A412

0.1

Generated by Doxygen 1.8.8

Thu Dec 11 2014 11:36:53

Contents

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 data Struct Reference

Data Fields

- unsigned int tempo
- mode mode
- tone key

3.1.1 Detailed Description

Definition at line 37 of file main.c.

3.1.2 Field Documentation

3.1.2.1 tone data::key

Definition at line 40 of file main.c.

3.1.2.2 mode data::mode

Definition at line 39 of file main.c.

3.1.2.3 unsigned int data::tempo

Definition at line 38 of file main.c.

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 Detailed Description

Definition at line 1 of file findEvents.c.

3.2.2 Field Documentation

3.2.2.1 int eventPlacement::afterTouch

Definition at line 4 of file findEvents.c.

3.2.2.2 int eventPlacement::channelPressure

Definition at line 7 of file findEvents.c.

3.2.2.3 int eventPlacement::controlChange

Definition at line 5 of file findEvents.c.

3.2.2.4 int eventPlacement::noteOff

Definition at line 3 of file findEvents.c.

3.2.2.5 int eventPlacement::noteOn

Definition at line 2 of file findEvents.c.

3.2.2.6 int eventPlacement::pitchWheel

Definition at line 8 of file findEvents.c.

3.2.2.7 int eventPlacement::programChange

Definition at line 6 of file findEvents.c.

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

- findEvents.c
- main.c

3.3 moodWeighting Struct Reference

Data Fields

- char name [25]

- int mode
- int tempo
- int toneLength
- int pitch

3.3.1 Detailed Description

Definition at line 48 of file main.c.

3.3.2 Field Documentation

3.3.2.1 int moodWeighting::mode

Definition at line 50 of file main.c.

3.3.2.2 char moodWeighting::name[25]

Definition at line 49 of file main.c.

3.3.2.3 int moodWeighting::pitch

Definition at line 53 of file main.c.

3.3.2.4 int moodWeighting::tempo

Definition at line 51 of file main.c.

3.3.2.5 int moodWeighting::toneLength

Definition at line 52 of file main.c.

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

3.4.1 Detailed Description

Definition at line 30 of file main.c.

3.4.2 Field Documentation

3.4.2.1 int note::average

Definition at line 34 of file main.c.

3.4.2.2 int note::length

Definition at line 33 of file main.c.

3.4.2.3 int note::octave

Definition at line 32 of file main.c.

3.4.2.4 int note::tone

Definition at line 31 of file main.c.

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 Detailed Description

Definition at line 43 of file main.c.

3.5.2 Field Documentation

3.5.2.1 char* points::parameter

Definition at line 44 of file main.c.

3.5.2.2 int points::point

Definition at line 45 of file main.c.

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

- main.c

Chapter 4

File Documentation

4.1 findEvents.c File Reference

Data Structures

- struct eventPlacement

Functions

- int main (void)
- void findEvents (int numbersInText, int hex[], eventPlacement placement[], note noteAr[], int ticks[])
- void insertPlacement1 (int hex[], int *place, int j, note noteAr[], int *n)
- void insertPlacement2 (int hex[], int *place, int j)
- int checkNextEvent (int hex[], int j)
- void findTicks (int numbersInText, int hex[], eventPlacement placement[], note noteAr[], int ticks[])
- void countTicks1 (int hex[], int *i, int deltaCounter, int ticks[], int *tickCounter)
- void countTicks2 (int hex[], int *i, int deltaCounter, int ticks[], int *tickCounter)

4.1.1 Function Documentation

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

Definition at line 54 of file findEvents.c.

```
54                                      {
55   switch (hex[j]){
56     case 0x90:
57     case 0x80:
58     case 0xA0:
59     case 0xB0:
60     case 0xC0:
61     case 0xD0:
62     case 0xE0: return 1; break;
63     default  : return 0; break;
64   }
65 }
```

4.1.1.2 void countTicks1 (int *hex[]*, int * *i*, int *deltaCounter*, int *ticks[]*, int * *tickCounter*)

Definition at line 103 of file findEvents.c.

```
103                                                                                                 {
104   while(deltaCounter < 7 && hex[(i + deltaCounter)] > 0x80)
105     ticks[tickCounter] += ((hex[(i + deltaCounter++)] - 0x80) * 128);
106   ticks[tickCounter++] += hex[(i + deltaCounter++)];
107   i += deltaCounter;
108 }
```

4.1.1.3 void countTicks2 (int *hex[]*, int * *i*, int *deltaCounter*, int *ticks[]*, int * *tickCounter*)

Definition at line 110 of file findEvents.c.

```
110                                                                                                 {
111   while(deltaCounter < 6 && hex[(i + deltaCounter)] > 0x80)
112     ticks[tickCounter] += ((hex[(i + deltaCounter++)] - 0x80) * 128);
113   ticks[tickCounter++] += hex[(i + deltaCounter++)];
114   i += deltaCounter;
115 }
```

4.1.1.4 void findEvents (int *numbersInText*, int *hex[]*, eventPlacement *placement[]*, note *noteAr[]*, int *ticks[]*)

Definition at line 16 of file findEvents.c.

```
16                                                                                                                  {
17   int noteOff = 0, noteOn = 0, afterTouch = 0, controlChange = 0,
18       programChange = 0, channelPressure = 0, pitchWheel = 0, i = 0, n = 0;
19
20   for(int j = 0; j < numbersInText; j++){
21     switch (hex[j]){
22       case 0x90: insertPlacement1(hex, &placement[noteOn++].noteOn, j, noteAr, &n);
             break;
23       case 0x80: insertPlacement1(hex, &placement[noteOff++].noteOff, j, noteAr, &n);
             break;
24       case 0xA0: insertPlacement1(hex, &placement[afterTouch++].afterTouch, j, noteAr, &n);
             break;
25       case 0xB0: insertPlacement1(hex, &placement[controlChange++].controlChange, j, noteAr
      , &n); break;
26       case 0xC0: insertPlacement2(hex, &placement[programChange++].programChange, j);
             break;
27       case 0xD0: insertPlacement2(hex, &placement[channelPressure++].channelPressure, j);
             break;
28       case 0xE0: insertPlacement1(hex, &placement[pitchWheel++].pitchWheel, j, noteAr, &n);
             break;
29       default  :                                                                                      break;
30     }
31   }
32   findTicks(numbersInText, hex, placement, noteAr, ticks);
33 }
```

4.1.1.5 void findTicks (int *numbersInText*, int *hex[]*, eventPlacement *placement[]*, note *noteAr[]*, int *ticks[]*)

Definition at line 67 of file findEvents.c.

```
67                                                                                                  {
68   int tickCounter = 0, deltaCounter1 = 3, deltaCounter2 = 2;
69
70   for(int j = 0; j < noteOn; j++){
71     for(int i = placement[j].noteOn; i < numbersInText; i++){
72       if(hex[i] == 0x80){
73         if(hex[i + 1] == noteAr[j])
74           break;
75         else{
76           countTicks1(hex, &i, deltaCounter1, ticks[], tickCounter);
77         }
78       }
79       else if(hex[i] == 0xA0){
80         if(hex[i + 1] == noteAr[j] && hex[i + 2] == 0x00)
81           break;
82         else{
83           countTicks1(hex, &i, deltaCounter1, ticks[], tickCounter);
84         }
85       }
86       else if(hex[i] == 0xD0){
```

```
87          if(hex[i + 1] == 0x00)
88            break;
89          else{
90            countTicks2(hex, &i, deltaCounter2, ticks[], tickCounter);
91          }
92        }
93        else if(hex[start] == 0xC0){
94          countTicks2(hex, &i, deltaCounter2, ticks[], tickCounter);
95        }
96        else{
97          countTicks1(hex, &i, deltaCounter1, ticks[], tickCounter);
98        }
99      }
100    }
101 }
```

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

Definition at line 35 of file findEvents.c.

```
35                                                                    {
36   int i = 3;
37   while(i < 7 && hex[(j + i++)] > 0x80);
38   if(checkNextEvent(hex, (j + i))){
39     *place = j;
40     if(hex[j] == 0x90){
41       fillNote(hex[j + 1], &noteAr[*n]);
42       *n += 1;
43     }
44   }
45 }
```

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

Definition at line 47 of file findEvents.c.

```
47                                                  {
48   int i = 2;
49   while(i < 6 && hex[(j + i++)] > 0x80);
50   if(checkNextEvent(hex, (j + i)))
51     *place = j;
52 }
```

4.1.1.8 int main (void)

Definition at line 11 of file findEvents.c.

```
11              {
12   int ticks[numbersInText];
13   return 0;
14 }
```

4.2 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 *)
- 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 *)
- int weightingMatrix (moodWeighting[], int, int, int, int)
- void findEvents (int, int[], eventPlacement[], note[], int[], int *)
- void insertPlacement1 (int[], int *, int, note[], int *)
- void insertPlacement2 (int[], int *, int)
- int checkNextEvent (int[], int)
- void findTicks (int, int[], eventPlacement[], note[], int[], int, int *)
- void countTicks1 (int[], int *, int, int[], int *)
- void countTicks2 (int[], int *, int, int[], int *)
- int sortResult (const void *, const void *)
- void deltaTimeToNoteLength (int *, 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 *)
- int main (int argc, const char *argv[])
- int sortTones (const void *a, const void *b)
- void findMode (note noteAr[], int totalNotes, data *data)

Variables

- int AMOUNT_OF_MOODS

4.2.1 Macro Definition Documentation

4.2.1.1 #define CHARS 1000

Definition at line 21 of file main.c.

4.2.1.2 #define SCALESIZE 7

Definition at line 23 of file main.c.

4.2.2 Typedef Documentation

4.2.2.1 typedef enum mode mode

4.2.2.2 typedef enum mood mood

4.2.2.3 typedef enum tone tone

4.2.3 Enumeration Type Documentation

4.2.3.1 enum mode

Enumerator

major

minor

Definition at line 26 of file main.c.

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

4.2.3.2 enum mood

Enumerator

glad

sad

Definition at line 28 of file main.c.

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

4.2.3.3 enum tone

Enumerator

C

Csharp

D

Dsharp

E

F

Fsharp

G

Gsharp

A

Asharp

B

Definition at line 27 of file main.c.

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

4.2.4 Function Documentation

4.2.4.1 void checkDirectory (char * *MIDIfile*)

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

Parameters

char]*	MIDIfile: a pointer to a string containing the name of the chosen input file

Definition at line 162 of file main.c.

```
162                                     {
163   DIR *dir;
164   struct dirent *musicDir;
165   if ((dir = opendir ("./Music")) != NULL) {
166     printf("Mulige numre\n");
167     /* print all the files and directories within specified directory */
168       while ((musicDir = readdir (dir)) != NULL) {
169         printf ("%s\n", musicDir->d_name);
170       }
171     closedir (dir);
172   }
173   else {
174   /* Could not open directory */
175     perror ("Failure while opening directory");
176     exit (EXIT_FAILURE);
177   }
178   printf("Indtast det valgte nummer\n");
179   scanf("%s", MIDIfile);
180 }
```

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

Definition at line 264 of file main.c.

```
264                                      {
265   switch (hex[j]){
266     case 0x90:
267     case 0x80:
```

```
268     case 0xA0:
269     case 0xB0:
270     case 0xC0:
271     case 0xD0:
272     case 0xE0: return 1; break;
273     default  : return 0; break;
274   }
275 }
```

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

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

Parameters

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

Definition at line 200 of file main.c.

```
200                                     {
201   int i = 0, res = 0;
202   for(i = 0; i < amount; i++){
203     if(hex[i] == 0x90){
204       res++;
205     }
206   }
207   return res;
208 }
```

4.2.4.4 void countTicks1 (int *hex[]*, int * *i*, int *deltaCounter*, int *ticks[]*, int * *tickCounter*)

Definition at line 313 of file main.c.

```
313                                                                                 {
314   while(deltaCounter < 7 && hex[(*i + deltaCounter)] > 0x80)
315     ticks[*tickCounter] += ((hex[(*i + deltaCounter++)] - 0x80) * 128);
316   ticks[*tickCounter++] += hex[(*i + deltaCounter++)];
317   i += deltaCounter;
318 }
```

4.2.4.5 void countTicks2 (int *hex[]*, int * *i*, int *deltaCounter*, int *ticks[]*, int * *tickCounter*)

Definition at line 320 of file main.c.

```
320                                                                                 {
321   while(deltaCounter < 6 && hex[(*i + deltaCounter)] > 0x80)
322     ticks[*tickCounter] += ((hex[(*i + deltaCounter++)] - 0x80) * 128);
323   ticks[*tickCounter++] += hex[(*i + deltaCounter++)];
324   i += deltaCounter;
325 }
```

4.2.4.6 void deltaTimeToNoteLength (int * *ticks*, int *ppqn*, int *size*, note * *noteAr*)

Definition at line 485 of file main.c.

```
485                                                                     {
486
487   for (int i = 0; i < size; i++){
488
489     double noteLength = ((double) (ticks[i])) / ((double) (ppqn/8));
490
491     if (noteLength < 1.5 && noteLength >= 0)
492       noteLength = 1;
493     else if (noteLength < 3 && noteLength >= 1.5)
```

```
494       noteLength = 2;
495     else if (noteLength < 6 && noteLength >= 3)
496       noteLength = 4;
497     else if (noteLength < 12 && noteLength >= 6)
498       noteLength = 8;
499     else if (noteLength < 24 && noteLength >= 12)
500       noteLength = 16;
501     else
502       noteLength = 32;
503 
504         noteAr[i].length = noteLength;
505     }
506 }
```

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

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

Parameters

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

Definition at line 331 of file main.c.

```
331                                         {
332   note->tone = inputTone % 12;
333   note->average = inputTone;
334   note->octave = inputTone / 12;
335 }
```

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

A function, that fills out the song data

Parameters

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

Definition at line 215 of file main.c.

```
215                                                         {
216   int j;
217   /*Find the mode of the song, initialised as minor atm*/
218   for(j = 0; j < numbersInText; j++){
219     /* finds the tempo */
220     if(hex[j] == 0xff && hex[j+1] == 0x51 && hex[j+2] == 0x03){
221       data->tempo =  60000000/((hex[j+3] << 16) | (hex[j+4] << 8) | (hex[j+5]));
222     }
223   }
224 }
```

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

Definition at line 226 of file main.c.

```
226 
          {
227   int noteOff = 0, noteOn = 0, afterTouch = 0, controlChange = 0,
228       programChange = 0, channelPressure = 0, pitchWheel = 0, n = 0;
229 
230   for(int j = 0; j < numbersInText; j++){
231     switch (hex[j]){
232       case 0x90: insertPlacement1(hex, &placement[noteOn++].noteOn, j, noteAr, &n);
             break;
```

```
233       case 0x80: insertPlacement1(hex, &placement[noteOff++].noteOff, j, noteAr, &n);
              break;
234       case 0xA0: insertPlacement1(hex, &placement[afterTouch++].afterTouch, j, noteAr, &n);
              break;
235       case 0xB0: insertPlacement1(hex, &placement[controlChange++].controlChange, j, noteAr
      , &n); break;
236       case 0xC0: insertPlacement2(hex, &placement[programChange++].programChange, j);
              break;
237       case 0xD0: insertPlacement2(hex, &placement[channelPressure++].channelPressure, j);
              break;
238       case 0xE0: insertPlacement1(hex, &placement[pitchWheel++].pitchWheel, j, noteAr, &n);
              break;
239       default  :                                                                    break;
240     }
241   }
242   findTicks(numbersInText, hex, placement, noteAr, ticks, noteOn, size);
243 }
```

4.2.4.10 void findMode (note * , int , data *)

4.2.4.11 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

note[]]	noteAr: An array of all the notes in the entire song
int]	totalNotes: The number of notes in the song

Definition at line 520 of file main.c.

```
520                                                     {
521   int x = 0, y = 0, z = 0, bar[4], sizeBar = 4, tempSpan = 999, span = 999, keynote = 0,
      mode = 0;
522
523   /*Goes through all notes of the song and puts them into an array*/
524   while(x < totalNotes){
525     for(y = 0; y < sizeBar; y++, x++){
526       bar[y] = noteAr[x].tone;
527     }
528
529     if(y == sizeBar){
530       span = 999;
531       /*Sort notes in acsending order*/
532       qsort(bar, sizeBar, sizeof(tone), sortTones);
533
534       /*Find the lowest possible tonespan over the entire array of notes*/
535       for(z = 0; z < 4; z++){
536     if((z + 1) > 3)
537           tempSpan = (bar[(z+1)%4]+12)-bar[z] + bar[(z+2)%4]-bar[(z+1)%4] + bar[(z+3)%4]-bar[(z+2)%4];
538         else if((z + 2) > 3)
539           tempSpan = bar[(z+1)]-bar[z] + (bar[(z+2)%4]+12)-bar[(z+1)%4] + bar[(z+3)%4]-bar[(z+2)%4];
540     else if((z +3) > 3)
541           tempSpan = bar[(z+1)]-bar[z] + bar[(z+2)]-bar[(z+1)] + (bar[(z+3)%4]+12)-bar[z];
542     else
543           tempSpan = bar[(z+1)]-bar[z] + bar[(z+2)]-bar[(z+1)] + bar[(z+3)]-bar[(z+2)];
544
545     if(tempSpan < span){
546           span = tempSpan;
547           keynote = bar[z];
548         }
549       }
550       mode += isInScale(keynote, bar, sizeBar);
551       printf("Moden er nu: %d\n", mode);
552     }
553   data->key = keynote;
554   if(mode > 0)
555     data->mode = major;
556   else if(mode < 0)
557     data->mode = minor;
558   }
559 }
```

4.2.4.12 int FindMoodAmount (FILE * *moods*)

Definition at line 621 of file main.c.

```
621                                {
622   int i = 1;
623   while(fgetc(moods) != EOF){
624     if(fgetc(moods) == '\n')
625       i++;
626   }
627   rewind(moods);
628   return i;
629 }
```

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

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

Definition at line 277 of file main.c.

```
277
                     {
278   int tickCounter = 0, deltaCounter1 = 3, deltaCounter2 = 2;
279
280   for(int j = 0; j < noteOn; j++){
281     for(int i = placement[j].noteOn; i < numbersInText; i++){
282       if(hex[i] == 0x80){
283         if(hex[i + 1] == noteAr[j].tone)
284           break;
285         else{
286           countTicks1(hex, &i, deltaCounter1, ticks, &tickCounter);
287         }
288       }
289       else if(hex[i] == 0xA0){
290         if(hex[i + 1] == noteAr[j].tone && hex[i + 2] == 0x00)
291           break;
292         else{
293           countTicks1(hex, &i, deltaCounter1, ticks, &tickCounter);
294         }
295       }
296       else if(hex[i] == 0xD0){
297         if(hex[i + 1] == 0x00)
298           break;
299         else{
300           countTicks2(hex, &i, deltaCounter2, ticks, &tickCounter);
301         }
302       }
303       else if(hex[i] == 0xC0){
304         countTicks2(hex, &i, deltaCounter2, ticks, &tickCounter);
305       }
306       else{
307         countTicks1(hex, &i, deltaCounter1, ticks, &tickCounter);
308       }
309     }
310   }
311 }
```

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

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

Parameters

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

Definition at line 186 of file main.c.

```
186                                {
187   int i = 0, c;
```

```
188   while( (c = fgetc(f)) != EOF && i < CHARS){
189     hexAr[i] = c;
190     i++;
191   }
192
193   return i;
194 }
```

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

Definition at line 444 of file main.c.

```
444                                                          {
445   printf("hej er her");
446   for(int i = 0; i < AMOUNT_OF_MOODS; i++){
447     fscanf(moods, "%s %d %d %d %d", moodArray[i].name , &moodArray[i].mode,
448                                     &moodArray[i].tempo, &moodArray[i].toneLength,
449                                     &moodArray[i].pitch);
450   }
451   printf("jeg er også her");
452 }
```

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

Definition at line 245 of file main.c.

```
245                                                                                {
246   int i = 3;
247   while(i < 7 && hex[(j + i++)] > 0x80);
248   if(checkNextEvent(hex, (j + i))){
249     *place = j;
250     if(hex[j] == 0x90){
251       fillNote(hex[j + 1], &noteAr[*n]);
252       *n += 1;
253     }
254   }
255 }
```

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

Definition at line 257 of file main.c.

```
257                                                  {
258   int i = 2;
259   while(i < 6 && hex[(j + i++)] > 0x80);
260   if(checkNextEvent(hex, (j + i)))
261     *place = j;
262 }
```

4.2.4.19 int isInMajor (int *toneLeap*)

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

Parameters

int]	toneLeap: An integer describing the processed tone leap

Returns

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

Definition at line 611 of file main.c.

```
611                             {
612   int major[] = {0, 2, 4, 5, 7, 9, 11};
613
614   for(int i = 0; i < SCALESIZE; i++){
615     if(toneLeap == major[i])
616       return 1;
617   }
618   return 0;
619 }
```

4.2.4.20 int isInMinor (int *toneLeap*)

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

Parameters

int]	toneLeap: An integer describing the processed tone leap

Returns

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

Definition at line 597 of file main.c.

```
597                             {
598   int minor[] = {0, 2, 3, 5, 7, 8, 10};
599
600   for(int i = 0; i < SCALESIZE; i++){
601     if(toneLeap == minor[i])
602       return 1;
603   }
604   return 0;
605 }
```

4.2.4.21 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

scale]	mode: An enum that describes the given mode
int]	keytone: The keytone of the processed scale
int]	otherTones[]: An array of the rest of the tones, which the function compares to the keytone and mode
int]	size: The number of tones in the otherTones array

Returns

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

Definition at line 568 of file main.c.

```
568                                                     {
569   int toneLeap, isMinor = 1, isMajor = 1;
570
571   for(int i = 0; i < size; i++){
572     if(otherTones[i] < keytone)
573       otherTones[i] += 12;
574       toneLeap = otherTones[i] - keytone;
575
576       if(isMinor)
577         isMinor = isInMinor(toneLeap);
578       if(isMajor)
579         isMajor = isInMajor(toneLeap);
580     }
581
582     if(isMinor && isMajor)
```

```
583       return 0;
584     else if(isMinor)
585       return -1;
586     if(isMajor)
587       return 1;
588
589     return 0;
590 }
```

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

Definition at line 94 of file main.c.

```
94                                      {
95   FILE *f;
96   char MIDIfile[25];
97   /*Variables*/
98   int numbersInText = 0, notes, i = 0, size = 0, moodOfMelodi = 0;
99   /* PLACEHOLDER FIX THIS */
100   int mode = 5, tempo = 5, toneLength = 5, pitch = 5;
101   FILE* moods = fopen("moods.txt", "r");
102   if(moods == NULL){
103     perror("Error: moods missing ");
104     exit(EXIT_FAILURE);
105   }
106   AMOUNT_OF_MOODS = FindMoodAmount(moods);
107   moodWeighting moodArray[AMOUNT_OF_MOODS];
108   data data = {0, major, D};
109   if (argv[1] == NULL){
110     checkDirectory(MIDIfile);
111     f = fopen(MIDIfile,"r");
112     if(f == NULL){
113       perror("Error opening file");
114       exit(EXIT_FAILURE);
115     }
116   }
117   else if(argv[1] != NULL){
118     f = fopen(argv[1],"r");
119     if(f == NULL){
120       perror("Error opening file");
121       exit(EXIT_FAILURE);
122     }
123   }
124
125   int *hex = (int *) malloc(CHARS * sizeof(int));
126   if(hex == NULL){
127     printf("Memory allocation failed, bye!");
128     exit(EXIT_FAILURE);
129   }
130   /*Reading the data from the file*/
131   numbersInText = getHex(f, hex);
132   fillSongData(&data, hex, numbersInText);
133   notes = countNotes(hex, numbersInText);
134   note *noteAr = (note*) malloc(notes * sizeof(note));
135   if(noteAr == NULL){
136     printf("Memory allocation failed, bye!");
137     exit(EXIT_FAILURE);
138   }
139   eventPlacement placement[numbersInText];
140   int ticks[numbersInText];
141   findEvents(numbersInText, hex, placement, noteAr, ticks, &size);
142   insertMoods(moodArray, moods);
143   settingPoints(&mode, &tempo, &toneLength, &pitch, data, notes, noteAr, &size);
144   printf("%d, %d, %d, %d\n", mode, tempo, toneLength, pitch);
145   for(i = 0; i < notes; i++)
146     printNote(noteAr[i]);
147   findMode(noteAr, notes, &data);
148   printSongData(data);
149   moodOfMelodi = weightingMatrix(moodArray, mode, tempo, toneLength, pitch);
150   printf("%d\n", moodOfMelodi);
151
152
153   /*Clean up and close*/
154   fclose(f);
155   free(hex);
156   free(noteAr);
157
158   return 0;
159 }
```

4.2.4.23 void printNote (note *note*)

A function to print the note

Parameters

note]	note: the note structure to be printed

Definition at line 340 of file main.c.

```
340                          {
341   printf("Tone: ");
342
343   switch (note.tone){
344     case C     : printf("C") ; break;
345     case Csharp: printf("C#"); break;
346     case D     : printf("D") ; break;
347     case Dsharp: printf("D#"); break;
348     case E     : printf("E") ; break;
349     case F     : printf("F") ; break;
350     case Fsharp: printf("F#"); break;
351     case G     : printf("G") ; break;
352     case Gsharp: printf("G#"); break;
353     case A     : printf("A") ; break;
354     case Asharp: printf("A#"); break;
355     case B     : printf("B") ; break;
356     default    : printf("Undefined note"); break;
357   }
358   printf(", octave: %d\n", note.octave);
359 }
```

4.2.4.24 void printSongData (data *data*)

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

Parameters

data]	data: the data to be printed

Definition at line 364 of file main.c.

```
364                              {
365   printf("Tempo: %d\nMode: ", data.tempo);
366   switch(data.mode){
367     case minor: printf("minor"); break;
368     case major: printf("major"); break;
369     default: printf("unknown mode"); break;
370   }
371   printf("\nKeytone: %d", data.key);
372   putchar('\n');
373 }
```

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

Definition at line 375 of file main.c.

```
375
            {
376   int deltaTime = 2, combined = 0, averageNote = 0;
377   switch(data.mode){
378     case minor: *mode = -5; break;
379     case major: *mode = 5; break;
380     default: *mode = 0; break;
381   }
382   if(data.tempo < 60)
383     *tempo = -5;
384   else if(data.tempo >= 60 && data.tempo < 70)
385     *tempo = -4;
386   else if(data.tempo >= 70 && data.tempo < 80)
387     *tempo = -3;
388   else if(data.tempo >= 80 && data.tempo < 90)
389     *tempo = -2;
390   else if(data.tempo >= 90 && data.tempo < 100)
391     *tempo = -1;
392   else if(data.tempo >= 100 && data.tempo < 120)
393     *tempo =  0;
```

```
394   else if(data.tempo >= 120 && data.tempo < 130)
395     *tempo =  1;
396   else if(data.tempo >= 130 && data.tempo < 140)
397     *tempo =  2;
398   else if(data.tempo >= 140 && data.tempo < 150)
399     *tempo =  3;
400   else if(data.tempo >= 150 && data.tempo < 160)
401     *tempo =  4;
402   else if(data.tempo >=  160)
403     *tempo =  5;
404
405   switch(deltaTime){
406     case 1: *length = -5; break;
407     case 2: *length = -4; break;
408     case 4: *length = -2; break;
409     case 8: *length =  0; break;
410     case 16: *length = 3; break;
411     case 32: *length = 5; break;
412   }
413   for (int i = 0; i < notes; i++){
414     combined += noteAr[i].average;
415   }
416   averageNote = combined/notes;
417
418   if(averageNote <= 16)
419     *octave = -5;
420   else if(averageNote >= 17 && averageNote <= 23)
421     *octave = -4;
422   else if(averageNote >= 24 && averageNote <= 30)
423     *octave = -3;
424   else if(averageNote >= 31 && averageNote <= 37)
425     *octave = -2;
426   else if(averageNote >= 38 && averageNote <= 44)
427     *octave = -1;
428   else if(averageNote >= 45 && averageNote <= 51)
429     *octave = 0;
430   else if(averageNote >= 52 && averageNote <= 58)
431     *octave = 1;
432   else if(averageNote >= 59 && averageNote <= 65)
433     *octave = 2;
434   else if(averageNote >= 66 && averageNote <= 72)
435     *octave = 3;
436   else if(averageNote >= 73 && averageNote <= 79)
437     *octave = 4;
438   else if(averageNote >=80)
439     *octave = 5;
440 }
```

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

Definition at line 478 of file main.c.

```
478                                                 {
479   int a = *(const int*)pa;
480   int b = *(const int*)pb;
481   return (b-a);
482 }
```

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

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

A function to sort integers in ascending order.

Definition at line 510 of file main.c.

```
510                                             {
511   int *i1 = (int*) a, *i2 = (int*) b;
512
513   return (int) *i1 - *i2;
514 }
```

4.2.4.29 int weightingMatrix (moodWeighting *moodArray[]*, int *mode*, int *tempo*, int *toneLength*, int *pitch*)

Definition at line 455 of file main.c.

```
455                                                                                                                 {
456   int result[AMOUNT_OF_MOODS];
457
458   for(int i = 0; i < AMOUNT_OF_MOODS; i++){
459     result[i] = 0;
460   }
461
462   for(int i = 0; i < AMOUNT_OF_MOODS; i++){
463     result[i] += (moodArray[i].mode * mode);
464     result[i] += (moodArray[i].tempo * tempo);
465     result[i] += (moodArray[i].toneLength * toneLength);
466     result[i] += (moodArray[i].pitch * pitch);
467   }
468
469   for(int i = 0; i < AMOUNT_OF_MOODS; i++){
470     printf("%s: %d\n", moodArray[i].name, result[i]);
471   }
472
473   qsort(result, AMOUNT_OF_MOODS, sizeof(int), sortResult);
474   return result[0];
475 }
```

4.2.5 Variable Documentation

4.2.5.1 int AMOUNT_OF_MOODS

Definition at line 22 of file main.c.

4.3 test.c File Reference

```
#include <stdlib.h>
#include <stdio.h>
```

Functions

- int main (void)
- void testFunk (void)

4.3.1 Function Documentation

4.3.1.1 int main (void)

Definition at line 3 of file test.c.

```
3               {
4   printf("Jonas er en kagemand!\nOg han har lange løg.\n");
5
6   return 0;
7 }
```

4.3.1.2 void testFunk (void)

Definition at line 12 of file test.c.

```
12                  {
13   int stuff = 1337;
14 }
```

Index