A412

0.1

Generated by Doxygen 1.8.8

Fri Dec 12 2014 08:23:15

Contents

1 Data Structure Index **1**

1.1 Data Structures 1

2 File Index **2**

2.1 File List 2

3 Data Structure Documentation **2**

3.1 data Struct Reference 2

3.1.1 Field Documentation 2

3.2 eventPlacement Struct Reference 2

3.2.1 Field Documentation 2

3.3 moodWeighting Struct Reference 3

3.3.1 Field Documentation 3

3.4 note Struct Reference 3

3.4.1 Field Documentation 3

3.5 points Struct Reference 4

3.5.1 Field Documentation 4

4 File Documentation **4**

4.1 main.c File Reference 4

4.1.1 Macro Definition Documentation 6

4.1.2 Typedef Documentation 6

4.1.3 Enumeration Type Documentation 6

4.1.4 Function Documentation 6

4.1.5 Variable Documentation 17

Index **18**

1 Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

data **2**

eventPlacement **2**

moodWeighting **3**

note **3**

points **4**

2 File Index

2.1 File List

Here is a list of all files with brief descriptions:

main.c 4

3 Data Structure Documentation

3.1 data Struct Reference

Data Fields

- unsigned int **tempo**
- **mode mode**
- **tone key**

3.1.1 Field Documentation

3.1.1.1 tone data::key

3.1.1.2 mode data::mode

3.1.1.3 unsigned int data::tempo

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

- **main.c**

3.2 eventPlacement Struct Reference

Data Fields

- int **noteOn**
- int **noteOff**
- int **afterTouch**
- int **controlChange**
- int **programChange**
- int **channelPressure**
- int **pitchWheel**

3.2.1 Field Documentation

3.2.1.1 int eventPlacement::afterTouch

3.2.1.2 int eventPlacement::channelPressure

3.2.1.3 int eventPlacement::controlChange

3.2.1.4 int eventPlacement::noteOff

3.2.1.5 int eventPlacement::noteOn

3.2.1.6 int eventPlacement::pitchWheel

3.2.1.7 int eventPlacement::programChange

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

- **main.c**

3.3 moodWeighting Struct Reference

Data Fields

- char **name** [25]
- int **mode**
- int **tempo**
- int **toneLength**
- int **pitch**

3.3.1 Field Documentation

3.3.1.1 int moodWeighting::mode

3.3.1.2 char moodWeighting::name[25]

3.3.1.3 int moodWeighting::pitch

3.3.1.4 int moodWeighting::tempo

3.3.1.5 int moodWeighting::toneLength

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

- **main.c**

3.4 note Struct Reference

Data Fields

- int **tone**
- int **octave**
- int **length**
- int **average**
- int **ticks**

3.4.1 Field Documentation

3.4.1.1 int note::average

3.4.1.2 int note::length

3.4.1.3 int note::octave

3.4.1.4 int note::ticks

3.4.1.5 int note::tone

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

- **main.c**

3.5 points Struct Reference

Data Fields

- char * **parameter**
- int **point**

3.5.1 Field Documentation

3.5.1.1 char* points::parameter

3.5.1.2 int points::point

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

- **main.c**

4 File Documentation

4.1 main.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <dirent.h>
```

Data Structures

- struct **note**
- struct **data**
- struct **points**
- struct **moodWeighting**
- struct **eventPlacement**

Macros

- #define **CHARS** 1000
- #define **SCALESIZE** 7

Typedefs

- typedef enum **mode mode**
- typedef enum **tone tone**
- typedef enum **mood mood**

Enumerations

- enum **mode** { **major**, **minor** }
- enum **tone** {
  C, **Csharp**, **D**, **Dsharp**,
  E, **F**, **Fsharp**, **G**,
  Gsharp, **A**, **Asharp**, **B** }
- enum **mood** { **glad**, **sad** }

Functions

- void **checkDirectory** (char ∗, DIR ∗)
- void **findNoteLength** (double x, int ∗, int ∗)
- void **printNote** (**note**)
- int **getHex** (FILE ∗, int[])
- void **fillSongData** (**data** ∗, int[], int)
- int **countNotes** (int[], int)
- void **fillNote** (int, **note** ∗)
- void **printSongData** (**data**)
- void **settingPoints** (int ∗, int ∗, int ∗, int ∗, **data**, int, **note**[], int ∗)
- void **insertMoods** (**moodWeighting**[], FILE ∗)
- void **weightingMatrix** (**moodWeighting**[], int, int, int, int, int ∗)
- void **findEvents** (int, int[], **eventPlacement**[], **note**[], int ∗)
- void **insertPlacement1** (int[], int ∗, int, **note**[], int ∗, int[])
- void **insertPlacement2** (int[], int ∗, int)
- int **checkNextEvent** (int[], int)
- void **findTicks** (int, int[], **eventPlacement**[], **note**[], int, int ∗, int[])
- void **countTicks1** (int[], int ∗, int, **note**[], int ∗)
- void **countTicks2** (int[], int ∗, int, **note**[], int ∗)
- int **sortResult** (const void ∗, const void ∗)
- void **deltaTimeToNoteLength** (int, int, **note** ∗)
- int **isInScale** (int, int[], int)
- int **isInMinor** (int)
- int **isInMajor** (int)
- int **sortToner** (const void ∗, const void ∗)
- void **findMode** (**note** ∗, int, **data** ∗)
- int **FindMoodAmount** (FILE ∗)
- void **printResults** (int, int, int, int, **moodWeighting**[], int[])
- int **main** (int argc, const char ∗argv[])
- int **sortTones** (const void ∗a, const void ∗b)
- void **checkScale** (int scales[], int **tone**, int key)
- void **findMode** (**note** noteAr[], int totalNotes, **data** ∗**data**)

Variables

- int **AMOUNT_OF_MOODS**

4.1.1 Macro Definition Documentation

4.1.1.1 #define CHARS 1000

4.1.1.2 #define SCALESIZE 7

4.1.2 Typedef Documentation

4.1.2.1 typedef enum mode mode

4.1.2.2 typedef enum mood mood

4.1.2.3 typedef enum tone tone

4.1.3 Enumeration Type Documentation

4.1.3.1 enum mode

Enumerator

major

minor

```
00026 {major, minor} mode;
```

4.1.3.2 enum mood

Enumerator

glad

sad

```
00028 {glad, sad} mood;
```

4.1.3.3 enum tone

Enumerator

C

Csharp

D

Dsharp

E

F

Fsharp

G

Gsharp

A

Asharp

B

```
00027 {C, Csharp, D, Dsharp, E, F, Fsharp, G, Gsharp, A, Asharp, B} tone;
```

4.1.4 Function Documentation

4.1.4.1 void checkDirectory (char * *MIDIfile,* DIR * *dir*)

A function to read music directory and prompt user to choose file

Parameters

MIDIfile	a pointer to a string containing the name of the chosen input file
dir	a pointer to a directory

```
00163                                                {
00164   struct dirent *musicDir;
00165   if ((dir = opendir ("./Music")) != NULL) {
00166     printf("Mulige numre\n");
00167       while ((musicDir = readdir (dir)) != NULL) {
00168         printf ("%s\n", musicDir->d_name);
00169       }
00170   }
00171   else {
00172     perror ("Failure while opening directory");
00173     exit (EXIT_FAILURE);
00174   }
00175   printf("Indtast det valgte nummer\n");
00176   scanf("%s", MIDIfile);
00177   chdir("./Music");
00178 }
```

4.1.4.2 int checkNextEvent (int *hex[]*, int *j*)

```
00264                                        {
00265   switch (hex[j]){
00266     case 0x90:
00267     case 0x80:
00268     case 0xA0:
00269     case 0xB0:
00270     case 0xC0:
00271     case 0xD0:
00272     case 0xE0: return 1; break;
00273     default  : return 0; break;
00274   }
00275 }
```

4.1.4.3 void checkScale (int *scales[]*, int *tone*, int *key*)

Checks if the tone given is within the scale of the key given.

Parameters

scales	An array containing the scalas
tone	An integer representing the tone to be checked
key	Integer representing the key the note is compared to

```
00518                                                  {
00519   if(tone < key)
00520     tone += 12;
00521   scales[key] = isInMajor(tone - key);
00522 }
```

4.1.4.4 int countNotes (int *hex[]*, int *amount*)

A function to count the number of notes in the entire song

Parameters

hex[]	an array with the stored information from the file
amount	an integer holding the total number of characters in the array

```
00198                                        {
00199   int i = 0, res = 0;
00200   for(i = 0; i < amount; i++){
00201     if(hex[i] == 0x90){
00202       res++;
00203     }
00204   }
00205   return res;
00206 }
```

4.1.4.5 void countTicks1 (int *hex[ ]*, int * *i*, int *deltaCounter*, note *noteAr[ ]*, int * *tickCounter*)

```
{
  noteAr[*tickCounter].ticks = 0;
  int tick = 0;
  while(deltaCounter < 7 && hex[(*i + deltaCounter)] > 0x80)
    tick += ((hex[(*i + deltaCounter++)] - 0x80) * 128);
  tick += hex[(*i + deltaCounter)];
  noteAr[*tickCounter].ticks += tick;
  *tickCounter += 1;
  *i += deltaCounter;
}
```

4.1.4.6 void countTicks2 (int *hex[ ]*, int * *i*, int *deltaCounter*, note *noteAr[ ]*, int * *tickCounter*)

```
{
  noteAr[*tickCounter].ticks = 0;
  int tick = 0;
  while(deltaCounter < 6 && hex[(*i + deltaCounter)] > 0x80)
    tick += ((hex[(*i + deltaCounter++)] - 0x80) * 128);
  tick += hex[(*i + deltaCounter)];
  noteAr[*tickCounter].ticks += tick;
  *tickCounter += 1;
  *i += deltaCounter;
}
```

4.1.4.7 void deltaTimeToNoteLength (int *ppqn*, int *size*, note * *noteAr*)

```
{

  for (int i = 0; i < size; i++){

    double noteLength = ((double) (noteAr[i].ticks)) / ((double) (ppqn/8));

    if (noteLength < 1.5 && noteLength >= 0)
      noteLength = 1;
    else if (noteLength < 3 && noteLength >= 1.5)
      noteLength = 2;
    else if (noteLength < 6 && noteLength >= 3)
      noteLength = 4;
    else if (noteLength < 12 && noteLength >= 6)
      noteLength = 8;
    else if (noteLength < 24 && noteLength >= 12)
      noteLength = 16;
    else
      noteLength = 32;

        noteAr[i].length = noteLength;
    }
}
```

4.1.4.8 void fillNote (int *inputTone*, note * *note*)

A function to fill out each of the structures of type note

Parameters

inputTone	the value of the hexadecimal collected on the "tone"-spot
*note**	a pointer to a note-structure

```
{
  note->tone = inputTone % 12;
  note->average = inputTone;
  note->octave = inputTone / 12;
}
```

4.1.4.9 void fillSongData (data * *data*, int *hex[ ]*, int *numbersInText*)

! A function, that fills out the song data

Parameters

∗data	a pointer to a structure containing the tempo and mode of the song
hex[]	the array of integers read from the file
numbersInText	the total amount of integers in the array

```
00214                                                                {
00215   int j;
00216   /*Find the mode of the song, initialised as minor atm*/
00217   for(j = 0; j < numbersInText; j++){
00218     /* finds the tempo */
00219     if(hex[j] == 0xff && hex[j+1] == 0x51 && hex[j+2] == 0x03){
00220       data->tempo =  60000000/((hex[j+3] << 16) | (hex[j+4] << 8) | (hex[j+5]));
00221     }
00222   }
00223 }
```

4.1.4.10 void findEvents (int *numbersInText*, int *hex[]*, eventPlacement *placement[]*, note *noteAr[]*, int ∗ *size*)

```
00225                                                                                                                                {
00226   int noteOff = 0, noteOn = 0, afterTouch = 0, controlChange = 0,
00227       programChange = 0, channelPressure = 0, pitchWheel = 0, n = 0, notes[numbersInText];
00228
00229   for(int j = 0; j < numbersInText; j++){
00230     switch (hex[j]){
00231       case 0x90: insertPlacement1(hex, &placement[noteOn++].noteOn, j, noteAr, &n, notes);
      break;
00232       case 0x80: insertPlacement1(hex, &placement[noteOff++].noteOff, j, noteAr, &n, notes);
      break;
00233       case 0xA0: insertPlacement1(hex, &placement[afterTouch++].afterTouch, j, noteAr, &n, notes);
      break;
00234       case 0xB0: insertPlacement1(hex, &placement[controlChange++].controlChange, j, noteAr, &n, notes);
      break;
00235       case 0xC0: insertPlacement2(hex, &placement[programChange++].programChange, j);
      break;
00236       case 0xD0: insertPlacement2(hex, &placement[channelPressure++].channelPressure, j);
      break;
00237       case 0xE0: insertPlacement1(hex, &placement[pitchWheel++].pitchWheel, j, noteAr, &n, notes);
      break;
00238       default   :
      break;
00239     }
00240   }
00241   findTicks(numbersInText, hex, placement, noteAr, noteOn, size, notes);
00242 }
```

4.1.4.11 void findMode (note ∗ , int , data ∗)

4.1.4.12 void findMode (note *noteAr[]*, int *totalNotes*, data ∗ *data*)

A function to find the mode of the song by first calculating the tone span over sets of notes in the song, and then comparing it to the definition of minor and major keys.

Parameters

noteAr	An array of all the notes in the entire song
totalNotes	The number of notes in the song
data	The song data

```
00529                                                          {
00530   int majors[12] = {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}, minors[12] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
00531   int x = 0, y = 0, z = 0, bar[4], sizeBar = 4, tempSpan = 999, span = 999, keynote = 0,
      mode = 0, tempNote = 0;
00532
00533   for(x = 0; x < totalNotes; x++){
00534     tempNote = noteAr[x].tone;
00535
00536     for(y = C; y <= B; y++){
00537       if(majors[y])
00538         checkScale(majors, tempNote, y);
00539     }
00540   }
00541
00542   for(y = 0; y < 12; y++){
00543     z = y;
```

```
00544     if(majors[z]){
00545       if((z - 3) < 0)
00546         z += 12;
00547     minors[z-3] = 1;
00548     }
00549   }
00550
00551   z = 0;  x = 0;
00552
00553   /*Goes through all notes of the song and puts them into an array, 4 at a time*/
00554   while(x < totalNotes){
00555     z = x;
00556     for(y = 0; y < sizeBar; y++, z++){
00557       if(z < totalNotes)
00558         bar[y] = noteAr[z].tone;
00559       else
00560         sizeBar = y;
00561     }
00562
00563     if(y == sizeBar){
00564       span = 999;
00565       /*Sort notes in ascending order*/
00566       qsort(bar, sizeBar, sizeof(tone), sortTones);
00567
00568       /*Finds the lowest possible tonespan over the array of 4 notes*/
00569       for(z = 0; z < sizeBar; z++){
00570     if((z + 1) > 3)
00571           tempSpan = (bar[(z+1)%4]+12)-bar[z] + bar[(z+2)%4]-bar[(z+1)%4] + bar[(z+3)%4]-bar[(z+2)%4];
00572         else if((z + 2) > 3)
00573           tempSpan = bar[(z+1)]-bar[z] + (bar[(z+2)%4]+12)-bar[(z+1)%4] + bar[(z+3)%4]-bar[(z+2)%4];
00574     else if((z +3) > 3)
00575           tempSpan = bar[(z+1)]-bar[z] + bar[(z+2)]-bar[(z+1)] + (bar[(z+3)%4]+12)-bar[z];
00576     else
00577           tempSpan = bar[(z+1)]-bar[z] + bar[(z+2)]-bar[(z+1)] + bar[(z+3)]-bar[(z+2)];
00578
00579     if(tempSpan < span && (majors[bar[z]] || minors[bar[z]])){
00580           span = tempSpan;
00581           keynote = bar[z];
00582         }
00583       }
00584       mode += isInScale(keynote, bar, sizeBar);
00585       x++;
00586     }
00587   }
00588   /*outputs result directly to the data struct*/
00589   if(mode > 0)
00590     data->mode = major;
00591   else if(mode < 0)
00592     data->mode = minor;
00593 }
```

4.1.4.13 int FindMoodAmount (FILE * *moods*)

```
00654                                 {
00655   int i = 1;
00656   while(fgetc(moods) != EOF){
00657     if(fgetc(moods) == '\n')
00658       i++;
00659   }
00660   rewind(moods);
00661   return i;
00662 }
```

4.1.4.14 void findNoteLength (double *x*, int * , int *)

4.1.4.15 void findTicks (int *numbersInText*, int *hex[]*, eventPlacement *placement[]*, note *noteAr[]*, int *noteOn*, int * *size*, int *notes[]*)

```
00277
                    {
00278   int tickCounter = 0, deltaCounter1 = 3, deltaCounter2 = 2;
00279
00280   for(int j = 0; j < noteOn; j++){
00281     for(int i = placement[j].noteOn; i < numbersInText; i++){
00282       if(hex[i] == 0x80){
00283         if(hex[i + 1] == notes[j])
00284           break;
00285         else
00286           countTicks1(hex, &i, deltaCounter1, noteAr, &tickCounter);
00287       }
00288       else if(hex[i] == 0xA0){
00289         if(hex[i + 1] == notes[j] && hex[i + 2] == 0x00)
```

```
00290           break;
00291         else
00292           countTicks1(hex, &i, deltaCounter1, noteAr, &tickCounter);
00293       }
00294       else if(hex[i] == 0xD0){
00295         if(hex[i + 1] == 0x00)
00296           break;
00297         else
00298           countTicks2(hex, &i, deltaCounter2, noteAr, &tickCounter);
00299       }
00300       else if(hex[i] == 0xC0)
00301         countTicks2(hex, &i, deltaCounter2, noteAr, &tickCounter);
00302       else
00303         countTicks1(hex, &i, deltaCounter1, noteAr, &tickCounter);
00304     }
00305   }
00306   *size = tickCounter;
00307 }
```

4.1.4.16 int getHex (FILE * *f,* int *hexAr[]*)

A function, that retrieves the hexadecimals from the files and also returns the number of files

Parameters

**f*	a pointer to the file the program is reading from
hexAr[]	an array of integers, that the information is stored in

```
00184                                 {
00185   int i = 0, c;
00186   while( (c = fgetc(f)) != EOF && i < CHARS){
00187     hexAr[i] = c;
00188     i++;
00189   }
00190
00191   return i;
00192 }
```

4.1.4.17 void insertMoods (moodWeighting *moodArray[],* FILE * *moods*)

```
00448                                                           {
00449   for(int i = 0; i < AMOUNT_OF_MOODS; i++){
00450     fscanf(moods, "%s %d %d %d %d", moodArray[i].name , &moodArray[i].mode,
00451                                     &moodArray[i].tempo, &moodArray[i].toneLength,
00452                                     &moodArray[i].pitch);
00453   }
00454 }
```

4.1.4.18 void insertPlacement1 (int *hex[],* int * *place,* int *j,* note *noteAr[],* int * *n,* int *notes[]*)

```
00244                                                                                                 {
00245   int i = 3;
00246   while(i < 7 && hex[(j + i++)] > 0x80);
00247   if(checkNextEvent(hex, (j + i))){
00248     *place = j;
00249     if(hex[j] == 0x90){
00250       notes[*n] = hex[j + 1];
00251       fillNote(hex[j + 1], &noteAr[*n]);
00252       *n += 1;
00253     }
00254   }
00255 }
```

4.1.4.19 void insertPlacement2 (int *hex[],* int * *place,* int *j*)

```
00257                                                 {
00258   int i = 2;
00259   while(i < 6 && hex[(j + i++)] > 0x80);
00260   if(checkNextEvent(hex, (j + i)))
00261     *place = j;
00262 }
```

4.1.4.20 int isInMajor (int *toneLeap*)

A function to check if the given tone leap is in the major scale.

Parameters

toneLeap	An integer describing the processed tone leap

Returns

a boolean value, returns 1 if the tone leap is in the major scale, 0 if it's not.

```
00643                               {
00644   int major[] = {0, 2, 4, 5, 7, 9, 11};
00645
00646   for(int i = 0; i < SCALESIZE; i++){
00647     if(toneLeap == major[i])
00648       return 1;
00649   }
00650   return 0;
00651 }
```

4.1.4.21 int isInMinor (int *toneLeap*)

A function to check if the given tone leap is in the minor scale.

Parameters

toneLeap	An integer describing the processed tone leap

Returns

a boolean value, returns 1 if the tone leap is in the minor scale, 0 if it's not.

```
00629                               {
00630   int minor[] = {0, 2, 3, 5, 7, 8, 10};
00631
00632   for(int i = 0; i < SCALESIZE; i++){
00633     if(toneLeap == minor[i])
00634       return 1;
00635   }
00636   return 0;
00637 }
```

4.1.4.22 int isInScale (int *keytone*, int *otherTones[]*, int *size*)

A function to check if a given scale in given keytone corresponds with the tones in the rest of the song.

Parameters

keytone	The keytone of the processed scale
otherTones	An array of the rest of the tones, which the function compares to the keytone and mode
size	The number of tones in the otherTones array

Returns

a boolean value, returns 1 if the mode is major, -1 if it's minor and 0, if wasn't possible to decide.

```
00601                                                       {
00602   int toneLeap, isMinor = 1, isMajor = 1;
00603
00604   for(int i = 0; i < size; i++){
00605     if(otherTones[i] < keytone)
00606       otherTones[i] += 12;
00607       toneLeap = otherTones[i] - keytone;
00608
00609       if(isMinor)
00610         isMinor = isInMinor(toneLeap);
00611       if(isMajor)
00612         isMajor = isInMajor(toneLeap);
00613     }
00614
00615     if(isMinor && isMajor)
00616       return 0;
```

```
00617     else if(isMinor)
00618       return -1;
00619     if(isMajor)
00620       return 1;
00621
00622     return 0;
00623 }
```

4.1.4.23 int main (int *argc*, const char * *argv[]*)

```
00096                                       {
00097   DIR *dir = 0;
00098   FILE *f;
00099   char MIDIfile[25];
00100   /*Variables*/
00101   int numbersInText = 0, notes, size = 0, mode = 5, tempo = 5, toneLength = 5, pitch = 5;
00102   FILE* moods = fopen("moods.txt", "r");
00103   if(moods == NULL){
00104     perror("Error: moods missing ");
00105     exit(EXIT_FAILURE);
00106   }
00107   AMOUNT_OF_MOODS = FindMoodAmount(moods);
00108   moodWeighting moodArray[AMOUNT_OF_MOODS];
00109   data data = {0, major, D};
00110   if (argv[1] == NULL){
00111     checkDirectory(MIDIfile, dir);
00112     f = fopen(MIDIfile, "r");
00113     if(f == NULL){
00114       perror("Error opening file");
00115       exit(EXIT_FAILURE);
00116     }
00117   }
00118   else if(argv[1] != NULL){
00119     f = fopen(argv[1],"r");
00120     if(f == NULL){
00121       perror("Error opening file");
00122       exit(EXIT_FAILURE);
00123     }
00124   }
00125   closedir (dir);
00126   int *hex = (int *) malloc(CHARS * sizeof(int));
00127   if(hex == NULL){
00128     printf("Memory allocation failed, bye!");
00129     exit(EXIT_FAILURE);
00130   }
00131   /*Reading the data from the file*/
00132   numbersInText = getHex(f, hex);
00133   fillSongData(&data, hex, numbersInText);
00134   notes = countNotes(hex, numbersInText);
00135   note *noteAr = (note*) malloc(notes * sizeof(note));
00136   if(noteAr == NULL){
00137     printf("Memory allocation failed, bye!");
00138     exit(EXIT_FAILURE);
00139   }
00140   eventPlacement placement[numbersInText];
00141   findEvents(numbersInText, hex, placement, noteAr, &size);
00142   deltaTimeToNoteLength(960, size, noteAr);
00143   insertMoods(moodArray, moods);
00144   findMode(noteAr, notes, &data);
00145   settingPoints(&mode, &tempo, &toneLength, &pitch, data, notes, noteAr, &size);
00146   printSongData(data);
00147   int result[AMOUNT_OF_MOODS];
00148   weightingMatrix(moodArray, mode, tempo, toneLength, pitch, result);
00149
00150   /*Clean up and close*/
00151   fclose(f);
00152   free(hex);
00153   free(noteAr);
00154
00155   /* Print results */
00156   printResults(mode, tempo, toneLength, pitch, moodArray, result);
00157
00158   return 0;
00159 }
```

4.1.4.24 void printNote (note *note*)

A function to print the note

Parameters

note	the note structure to be printed

```
{
  printf("Tone: ");

  switch (note.tone){
    case C     : printf("C") ; break;
    case Csharp: printf("C#"); break;
    case D     : printf("D") ; break;
    case Dsharp: printf("D#"); break;
    case E     : printf("E") ; break;
    case F     : printf("F") ; break;
    case Fsharp: printf("F#"); break;
    case G     : printf("G") ; break;
    case Gsharp: printf("G#"); break;
    case A     : printf("A") ; break;
    case Asharp: printf("A#"); break;
    case B     : printf("B") ; break;
    default    : printf("Undefined note"); break;
  }
  printf(", octave: %d\n", note.octave);
}
```

4.1.4.25 void printResults (int *mode,* int *tempo,* int *toneLength,* int *pitch,* moodWeighting *moodArray[ ],* int *result[ ]*)

```
{
  printf("\n\n\n");
  printf(" Mode:");
  if(mode < 0)
    printf("         %d\n", mode);
  else
    printf("          %d\n", mode);
  printf(" Tempo:");
  if(tempo < 0)
    printf("        %d\n", tempo);
  else
    printf("         %d\n", tempo);
  printf(" Tone length:");
  if(toneLength < 0)
    printf(" %d\n", toneLength);
  else
    printf("  %d\n", toneLength);
  printf(" Pitch:");
  if(pitch < 0)
    printf("        %d\n", pitch);
  else
    printf("         %d\n", pitch);
  printf("\n\n\n                                    WEIGHTINGS            \n");
  printf("                           Mode | Tempo | Tone length | Pitch\n");

  for(int i = 0; i < AMOUNT_OF_MOODS; i++){
    printf(" %s", moodArray[i].name);
    for(int j = strlen(moodArray[i].name); j < 26; j++)
      printf(" ");
    if(moodArray[i].mode > -1)
      printf(" ");
    printf(" %d", moodArray[i].mode);
    for(int j = 0; j < 2; j++)
      printf(" ");
    printf("| ");
    if(moodArray[i].tempo > -1)
      printf(" ");
    printf(" %d", moodArray[i].tempo);
    for(int j = 0; j < 3; j++)
      printf(" ");
    printf("|     ");
    if(moodArray[i].toneLength > -1)
      printf(" ");
    printf(" %d", moodArray[i].toneLength);
    for(int j = 0; j < 6; j++)
      printf(" ");
    printf("|  ");
    if(moodArray[i].pitch > -1)
      printf(" ");
    printf(" %d\n", moodArray[i].pitch);
  }
  printf("\n\n\n");

  for(int i = 0; i < AMOUNT_OF_MOODS; i++){
    if(mode < 0)
      printf(" %d * ", mode);
```

```
00720     else
00721       printf(" %d * ", mode);
00722     if(moodArray[i].mode < 0)
00723       printf("%d + ", moodArray[i].mode);
00724     else
00725       printf(" %d + ", moodArray[i].mode);
00726     if(tempo < 0)
00727       printf("%d * ", tempo);
00728     else
00729       printf(" %d * ", tempo);
00730     if(moodArray[i].tempo < 0)
00731       printf("%d + ", moodArray[i].tempo);
00732     else
00733       printf(" %d + ", moodArray[i].tempo);
00734     if(toneLength < 0)
00735       printf("%d * ", toneLength);
00736     else
00737       printf(" %d * ", toneLength);
00738     if(moodArray[i].toneLength < 0)
00739       printf("%d + ", moodArray[i].toneLength);
00740     else
00741       printf(" %d + ", moodArray[i].toneLength);
00742     if(pitch < 0)
00743       printf("%d * ", pitch);
00744     else
00745       printf(" %d * ", pitch);
00746     if(moodArray[i].pitch < 0)
00747       printf("%d = ", moodArray[i].pitch);
00748     else
00749       printf(" %d = ", moodArray[i].pitch);
00750     if(result[i] < 0)
00751       printf("%d\n", result[i]);
00752     else
00753       printf(" %d\n", result[i]);
00754   }
00755   int moodOfMelodi = 0;
00756   for(int i = 0; i < AMOUNT_OF_MOODS; i++){
00757     if(moodOfMelodi < result[i])
00758       moodOfMelodi = i;
00759   }
00760   int test = 0;
00761
00762   if(!strcmp(moodArray[moodOfMelodi].name, "Happy")){
00763     printf("\n\n\n Sad ");
00764     while(test < 51){
00765       if(test == 25)
00766         printf("|");
00767       else if(test == (((result[moodOfMelodi] + 100) / 4)))
00768         printf("[");
00769       else if(test == (((result[moodOfMelodi] + 100) / 4) + 2))
00770         printf("]");
00771       else
00772         printf("-");
00773       test++;
00774     }
00775     printf(" Happy\n\n\n");
00776   }
00777   else if(!strcmp(moodArray[moodOfMelodi].name, "Sad")){
00778     printf("\n\n\n Sad ");
00779     while(test < 51){
00780       if(test == 25)
00781         printf("|");
00782       else if(test == (((-(result[moodOfMelodi]) + 100) / 4)))
00783         printf("[");
00784       else if(test == (((-(result[moodOfMelodi]) + 100) / 4) + 2))
00785         printf("]");
00786       else
00787         printf("-");
00788       test++;
00789     }
00790     printf(" Happy\n\n\n");
00791   }
00792
00793   printf("\n The mood of the melodi is %s\n", moodArray[moodOfMelodi].name);
00794 }
```

4.1.4.26 void printSongData (data *data*)

A function to print out the overall data of the song, tempo and mode

Parameters

data	the data to be printed

```
00368                                {
00369   printf("Tempo: %d\nMode: ", data.tempo);
00370   switch(data.mode){
00371     case minor: printf("minor"); break;
00372     case major: printf("major"); break;
00373     default: printf("unknown mode"); break;
00374   }
00375   printf("\nKeytone: %d", data.key);
00376   putchar('\n');
00377 }
```

4.1.4.27 void settingPoints (int * *mode*, int * *tempo*, int * *length*, int * *octave*, data *data*, int *notes*, note *noteAr[]*, int * *size*)

```
00379
            {
00380   int deltaTime = 2, combined = 0, averageNote = 0;
00381   switch(data.mode){
00382     case minor: *mode = -5; break;
00383     case major: *mode = 5; break;
00384     default: *mode = 0; break;
00385   }
00386   if(data.tempo < 60)
00387     *tempo = -5;
00388   else if(data.tempo >= 60 && data.tempo < 70)
00389     *tempo = -4;
00390   else if(data.tempo >= 70 && data.tempo < 80)
00391     *tempo = -3;
00392   else if(data.tempo >= 80 && data.tempo < 90)
00393     *tempo = -2;
00394   else if(data.tempo >= 90 && data.tempo < 100)
00395     *tempo = -1;
00396   else if(data.tempo >= 100 && data.tempo < 120)
00397     *tempo =  0;
00398   else if(data.tempo >= 120 && data.tempo < 130)
00399     *tempo =  1;
00400   else if(data.tempo >= 130 && data.tempo < 140)
00401     *tempo =  2;
00402   else if(data.tempo >= 140 && data.tempo < 150)
00403     *tempo =  3;
00404   else if(data.tempo >= 150 && data.tempo < 160)
00405     *tempo =  4;
00406   else if(data.tempo >=  160)
00407     *tempo =  5;
00408
00409   switch(deltaTime){
00410     case 1: *length = -5; break;
00411     case 2: *length = -4; break;
00412     case 4: *length = -2; break;
00413     case 8: *length =  0; break;
00414     case 16: *length = 3; break;
00415     case 32: *length = 5; break;
00416   }
00417   for (int i = 0; i < notes; i++){
00418     combined += noteAr[i].average;
00419   }
00420   averageNote = combined/notes;
00421
00422   if(averageNote <= 16)
00423     *octave = -5;
00424   else if(averageNote >= 17 && averageNote <= 23)
00425     *octave = -4;
00426   else if(averageNote >= 24 && averageNote <= 30)
00427     *octave = -3;
00428   else if(averageNote >= 31 && averageNote <= 37)
00429     *octave = -2;
00430   else if(averageNote >= 38 && averageNote <= 44)
00431     *octave = -1;
00432   else if(averageNote >= 45 && averageNote <= 51)
00433     *octave = 0;
00434   else if(averageNote >= 52 && averageNote <= 58)
00435     *octave = 1;
00436   else if(averageNote >= 59 && averageNote <= 65)
00437     *octave = 2;
00438   else if(averageNote >= 66 && averageNote <= 72)
00439     *octave = 3;
00440   else if(averageNote >= 73 && averageNote <= 79)
00441     *octave = 4;
00442   else if(averageNote >=80)
```

```
00443     *octave = 5;
00444 }
```

4.1.4.28 int sortResult (const void * *pa,* const void * *pb*)

```
00475                                                {
00476   int a = *(const int*)pa;
00477   int b = *(const int*)pb;
00478   return (b-a);
00479 }
```

4.1.4.29 int sortToner (const void * , const void *)

4.1.4.30 int sortTones (const void * *a,* const void * *b*)

A function to sort integers in ascending order, used by qsort

```
00507                                               {
00508   int *i1 = (int*) a, *i2 = (int*) b;
00509
00510   return *i1 - *i2;
00511 }
```

4.1.4.31 void weightingMatrix (moodWeighting *moodArray[],* int *mode,* int *tempo,* int *toneLength,* int *pitch,* int * *result*)

```
00457
      {
00458   for(int i = 0; i < AMOUNT_OF_MOODS; i++){
00459     result[i] = 0;
00460   }
00461
00462   for(int i = 0; i < AMOUNT_OF_MOODS; i++){
00463     result[i] += (moodArray[i].mode * mode);
00464     result[i] += (moodArray[i].tempo * tempo);
00465     result[i] += (moodArray[i].toneLength * toneLength);
00466     result[i] += (moodArray[i].pitch * pitch);
00467   }
00468
00469   for(int i = 0; i < AMOUNT_OF_MOODS; i++){
00470     printf("%s: %d\n", moodArray[i].name, result[i]);
00471   }
00472 }
```

4.1.5 Variable Documentation

4.1.5.1 int AMOUNT_OF_MOODS

Index

A
    main.c, 6
AMOUNT_OF_MOODS
    main.c, 17
afterTouch
    eventPlacement, 2
Asharp
    main.c, 6
average
    note, 3

B
    main.c, 6

C
    main.c, 6
CHARS
    main.c, 6
channelPressure
    eventPlacement, 2
checkDirectory
    main.c, 6
checkNextEvent
    main.c, 7
checkScale
    main.c, 7
controlChange
    eventPlacement, 2
countNotes
    main.c, 7
countTicks1
    main.c, 7
countTicks2
    main.c, 8
Csharp
    main.c, 6

D
    main.c, 6
data, 2
    key, 2
    mode, 2
    tempo, 2
deltaTimeToNoteLength
    main.c, 8
Dsharp
    main.c, 6

E
    main.c, 6
eventPlacement, 2
    afterTouch, 2
    channelPressure, 2
    controlChange, 2
    noteOff, 2
    noteOn, 2
    pitchWheel, 2
    programChange, 3

F
    main.c, 6
fillNote
    main.c, 8
fillSongData
    main.c, 8
findEvents
    main.c, 9
findMode
    main.c, 9
FindMoodAmount
    main.c, 10
findNoteLength
    main.c, 10
findTicks
    main.c, 10
Fsharp
    main.c, 6

G
    main.c, 6
getHex
    main.c, 11
glad
    main.c, 6
Gsharp
    main.c, 6

insertMoods
    main.c, 11
insertPlacement1
    main.c, 11
insertPlacement2
    main.c, 11
isInMajor
    main.c, 11
isInMinor
    main.c, 12
isInScale
    main.c, 12

key
    data, 2

length
    note, 3

main
    main.c, 13
main.c, 4
    A, 6
    AMOUNT_OF_MOODS, 17
    Asharp, 6
    B, 6

C, 6
CHARS, 6
checkDirectory, 6
checkNextEvent, 7
checkScale, 7
countNotes, 7
countTicks1, 7
countTicks2, 8
Csharp, 6
D, 6
deltaTimeToNoteLength, 8
Dsharp, 6
E, 6
F, 6
fillNote, 8
fillSongData, 8
findEvents, 9
findMode, 9
FindMoodAmount, 10
findNoteLength, 10
findTicks, 10
Fsharp, 6
G, 6
getHex, 11
glad, 6
Gsharp, 6
insertMoods, 11
insertPlacement1, 11
insertPlacement2, 11
isInMajor, 11
isInMinor, 12
isInScale, 12
main, 13
major, 6
minor, 6
mode, 6
mood, 6
printNote, 13
printResults, 14
printSongData, 15
SCALESIZE, 6
sad, 6
settingPoints, 16
sortResult, 17
sortToner, 17
sortTones, 17
tone, 6
weightingMatrix, 17
major
main.c, 6
minor
main.c, 6
mode
data, 2
main.c, 6
moodWeighting, 3
mood
main.c, 6
moodWeighting, 3
mode, 3
name, 3
pitch, 3
tempo, 3
toneLength, 3
name
moodWeighting, 3
note, 3
average, 3
length, 3
octave, 3
ticks, 3
tone, 3
noteOff
eventPlacement, 2
noteOn
eventPlacement, 2
octave
note, 3
parameter
points, 4
pitch
moodWeighting, 3
pitchWheel
eventPlacement, 2
point
points, 4
points, 4
parameter, 4
point, 4
printNote
main.c, 13
printResults
main.c, 14
printSongData
main.c, 15
programChange
eventPlacement, 3
SCALESIZE
main.c, 6
sad
main.c, 6
settingPoints
main.c, 16
sortResult
main.c, 17
sortToner
main.c, 17
sortTones
main.c, 17
tempo
data, 2
moodWeighting, 3

ticks
    note, 3
tone
    main.c, 6
    note, 3
toneLength
    moodWeighting, 3

weightingMatrix
    main.c, 17