CS124 Lab3 - Music Program

Generated by Doxygen 1.8.8

Sat Oct 7 2017 07:11:12

Contents

1 abc

```
debian@debian:~/lab3$ g++ -std=c++
debian@debian:~/lab3$ ./lab
Music has 56 Characters
Music has 56 Characters
play -qn synth 0.750000 pluck G
play -qn synth 1.000000 pluck G
play -qn synth 0.250000 pluck B
play -qn synth 0.250000 pluck G
play -qn synth 0.250000 pluck B
play -qn synth 1.000000 pluck A
play -qn synth 0.500000 pluck B
play -qn synth 0.500000 pluck G
play -qn synth 1.000000 pluck E
play -qn synth 0.500000 pluck G
play -qn synth 1.000000 pluck D
play -qn synth 0.500000 pluck D
play -qn synth 0.500000 pluck D
play -qn synth 0.750000 pluck G
play -qn synth 0.500000 pluck G
play -qn synth 0.500000 pluck B
play -qn synth 0.000000 pluck 8
```

2 Specification

This program lets the user play the song "Amazing Grace" instrumental with the use of stacks, pointers, and abc notation.

3 Analysis

This program runs with the notes of the song being printed out for the user to see as the program plays the song. Once the song ends, it ends.

4 Design

When the program executes, it will automatically start playing the song "Amazing Grace" and while the song plays, it will display each chord that is playing in the song. As the program continues to print out the abc notation and play the notation, it will eventually come to an end and automatically terminate the program.

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 <lab3.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

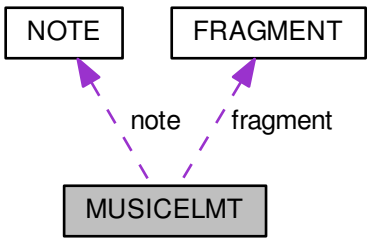
The documentation for this struct was generated from the following file:

- lab3.h

7.2 MUSICELMT Struct Reference

`#include <lab3.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:

- lab3.h

7.3 NOTE Struct Reference

`#include <lab3.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:

- lab3.h

7.4 STACK Struct Reference

`#include <lab3.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:

- lab3.h

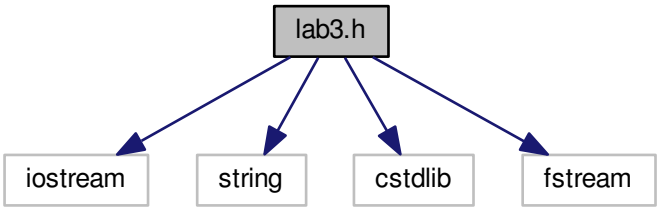
8 File Documentation

8.1 lab.dox File Reference

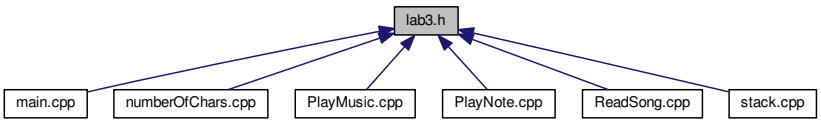
8.2 lab3.h File Reference

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

Include dependency graph for lab3.h:



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



Classes

- struct STACK
- struct NOTE
- struct FRAGMENT
- struct MUSICELMT

Enumerations

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

Functions

- STATUS Create (STACK &stack, int size)
- STATUS Push (STACK &stack, int item)
- STATUS Pop (STACK &stack, int &item)
- void Destroy (STACK &stack)
- bool isEmpty (STACK &stack)
- bool isFull (STACK &stack)
- int NoElements (STACK &stack)
- void readSong (std::ifstream &f, MUSICELMT music[], int n)
- void PlayMusic (MUSICELMT music[], float tempo)
- void PlayNote (NOTE &note, float tempo)
- int numberOfChars (std::ifstream &f)

8.2.1 Enumeration Type Documentation

8.2.1.1 enum PLAY

Enumerator

PLAYNOTE

PLAYFRAGMENT

PLAYSTOP

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

8.2.1.2 enum STATUS

Enumerator

FAILED

OK

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

8.2.2 Function Documentation

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

```
10 {
11     stack.buf = new int[size];
12     if (!stack.buf)
13         return FAILED;
14     stack.size = size;
15     stack.sp = 0;
16     return OK;
17 }
```

8.2.2.2 void Destroy (STACK & *stack*)

```
44 {
45     delete [] stack.buf;
46 }
```

8.2.2.3 bool isEmpty (STACK & *stack*) [inline]

```
20                                  {
21         return bool(stack.sp ==0);
22 }
```

8.2.2.4 bool isFull (STACK & *stack*) [inline]

```
24                                  {
25         return bool(stack.sp ==stack.size);
26 }
```

8.2.2.5 int NoElements (STACK & *stack*) [inline]

```
28                                    {
29         return stack.sp;
30 }
```

8.2.2.6 int numberOfChars (std::ifstream & *f*)

```
4 {
5
6     int n = 0;
7     char c;
8     while(f >> c)
9         n++;
10     //count characters
11     return n;
12
13 }
```

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

```
4 {
5
6     const int MAXSTACK = 400, MAXARRAY = 9999;
7     STACK stack;
8     PLAY type;
9
10     if (Create(stack, MAXSTACK) == FAILED)
11         {
12             std::cerr << "*** MUSIC: Stack allocation error. ***\n" << std::endl;
13             return;
14         }
15
16     int current = 0;
17     int finish = MAXARRAY;
```

```

    while (OK)
        {

            type = music[current].type;
            if (current <= finish && type != PLAYSTOP)
                {
                    if (type == PLAYNOTE)
                        PlayNote(music[current++].note, tempo);
                    else if (type == PLAYFRAGMENT)
                        {
                            Push(stack, current+1);
                            Push(stack, finish);
                            finish = music[current].fragment.finish;
                            current = music[current].fragment.start;
                        }
                }
            else if (!isEmpty(stack))
                {
                  Pop(stack, finish);
                  Pop(stack, current);
                }
            else
                //nothing else to do
                break;

        }
    Destroy(stack);
}
```

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

```
{
    std::ifstream inputFile("music");
    int n = numberOfChars(inputFile);
    std::cout << "Music has " << n << " Characters\n";
    inputFile.close();
    MUSICELMT *music;
     music = new MUSICELMT[n];
     inputFile.open("music");
     readSong(inputFile, music, n);
     inputFile.close();
        for(int i=0; i< n; i++)
        {
            if(music[i].type == 'n')
            std::cout << music[i].note.tone << " "
            << music[i].note.duration << std::endl;
        else if(music[i].type == 'f')
            std::cout << music[i].fragment.start << " "
                << music[i].fragment.finish << std::endl;
        }
        std::string s1 = "play -qn synth ";
        std::string s2 = " pluck ";
        for (int i = 0; i < n; i++)
        {
        std::string ms = s1 + std:: to_string(music[i].note.duration/16.0)
                        + s2 + music[i].note.tone;
        std::cout << ms << std::endl;
        system(ms.c_str());

    }
}
```

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

```
{

    if (stack.sp == 0)
        return FAILED;
    stack.sp--;
    item = stack.buf[stack.sp];
    return OK;

}
```

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

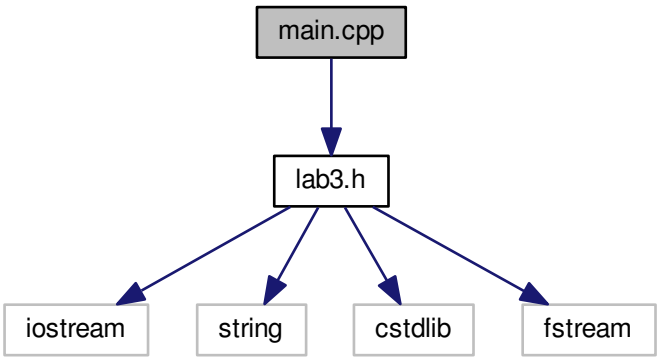
```
23 {
24
25     if (stack.sp == stack.size)
26         return FAILED;
27     stack.buf[stack.sp] = item;
28     stack.sp++;
29     return OK;
30 }
```

8.2.2.11 void readSong (std::ifstream & *f*, MUSICELMT *music[]*, int *n*)

```
5 {
6     int i = 0; char type;
7     while(f >> type)
8     {
9         //type = music[i].type;
10         if (type == 'n')
11             {
12             f >> music[i].note.tone >> music[i].note.duration;
13             music[i].type = PLAYNOTE;
14             }
15         else if (type == 'f')
16             {
17             f >> music[i].fragment.start >> music[i].fragment.
    finish;
18             music[i].type = PLAYFRAGMENT;
19             }
20         i++;
21     }
22     music[i].type = PLAYSTOP;
23 }
```

8.3 main.cpp File Reference

`#include "lab3.h"`
Include dependency graph for main.cpp:



Functions

- int main ()

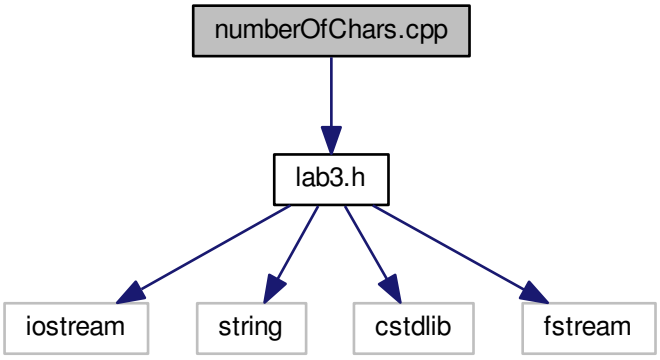
8.3.1 Function Documentation

8.3.1.1 int main ( )

```
4 {
5     const float tempo = 1.2;
6     std::ifstream inputFile("music");
7     int n = numberOfChars(inputFile);
8     std::cout << "Music has " << n << " Characters\n";
9     inputFile.close();
10     MUSICELMT *music;
11     music = new MUSICELMT[n];
12     inputFile.open("music");
13     readSong(inputFile, music, n);
14     PlayMusic(music, tempo);
15     inputFile.close();
16
17 }
```

8.4 numberOfChars.cpp File Reference

`#include "lab3.h"`
Include dependency graph for numberOfChars.cpp:



Functions

- int numberOfChars (std::ifstream &f)

8.4.1 Function Documentation

8.4.1.1 int numberOfChars (std::ifstream & *f*)

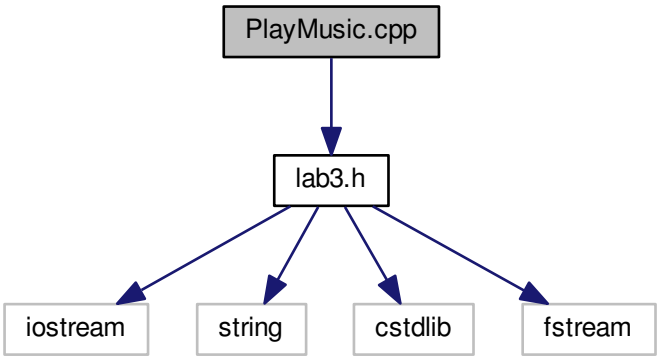
```
4 {
5
6     int n = 0;
7     char c;
8     while(f >> c)
9         n++;
10     //count characters
11     return n;
```

```
12
13 }
```

8.5 PlayMusic.cpp File Reference

```
#include "lab3.h"
```

Include dependency graph for PlayMusic.cpp:



Functions

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

8.5.1 Function Documentation

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

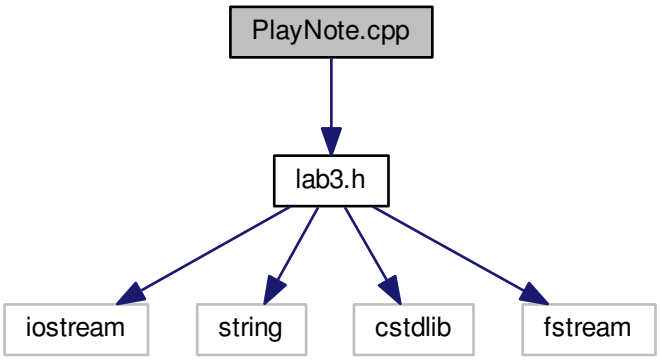
```
4 {
5
6     const int MAXSTACK = 400, MAXARRAY = 9999;
7     STACK stack;
8     PLAY type;
9
10     if (Create(stack, MAXSTACK) == FAILED)
11         {
12             std::cerr << "*** MUSIC: Stack allocation error. ***\n" << std::endl;
13             return;
14         }
15
16     int current = 0;
17     int finish = MAXARRAY;
18
19
20     while (OK)
21         {
22
23             type = music[current].type;
24             if (current <= finish && type != PLAYSTOP)
25                 {
26                     if (type == PLAYNOTE)
27                         PlayNote(music[current++].note, tempo);
28                     else if (type == PLAYFRAGMENT)
```

```
29                          {
30                              Push(stack, current+1);
31                              Push(stack, finish);
32                              finish = music[current].fragment.finish;
33                              current = music[current].fragment.start;
34                          }
35                  }
36              else if (!isEmpty(stack))
37                  {
38                    Pop(stack, finish);
39                    Pop(stack, current);
40                  }
41              else
42                  //nothing else to do
43                  break;
44
45          }
46      Destroy(stack);
47  }
```

8.6 PlayNote.cpp File Reference

```
#include "lab3.h"
```

Include dependency graph for PlayNote.cpp:



Functions

- void PlayNote (NOTE &note, float tempo)

8.6.1 Function Documentation

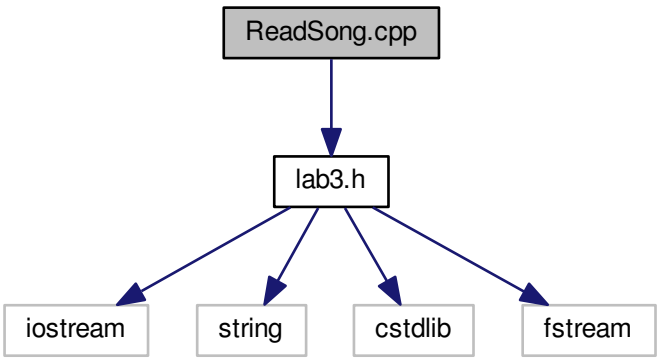
8.6.1.1 void PlayNote (NOTE & *note*, float *tempo*)

```
4 {
5      std::ifstream inputFile("music");
6      int n = numberOfChars(inputFile);
7      std::cout << "Music has " << n << " Characters\n";
8      inputFile.close();
9      MUSICELMT *music;
10      music = new MUSICELMT[n];
11      inputFile.open("music");
```

```
12      readSong(inputFile, music, n);
13      inputFile.close();
14          for(int i=0; i< n; i++)
15          {
16              if(music[i].type == 'n')
17              std::cout << music[i].note.tone << " "
18              << music[i].note.duration << std::endl;
19          else if(music[i].type == 'f')
20              std::cout << music[i].fragment.start << " "
21                  << music[i].fragment.finish << std::endl;
22          }
23          std::string s1 = "play -qn synth ";
24          std::string s2 = " pluck ";
25          for (int i = 0; i < n; i++)
26          {
27          std::string ms = s1 + std:: to_string(music[i].note.duration/16.0)
28                          + s2 + music[i].note.tone;
29          std::cout << ms << std::endl;
30          system(ms.c_str());
31
32      }
33  }
```

8.7 ReadSong.cpp File Reference

`#include "lab3.h"`
Include dependency graph for ReadSong.cpp:



Functions

- void readSong (std::ifstream &f, MUSICELMT music[], int n)

8.7.1 Function Documentation

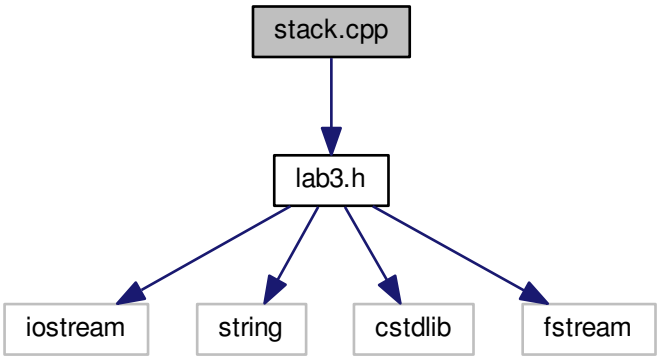
8.7.1.1 void readSong (std::ifstream & *f*, MUSICELMT *music[]*, int *n*)

```
5  {
6      int i = 0; char type;
7      while(f >> type)
8      {
9          //type = music[i].type;
```

```
10          if (type == 'n')
11              {
12              f >> music[i].note.tone >> music[i].note.duration;
13              music[i].type = PLAYNOTE;
14              }
15          else if (type == 'f')
16              {
17              f >> music[i].fragment.start >> music[i].fragment.
    finish;
18              music[i].type = PLAYFRAGMENT;
19              }
20          i++;
21      }
22      music[i].type = PLAYSTOP;
23  }
```

8.8 stack.cpp File Reference

`#include "lab3.h"`
Include dependency graph for stack.cpp:



Functions

- STATUS Create (STACK &stack, int size)
- STATUS Push (STACK &stack, int item)
- STATUS Pop (STACK &stack, int &item)
- void Destroy (STACK &stack)

8.8.1 Function Documentation

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

```
10 {
11      stack.buf = new int[size];
12      if (!stack.buf)
13          return FAILED;
14      stack.size = size;
15      stack.sp = 0;
16      return OK;
17 }
```

8.8.1.2 void Destroy (STACK & *stack*)

```
44 {
45     delete [] stack.buf;
46 }
```

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

```
33 {
34
35     if (stack.sp == 0)
36         return FAILED;
37     stack.sp--;
38     item = stack.buf[stack.sp];
39     return OK;
40
41 }
```

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

```
23 {
24
25     if (stack.sp == stack.size)
26         return FAILED;
27     stack.buf[stack.sp] = item;
28     stack.sp++;
29     return OK;
30 }
```

Index