lab3

Generated by Doxygen 1.8.8

Tue Oct 17 2017 14:35:42

Contents

1 Specification

This is the Stacks Music Program. Once this program is executed the music will begin to play. It will play the song Yankee Doodle. The song repeats itself in a few places as well as overall. The user will see the exact notes and duration for which the song is played. The music is stored in stacks and will be

Features:

1) A user can enter another song into the program provided he has the music notation for it

2) The ammount of notes can be shortened because the song can replay parts of itself in "fragments"

3) Delete the stack so everything is done when the program exits.

2 Analysis

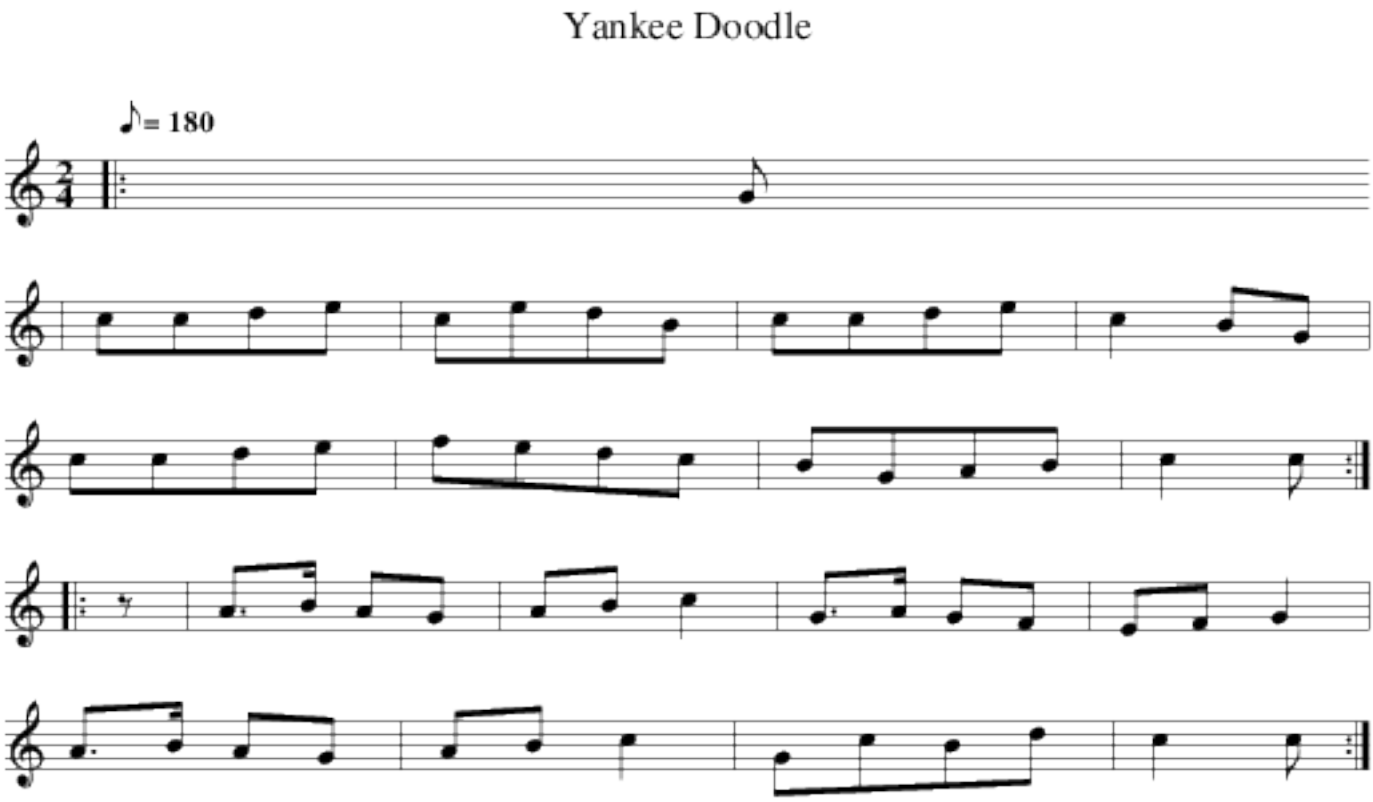
When the program runs the song will immediatly begin to play. The user will be gin to see the notes and the duration of each not being played. The user will also be able to see the pointers tha controll the fragment replays. This is not necessary but I felt that the user would be interested to see if the program is working properly.

3 Design

There are 8 parts to the lab. The most notable are the push and pop functions. Those two control everything in the stack. Wer also have the create and destroy fucntions, which creat/destroy the stack. We then uses specific functions to play the music, play the notes, and read the song. The play notes allow individual notes to be played while play music plays the song as a whole.

4 Test

This test shows that the program functions The image of the code running is only a portion of the whole song. The song is much larger than what the picture carried. I also created a chart showing how the stack works. It deatails where the pointers are pointing during the playnotes and play fragments. This is for part of the song since the song is much longer.



width=2in

```
play -qn synth 0.500000 pluck e
note played
5 stack pop 23
14 stack pop 23
14
play -qn synth 0.500000 pluck f
note played
play -qn synth 0.500000 pluck e
note played
play -qn synth 0.500000 pluck d
play FAIL synth: invalid freq
note played
play -qn synth 0.500000 pluck c
note played
play -qn synth 0.500000 pluck B
note played
play -qn synth 0.500000 pluck G
note played
play -qn synth 0.500000 pluck A
note played
```

type	current	finish	stack elements
p.note	0-8	9999	empty
p.fragment	9	9999	10/9999
p.note	1-4	4	empty
p.note	10-12	9999	empty

5 Class Index

5.1 Class List

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

6 File Index

6.1 File List

Here is a list of all files with brief descriptions:

7 Class Documentation

7.1 FRAGMENT Struct Reference

`#include <lab.h>`

Public Attributes

- int start
- int finish

7.1.1 Member Data Documentation

7.1.1.1 int FRAGMENT::finish

7.1.1.2 int FRAGMENT::start

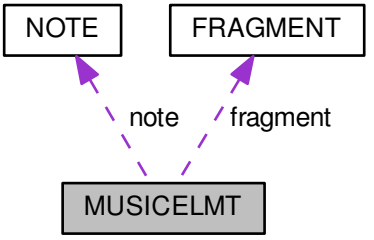
The documentation for this struct was generated from the following file:

- lab.h

7.2 MUSICELMT Struct Reference

`#include <lab.h>`

Collaboration diagram for MUSICELMT:



Public Attributes

- PLAY type
- union {
  NOTE note
  FRAGMENT fragment
  };

7.2.1 Member Data Documentation

7.2.1.1 union { ... }

7.2.1.2 FRAGMENT MUSICELMT::fragment

7.2.1.3 NOTE MUSICELMT::note

7.2.1.4 PLAY MUSICELMT::type

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

- lab.h

7.3 NOTE Struct Reference

```
#include <lab.h>
```

Public Attributes

- char tone
- int duration

7.3.1 Member Data Documentation

7.3.1.1 int NOTE::duration

7.3.1.2 char NOTE::tone

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

- lab.h

7.4 STACK Struct Reference

`#include <lab.h>`

Public Attributes

- int size
- int * buf
- int sp

7.4.1 Member Data Documentation

7.4.1.1 int* STACK::buf

7.4.1.2 int STACK::size

7.4.1.3 int STACK::sp

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

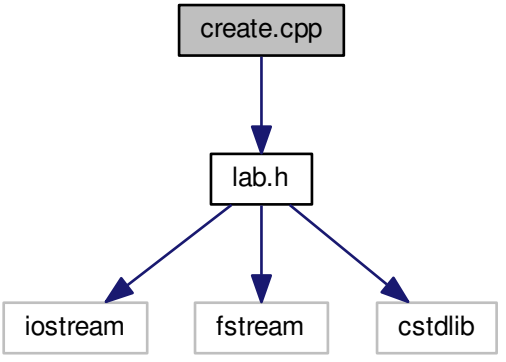
- lab.h

8 File Documentation

8.1 create.cpp File Reference

```
#include "lab.h"
```

Include dependency graph for create.cpp:



Functions

- STATUS Create (STACK &stack, int size)

8.1.1 Function Documentation

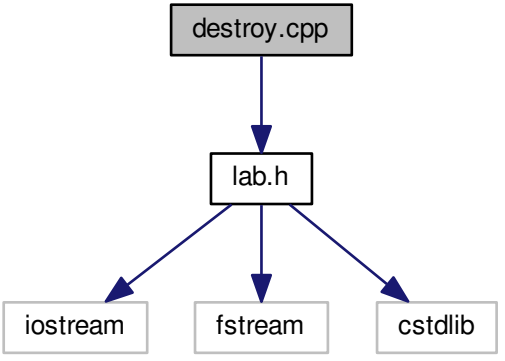
8.1.1.1 STATUS Create (STACK & *stack,* int *size*)

```
4 {
5     stack.buf = new int[size];
6     if (!stack.buf)
7         return FAILED;
8     stack.size = size;
9     stack.sp = 0;
10     return OK;
11 }
```

8.2 destroy.cpp File Reference

```
#include "lab.h"
```

Include dependency graph for destroy.cpp:



Functions

- STATUS Destroy (STACK &stack)

8.2.1 Function Documentation

8.2.1.1 STATUS Destroy (STACK & *stack*)

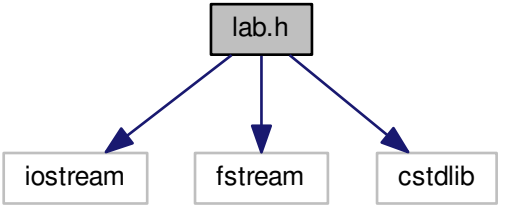
This is the function that destroys the stack. it taks in stack as the parameter

```
7 {
8     delete [] stack.buf;
9 }
```

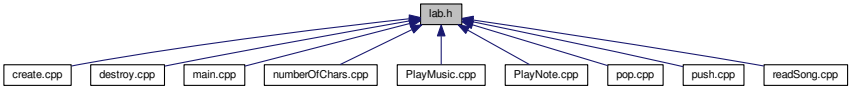
8.3 lab.h File Reference

```
#include <iostream>
#include <fstream>
#include <cstdlib>
```

Include dependency graph for lab.h:



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



Classes

- struct NOTE

- struct FRAGMENT
- struct MUSICELMT
- struct STACK

Enumerations

- enum PLAY { PLAYNOTE, PLAYFRAGMENT, PLAYSTOP }
- enum STATUS { FAILED, OK }

Functions

- void PlayNote (NOTE &note, float tempo)
- void PlayMusic (MUSICELMT music[], float tempo)
- int numberofchars (std::ifstream &f)
- void readsong (std::ifstream &f, MUSICELMT m[], int n)
- void PlayMUsic (MUSICELMT music[], float tempo)
- STATUS Create (STACK &stack, int size)
- STATUS Push (STACK &stack, int item)
- STATUS Pop (STACK &stack, int &item)
- STATUS Destroy (STACK &stack)
- bool IsEmpty (STACK &stack)

8.3.1 Enumeration Type Documentation

8.3.1.1 enum PLAY

This is the header file. This contains the several function declarations as well as struct definistions. It also calls the system libraries that we use.

Enumerator

PLAYNOTE

PLAYFRAGMENT

PLAYSTOP

```
12 {PLAYNOTE,PLAYFRAGMENT,PLAYSTOP};
```

8.3.1.2 enum STATUS

Enumerator

FAILED

OK

```
13 {FAILED, OK};
```

8.3.2 Function Documentation

8.3.2.1 STATUS Create (STACK & *stack*, int *size*)

```
4 {
5     stack.buf = new int[size];
6     if (!stack.buf)
7         return FAILED;
8     stack.size = size;
9     stack.sp = 0;
10     return OK;
11 }
```

8.3.2.2 STATUS Destroy (STACK & *stack*)

This is the function that destroys the stack. it taks in stack as the parameter

```
7 {
8     delete [] stack.buf;
9 }
```

8.3.2.3 bool IsEmpty (STACK & *stack*) [inline]

```
52 {
53     return bool(stack.sp == 0);
54 }
```

8.3.2.4 int numberofchars (std::ifstream & *f*)

This program counts the number of chareters in the abc notion

```
6 {
7     int n = 0;
8     //count characters
9     char c;
10     while(f >> c)
11         n++;
12     return n;
13 }
```

8.3.2.5 void PlayMusic (MUSICELMT *music[]*, float *tempo*)

This function is responsible for playing the music. its parameters are a music structure and a float value. It uses a while loop to conrol and play notes or fragments. See the line comments for more details

```
9 {
10      const int MAXSTACK = 400, MAXARRAY = 9999;
11      STACK stack;
12      PLAY type;
13
14      if (Create(stack, MAXSTACK) == FAILED) {
15          cerr << "*** MUSIC Stack allocation error. ***\n";
16          return;
17      }
18
19      int current = 0;
20      int finish = MAXARRAY;
21
22
23      while (OK) {
24          type = music[current].type;
25
26          if(current <= finish && type != PLAYSTOP){
27              if(type == PLAYNOTE) {
28                  PlayNote(music[current++].note, tempo); //This plays the notes
29                  //It also updates the current pointer everytime it does
30              }
31              else if (type == PLAYFRAGMENT){
32                  Push(stack, ++current);
33                  //This adds elements into the array as well as increments the pointer
34                  Push(stack, finish);
35                  //This adds elements into the array an dcitates where the end pointer is
36
37                  finish = music[--current].fragment.finish;
38                  //This is what finish is set to so that the fragment can play
39                  current = music[current].fragment.start;
40                  //This is what current is updated to once the fragment plays
```

```
41          }
42      }
43          else if (!IsEmpty (stack)) {
44              Pop(stack, finish);
45              //This pops the items at the end of the stack
46              Pop(stack, current);
47              //This posp the items at the current pointer in the stack
48              //This is then played
49          }
50          else
51              break;
52
53      }
54      Destroy(stack);
55      //Destroys the stack
56  }
```

8.3.2.6 void PlayMUsic (MUSICELMT *music[],* float *tempo*)

8.3.2.7 void PlayNote (NOTE & *note,* float *tempo*)

This function plays the individual notes and fragments This creates a single string that the console can read as an instruction to play the note for a set duration

```
8  {
9       std::string s1 = "play -qn synth ";
10      std::string s2 = " pluck ";
11
12      string ms = s1 + std::to_string(note.duration/16.0) +
13                      s2 + note.tone;
14          cout << ms << endl;
```

```
15          system(ms.c_str());
16
17
18
19 }
```

8.3.2.8 STATUS Pop (STACK & *stack*, int & *item*)

This is a pop function it pos the elements off the top of the stack See inline comments for more clarification

```
7 {
8     if (stack.sp == 0) //Check if stack is empty
9         return FAILED;
10     stack.sp--; //decrement stack to the bottom element with stuff in it
11     item = stack.buf[stack.sp]; // set a variable to equal what is in the element to use later
12
13     return OK;
14 }
```

8.3.2.9 STATUS Push (STACK & *stack*, int *item*)

This is the function to push the stack see the individual comments to understand it more

```
7 {
8     if (stack.sp == stack.size) //check if the stack is full
9         return FAILED;
10     stack.buf[stack.sp] = item; // add the element into the empty space where the stack pointer was
       pointing
11     stack.sp++; //increment the stack pointer to add more into the stack
12     return OK;
13 }
```

8.3.2.10 void readsong (std::ifstream & *f*, MUSICELMT *m[]*, int *n*)

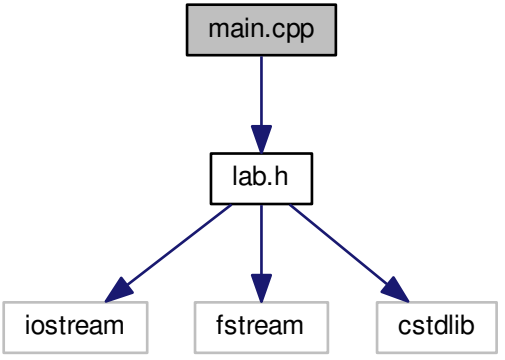
This function reads the song as a whole It uses 2 indicators for readings a fragment or a note For note it takes in its tone and duration and for fragment it takes in its start and finish This function is read in main

```
10 {
11     int i = 0;
12     char type;
13     while(f>> type)
14      {
15         if(type == 'r') {
16             f >> m[i].note.tone >> m[i].note.duration;
17             m[i].type = PLAYNOTE;
18         }
19         else if (type == 'f') {
20             f >> m[i].fragment.start >> m[i].fragment.
    finish;
21             m[i].type = PLAYFRAGMENT;
22     }
23     i++;
24 }
25
26     m[i].type = PLAYSTOP;
27 }
```

8.4 main.cpp File Reference

```
#include "lab.h"
```

Include dependency graph for main.cpp:



Functions

- int main ()

8.4.1 Function Documentation

8.4.1.1 int main ( )

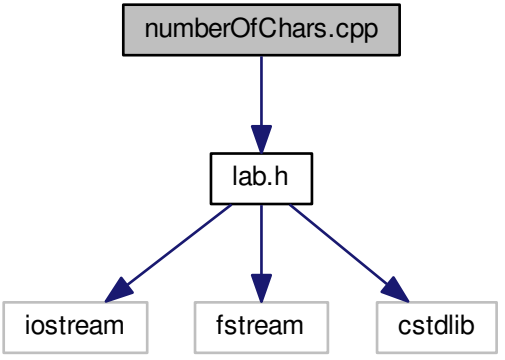
This is the main function of the music program. This function call all the other files and uses them to play the song. See the individual commetns to see what everything does

```
8 {
9     std::ifstream ifs("music"); // declare file variable
10     int n = numberofchars(ifs);
11     cout << "Music has " << n
12          << " characters" << endl;
13
14     ifs.close();
15     MUSICELMT *music;
16     music = new MUSICELMT[n]; //Declare new MUSICELMT
17
18     ifs.open("music"); // open file
19     readsong(ifs,music,n); //run the readsong program to see what everything is
20
21
22     PlayMusic (music, 80); //Play the music at a tempo of 80 beats per minute
23
24 }
```

8.5 numberOfChars.cpp File Reference

```
#include "lab.h"
```

Include dependency graph for numberOfChars.cpp:



Functions

- int numberofchars (std::ifstream &f)

8.5.1 Function Documentation

8.5.1.1 int numberofchars (std::ifstream & *f*)

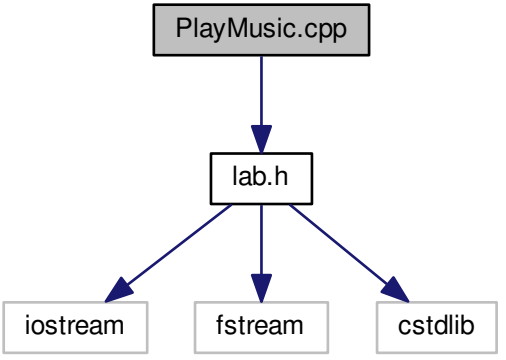
This program counts the number of chareters in the abc notion

```
6 {
7     int n = 0;
8     //count characters
9     char c;
10     while(f >> c)
11         n++;
12     return n;
13 }
```

8.6 PlayMusic.cpp File Reference

```
#include "lab.h"
```

Include dependency graph for PlayMusic.cpp:



Functions

- void PlayMusic (MUSICELMT music[], float tempo)

8.6.1 Function Documentation

8.6.1.1 void PlayMusic (MUSICELMT *music[],* float *tempo*)

This function is responsible for playing the music. its parameters are a music structure and a float value. It uses a while loop to conrol and play notes or fragments. See the line comments for more details

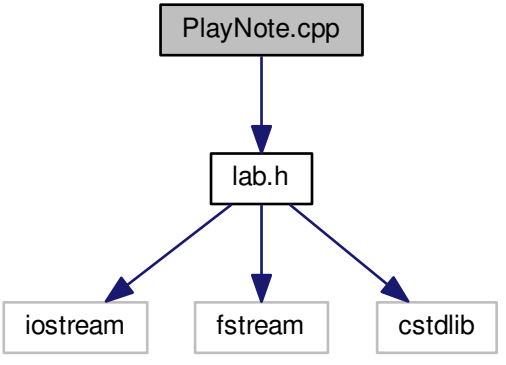
```
9 {
10      const int MAXSTACK = 400, MAXARRAY = 9999;
11      STACK stack;
12      PLAY type;
13
14      if (Create(stack, MAXSTACK) == FAILED) {
15          cerr << "*** MUSIC Stack allocation error. ***\n";
16          return;
17      }
18
19      int current = 0;
20      int finish = MAXARRAY;
21
22
23      while (OK) {
24          type = music[current].type;
25
26          if(current <= finish && type != PLAYSTOP){
27              if(type == PLAYNOTE) {
28                  PlayNote(music[current++].note, tempo); //This plays the notes
29                  //It also updates the current pointer everytime it does
30              }
31              else if (type == PLAYFRAGMENT){
32                  Push(stack, ++current);
33                  //This adds elements into the array as well as increments the pointer
```

```
34                 Push(stack, finish);
35                 //This adds elements into the array an dcitates where the end pointer is
36
37                 finish = music[--current].fragment.finish;
38                 //This is what finish is set to so that the fragment can play
39                 current = music[current].fragment.start;
40                 //This is what current is updated to once the fragment plays
41         }
42     }
43         else if (!IsEmpty (stack)) {
44             Pop(stack, finish);
45             //This pops the items at the end of the stack
46             Pop(stack, current);
47             //This posp the items at the current pointer in the stack
48             //This is then played
49         }
50         else
51             break;
52
53     }
54     Destroy(stack);
55     //Destroys the stack
56 }
```

8.7 PlayNote.cpp File Reference

```
#include "lab.h"
```

Include dependency graph for PlayNote.cpp:



Functions

- void PlayNote (NOTE &note, float tempo)

8.7.1 Function Documentation

8.7.1.1 void PlayNote (NOTE & *note*, float *tempo*)

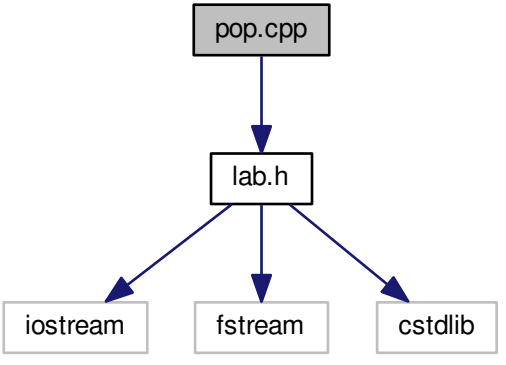
This function plays the individual notes and fragments This creates a single string that the console can read as an instruction to play the note for a set duration

```
8 {
9     std::string s1 = "play -qn synth ";
10     std::string s2 = " pluck ";
11
12     string ms = s1 + std::to_string(note.duration/16.0) +
13                     s2 + note.tone;
14         cout << ms << endl;
15         system(ms.c_str());
16
17
18
19 }
```

8.8 pop.cpp File Reference

```
#include "lab.h"
```

Include dependency graph for pop.cpp:



Functions

- STATUS Pop (STACK &stack, int &item)

8.8.1 Function Documentation

8.8.1.1 STATUS Pop (STACK & *stack*, int & *item*)

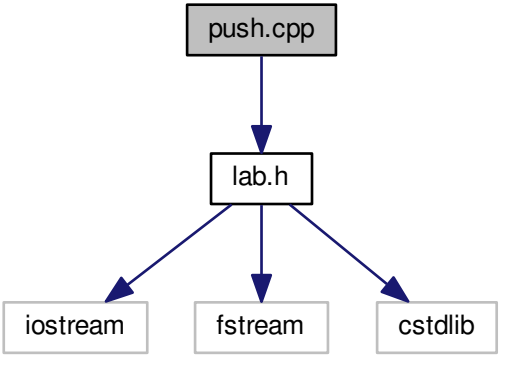
This is a pop function it pos the elements off the top of the stack See inline comments for more clarification

```
7 {
8     if (stack.sp == 0) //Check if stack is empty
9         return FAILED;
10     stack.sp--; //decrement stack to the bottom element with stuff in it
11     item = stack.buf[stack.sp]; // set a variable to equal what is in the element to use later
12 
13     return OK;
14 }
```

8.9 push.cpp File Reference

```
#include "lab.h"
```

Include dependency graph for push.cpp:



Functions

- STATUS Push (STACK &stack, int item)

8.9.1 Function Documentation

8.9.1.1 STATUS Push (STACK & *stack*, int *item*)

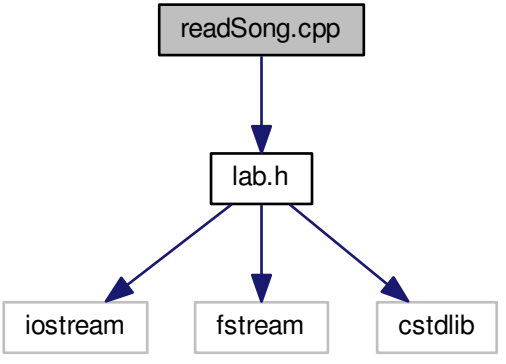
This is the function to push the stack see the individual comments to understand it more

```
7 {
8     if (stack.sp == stack.size) //check if the stack is full
9         return FAILED;
10     stack.buf[stack.sp] = item; // add the element into the empty space where the stack pointer was
       pointing
11     stack.sp++; //increment the stack pointer to add more into the stack
12     return OK;
13 }
```

8.10 readSong.cpp File Reference

```
#include "lab.h"
```

Include dependency graph for readSong.cpp:



Functions

- void readsong (std::ifstream &f, MUSICELMT m[], int n)

8.10.1 Function Documentation

8.10.1.1 void readsong (std::ifstream & *f*, MUSICELMT *m[]*, int *n*)

This function reads the song as a whole It uses 2 indicators for readings a fragment or a note For note it takes in its tone and duration and for fragment it takes in its start and finish This function is read in main

```
10 {
11     int i = 0;
12     char type;
13     while(f>> type)
14      {
15         if(type == 'r') {
16             f >> m[i].note.tone >> m[i].note.duration;
17             m[i].type = PLAYNOTE;
18         }
19         else if (type == 'f') {
20             f >> m[i].fragment.start >> m[i].fragment.
      finish;
21             m[i].type = PLAYFRAGMENT;
22     }
23     i++;
24 }
25
26     m[i].type = PLAYSTOP;
27 }
```

8.11 specification.dox File Reference

Index