A412

0.1

Generated by Doxygen 1.8.8

Tue Dec 9 2014 11:16:10

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

3.1.1 Detailed Description

Definition at line 33 of file main.c.

3.1.2 Field Documentation

3.1.2.1 mode data::mode

Definition at line 35 of file main.c.

3.1.2.2 unsigned int data::tempo

Definition at line 34 of file main.c.

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

- main.c

3.2 moodWeighting Struct Reference

Data Fields

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

3.2.1 Detailed Description

Definition at line 43 of file main.c.

3.2.2 Field Documentation

3.2.2.1 int moodWeighting::mode

Definition at line 44 of file main.c.

3.2.2.2 int moodWeighting::pitch

Definition at line 47 of file main.c.

3.2.2.3 int moodWeighting::tempo

Definition at line 45 of file main.c.

3.2.2.4 int moodWeighting::toneLength

Definition at line 46 of file main.c.

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

- main.c

3.3 note Struct Reference

Data Fields

- int tone
- int octave
- int lenght

3.3.1 Detailed Description

Definition at line 27 of file main.c.

3.3.2 Field Documentation

3.3.2.1 int note::lenght

Definition at line 30 of file main.c.

3.3.2.2 int note::octave

Definition at line 29 of file main.c.

3.3.2.3 int note::tone

Definition at line 28 of file main.c.

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

- main.c

3.4 points Struct Reference

Data Fields

- char ∗ parameter
- int point

3.4.1 Detailed Description

Definition at line 38 of file main.c.

3.4.2 Field Documentation

3.4.2.1 char∗ points::parameter

Definition at line 39 of file main.c.

3.4.2.2 int points::point

Definition at line 40 of file main.c.

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

- main.c

Chapter 4

File Documentation

4.1 main.c File Reference

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

Data Structures

- struct note
- struct data
- struct points
- struct moodWeighting

Macros

- #define CHARS 1000
- #define AMOUNT_OF_MOODS 2

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 = 6, Fsharp, G,
  Gsharp, A, Asharp, B }
- enum mood { glad, sad }

Functions

- 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 insertMoods (moodWeighting[])
- int weightingMatrix (moodWeighting[], int, int, int, int)
- void findEvents (int, int[], note[])
- int sortResult (const void ∗, const void ∗)
- int deltaTimeToNoteLength (int, int)
- int main (int argc, const char ∗argv[])
- void settingPoints (int ∗mode, int ∗tempo, int ∗length, int ∗octave, data data)

4.1.1 Macro Definition Documentation

4.1.1.1 #define AMOUNT_OF_MOODS 2

Definition at line 20 of file main.c.

4.1.1.2 #define CHARS 1000

Definition at line 19 of file main.c.

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

Definition at line 23 of file main.c.

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

4.1.3.2 enum mood

Enumerator

glad

sad

Definition at line 25 of file main.c.

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

4.1.3.3 enum tone

Enumerator

C

Csharp

D

Dsharp

E

F

Fsharp

G

Gsharp

A

Asharp

B

Definition at line 24 of file main.c.

```
24 {C, Csharp, D, Dsharp, E, F = 6, Fsharp, G, Gsharp,
      A, Asharp, B} tone;
```

4.1.4 Function Documentation

4.1.4.1 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 121 of file main.c.

```
121                                         {
122   int i = 0, res = 0;
123   for(i = 0; i < amount; i++){
124     if(hex[i] == 0x90){
125       res++;
126     }
127   }
128   return res;
129 }
```

4.1.4.2 int deltaTimeToNoteLength (int *ticks*, int *ppqn*)

Definition at line 319 of file main.c.

```
319                                              {
320   return (int) (round((4*ticks)/ppqn));
321 }
```

4.1.4.3 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 203 of file main.c.

```
203                                       {
204   note->tone = inputTone % 12;
205   note->octave = inputTone / 12;
206 }
```

4.1.4.4 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 136 of file main.c.

```
136                                                           {
137   int j;
138   /*Find the mode of the song, initialised as minor atm*/
139   data->mode = minor;
140   for(j = 0; j < numbersInText; j++){
141     /* finds the tempo */
142     if(hex[j] == 0xff){
143       if(hex[j+1] == 0x51){
144         data->tempo =  60000000/((hex[j+3] << 16) | (hex[j+4] << 8) | (hex[j+5]));
145       }
146     }
147   }
148 }
```

4.1.4.5 void findEvents (int *numbersInText*, int *hex[ ]*, note *noteAr[ ]*)

Definition at line 150 of file main.c.

```
150                                                            {
151   int note = 0x01, eventType = 0x01, counter = 0, i = 0;
152   /*Read and proces the hex array*/
153   for(int j = 0; j < numbersInText; j++){
154     /* Hops over any noto-on, note-off or metaevent start
155        Also stores the tones read after a note-on         */
156     if(hex[j] == 0x00 && (hex[j + 1] == 0x90 || hex[j + 1] == 0xff)){
157       counter = 1;
158       j += 4;
159       if(hex[j - 3] == 0x90){
160         note = hex[j - 2];
161         fillNote(hex[j - 2], &noteAr[i]);
162         i++;
163       }
164       else{
165         eventType = hex[j - 2];
166       }
167     }
168     else if(hex[j] == 0x80 && hex[j + 1] == note){
169       j += 2;
170       note = 0x01;
171       counter = 0;
172     }
173     if(counter){
174       /* Here you can check for parameters inside a meta-event or MIDI-event */
175     }
176     else{
177       /* Here you can check for parameters outside a meta-event or MIDI-event
178          e.g. between a note-off and the next MIDI-event or a meta-event      */
179     }
180   }
181 }
```

4.1.4.6 void findNoteLength (double *x,* int ∗ *high,* int ∗ *low*)

A function to calculate the notelenght - tba

Definition at line 185 of file main.c.

```
185                                                  {
186   double func = 16*((x*x)*(0.0000676318287050830)+(0.0128675448628599*x)-2.7216713227147);
187   double temp = func;
188   double temp2 = (int) temp;
189 
190   if (!(temp - (double) temp2 < 0.5)){
191     func += 1;
192   }
193 
194   printf("x: %f og func: %f\n", x, func);
195   *high = (int) func;
196   *low = 16;
197 }
```

4.1.4.7 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 106 of file main.c.

```
106                               {
107   int i = 0, c;
108 
109   while( (c = fgetc(f)) != EOF && i < CHARS){
110     hexAr[i] = c;
111     i++;
112   }
113 
114   return i;
115 }
```

4.1.4.8 void insertMoods (moodWeighting *moodArray[]*)

Definition at line 286 of file main.c.

```
286                                          {
287   moodArray[glad].mode            = 3;
```

```
  moodArray[glad].tempo          = 4;
  moodArray[glad].toneLength     = 2;
  moodArray[glad].pitch          = 1;

  moodArray[sad].mode            = -4;
  moodArray[sad].tempo           = -5;
  moodArray[sad].toneLength      = -3;
  moodArray[sad].pitch           = 0;
}
```

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

Definition at line 64 of file main.c.

```
                                        {
  /*Variables*/
  int numbersInText = 0, notes, i = 0, moodOfMelodi = 0;
  /* PLACEHOLDER FIX THIS */
  int mode = 5, tempo = 5, toneLength = 5, pitch = 5;
  moodWeighting moodArray[AMOUNT_OF_MOODS];
  data data;
  FILE *f = fopen(argv[1],"r");
  int *hex = (int *) malloc(CHARS * sizeof(int));
  if(hex == NULL){
    printf("Memory allokation failed, bye!");
    exit(EXIT_FAILURE);
  }

  /*Reading the data from the file*/
  numbersInText = getHex(f, hex);
  fillSongData(&data, hex, numbersInText);
  notes = countNotes(hex, numbersInText);
  note *noteAr = (note*) malloc(notes * sizeof(note));
  if(noteAr == NULL){
    printf("Memory allocation failed, bye!");
    exit(EXIT_FAILURE);
  }
  findEvents(numbersInText, hex, noteAr);
  insertMoods(moodArray);
  for(i = 0; i < notes; i++)
    printNote(noteAr[i]);
  printSongData(data);
  moodOfMelodi = weightingMatrix(moodArray, mode, tempo, toneLength, pitch);

  /*Clean up and close*/
  fclose(f);
  free(hex);
  free(noteAr);

  return 0;
}
```

4.1.4.10 void printNote (note *note*)

A function to print the note

Parameters

note]	note: the note structure to be printed

Definition at line 211 of file main.c.

```
                          {
  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);
}
```

4.1.4.11 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 235 of file main.c.

```
                             {
  printf("Tempo: %d\nMode: ", data.tempo);
  switch(data.mode){
    case minor: printf("minor"); break;
    case major: printf("major"); break;
    default: printf("unknown mode"); break;
  }
  putchar('\n');
}
```

4.1.4.12 void settingPoints (int ∗ *mode,* int ∗ *tempo,* int ∗ *length,* int ∗ *octave,* data *data*)

Definition at line 245 of file main.c.

```
                                                                              {
  int deltaTime = deltaTimeToNoteLength(480, 960);
  switch(data.mode){
    case minor: *mode = -5; break;
    case major: *mode = 5; break;
  }
  if(data.tempo < 60)
    *tempo = -5;
  else if(data.tempo >= 60 && data.tempo < 70)
    *tempo = -4;
  else if(data.tempo >= 70 && data.tempo < 80)
    *tempo = -3;
  else if(data.tempo >= 80 && data.tempo < 90)
    *tempo = -2;
  else if(data.tempo >= 90 && data.tempo < 100)
    *tempo = -1;
  else if(data.tempo >= 100 && data.tempo < 120)
    *tempo =  0;
  else if(data.tempo >= 120 && data.tempo < 130)
    *tempo =  1;
  else if(data.tempo >= 130 && data.tempo < 140)
    *tempo =  2;
  else if(data.tempo >= 140 && data.tempo < 150)
    *tempo =  3;
  else if(data.tempo >= 150 && data.tempo < 160)
    *tempo =  4;
  else if(data.tempo >=  160)
    *tempo =  5;

  switch(deltaTime){
    case 1: *length = -5; break;
    case 2: *length = -4; break;
    case 4: *length = -2; break;
    case 8: *length =  0; break;
    case 16: *length = 3; break;
    case 32: *length = 5; break;
  }
}
```

4.1.4.13 int sortResult (const void ∗ *pa,* const void ∗ *pb*)

Definition at line 312 of file main.c.

```
                                           {
  int a = *(const int*)pa;
  int b = *(const int*)pb;
  return (b-a);
}
```

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

Definition at line 299 of file main.c.

```
                                                                                                      {
  int result[AMOUNT_OF_MOODS] = {0};
  for(int i = 0; i < AMOUNT_OF_MOODS; i++){
    result[i] += (moodArray[i].mode * mode);
    result[i] += (moodArray[i].tempo * tempo);
    result[i] += (moodArray[i].toneLength * toneLength);
    result[i] += (moodArray[i].pitch * pitch);
  }
  qsort(result, AMOUNT_OF_MOODS, sizeof(int), sortResult);
  return result[0];
}
```

4.2 test.c File Reference

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

Functions

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

4.2.1 Function Documentation

4.2.1.1 int main (void)

Definition at line 3 of file test.c.

```
              {
  printf("Jonas er en kagemand!\nOg han har lange løg.\n");

  return 0;
}
```

4.2.1.2 void testFunk (void)

Definition at line 12 of file test.c.

```
                    {
  int stuff = 1337;
}
```

Index