A412

0.1

Generated by Doxygen 1.8.8

Thu Dec 11 2014 14:13:10

Contents

1 Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

2 File Index

2.1 File List

Here is a list of all files with brief descriptions:

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 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 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**

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::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 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*)

```
                                  {
  switch (hex[j]){
    case 0x90:
    case 0x80:
    case 0xA0:
    case 0xB0:
    case 0xC0:
    case 0xD0:
    case 0xE0: return 1; break;
    default  : return 0; break;
  }
}
```

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

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

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

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

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

```
                                                                                          {
  int noteOff = 0, noteOn = 0, afterTouch = 0, controlChange = 0,
      programChange = 0, channelPressure = 0, pitchWheel = 0, i = 0, n = 0;

  for(int j = 0; j < numbersInText; j++){
    switch (hex[j]){
      case 0x90: insertPlacement1(hex, &placement[noteOn++].noteOn, j, noteAr, &n);             break;
      case 0x80: insertPlacement1(hex, &placement[noteOff++].noteOff, j, noteAr, &n);           break;
      case 0xA0: insertPlacement1(hex, &placement[afterTouch++].afterTouch, j, noteAr, &n);     break;
      case 0xB0: insertPlacement1(hex, &placement[controlChange++].controlChange, j, noteAr, &n); break;
      case 0xC0: insertPlacement2(hex, &placement[programChange++].programChange, j);           break;
      case 0xD0: insertPlacement2(hex, &placement[channelPressure++].channelPressure, j);       break;
      case 0xE0: insertPlacement1(hex, &placement[pitchWheel++].pitchWheel, j, noteAr, &n);     break;
      default  :                                                                                break;
    }
  }
  findTicks(numbersInText, hex, placement, noteAr, ticks);
}
```

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

```
                                                                                         {
  int tickCounter = 0, deltaCounter1 = 3, deltaCounter2 = 2;

  for(int j = 0; j < noteOn; j++){
    for(int i = placement[j].noteOn; i < numbersInText; i++){
      if(hex[i] == 0x80){
        if(hex[i + 1] == noteAr[j])
          break;
        else{
          countTicks1(hex, &i, deltaCounter1, ticks[], tickCounter);
        }
      }
      else if(hex[i] == 0xA0){
        if(hex[i + 1] == noteAr[j] && hex[i + 2] == 0x00)
          break;
        else{
          countTicks1(hex, &i, deltaCounter1, ticks[], tickCounter);
        }
      }
```

```
00086       else if(hex[i] == 0xD0){
00087         if(hex[i + 1] == 0x00)
00088           break;
00089         else{
00090           countTicks2(hex, &i, deltaCounter2, ticks[], tickCounter);
00091         }
00092       }
00093       else if(hex[start] == 0xC0){
00094         countTicks2(hex, &i, deltaCounter2, ticks[], tickCounter);
00095       }
00096       else{
00097         countTicks1(hex, &i, deltaCounter1, ticks[], tickCounter);
00098       }
00099     }
00100   }
00101 }
```

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

```
00035                                                                    {
00036   int i = 3;
00037   while(i < 7 && hex[(j + i++)] > 0x80);
00038   if(checkNextEvent(hex, (j + i))){
00039     *place = j;
00040     if(hex[j] == 0x90){
00041       fillNote(hex[j + 1], &noteAr[*n]);
00042       *n += 1;
00043     }
00044   }
00045 }
```

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

```
00047                                                 {
00048   int i = 2;
00049   while(i < 6 && hex[(j + i++)] > 0x80);
00050   if(checkNextEvent(hex, (j + i)))
00051     *place = j;
00052 }
```

4.1.1.8 int main (void)

```
00011               {
00012   int ticks[numbersInText];
00013   return 0;
00014 }
```

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

4.2.1.2 #define SCALESIZE 7

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

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

4.2.3.2 enum mood

Enumerator

glad

sad

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

4.2.3.3 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;
```

4.2.4 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

```
00162                                   {
00163   DIR *dir;
00164   struct dirent *musicDir;
00165   if ((dir = opendir ("./Music")) != NULL) {
00166     printf("Mulige numre\n");
00167     /* print all the files and directories within specified directory */
00168       while ((musicDir = readdir (dir)) != NULL) {
00169         printf ("%s\n", musicDir->d_name);
00170       }
00171     closedir (dir);
00172   }
00173   else {
00174   /* Could not open directory */
00175     perror ("Failure while opening directory");
00176     exit (EXIT_FAILURE);
00177   }
00178   printf("Indtast det valgte nummer\n");
00179   scanf("%s", MIDIfile);
00180 }
```

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

```
00265                                   {
00266   switch (hex[j]){
00267     case 0x90:
00268     case 0x80:
00269     case 0xA0:
00270     case 0xB0:
00271     case 0xC0:
00272     case 0xD0:
00273     case 0xE0: return 1; break;
00274     default  : return 0; break;
00275   }
00276 }
```

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

```
00200                                    {
00201   int i = 0, res = 0;
00202   for(i = 0; i < amount; i++){
00203     if(hex[i] == 0x90){
00204       res++;
00205     }
00206   }
00207   return res;
00208 }
```

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

```
00314                                                                                   {
00315   while(deltaCounter < 7 && hex[(*i + deltaCounter)] > 0x80)
00316     ticks[*tickCounter] += ((hex[(*i + deltaCounter++)] - 0x80) * 128);
00317   ticks[*tickCounter++] += hex[(*i + deltaCounter++)];
00318   i += deltaCounter;
00319 }
```

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

```
00321                                                                                   {
00322   while(deltaCounter < 6 && hex[(*i + deltaCounter)] > 0x80)
00323     ticks[*tickCounter] += ((hex[(*i + deltaCounter++)] - 0x80) * 128);
00324   ticks[*tickCounter++] += hex[(*i + deltaCounter++)];
00325   i += deltaCounter;
00326 }
```

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

```
00484                                                                                {
00485
00486   for (int i = 0; i < size; i++){
00487
00488     double noteLength = ((double) (ticks[i])) / ((double) (ppqn/8));
00489
00490     if (noteLength < 1.5 && noteLength >= 0)
00491       noteLength = 1;
00492     else if (noteLength < 3 && noteLength >= 1.5)
00493       noteLength = 2;
00494     else if (noteLength < 6 && noteLength >= 3)
00495       noteLength = 4;
00496     else if (noteLength < 12 && noteLength >= 6)
00497       noteLength = 8;
00498     else if (noteLength < 24 && noteLength >= 12)
00499       noteLength = 16;
00500     else
00501       noteLength = 32;
00502
00503         noteAr[i].length = noteLength;
00504     }
00505 }
```

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

```
00332                                          {
00333   note->tone = inputTone % 12;
00334   note->average = inputTone;
00335   note->octave = inputTone / 12;
00336 }
```

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

```
00216                                                          {
00217   int j;
00218   /*Find the mode of the song, initialised as minor atm*/
00219   for(j = 0; j < numbersInText; j++){
00220     /* finds the tempo */
00221     if(hex[j] == 0xff && hex[j+1] == 0x51 && hex[j+2] == 0x03){
00222       data->tempo =  60000000/((hex[j+3] << 16) | (hex[j+4] << 8) | (hex[j+5]));
00223     }
00224   }
00225 }
```

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

```
00227
          {
00228   int noteOff = 0, noteOn = 0, afterTouch = 0, controlChange = 0,
00229       programChange = 0, channelPressure = 0, pitchWheel = 0, n = 0;
00230
00231   for(int j = 0; j < numbersInText; j++){
00232     switch (hex[j]){
00233       case 0x90: insertPlacement1(hex, &placement[noteOn++].noteOn, j, noteAr, &n);              break;
00234       case 0x80: insertPlacement1(hex, &placement[noteOff++].noteOff, j, noteAr, &n);            break;
00235       case 0xA0: insertPlacement1(hex, &placement[afterTouch++].afterTouch, j, noteAr, &n);      break;
```

```
00236       case 0xB0: insertPlacement1(hex, &placement[controlChange++].controlChange, j, noteAr, &n); break;
00237       case 0xC0: insertPlacement2(hex, &placement[programChange++].programChange, j);             break;
00238       case 0xD0: insertPlacement2(hex, &placement[channelPressure++].channelPressure, j);         break;
00239       case 0xE0: insertPlacement1(hex, &placement[pitchWheel++].pitchWheel, j, noteAr, &n);       break;
00240       default  :                                                                                  break;
00241     }
00242   }
00243   findTicks(numbersInText, hex, placement, noteAr, ticks, noteOn, size);
00244 }
```

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

```
00519                                                       {
00520   int x = 0, y = 0, z = 0, bar[4], sizeBar = 4, tempSpan = 999, span = 999, keynote = 0,
      mode = 0;
00521
00522   /*Goes through all notes of the song and puts them into an array*/
00523   while(x < totalNotes){
00524     for(y = 0; y < sizeBar; y++, x++){
00525       bar[y] = noteAr[x].tone;
00526     }
00527
00528     if(y == sizeBar){
00529       span = 999;
00530       /*Sort notes in acsending order*/
00531       qsort(bar, sizeBar, sizeof(tone), sortTones);
00532
00533       /*Find the lowest possible tonespan over the entire array of notes*/
00534       for(z = 0; z < 4; z++){
00535     if((z + 1) > 3)
00536           tempSpan = (bar[(z+1)%4]+12)-bar[z] + bar[(z+2)%4]-bar[(z+1)%4] + bar[(z+3)%4]-bar[(z+2)%4];
00537         else if((z + 2) > 3)
00538           tempSpan = bar[(z+1)]-bar[z] + (bar[(z+2)%4]+12)-bar[(z+1)%4] + bar[(z+3)%4]-bar[(z+2)%4];
00539     else if((z +3) > 3)
00540           tempSpan = bar[(z+1)]-bar[z] + bar[(z+2)]-bar[(z+1)] + (bar[(z+3)%4]+12)-bar[z];
00541     else
00542           tempSpan = bar[(z+1)]-bar[z] + bar[(z+2)]-bar[(z+1)] + bar[(z+3)]-bar[(z+2)];
00543
00544     if(tempSpan < span){
00545           span = tempSpan;
00546           keynote = bar[z];
00547         }
00548       }
00549       mode += isInScale(keynote, bar, sizeBar);
00550       printf("Moden er nu: %d\n", mode);
00551     }
00552   data->key = keynote;
00553   if(mode > 0)
00554     data->mode = major;
00555   else if(mode < 0)
00556     data->mode = minor;
00557   }
00558 }
```

4.2.4.12 int FindMoodAmount (FILE * *moods*)

```
00620                                 {
00621   int i = 1;
00622   while(fgetc(moods) != EOF){
00623     if(fgetc(moods) == '\n')
00624       i++;
00625   }
00626   rewind(moods);
00627   return i;
00628 }
```

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*)

```
00278
                    {
00279   int tickCounter = 0, deltaCounter1 = 3, deltaCounter2 = 2;
00280
00281   for(int j = 0; j < noteOn; j++){
00282     for(int i = placement[j].noteOn; i < numbersInText; i++){
00283       if(hex[i] == 0x80){
00284         if(hex[i + 1] == noteAr[j].tone)
00285           break;
00286         else{
00287           countTicks1(hex, &i, deltaCounter1, ticks, &tickCounter);
00288         }
00289       }
00290       else if(hex[i] == 0xA0){
00291         if(hex[i + 1] == noteAr[j].tone && hex[i + 2] == 0x00)
00292           break;
00293         else{
00294           countTicks1(hex, &i, deltaCounter1, ticks, &tickCounter);
00295         }
00296       }
00297       else if(hex[i] == 0xD0){
00298         if(hex[i + 1] == 0x00)
00299           break;
00300         else{
00301           countTicks2(hex, &i, deltaCounter2, ticks, &tickCounter);
00302         }
00303       }
00304       else if(hex[i] == 0xC0){
00305         countTicks2(hex, &i, deltaCounter2, ticks, &tickCounter);
00306       }
00307       else{
00308         countTicks1(hex, &i, deltaCounter1, ticks, &tickCounter);
00309       }
00310     }
00311   }
00312 }
```

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

```
00186                                   {
00187   int i = 0, c;
00188   while( (c = fgetc(f)) != EOF && i < CHARS){
00189     hexAr[i] = c;
00190     i++;
00191   }
00192
00193   return i;
00194 }
```

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

```
00445                                                           {
00446   for(int i = 0; i < AMOUNT_OF_MOODS; i++){
00447     fscanf(moods, "%s %d %d %d %d", moodArray[i].name , &moodArray[i].mode,
00448                                     &moodArray[i].tempo, &moodArray[i].toneLength,
00449                                     &moodArray[i].pitch);
00450   }
00451 }
```

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

```
00246                                                                                   {
00247   int i = 3;
00248   while(i < 7 && hex[(j + i++)] > 0x80);
```

```
00249   if(checkNextEvent(hex, (j + i))){
00250     *place = j;
00251     if(hex[j] == 0x90){
00252       fillNote(hex[j + 1], &noteAr[*n]);
00253       *n += 1;
00254     }
00255   }
00256 }
```

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

```
00258                                                   {
00259   int i = 2;
00260   while(i < 6 && hex[(j + i++)] > 0x80);
00261   if(checkNextEvent(hex, (j + i)))
00262     *place = j;
00263 }
```

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.

```
00610                           {
00611   int major[] = {0, 2, 4, 5, 7, 9, 11};
00612
00613   for(int i = 0; i < SCALESIZE; i++){
00614     if(toneLeap == major[i])
00615       return 1;
00616   }
00617   return 0;
00618 }
```

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.

```
00596                           {
00597   int minor[] = {0, 2, 3, 5, 7, 8, 10};
00598
00599   for(int i = 0; i < SCALESIZE; i++){
00600     if(toneLeap == minor[i])
00601       return 1;
00602   }
00603   return 0;
00604 }
```

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.

```
00567                                                    {
00568   int toneLeap, isMinor = 1, isMajor = 1;
00569
00570   for(int i = 0; i < size; i++){
00571     if(otherTones[i] < keytone)
00572       otherTones[i] += 12;
00573       toneLeap = otherTones[i] - keytone;
00574
00575       if(isMinor)
00576         isMinor = isInMinor(toneLeap);
00577       if(isMajor)
00578         isMajor = isInMajor(toneLeap);
00579     }
00580
00581     if(isMinor && isMajor)
00582       return 0;
00583     else if(isMinor)
00584       return -1;
00585     if(isMajor)
00586       return 1;
00587
00588     return 0;
00589 }
```

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

```
00094                                       {
00095   FILE *f;
00096   char MIDIfile[25];
00097   /*Variables*/
00098   int numbersInText = 0, notes, i = 0, size = 0, moodOfMelodi = 0;
00099   /* PLACEHOLDER FIX THIS */
00100   int mode = 5, tempo = 5, toneLength = 5, pitch = 5;
00101   FILE* moods = fopen("moods.txt", "r");
00102   if(moods == NULL){
00103     perror("Error: moods missing ");
00104     exit(EXIT_FAILURE);
00105   }
00106   AMOUNT_OF_MOODS = FindMoodAmount(moods);
00107   moodWeighting moodArray[AMOUNT_OF_MOODS];
00108   data data = {0, major, D};
00109   if (argv[1] == NULL){
00110     checkDirectory(MIDIfile);
00111     f = fopen(MIDIfile,"r");
00112     if(f == NULL){
00113       perror("Error opening file");
00114       exit(EXIT_FAILURE);
00115     }
00116   }
00117   else if(argv[1] != NULL){
00118     f = fopen(argv[1],"r");
00119     if(f == NULL){
00120       perror("Error opening file");
00121       exit(EXIT_FAILURE);
00122     }
00123   }
00124
00125   int *hex = (int *) malloc(CHARS * sizeof(int));
00126   if(hex == NULL){
00127     printf("Memory allocation failed, bye!");
00128     exit(EXIT_FAILURE);
00129   }
00130   /*Reading the data from the file*/
00131   numbersInText = getHex(f, hex);
00132   fillSongData(&data, hex, numbersInText);
00133   notes = countNotes(hex, numbersInText);
00134   note *noteAr = (note*) malloc(notes * sizeof(note));
00135   if(noteAr == NULL){
```

```
00136     printf("Memory allocation failed, bye!");
00137     exit(EXIT_FAILURE);
00138   }
00139   eventPlacement placement[numbersInText];
00140   int ticks[numbersInText];
00141   findEvents(numbersInText, hex, placement, noteAr, ticks, &size);
00142   insertMoods(moodArray, moods);
00143   settingPoints(&mode, &tempo, &toneLength, &pitch, data, notes, noteAr, &size);
00144   printf("%d, %d, %d, %d\n", mode, tempo, toneLength, pitch);
00145   for(i = 0; i < notes; i++)
00146     printNote(noteAr[i]);
00147   findMode(noteAr, notes, &data);
00148   printSongData(data);
00149   moodOfMelodi = weightingMatrix(moodArray, mode, tempo, toneLength, pitch);
00150   printf("%d\n", moodOfMelodi);
00151
00152
00153   /*Clean up and close*/
00154   fclose(f);
00155   free(hex);
00156   free(noteAr);
00157
00158   return 0;
00159 }
```

4.2.4.23 void printNote (note *note*)

A function to print the note

Parameters

	note]: note: the note structure to be printed

```
00341                         {
00342   printf("Tone: ");
00343
00344   switch (note.tone){
00345     case C     : printf("C") ; break;
00346     case Csharp: printf("C#"); break;
00347     case D     : printf("D") ; break;
00348     case Dsharp: printf("D#"); break;
00349     case E     : printf("E") ; break;
00350     case F     : printf("F") ; break;
00351     case Fsharp: printf("F#"); break;
00352     case G     : printf("G") ; break;
00353     case Gsharp: printf("G#"); break;
00354     case A     : printf("A") ; break;
00355     case Asharp: printf("A#"); break;
00356     case B     : printf("B") ; break;
00357     default    : printf("Undefined note"); break;
00358   }
00359   printf(", octave: %d\n", note.octave);
00360 }
```

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

```
00365                           {
00366   printf("Tempo: %d\nMode: ", data.tempo);
00367   switch(data.mode){
00368     case minor: printf("minor"); break;
00369     case major: printf("major"); break;
00370     default: printf("unknown mode"); break;
00371   }
00372   printf("\nKeytone: %d", data.key);
00373   putchar('\n');
00374 }
```

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

```
00376
```

```
            {
00377   int deltaTime = 2, combined = 0, averageNote = 0;
00378   switch(data.mode){
00379     case minor: *mode = -5; break;
00380     case major: *mode = 5; break;
00381     default: *mode = 0; break;
00382   }
00383   if(data.tempo < 60)
00384     *tempo = -5;
00385   else if(data.tempo >= 60 && data.tempo < 70)
00386     *tempo = -4;
00387   else if(data.tempo >= 70 && data.tempo < 80)
00388     *tempo = -3;
00389   else if(data.tempo >= 80 && data.tempo < 90)
00390     *tempo = -2;
00391   else if(data.tempo >= 90 && data.tempo < 100)
00392     *tempo = -1;
00393   else if(data.tempo >= 100 && data.tempo < 120)
00394     *tempo =  0;
00395   else if(data.tempo >= 120 && data.tempo < 130)
00396     *tempo =  1;
00397   else if(data.tempo >= 130 && data.tempo < 140)
00398     *tempo =  2;
00399   else if(data.tempo >= 140 && data.tempo < 150)
00400     *tempo =  3;
00401   else if(data.tempo >= 150 && data.tempo < 160)
00402     *tempo =  4;
00403   else if(data.tempo >=  160)
00404     *tempo =  5;
00405
00406   switch(deltaTime){
00407     case 1: *length = -5; break;
00408     case 2: *length = -4; break;
00409     case 4: *length = -2; break;
00410     case 8: *length =  0; break;
00411     case 16: *length = 3; break;
00412     case 32: *length = 5; break;
00413   }
00414   for (int i = 0; i < notes; i++){
00415     combined += noteAr[i].average;
00416   }
00417   averageNote = combined/notes;
00418
00419   if(averageNote <= 16)
00420     *octave = -5;
00421   else if(averageNote >= 17 && averageNote <= 23)
00422     *octave = -4;
00423   else if(averageNote >= 24 && averageNote <= 30)
00424     *octave = -3;
00425   else if(averageNote >= 31 && averageNote <= 37)
00426     *octave = -2;
00427   else if(averageNote >= 38 && averageNote <= 44)
00428     *octave = -1;
00429   else if(averageNote >= 45 && averageNote <= 51)
00430     *octave = 0;
00431   else if(averageNote >= 52 && averageNote <= 58)
00432     *octave = 1;
00433   else if(averageNote >= 59 && averageNote <= 65)
00434     *octave = 2;
00435   else if(averageNote >= 66 && averageNote <= 72)
00436     *octave = 3;
00437   else if(averageNote >= 73 && averageNote <= 79)
00438     *octave = 4;
00439   else if(averageNote >=80)
00440     *octave = 5;
00441 }
```

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

```
00477                                                {
00478   int a = *(const int*)pa;
00479   int b = *(const int*)pb;
00480   return (b-a);
00481 }
```

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.

```
00509                                           {
```

```
00510   int *i1 = (int*) a, *i2 = (int*) b;
00511
00512   return (int) *i1 - *i2;
00513 }
```

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

```
00454                                                                                                                {
00455   int result[AMOUNT_OF_MOODS];
00456
00457   for(int i = 0; i < AMOUNT_OF_MOODS; i++){
00458     result[i] = 0;
00459   }
00460
00461   for(int i = 0; i < AMOUNT_OF_MOODS; i++){
00462     result[i] += (moodArray[i].mode * mode);
00463     result[i] += (moodArray[i].tempo * tempo);
00464     result[i] += (moodArray[i].toneLength * toneLength);
00465     result[i] += (moodArray[i].pitch * pitch);
00466   }
00467
00468   for(int i = 0; i < AMOUNT_OF_MOODS; i++){
00469     printf("%s: %d\n", moodArray[i].name, result[i]);
00470   }
00471
00472   qsort(result, AMOUNT_OF_MOODS, sizeof(int), sortResult);
00473   return result[0];
00474 }
```

4.2.5 Variable Documentation

4.2.5.1 int AMOUNT_OF_MOODS

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)

```
00003             {
00004   printf("Jonas er en kagemand!\nOg han har lange løg.\n");
00005
00006   return 0;
00007 }
```

4.3.1.2 void testFunk (void)

```
00012                   {
00013   int stuff = 1337;
00014 }
```

Index