

LAB 3: MUSICAL STACKS BY:AMULDEEP DHILLON

We want to use a stack to save our location in our music sheet and then eventually return to it after the repeat.

We will achieve this by:

- Pushing the next note and the end to the stack
- Then popping those notes back into the function

Jingle Bells

We will repeat measures 1-6



Music.h

```
#include <cstdlib>
#include <iostream>
#include <sstream>
struct Note{int tone; int duration;};
struct Fragment{int start; int finish;};
enum Play {Playnote,Playfragment,Stop};
struct MusicElement{
    Play type;
    union {
        Note note;
        Fragment fragment;
    };
};
void playMusic(MusicElement music[], double tempo, int lastNote);
std::string convertNote(int note);
std::string convertDuration(int duration);
```

Contains all shared variables and functions

Stack.h

```
#pragma once
struct STACK
{
    int size;
    int* buf;
    int sp;
};

bool create(STACK &stack, int size);
bool push(STACK &stack, int item);
bool pop(STACK &stack, int &item);
void Destroy(STACK &stack);

inline bool IsEmpty(STACK &stack){
    return (stack.sp == 0);
}

inline bool IsFull(STACK &stack){
    return (stack.sp = stack.size);
}
```

Contains all stack variables and functions

- Also includes functions:
 - IsEmpty
 - IsFull

Main

Contains all the notes and executes all the functions

```
#include "music.h"
const double tempo = 1.2;
static MusicElement music[] = {{Playnote,{16,8}}, {Playnote,{16,8}},
    {Playnote,{16,16}}, {Playnote,{16,8}}, {Playnote,{16,8}},
    {Playnote,{16,16}}, {Playnote,{16,8}}, {Playnote,{18,8}},
    {Playnote,{14,8}}, {Playnote,{15,8}}, {Playnote,{16,32}},
    {Playnote,{17,8}}, {Playnote,{17,8}}, {Playnote,{17,8}},
    {Playnote,{17,8}}, {Playnote,{17,8}}, {Playnote,{16,8}},
    {Playnote,{16,8}}, {Playnote,{16,8}}, {Playnote,{16,8}},
    {Playnote,{15,8}}, {Playnote,{15,8}}, {Playnote,{16,8}},
    {Playnote,{15,16}}, {Playnote,{18,16}},
    {Playfragment,{0,18}},
    {Playnote,{18,8}}, {Playnote,{18,8}}, {Playnote,{17,8}},
    {Playnote,{15,8}}, {Playnote,{14,32}},
    {Stop,{0,0}}};
const int LastNote = sizeof(music)/sizeof(MusicElement);
int main()
{
    playMusic(music,tempo,LastNote);
}
```

Play Music

```
#include "music.h"
#include "stack.h"
void playMusic(MusicElement music[],double tempo, int lastNote)
{
    STACK stack; Play type; int current = 0; int finish = lastNote;
    const int MAXSTACK = 400;
    if(create(stack, MAXSTACK) == false){
        std::cerr << "**MUSIC: Stack allocation error. ***\n";return;}
    while(true)
    {type = music[current].type;if(current <= finish && type != Stop){
        if(type == Playnote){
            std::string currentNote = convertNote(music[current].note.tone);
            std::string currentDuration =
            convertDuration(music[current].note.duration);current++;
            std::cout << "play " << currentNote <<
            " for " << currentDuration << " counts." << std::endl;
            std::string s = "play -qn synth ";std::string p = " pluck ";
            system((s +(currentDuration) + p + (currentNote)).c_str());}
        else if (type == Playfragment)
        {push(stack, current + 1);push(stack, finish);
            finish = music[current].fragment.finish;
            current = music[current].fragment.start;}}
    else if(not IsEmpty(stack)){pop(stack,finish);
        pop(stack,current);}else break;}Destroy(stack);}
```

- Checks Play type
- Plays note on Play Note
- Plays fragment and save next location
- Stops at the end

Create

```
#include "stack.h"
```

```
bool create(STACK &stack, int size){  
    stack.buf = new int[size];  
    if(!stack.buf)  
        return false;  
    stack.size = size;  
    stack.sp = 0;  
    return true;  
}
```

Creates a stack using an array

Convert Note

```
#include "music.h"

std::string convertNote(int note){
    std::string realNote;

    if(note == 12){
        realNote="A";}
    else if(note == 13){
        realNote="B";}
    else if(note == 14){
        realNote="C";}
    else if(note == 15){
        realNote="D";}
    else if(note == 16){
        realNote="E";}
    else if(note == 17){
        realNote="F";}
    else if(note == 18){
        realNote="G";}
    return realNote;
}
```

Tranform the number in the array into it's appropriate note in music notation

Convert Duration

```
#include "music.h"
std::string convertDuration(int duration){
    std::string realDuration;

    if(duration == 4){
        realDuration = ".25";}
    else if(duration == 8){
        realDuration = ".5";}
    else if(duration == 12){
        realDuration == ".75";}
    else if (duration == 16){
        realDuration = "1";}
    else if (duration == 24){
        realDuration = "1.5";}
    else if (duration == 32){
        realDuration = "2";}

    return realDuration;
}
```

Transform the number in the array into it's appropriate timing in the terminal

Push

```
#include "stack.h"

bool push(STACK &stack, int item){
    if(stack.sp == stack.size)
        return false;
    stack.buf[stack.sp] = item;
    stack.sp++;
    return true;
}
```

Pushes the next note into the stack
Then increments

Pop

```
#include "stack.h"

bool pop(STACK &stack, int &item){
    if(stack.sp == 0)
        return false;
    stack.sp--;
    item = stack.buf[stack.sp];

    return true;
}
```

Decrements then
Pops the data from the stack

Destroy

```
#include "stack.h"

void Destroy(STACK &stack){
    delete[] stack.buf;
}
```

Destroys the array at the end

TEST

first part of the
song jingle bells

```
debian@debian:~/cs124/lab3$ ./lab
play E for .5 counts.
play E for .5 counts.
play E for 1 counts.
play E for .5 counts.
play E for .5 counts.
play E for 1 counts.
play E for .5 counts.
play G for .5 counts.
play C for .5 counts.
play D for .5 counts.
play E for 2 counts.
play F for .5 counts.
play F for .5 counts.
play F for .5 counts.
play F for .5 counts.
play F for .5 counts.
play E for .5 counts.
play E for .5 counts.
play E for .5 counts.
play E for .5 counts.
play D for .5 counts.
play D for .5 counts.
play E for .5 counts.
play D for 1 counts.
play G for 1 counts.
```

```
play E for .5 counts.
play E for .5 counts.
play E for 1 counts.
play E for .5 counts.
play E for .5 counts.
play E for 1 counts.
play E for .5 counts.
play G for .5 counts.
play C for .5 counts.
play D for .5 counts.
play E for 2 counts.
play F for .5 counts.
play F for .5 counts.
play F for .5 counts.
play F for .5 counts.
play F for .5 counts.
play E for .5 counts.
play E for .5 counts.
play E for .5 counts.
play E for .5 counts.
play E for .5 counts.
play G for .5 counts.
play G for .5 counts.
play F for .5 counts.
play D for .5 counts.
play C for 2 counts.
```

second part of the
song jingle bells

no memory leaks

```
==2545== HEAP SUMMARY:
==2545==    in use at exit: 0 bytes in 0 blocks
==2545==   total heap usage: 344 allocs, 344 frees, 12,246 bytes allocated
==2545==
==2545== All heap blocks were freed -- no leaks are possible
==2545==
==2545== For counts of detected and suppressed errors, rerun with: -v
==2545== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
debian@debian:~/cs124/lab3$
```