise);
            comparison[side] = comp > DET_THRESH;
        }

        string which;

        if(comparison[Side::Left] && !comparison[Side::Right]) {
            which = "The left";
        } else if(!comparison[Side::Left] && comparison[Side::Right]) {
            which = "The right";
        } else if (comparison[Side::Left] && comparison[Side::Right]) {
            which = "Both";
        } else {
            which = "Neither";
        }

        cout << which << " side seems to show evidence of a vasospasm." << endl;
    } catch(runtime_error ex) {
        cout << "These values will be stored as the baseline parameters to "
            "which all future parameters are compared." << endl;
    }
}
```

Here is the call graph for this function:



8.14 main.cpp

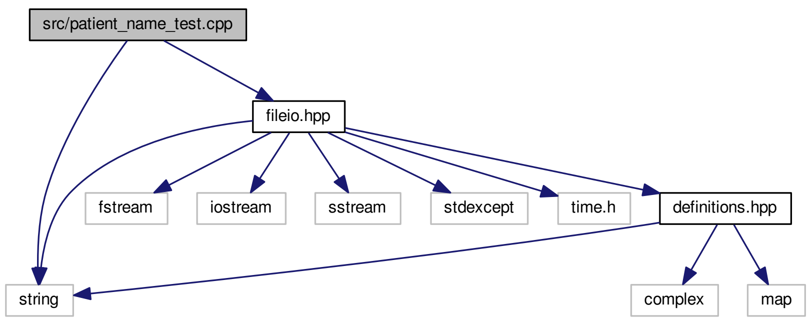
```
#include <array>
#include <cstdio>
#include <cstdlib>
#include <fcntl.h>
#include <iostream>
#include <map>
#include <pthread.h>
#include <string>
#include <unistd.h>

#include "definitions.hpp"
#include "fileio.hpp"
#include "process.hpp"

using namespace std;
using namespace avda;

int main(int argc, char** argv) {
    // Recorded audio buffer
    float32* buffer = (float32*)std::malloc(BUFFER_SIZE);
    bool cont = true;  // whether to continue in the recording loop
    DataParams params[REC_COUNT];  // holds DataParam's from recordings
    string filename = PatientName();  // generate name for patient's file
    map<Side, DataParams> results;  // parameters by side

    // arecord command
    const string recCommand = string("arecord -t raw -d ")
        + to_string(DURATION) + string(" -D plughw:1,0 -f FLOAT -q -r ")
        + to_string(SAMPLE_FREQ) + string(" ") + TEMP_FILE;

    // Recording
    while(cont) {
        for(uint8 i = 0; i < REC_COUNT; i++) {
            // prompt
            cout << "Press [ ENTER ] to begin analysis for the "
                << (i < REC_COUNT / 2 ? "left" : "right") << " side, depth #"
                << (((i >= REC_COUNT / 2) ? (i - REC_COUNT / 2) : i) + 1)
                << " ";
            getchar();  // wait for ENTER to be pressed
            cout << "Analyzing..." << endl;

            system(recCommand.c_str());
            usleep(DURATION*1000000 + 1500000);  // sleep DURATION + 1.5 seconds

            int file = open(TEMP_FILE.c_str(), O_RDONLY);  // open temp file
            int retRead = read(file, buffer, BUFFER_SIZE);  // copy to buffer
            close(file);  // close temp file
            remove(TEMP_FILE.c_str());  // delete temp file

            // if something goes wrong reading the temp file, program exits
            if(file < 0 || retRead < BUFFER_SIZE) {
                cerr << "An error occurred reading the doppler audio! "
                    "The program will now exit." << endl;
                return ERROR;
            }

            // process and store parameters
            params[i] = process(buffer, SAMPLE_COUNT,
      SAMPLE_FREQ);
            cout << "The analysis is complete." << endl << endl;
        }

        // calculate averaged parameters
        results[Side::Left] = average(params, REC_COUNT / 2);
        results[Side::Right] = average(&params[REC_COUNT / 2], REC_COUNT / 2);

        cout << "Analysis is complete." << endl << endl;

        // print averaged side analysis
        for(int i = 0; i < 2; i++) {
            Side side = (Side)i;
            cout << (side == Side::Left ? "[LEFT]" : "[RIGHT]") << endl;
            cout << "Drop-off frequency: " << (uint16)(results[side].freq + 0.5)
                << " Hz" << endl;
            cout << "Average relative noiseband power: "
                << (sint16)(results[side].noise - 0.5) << " dB" << endl <<endl;
        }

        cont = results[Side::Left].freq > MAX_DROP_FREQ
            || results[Side::Right].freq > MAX_DROP_FREQ;

        if(cont) {
            cout << "An error in aquisition of the doppler audio has occurred! "
                "Ensure the connection from the doppler machine to this device "
                "is secure and the connection uninterruptable." << endl << endl;
        }
    }

    free(buffer);  // free buffer to prevent memory leak
    WriteParams(results, filename);

    // examine likelihood of avdaspasm
    try {
        map<Side, DataParams> baseParams = ReadParams(filename);
        map<Side, bool> comparison;

        for(uint8 i = 0; i < 2; i++) {
            Side side = (Side)i;
            float comp = (results[side].freq - baseParams[side].freq)
                * (baseParams[side].noise - results[side].noise);
            comparison[side] = comp > DET_THRESH;
        }

        string which;

        if(comparison[Side::Left] && !comparison[Side::Right]) {
            which = "The left";
        } else if(!comparison[Side::Left] && comparison[Side::Right]) {
            which = "The right";
        } else if (comparison[Side::Left] && comparison[Side::Right]) {
            which = "Both";
        } else {
            which = "Neither";
        }

        cout << which << " side seems to show evidence of a vasospasm." << endl;
    } catch(runtime_error ex) {
        cout << "These values will be stored as the baseline parameters to "
            "which all future parameters are compared." << endl;
    }
}
```

8.15 src/patient_name_test.cpp File Reference

Contains a program to test the PatientName() function.

```
#include <string>
#include "fileio.hpp"
```

Include dependency graph for patient_name_test.cpp:



Functions

- int main (int argc, char ∗∗argv)

8.15.1 Detailed Description

Contains a program to test the PatientName() function.

Author

Samuel Andrew Wisner, awisner94@gmail.com

Definition in file patient_name_test.cpp.

8.15.2 Function Documentation

8.15.2.1 int main (int *argc,* char ∗∗ *argv*)

Tests the PatientName() function from fileio.hpp.

Definition at line 17 of file patient_name_test.cpp.

```
00017                          {
00018     string filename = PatientName();
00019     cout << filename;
00020 }
```

Here is the call graph for this function:

main
avda::PatientName

8.16 patient_name_test.cpp

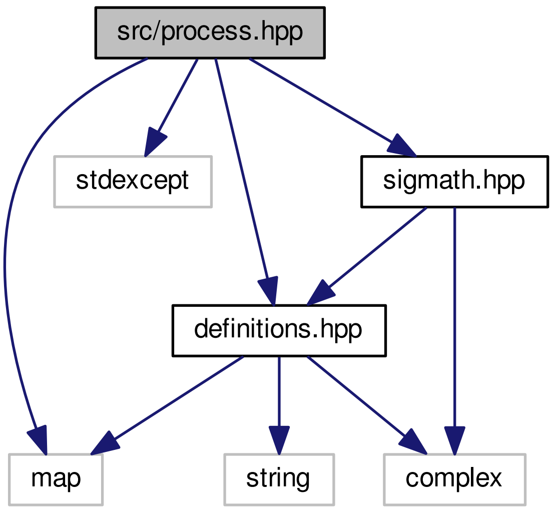
```

#include <string>

#include "fileio.hpp"

using namespace std;
using namespace avda;

int main(int argc, char** argv) {
    string filename = PatientName();
    cout << filename;
}
```

8.17 src/process.hpp File Reference

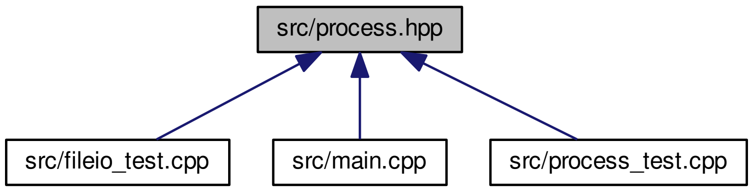
Contains functions related to the program's threaded processing of audio data.

```
#include <map>
#include <stdexcept>
#include "definitions.hpp"
#include "sigmath.hpp"
```

Include dependency graph for process.hpp:



This graph shows which files directly or indirectly include this file:



Namespaces

- avda

Functions

- DataParams avda::process (float32 ∗data, uint32 size, float32 samplingRate)

8.17.1 Detailed Description

Contains functions related to the program's threaded processing of audio data.

Author

Samuel Andrew Wisner, awisner94@gmail.com

Definition in file process.hpp.

8.18 process.hpp

```
#ifndef process_H
#define process_H

#include <map>
#include <stdexcept>

#include "definitions.hpp"
#include "sigmath.hpp"

namespace avda {
    DataParams process(float32* data, uint32 size,
float32 samplingRate) {
        if((size & (size - 1) != 0) || size < 2) {
            throw std::invalid_argument(
                    "The number of samples is not a power of two!");
        }

        // declare function-scoped variables
        uint32 freqSize = size / 2;
        cfloat32* cdata = (cfloat32*)std::malloc(size * sizeof(
cfloat32));
        float32* fdata = (float32*)std::malloc(freqSize * sizeof(
float32));
        float32* origdata = (float32*)std::malloc(freqSize * sizeof(
float32));

        // convert data to complex numbers for fft()
        for(uint32 i = 0; i < size; i++) {
            cdata[i] = data[i];
        }

        // find frequency spectrum in relative decibels
        fft(cdata, size);
        mag(cdata, fdata, freqSize);
        Maximum maximum = max(fdata, freqSize);

        for(uint32 i = 0; i < freqSize; i++) {
            fdata[i] /= maximum.value;
        }

        decibels(fdata, freqSize);

        for(uint32 i = 0; i < freqSize; i++) {
            origdata[i] = fdata[i];
        }

        /*
         * Run spectrum values through moving-average filter to smooth the
         * curve and make it easier to determine the derivative.
         */
        smooth(fdata, freqSize, 20);

        /*
         * Find the derivative of the smoothed spectrum. Bote that both this
         * filter and the previous are necessary to the algorithm.
         */
        diff(fdata, freqSize);
        smooth(fdata, freqSize, 100);
        absolute(fdata, freqSize);

        // find the parameters of this specific recording
        uint16 offset = 1000;
        absolute(&fdata[offset], freqSize - offset);
        maximum = max(&fdata[offset], freqSize - offset);
        uint32 index = maximum.index + offset;

        DataParams params;
        params.freq = index * (float)SAMPLE_FREQ / freqSize / 2;
        params.noise = average(&origdata[index + offset],
                freqSize - offset - index);

        free(cdata);
        free(fdata);
```

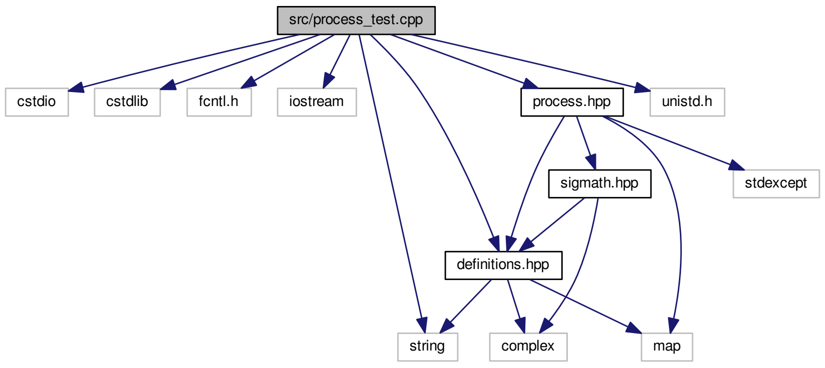
```
            return params;

        }
    }

#endif
```

8.19 src/process_test.cpp File Reference

Contains a program to test the process() function.

```
#include <cstdio>
#include <cstdlib>
#include <fcntl.h>
#include <iostream>
#include <string>
#include <unistd.h>
#include "definitions.hpp"
#include "process.hpp"
```

Include dependency graph for process_test.cpp:



Macros

- #define COUNT 131072

Functions

- int main (int argc, char **argv)

8.19.1 Detailed Description

Contains a program to test the process() function.

Author

Samuel Andrew Wisner, awisner94@gmail.com
Nicholas K. Nolan

Definition in file process_test.cpp.

8.19.2 Macro Definition Documentation

8.19.2.1 #define COUNT 131072

Definition at line 18 of file process_test.cpp.

8.19.3 Function Documentation

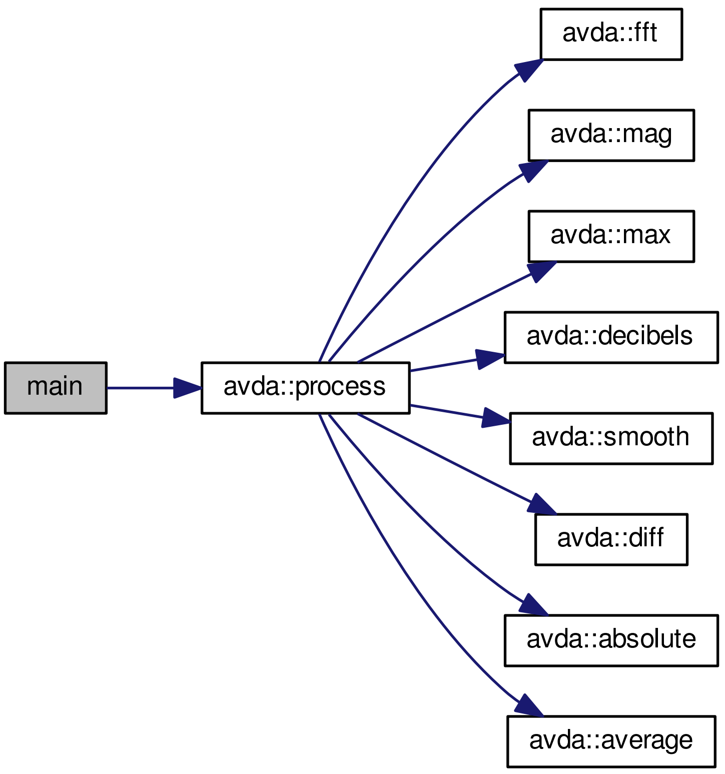
8.19.3.1 int main (int *argc,* char ∗∗ *argv*)

Tests the process() function from process.hpp.

Definition at line 26 of file process_test.cpp.

```
00026                                   {
00027     int file = open("/home/pi/avda/etc/audio/test.raw", O_RDONLY);
00028
00029     if(file < 0) {
00030         cerr << "File unreadable!" << endl;
00031         return -1;
00032     }
00033
00034     float32* buffer = (float32*)malloc(COUNT * sizeof(float32));
00035     int charRead = read(file, buffer, COUNT * sizeof(float32));
00036
00037     if(charRead < COUNT) {
00038         cerr << "Too few bytes read!" << endl;
00039         return -1;
00040     }
00041
00042     close(file);
00043
00044     DataParams params = process(buffer, COUNT, SAMPLE_FREQ);
00045     free(buffer);
00046     cout << "Cutoff: " << params.freq << endl;
00047     cout << "Noise: " << params.noise << endl;
00048 }
```

Here is the call graph for this function:



8.20 process_test.cpp

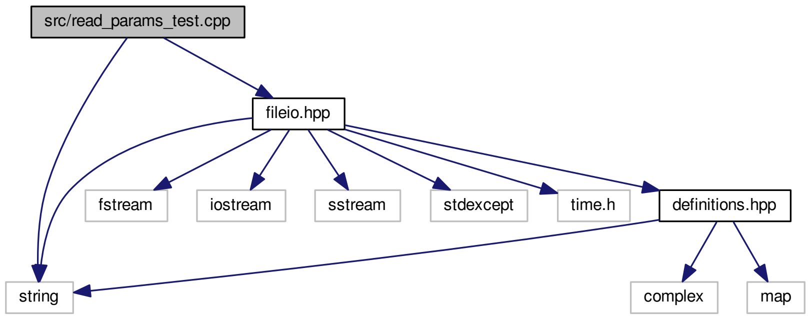
```

#include <cstdio>
#include <cstdlib>
#include <fcntl.h>
#include <iostream>
#include <string>
#include <unistd.h>

#include "definitions.hpp"
#include "process.hpp"

#define COUNT 131072

using namespace std;
using namespace avda;

int main(int argc, char** argv) {
    int file = open("/home/pi/avda/etc/audio/test.raw", O_RDONLY);

    if(file < 0) {
        cerr << "File unreadable!" << endl;
        return -1;
    }

    float32* buffer = (float32*)malloc(COUNT * sizeof(float32));
    int charRead = read(file, buffer, COUNT * sizeof(float32));

    if(charRead < COUNT) {
        cerr << "Too few bytes read!" << endl;
        return -1;
    }

    close(file);

    DataParams params = process(buffer, COUNT, SAMPLE_FREQ);
    free(buffer);
    cout << "Cutoff: " << params.freq << endl;
    cout << "Noise: " << params.noise << endl;
}
```

8.21 src/read_params_test.cpp File Reference

Contains a program test the PatientName() function.

```
#include <string>
#include "fileio.hpp"
```

Include dependency graph for read_params_test.cpp:



Functions

- int main (int argc, char **argv)

8.21.1 Detailed Description

Contains a program test the PatientName() function.

Author

Samuel Andrew Wisner, awisner94@gmail.com

Definition in file read_params_test.cpp.

8.21.2 Function Documentation

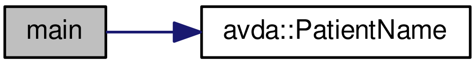
8.21.2.1 int main (int *argc*, char ** *argv*)

Tests the PatientName() function in fileio.hpp.

Definition at line 17 of file read_params_test.cpp.

```
                           {
    string filename = PatientName();
    cout << filename;
}
```

Here is the call graph for this function:



8.22 read_params_test.cpp

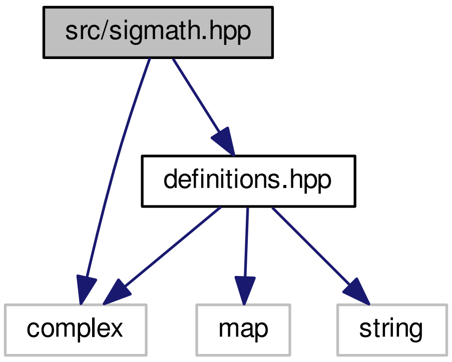
```
#include <string>

#include "fileio.hpp"

using namespace std;
using namespace avda;

int main(int argc, char** argv) {
    string filename = PatientName();
    cout << filename;
}
```

8.23 src/sigmath.hpp File Reference

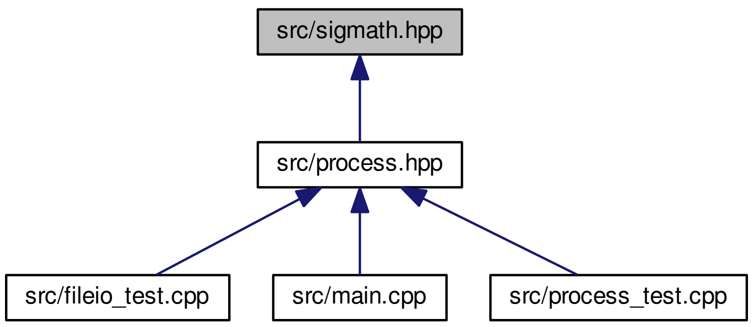
contains the functions necessary to perform the mathematical operations required by this program

```
#include <complex>
#include "definitions.hpp"
```

Include dependency graph for sigmath.hpp:



This graph shows which files directly or indirectly include this file:



Namespaces

- avda

Functions

- void avda::absolute (float32 ∗data, uint32 size)
- float32 avda::average (float32 ∗data, uint32 size)
- DataParams avda::average (DataParams ∗params, uint8 size)
- void avda::decibels (float32 ∗data, uint32 size)
- void avda::diff (float32 ∗data, uint32 size)
- void avda::fft (cfloat32 ∗data, uint32 size)
- void avda::mag (cfloat32 ∗orig, float32 ∗newmags, uint32 size)
- Maximum avda::max (float32 ∗data, uint32 size)
- void avda::smooth (float32 ∗data, uint32 size, uint16 order)

8.23.1 Detailed Description

contains the functions necessary to perform the mathematical operations required by this program

Author

Samuel Andrew Wisner, awisner94@gmail.com
Nicholas K. Nolan

Definition in file sigmath.hpp.

8.24 sigmath.hpp

```

#ifndef sigmath_H
#define sigmath_H

#include <complex>
#include "definitions.hpp"

```

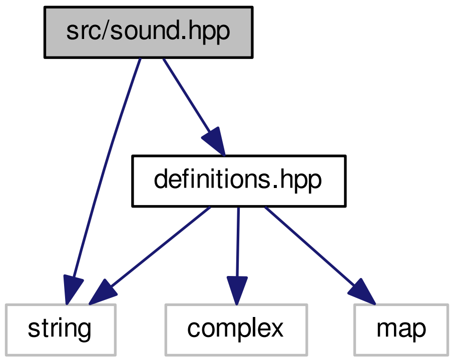
```
namespace avda {
    // PROTOTYPES

    void absolute(float32* data, uint32 size);

    float32 average(float32* data, uint32 size);

    DataParams average(DataParams* params, uint8 size);

    void decibels(float32* data, uint32 size);

    void diff(float32* data, uint32 size);

    void fft(cfloat32* data, uint32 size);

    void mag(cfloat32* orig, float32* newmags, uint32 size);

    Maximum max(float32* data, uint32 size);

    void smooth(float32* data, uint32 size, uint16 order);

    // DEFINITIONS

    void absolute(float32* data, uint32 size) {
        for(uint32 i = 0; i < size; i++) {
            data[i] = fabsf(data[i]);
        }
    }

    float32 average(float32* data, uint32 size) {
        float32 ave;

        for(uint32 i = 0; i < size; i++) {
            ave += data[i];
        }

        ave = ave / size;
        return ave;
    }

    DataParams average(DataParams* params, uint8 size) {
        DataParams ave;

        for(uint8 i = 0; i < size; i++) {
            //freq is an attribute. this is how to add structure attributes
            ave.freq += params[i].freq;
            ave.noise += params[i].noise;
        }

        ave.freq /= size;
        ave.noise /= size;
        return ave;
    }

    void decibels(float32* data, uint32 size) {
        for(uint32 i = 0; i < size; i++) {
            data[i] = 20 * log10(data[i]);
        }
    }

    void diff(float32* data, uint32 size) {
        float32 temp[size];
        temp[0] = 0;

        for(uint32 i = 1; i < size; i++) {
            temp[i] = data[i] - data[i-1];
        }

        for(uint32 i = 0; i < size; i++) {
            data[i] = temp[i];
        }
    }

    void fft(cfloat32* data, uint32 size) {
        // DFT
        uint32 k = size;
        uint32 n;
        float32 thetaT = M_PI / size;
        cfloat32 phiT(cos(thetaT), sin(thetaT));
        cfloat32 T;

        while(k > 1) {
            n = k;
            k >>= 1;
            phiT = phiT * phiT;
            T = 1.0L;

            for(uint32 l = 0; l < k; l++) {
                for(uint32 a = l; a < size; a += n) {
                    uint32 b = a + k;
                    cfloat32 t = data[a] - data[b];
                    data[a] += data[b];
                    data[b] = t * T;
                }

                T *= phiT;
            }
        }

        // Decimate
        uint32 m = (uint32)log2(size);

        for(uint32 a = 0; a < size; a++) {
            uint32 b = a;

            // Reverse bits
            b = (((b & 0xaaaaaaaa) >> 1) | ((b & 0x55555555) << 1));
            b = (((b & 0xcccccccc) >> 2) | ((b & 0x33333333) << 2));
            b = (((b & 0xf0f0f0f0) >> 4) | ((b & 0x0f0f0f0f) << 4));
            b = (((b & 0xff00ff00) >> 8) | ((b & 0x00ff00ff) << 8));
            b = ((b >> 16) | (b << 16)) >> (32 - m);

            if (b > a)
            {
                cfloat32 t = data[a];
                data[a] = data[b];
                data[b] = t;
            }
        }
    }

    void mag(cfloat32* orig, float32* newmags, uint32 size) {
        //loop to run throught the length of array orig
        for(uint32 n = 0; n < size; n++) {
            /*
             * abs should calculate the magnitude of complex array elements.
             * saves to new array
             */
            newmags[n] = std::abs(orig[n]);
        }
    }

    Maximum max(float32* data, uint32 size) {
        Maximum m;

        //loop to run through the length of array data
        for (uint32 i = 0; i < size; i++) {
            /*
             * when value at data[i] is above max.value,
             * sets max.value equal to data[i] and max.index equal to i
             */
            if (data[i] > m.value) {
                m.value = data[i];
                m.index = i;
            }
        }

        return m;
    }

    void smooth(float32* data, uint32 size, uint16 order) {
        float32 coeff = 1 / (float32)order;
        float32 temp[size];

        for(uint32 i = 0; i < size; i++) {
            temp[i] = 0;

            for(uint16 j = 0; j < order && j <= i; j++) {
                temp[i] += data[i - j];
            }

            temp[i] *= coeff;
        }

        for(uint32 i = 0; i < size; i++) {
            data[i] = temp[i];
        }
    }
}

#endif
```

8.25 src/sound.hpp File Reference

contains the function(s) relating to sound

```
#include <string>
#include "definitions.hpp"
```

Include dependency graph for sound.hpp:



Namespaces

- avda

Functions

- void avda::play (auto filename)

8.25.1 Detailed Description

contains the function(s) relating to sound

Author

Samuel Andrew Wisner, awisner94@gmail.com

Definition in file sound.hpp.

8.26 sound.hpp

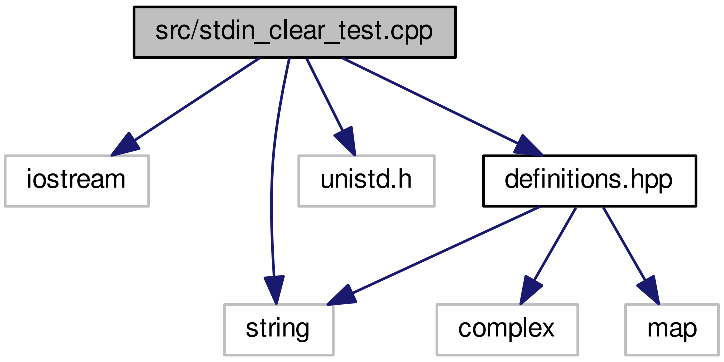
```
#ifndef sound_H
#define sound_H

#include <string>

#include "definitions.hpp"

namespace avda {
    void play(auto filename) {

    }
}

#endif
```

8.27 src/stdin_clear_test.cpp File Reference

Contains a program to test clearing the stdin buffer.

```
#include <iostream>
#include <string>
#include <unistd.h>
#include "definitions.hpp"
```

Include dependency graph for stdin_clear_test.cpp:



Macros

- #define COUNT 80

Functions

- int main (int argc, char **argv)

8.27.1 Detailed Description

Contains a program to test clearing the stdin buffer.

Author

Samuel Andrew Wisner, awisner94@gmail.com
Nicholas K. Nolan

Definition in file stdin_clear_test.cpp.

8.27.2 Macro Definition Documentation

8.27.2.1 #define COUNT 80

Definition at line 14 of file stdin_clear_test.cpp.

8.27.3 Function Documentation

8.27.3.1 int main (int *argc*, char ∗∗ *argv*)

Tests the ability to clear the stdin buffer.

Definition at line 22 of file stdin_clear_test.cpp.

```
00022                               {
00023     char text1[COUNT];
00024     char text2[COUNT];
00025
00026     cout << "Enter text to ignore: ";
00027     cout.flush();
00028     read(STDIN_FILENO, &text1, COUNT);
00029 //   fflush(stdin);
00030     cout << endl << "Enter text to print: ";
00031     cout.flush();
00032     read(STDIN_FILENO, &text2, COUNT);
00033     cout << endl << "In buffer: " << text2 << endl;
00034 }
```

8.28 stdin_clear_test.cpp

```
00001
00008 #include <iostream>
00009 #include <string>
00010 #include <unistd.h>
00011
00012 #include "definitions.hpp"
00013
00014 #define COUNT 80
00015
00016 using namespace std;
00017 using namespace avda;
00018
00022 int main(int argc, char** argv) {
00023     char text1[COUNT];
00024     char text2[COUNT];
00025
00026     cout << "Enter text to ignore: ";
00027     cout.flush();
00028     read(STDIN_FILENO, &text1, COUNT);
00029 //   fflush(stdin);
00030     cout << endl << "Enter text to print: ";
00031     cout.flush();
00032     read(STDIN_FILENO, &text2, COUNT);
00033     cout << endl << "In buffer: " << text2 << endl;
00034 }
```

Index