2015

ANALYSIS

Hassan Iqbal

Syed Qambar Ahmed Rizvi

AUDIO PLAYER

Description of the Problem

Fact Finding Technique

We will be using interviews to exactly investigate the difficulties being faced; and analyze as to how and what are the requirements of the prospective users of the media player.
Below is the interview of a student.

Interview

Interviewer: Qambar Rizvi
Interviewee: Qarib (Alevel Student)

Q. Your name, for the sake of record, Sir?
A. Qarib Ahmed.

Q. Why do you feel there is a need for an audio player?
A. The available audio players take up a lot of time in searching once the playlists become too large for them to handle. Although they have fancy design but I would prefer a simple player which is efficient.

Q. What are the different features you are looking for in the audio player?
A. Well, it should have the general features like play, stop, pause,etc; moreover it should have the option to create a playlist of favourites, and to mark songs that are to be played next, it should also allow us to repeatedly play a song, etc.

Q. How do you want the Player to perform search operation?
A. It should ask for the name of the song I want to search for and then display that song along with all the options that can be performed.

Q. Do you want to have the player to perform shuffle?
A. Yes, that will be a nice option as I do get bored while listening to the songs in the same order.

I will indeed try to design the software which may fulfill all your requirements.

It was nice meeting you,
Thank you!

Abstract

The task at hand is to make an audio player which should perform a number of functions efficiently. The player will be able to play mp3 format files. The player should be able to handle large number of songs efficiently.

The HQ Media Player will enable user to play audio files. It will allow the user the freedom of playing any song from a list. It will do so by giving the user the ability to search, select a song from recently played playlist, most played playlist, favorite's playlist, newly added playlist etc. Moreover it will also provide the user with the list of the songs which have never been played. The user can turn the 'shuffle', 'repeat' on. The player will have the option of moving to next or previous song within a playlist. The player will sort the available audios with respect to different characteristics like artist, album etc. It will also allow the user to add a song in user defined or predefined playlists. It facilitates the user in queuing the songs to be played.

Extensions:

(This depends upon the availability of time and open source functionalities available)

Pre-defined equalizers

Allowing the user the flexibly to customize equalizer settings.

Linking each song with a specific equalizer setting.

OBJECTIVES

General Aims

The player must meet the following objectives:

- To update and maintain lists of songs;
- To update and maintain songs according to their specific search requirements, i-e, characteristics;
- To search for a specific song using its name;
- To accelerate efficiency and speed in operations like searching, updating, deleting and editing of relevant information;
- To guarantee system security by making certain that essential information is not accidentally deleted or missed out.
- To be easy to operate and understand in order to motivate the users, increasing their interest and in the long run customer satisfaction;

Technical objectives

1) Enabling addition of songs and saving it using different data structures for efficient retrieval and search operations

2) Enabling the use of validation rules, input masks and format checks to guarantee integrity of data and minimize data input error. Fixed length checks like specifying field sizes can be used to lessen the space required to store data;

3) Menus will be used to give user a set of options to choose from so that input errors are minimized.

4) Search options will be provided using data structures such as tries to optimize efficiency.

5) Make a simple yet an eye-catching user interface.

Problem Faced

The major problem we faced was playing a song using a function in VS. We did extensive research and tried many functions to play an audio file.

After trying a number of alternative methods we came to a point where we decided to change the project completely and shift to making a phone directory.

However once we received a reply to Hassan's query on stackoverflow we started implementation of audio player once again.

0

For a semester project, I am planning to make a media player in visual studio c++ console application which will provide the functions of play, pause, previous, next, shuffle, repeat, different playlists of recently played, most played songs, search a song etc.

However, I can't find a way to do that without using multi-threading (which I dont know at the moment). To avoid multi-threading, I was thinking of making use of windows media player .dll if that is possible. I am expecting to give a path to play function which plays the song in background and then changes the song when I give it another path using the 'Next Song' function. Kindly tell how to do that if this is possible. Thank you.

c++ visual-studio-2013

share improve this question

asked Nov 23 '14 at 11:13
Hassan Iqbal
3 • 1

That's pretty ambitious. Are you familiar with VC++? – Chief Two Pencils Nov 23 '14 at 11:23

Yes. I am writing the code on Visual Studio 2013 in c++ win32 console application. – Hassan Iqbal Nov 23 '14 at 11:31

This might be helpful to you. – Chief Two Pencils Nov 23 '14 at 11:38

add a comment

1 Answer

active oldest votes

0

For your console application. You can simply call the system to launch windows media player

```
system ("start wmplayer.exe -p C:\\Folder\\Music\\Sample.mp3");
```

this will start your player. You can change track by doing this

```
system ("start wmplayer.exe -p C:\\Folder\\Music\\Sample2.mp3");
```

this will run Sample2.mp3. You can store other information in your program like for playlist store information in an array of string and start them individually.

Moreover if you want to don't want to see the WM Player interference you can use some sort of script like https://gallery.technet.microsoft.com/scriptcenter/2c3caa06-ca29-4faa-a16d-7db57e80428b I think this is the simplest solution of your problem.

Once we decided to add a graphical user interface one of our friends Abdullah Ahsan suggested the use of Open framework and ofxUI.

2015

IMPLEMENTATION

Hassan Iqbal

Syed Qambar Ahmed Rizvi

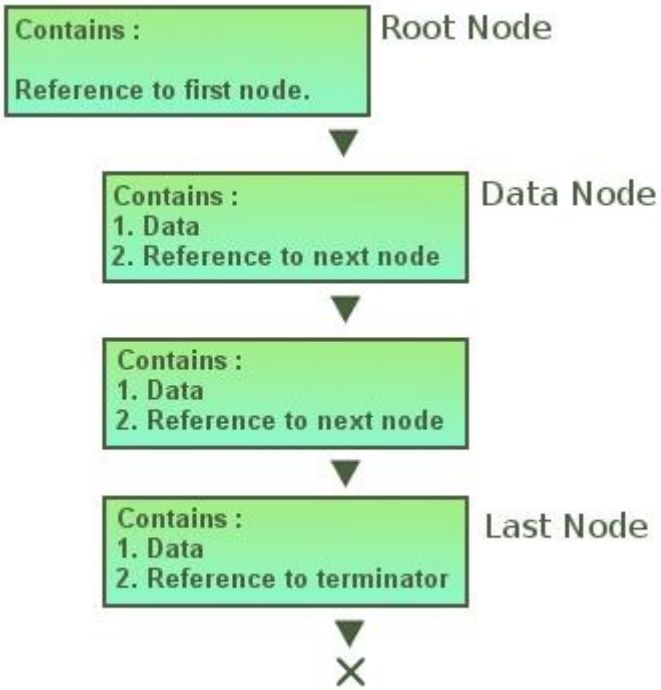
Method of solution related to problem

OpenFreamworks toolkit is being used to provide the back bone for the project. It is an open source C++ toolkit distributed under the MIT License.
The ofSoundPlayer class is being used and it allows us to load sound files and control and manipulate their playback and properties like pan, speed, volume etc.

Data Structures

Singly Linked List

A linked list consists of nodes which are responsible for holding the data as well as connecting a particular node to the next node in the list.

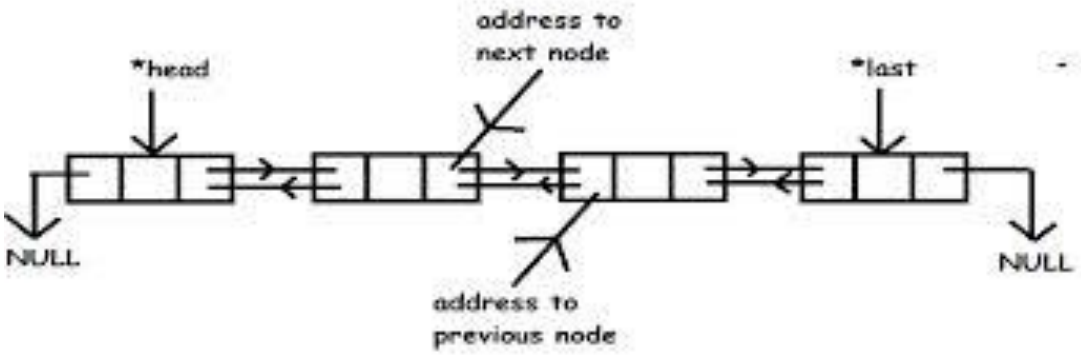


Source: www.codeproject.com

The advantage of list over an array is that it allows easier insertion and deletion, meaning new songs can be far easily added into the favorites playlist as unlike an array linked list does not uses a continuous chunk of memory. An array requires shifting of its elements if a new song is added or deleted from the start which causes an increases in the time complexity form O (1) to O (n).

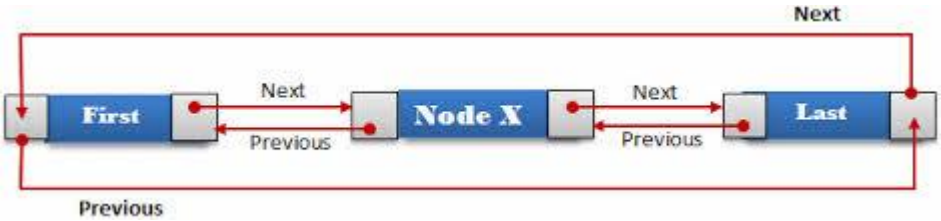
Doubly Linked List

It is a data structure which contains sequentially linked records referred to as nodes. The major difference between the nodes of doubly linked list from the nodes of singly linked list is that they also contain the address to the previous node and hence provide a link to it too.



Source: www.cprogramto.com

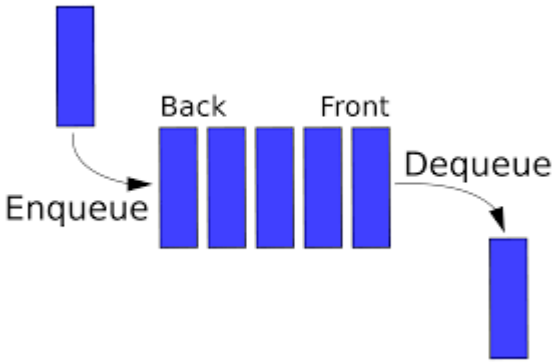
It is being used to implement the playlist as it offers easier movement within a list and the user is provided with the option to move to the next song or previous song. Moreover getting the last or first song is also comparatively easier as it has less overhead as compared these functions being implemented using a singly linked list. We also wanted to provide the user with the option to repeat the songs of a playlist and hence we used a circular doubly linked list.



Source: function.com

Queue

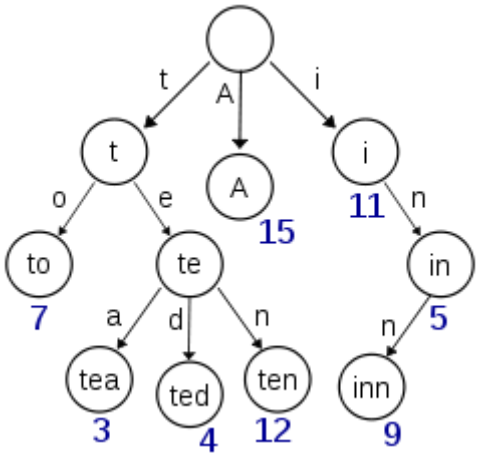
An important functionality of an audio player is queuing of songs that is allowing the user to choose which song to play once the current song ends. Interestingly the data structure used to implement this functionality is also called queue.
Queues are designed to function as first-in first-out environment (FIFO). The user are allowed to push the songs at the back of the queue while the songs in the front gets popped out and played.
The linked list implementation for queue is used as it offers better worst-case time O(1) as compared to the worst-case time of an array based implementation which is O(n) as the array may be required to grow in size.
Source: stackoverflow.com



Source: en.wikipedia.org

Tries

It is a variant of the tree data structure. It is used to store pieces of data that have a key which is used to identify the data. It uses the tree data structure and allows strings with similar character prefixes are able to use the same prefix data and store only the tails as separate data. One character of the string is stores at each level of the tree.

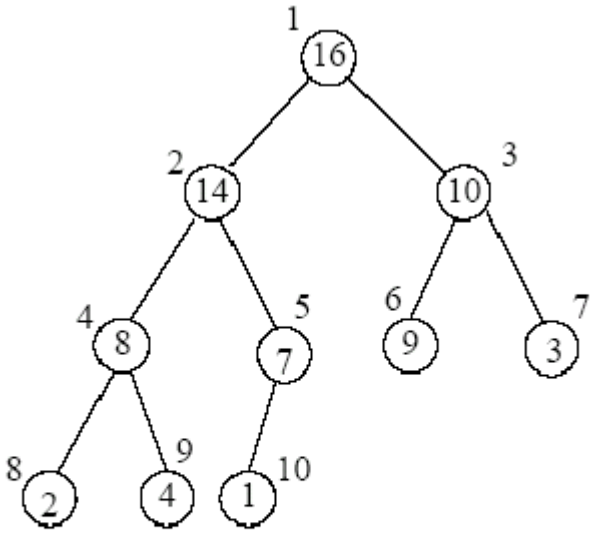


A trie for keys "A", "to", "tea", "ted", "ten", "i", "in", and "inn".
Source: www.wikipedia.com

We needed to provide the user with the option to search for a song using its name. For this purpose we chose tries, as it provides the most optimized way to search through the songs. A hash function could have been used, but avoiding collisions in it would have been near impossible as some songs have very similar names, which are being used as the key. Looking up of data is faster in a trie as the worst case is O(m), where m is the length of the key sting. While in hash table we would have to account for collisions of keys by either introducing buckets or multiple hash functions both of which increases the processing time.
Hence a trie is used the node of which contain a pointer for data(song info) and is linked to 26 children nodes.

Heap

It is a complete binary tree which follows the heap ordering property. There are two types of ordering min-heap and max-heap.



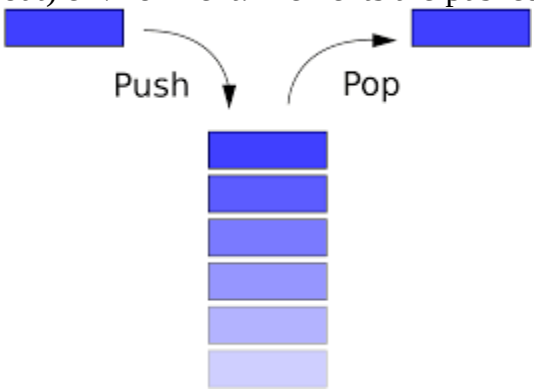
1	2	3	4	5	6	7	8	9	10
16	14	10	8	7	9	3	2	4	1

Figure 1: Binary tree and array representation for the MaxHeap containing elements (that has the priorities) [16,14,10,8,7,9,3,2,4,1].
Source: www.cse.hut.fi

We wanted to create a list of songs according to the number of times they have been played. Hence all the songs are read from a text file along with the number of times they have been played. They are used to populate a max-heap array based data structure, which ensure that the array is populated in a slightly ordered format as each song is inserted, O(log n).
Meaning that the number of times each song is played is less than or equal to the value of its parent, with the maximum time played song at the root.
Using heap we are also able to provide the user with the option to view songs which have been never played.
Now during the runtime if the order of the most played songs changes we are using bubble sort to change the arrangement of the songs. Although we used heap to initially populate the most played list since it offered the best O(log n) but considering that the array is already sorted in the best case hence bubble sort with a flag will be the most suited here, giving O(n) time complexity.

Stack

Stack are a type of container adaptor, coded so that they provide a LIFO(last-in first-out) environment. Elements are pushed and popped from the top of the stack.



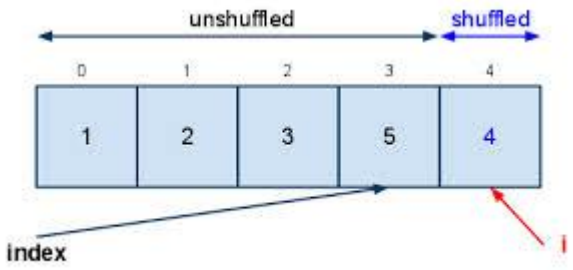
Source: en.wikipedia.org

We wanted to keep track of the songs being played in our audio player that is we wanted to maintain a list of the recently played songs. For this purpose stack was used since the song being played was pushed on top of the stack and popping the stack provided the user with the last played song.

Fisher-Yates Shuffle

It is an algorithm to generate a random permutation of a finite set that is to shuffle a

set. The algorithm is unbiased so that every permutation is equally likely. The approach being used by us is also very efficient as it requires time proportional to the number of elements and no additional storage.



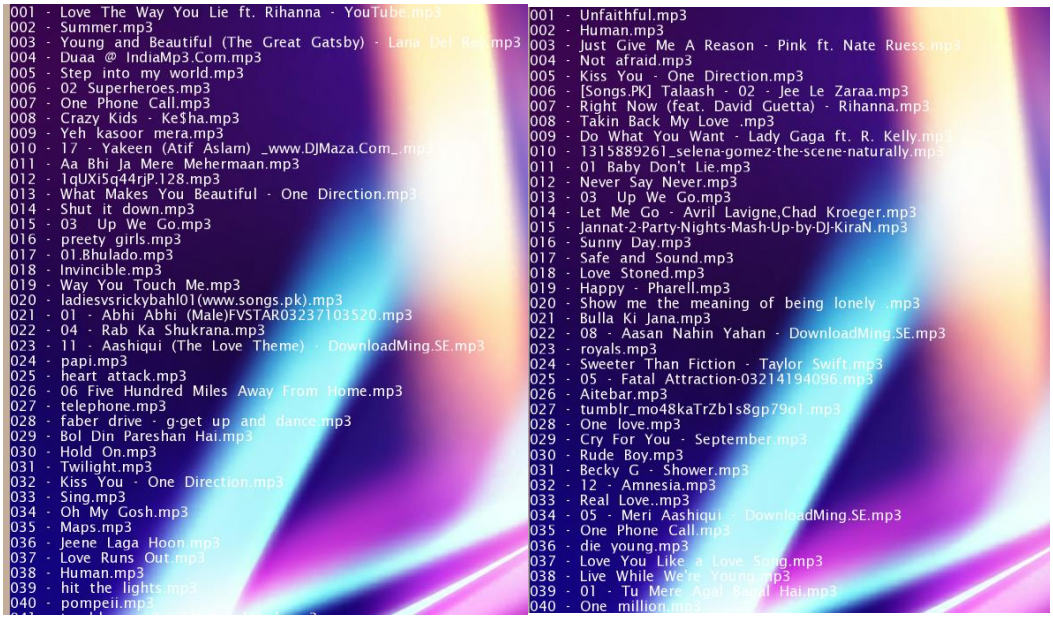
Source:codesam.blogspot.com

The Fisher–Yates shuffle, in its original form, was described in 1938 by Ronald A. Fisher and Frank Yates in their book *Statistical tables for biological, agricultural and medical research*.[1] Their description of the algorithm used pencil and paper; a table of random numbers provided the randomness. The basic method given for generating a random permutation of the numbers 1 through *N* goes as follows:

1. Write down the numbers from 1 through *N*.
2. Pick a random number *k* between one and the number of unstruck numbers remaining (inclusive).
3. Counting from the low end, strike out the *k*th number not yet struck out, and write it down elsewhere.
4. Repeat from step 2 until all the numbers have been struck out.
5. The sequence of numbers written down in step 3 is now a random permutation of the original numbers.

Source: www.wikipedia.com

The code written by us shuffles the elements of the array in-place that is it uses the back of the array to store the shuffled elements and hence there is no need to generate a separate copy, reducing the time and memory overhead. The algorithm is optimal and produces true random permutation as the time is being seeded to generate the first random number.



File Handling

The program generates a text file in which it stores all the songs and required info such as their path and number of times they have been played. As the program is run it reads this text files and populate all the data structures that it needs such as heap, stack, queue, etc.

Accurate method of solution

Serial No.	Objectives	Achieved	Not Achieved
1	Enabling addition of songs and saving it using different data structures for efficient retrieval and search operations	✓	
2	Enabling the use of validation rules, input masks and format checks to guarantee integrity of data and minimize data input error. Fixed length checks like specifying field sizes can be used to lessen the space required to store data	✓	
3	Menus will be used to give user a set of options to choose from so that input errors are minimized.	✓	
4	Search options will be provided using data structures such as tries to optimize efficiency.	✓	
5	Make a simple yet an eye-catching user interface.	✓	

2015

TESTING

Hassan Iqbal

Syed Qambar Ahmed Rizvi

Test Strategy

Before the audio player is ready to use, it needs to be thoroughly tested. All the sections of the new system will be tested to make sure that it works appropriately and accurately. Use of real life data will help in the revelation of hidden errors. Normal, Abnormal and Extreme data will be used to test the program.

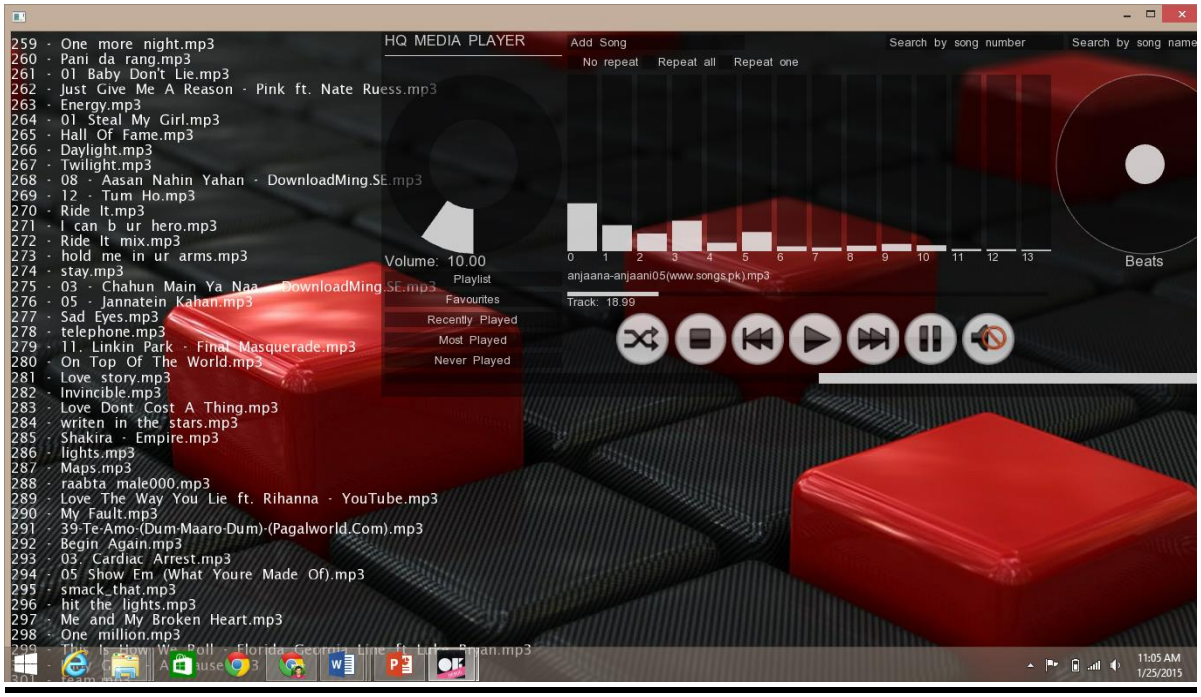
Input data, process data and output data will be tested. These tests will be used to analyze all the aspects of the system.

Testing

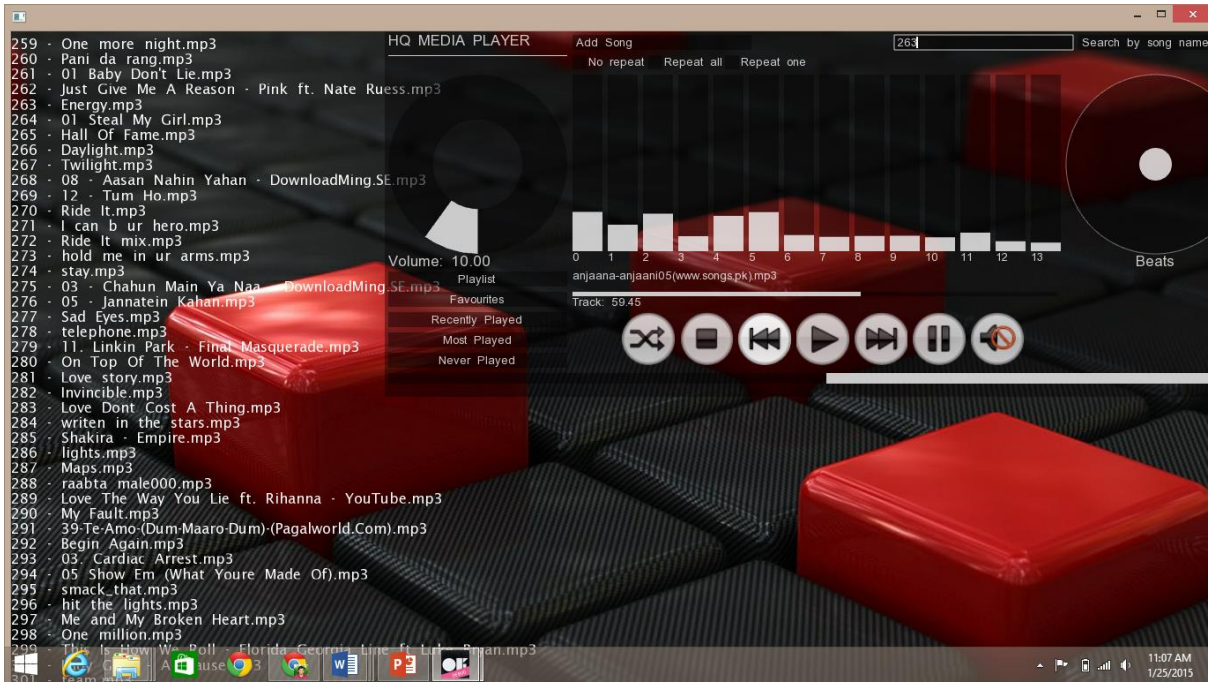
Test type Normal

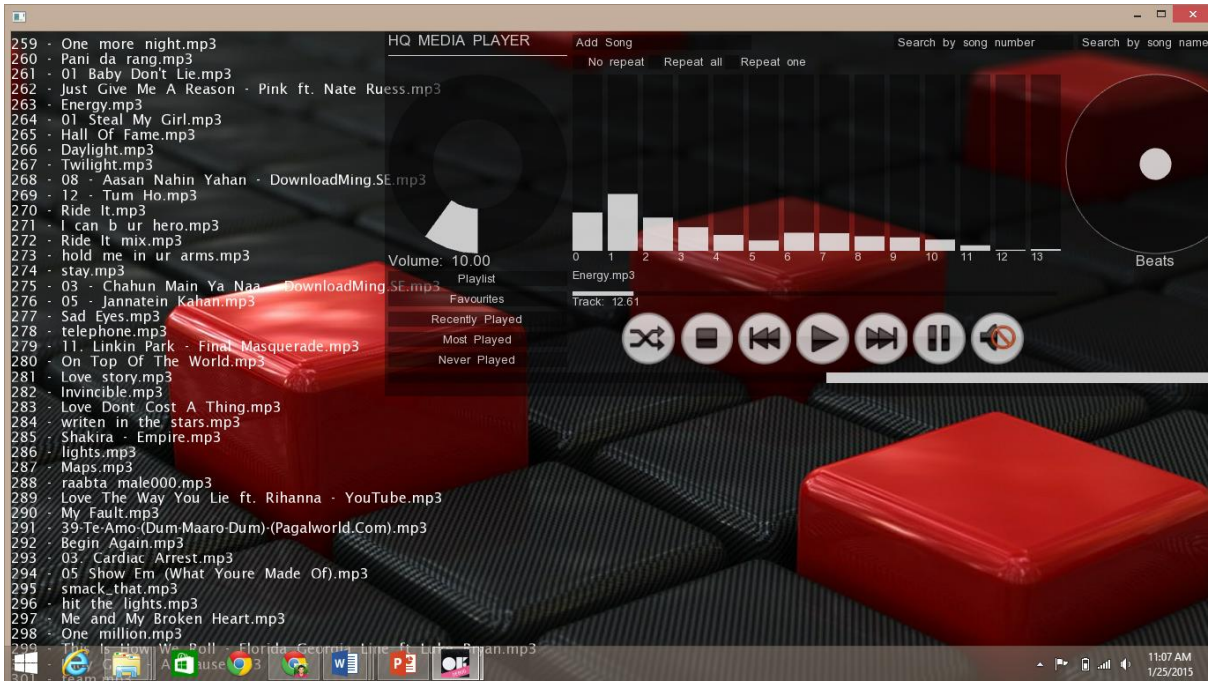
Objective Number	Test Number	Test For	Test Data	Expected Result	Actual Result	Evidence Number
1	1	Adding songs	Directory address	Songs are added	Songs were added	1
2	2	Playing a song from playlist	Song number	Song should be played	Song played	2
3	3	Checking buttons	Clicking play	Playlist gets played	Song played	3
3	4	Checking buttons	Clicking shuffle	Playlist shuffled	Playlist shuffled	4
4	5	Search by name	Name of song	List of relevant songs displyed	List of relevant songs displayed	5

Evidence 1

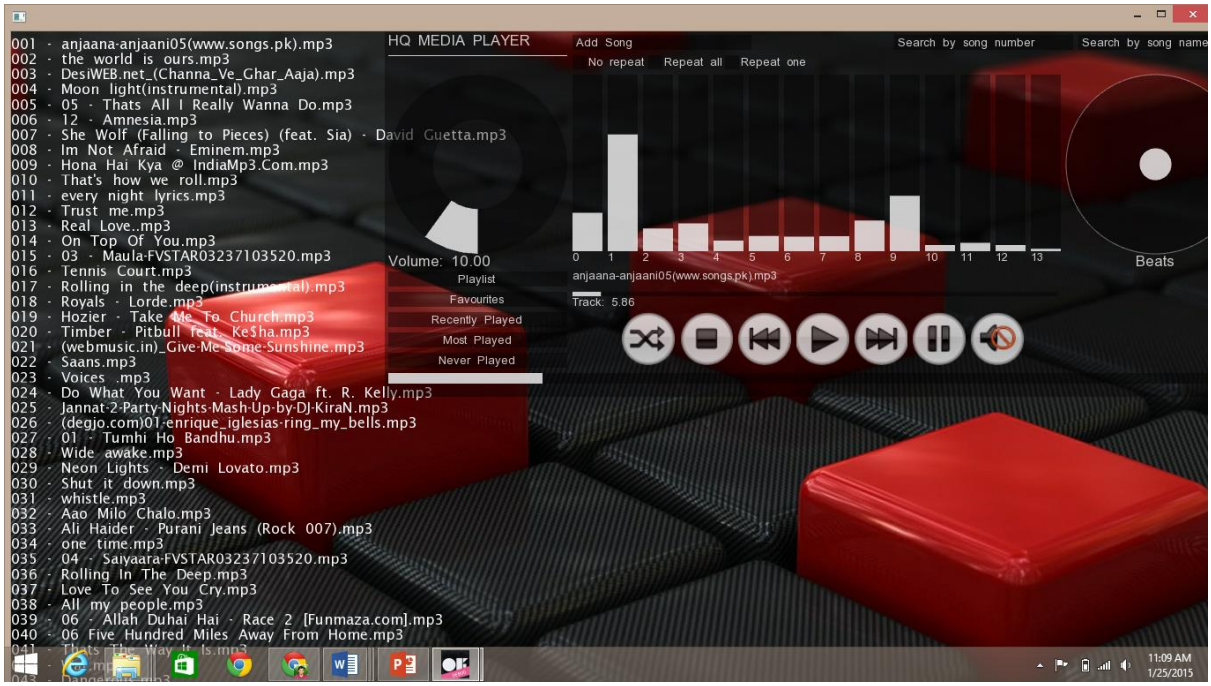


Evidence 2


HQ MEDIA PLAYER
Add Song
No repeat
Repeat all
Repeat one
Search by song name
Volume: 10.00
Playlist
Favourites
Recently Played
Most Played
Never Played
anjaana-anjaani05(www.songs.pk).mp3
Track: 59.45
Beats
11:07 AM
1/25/2015

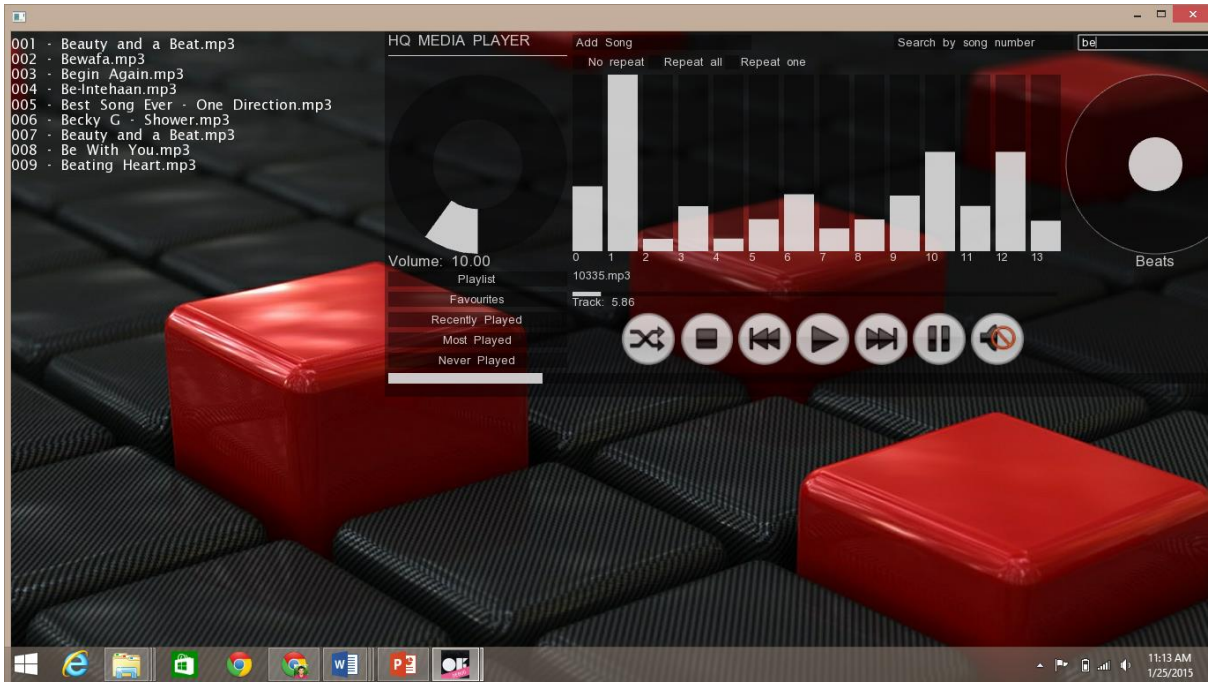

HQ MEDIA PLAYER
Add Song
No repeat
Repeat all
Repeat one
Search by song number
Search by song name
Volume: 10.00
Playlist
Favourites
Recently Played
Most Played
Never Played
Energy.mp3
Track: 12.61
Beats
11:07 AM
1/25/2015

Evidence 3



Evidence 4

Evidence 5

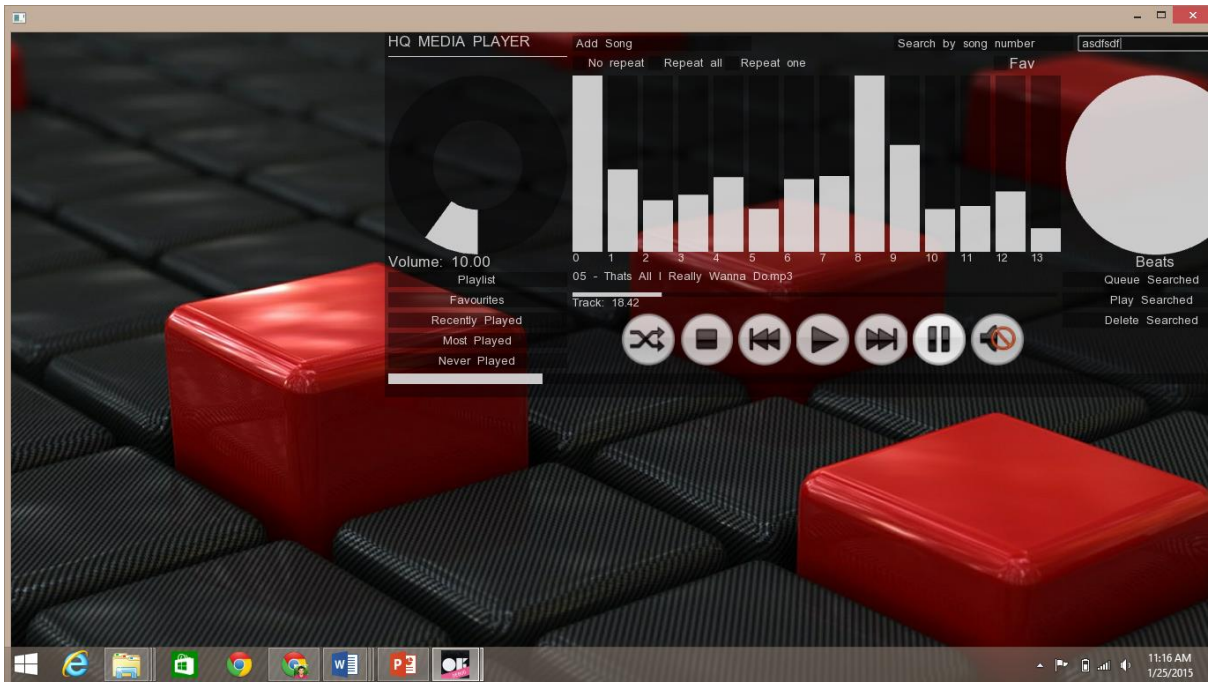


Test type Abnormal

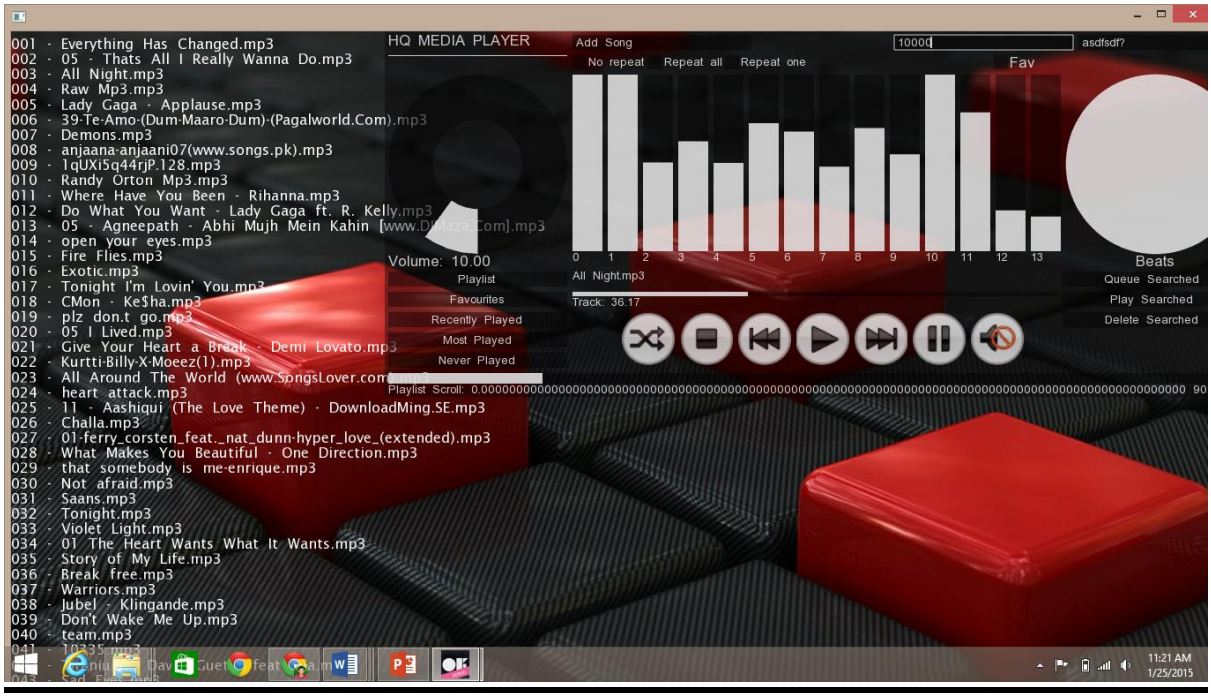
Objective Number	Test Number	Test For	Test Data	Expected Result	Actual Result	Evidence Number
1	6	Adding songs	Wrong data	No change	No change	6
2	7	Searching	Wrong name	No change	No change	7
3	8	Search by number	Non existent number	No change	No change	8

Evidence 6

Evidence 7



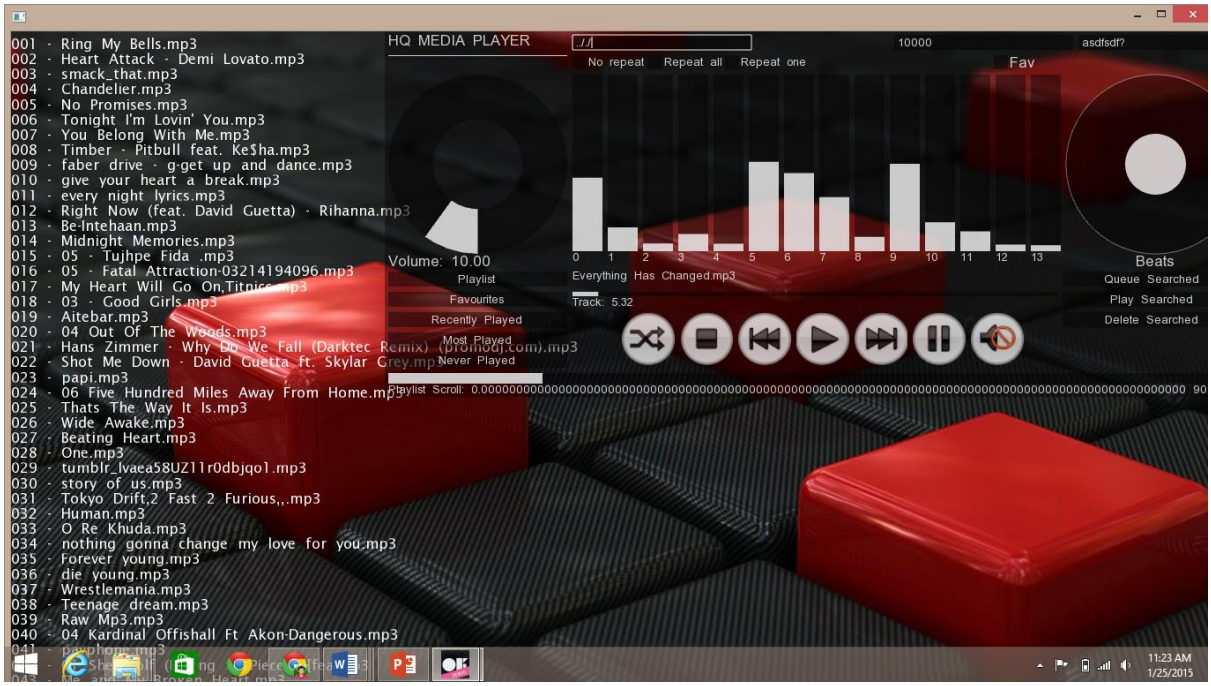
Evidence 8



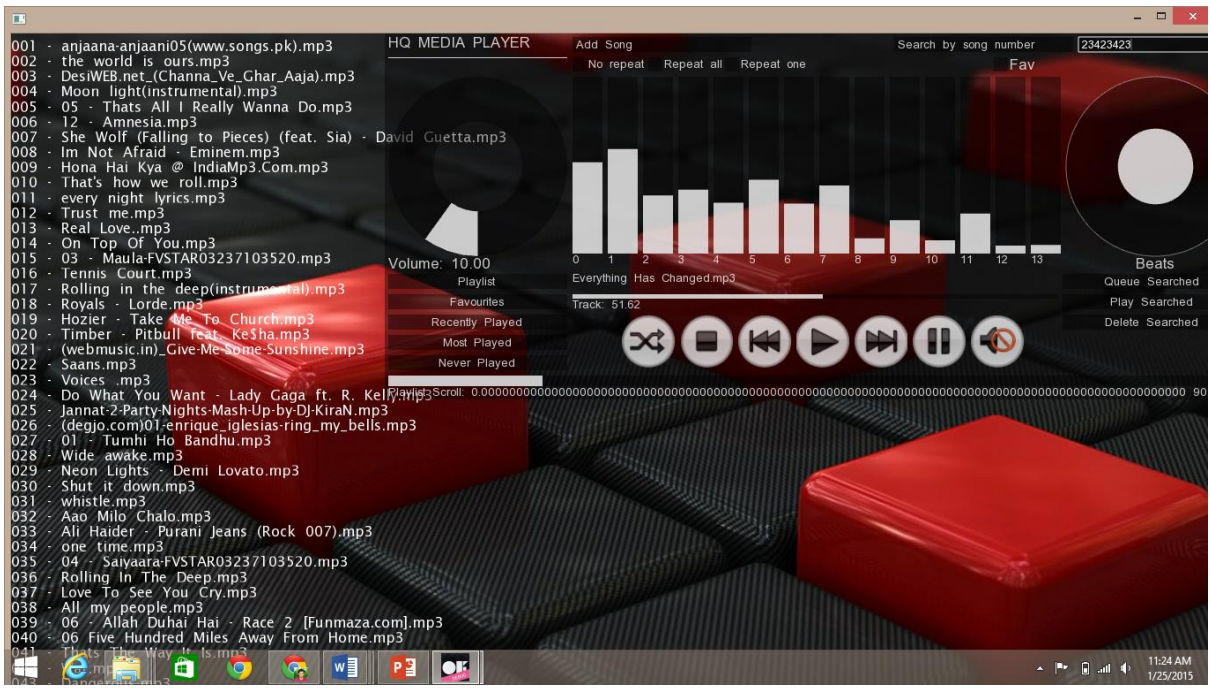
Test type Extreme

Objective Number	Test Number	Test For	Test Data	Expected Result	Actual Result	Evidence
1	9	Adding song	symbols	No change	No change	9
2	10	Searching by name	numbers	No change	No change	10
3	11	buttons	Random clicking	No change	No change	11

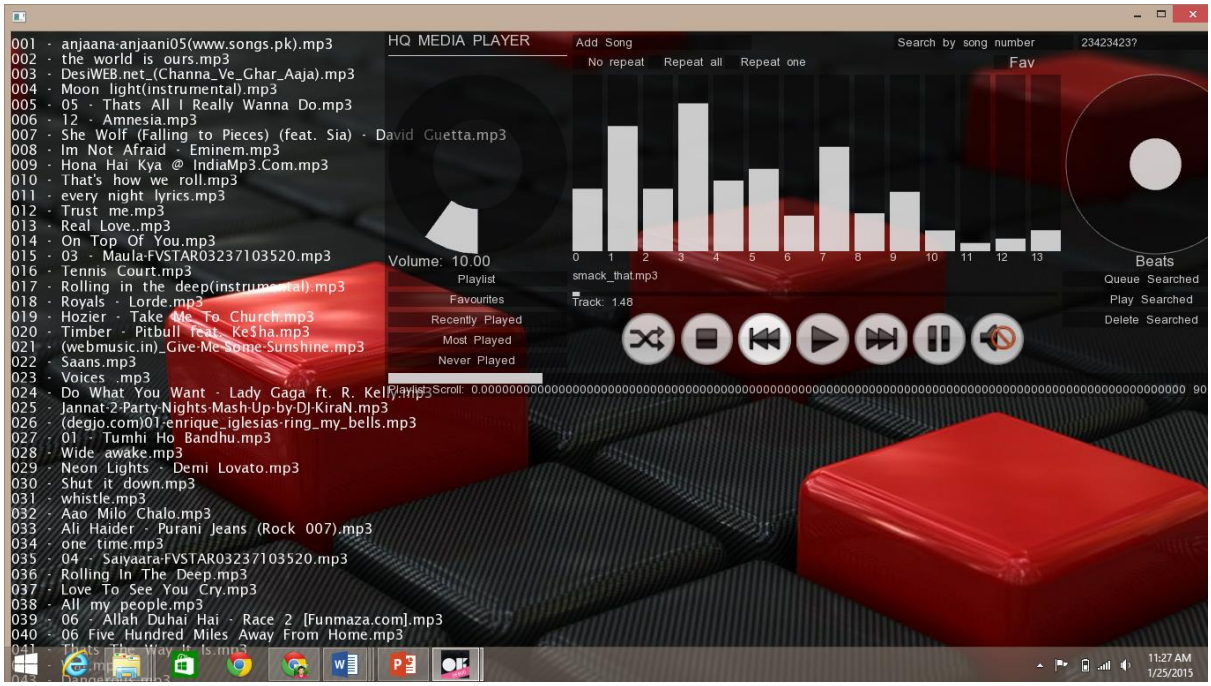
Evidence 9



Evidence 10



Evidence 11


HQ MEDIA PLAYER
Add Song
Search by song number
23423423?
No repeat
Repeat all
Repeat one
Fav
001 - anjaana-anjaani05(www.songs.pk).mp3
002 - the world is ours.mp3
003 - DesiWEB.net_(Channa_Ve_Ghar_Aaja).mp3
004 - Moon light(instrumental).mp3
005 - 05 - Thats All I Really Wanna Do.mp3
006 - 12 - Amnesia.mp3
007 - She Wolf (Falling to Pieces) (feat. Sia) - David Guetta.mp3
008 - Im Not Afraid - Eminem.mp3
009 - Hona Hai Kya @ IndiaMp3.Com.mp3
010 - That's how we roll.mp3
011 - every night lyrics.mp3
012 - Trust me.mp3
013 - Real Love..mp3
014 - On Top Of You.mp3
015 - 03 - Maula-FVSTAR03237103520.mp3
016 - Tennis Court.mp3
017 - Rolling in the deep(instrumental).mp3
018 - Royals - Lorde.mp3
019 - Hozier - Take Me To Church.mp3
020 - Timber - Pitbull feat. Ke$ha.mp3
021 - (webmusic.in)_Give-Me-some-Sunshine.mp3
022 - Saans.mp3
023 - Voices .mp3
024 - Do What You Want - Lady Gaga ft. R. Kelly.mp3
025 - Jannat-2-Party-Nights-Mash-Up-by-DJ-KiraN.mp3
026 - (degjo.com)01-enrique_iglesias-ring_my_bells.mp3
027 - 01 - Tumhi Ho Bandhu.mp3
028 - Wide awake.mp3
029 - Neon Lights - Demi Lovato.mp3
030 - Shut it down.mp3
031 - whistle.mp3
032 - Aao Milo Chalo.mp3
033 - Ali Haider - Purani Jeans (Rock 007).mp3
034 - one time.mp3
035 - 04 - Saiyaara-FVSTAR03237103520.mp3
036 - Rolling In The Deep.mp3
037 - Love To See You Cry.mp3
038 - All my people.mp3
039 - 06 - Allah Duhai Hai - Race 2 [Funmaza.com].mp3
040 - 06 Five Hundred Miles Away From Home.mp3
Volume: 10.00
Playlist
Favourites
Recently Played
Most Played
Never Played
smack_that.mp3
Track: 1.48
Beats
Queue Searched
Play Searched
Delete Searched
11:27 AM
1/25/2015

Evaluation

At this juncture we have to evaluate the new system to reach an assessment whether it achieves all our predetermined objectives. This will enable us to remove any hiccups which if found.

Objective 1

Enabling addition of songs and saving it using different data structures for efficient retrieval and search operations

The user has the option to smoothly add a song. File handling is used to store the data permanently while various data structures are populated each time the program is run to provide optimized functionalities like search, queue etc.

Test number: 1
Test number: 6
Test number: 9

Objective 2

Enabling the use of validation rules, input masks and format checks to guarantee integrity of data and minimize data input error. Fixed length checks like specifying field sizes can be used to lessen the space required to store data;

The new system has been set to automatically react with an error message if any invalid data is entered. For this purpose conditional statements have been used to reject any invalid data entry.

Test number: 2
Test number: 7
Test number: 10

Objective 3

Menus will be used to give user a set of options to choose from so that input errors are minimized.

A user friendly interface has been developed which ensure a clear representation of the options available to reduce input errors.

Test number: 3
Test number: 4
Test number: 8

Test number: 11

Objective 4

Search options will be provided using data structures such as tries to optimize efficiency.

Trie along with double linked list have been used to provide the user with searching options

Test number: 5

Overall the system has been developed to work optimally never making the user wait for more than a blink of an eye, this being an important feature of the program.
Furthermore, the space occupied on the Ram has also been reduced by using algorithms which do not make useless copies of the same data.

System Development

This audio player will provide its user with a smooth and novel experience of listening to their favorite songs without the hassles of an over complicated player. However, this may require some further developments, which are as follows:

- The system needs to be neatly bundled into an installation package so that the user can easily install it on his or her work station.

Code

Main.cpp

```
#include"ofApp.h"

int main()
{
	try{
		ofSetupOpenGL(1366,768,OF_WINDOW);		// <-------- setup the GL context

		// this kicks off the running of HQ Media Player
		ofRunApp(new ofApp());
		return 0;
	}
	catch (int n){
		if (n == -1){
			cout << "Extension not valid" << endl;
		}
		if (n == -2){
			cout << "This number is not in the middle" << endl;
		}

	}
}

//========================================================================
```

ofApp.h

```
#pragma once

#include "HQPlayer.h"
#include "ofMain.h"
#include "ofxUI.h"


class ofApp : public ofBaseApp{
public:
	void setup();//setups the program by reading all data from files and setting gui and default values of widgets
	void update();//updates in background the values changing
	void draw();//draws constantly changing features like slider values etc
	void exit(); //writes files , delete (frees memory)
	ofTrueTypeFont* listplay;
	void keyPressed(int key);
	void keyReleased(int key);
	void mouseMoved(int x, int y );
	void mouseDragged(int x, int y, int button);
	void mousePressed(int x, int y, int button);
	void mouseReleased(int x, int y, int button);
	void windowResized(int w, int h);
	void dragEvent(ofDragInfo dragInfo);
	void gotMessage(ofMessage msg);

	void drawGrid(float x, float y);
	ofxUIRangeSlider* range;
	void setGUI();
	ofxUIToggle* markfavSearch;
	ofxUILabelButton* queueSearch;
	ofxUILabelButton* playSearch;
	ofxUILabelButton* deleteSearch;
	song* searchedSong;
	int searchedIndex;
	ofxUISuperCanvas *gui3;
	HQPlayer* HQplayer;
	ofxUILabel* current;
	ofxUITextInput *textInput;
	ofxUITextInput *textInput1;
	ofxUITextInput *textInput2;
	ofxUISlider * track;
```

```
        ofxUISlider * equalizer0;
        ofxUISlider * equalizer1;
        ofxUISlider * equalizer2;
        ofxUISlider * equalizer3;
        ofxUISlider * equalizer4;
        ofxUISlider * equalizer5;
        ofxUISlider * equalizer6;
        ofxUISlider * equalizer7;
        ofxUISlider * equalizer8;
        ofxUISlider * equalizer9;
        ofxUISlider * equalizer10;
        ofxUISlider * equalizer11;
        ofxUISlider * equalizer12;
        ofxUISlider * equalizer13;
        ofxUICircleSlider* beats;

        bool hideGUI;

        float red, green, blue;
        bool bdrawGrid;
        bool bdrawPadding;
        Heap* mostPlayed;
        void guiEvent(ofxUIEventArgs &e);
        string playlist;
        int min,max;
        int selection;
        int nBandsToGet;
        float       * fftSmoothed;
        ofImage *img;


};
```

ofApp.cpp

```
;
        red = 233; blue = 233; green = 233;
        hideGUI = false;
        bdrawGrid = false;
        bdrawPadding = false;
        textInput = NULL;
        img = new ofImage();
        img->loadImage("1.jpg");

        nBandsToGet = 128;

        // the fft needs to be smoothed out, so we create an array of floats
        // for that purpose:
        fftSmoothed = new float[nBandsToGet];
        for (int i = 0; i < nBandsToGet; i++){
                fftSmoothed[i] = 0;
        }

        setGUI();
        song* readSong= new song();
        ifstream infile("playlist.txt");
        if(!infile)
                cout<<"Couldn't load the Playlist file"<<endl;
        else
        {
                while(!infile.eof())
                {

                        infile>>(*readSong);
                        if(!infile.good())
                                break;
                        song* newSong=new song(readSong->count,readSong->get_name(),readSong->get_directory());
                        HQplayer->get_playlist()->append(newSong);
                        HQplayer->insert_trie(newSong);

                }
        }
        infile.close();
        infile.open("favourites.txt");
```

```
	if(!infile)
		cout<<"Couldn't load the Favourites file"<<endl;
	else
	{
		while(!infile.eof())
		{

			infile>>(*readSong);
			if(!infile.good())
				break;
			song* newSong=new song(readSong->count,readSong->get_name(),readSong->get_directory());
			HQplayer->get_favourites()->append(newSong);
		}
	}
	infile.close();
	int count=HQplayer->get_playlist()->count;
	mostPlayed= new Heap(count);
	HQplayer->get_playlist()->move_to_start();
	for(int i=0;i<count;i++)
	{
		mostPlayed->insert(HQplayer->get_playlist()->get_song());
		HQplayer->get_playlist()->next();
	}
	HQplayer->get_playlist()->move_to_start();

	//gui3->loadSettings("gui3Settings.xml");
}

//--------------------------------------------------------------
void ofApp::update(){
	track->setValue(HQplayer->get_position()*100);
	if(track->getValue()<=100.0&&track->getValue()>=99.0){
		current->setLabel(HQplayer->next());
	}
	mostPlayed->sort();
	if(selection==1)
		playlist=HQplayer->display(min,max);
	else if (selection==2)
		playlist=HQplayer->display_favourites(min,max);
	else if (selection==3)
		playlist=HQplayer->get_recent()->display(min,max);
	else if (selection==4)
		playlist=mostPlayed->display(min,max);
	else if (selection==5)
		playlist=mostPlayed->neverdisplay(min,max);
	else if (selection ==6)
		playlist= HQplayer->get_trie()->display(textInput1->getTextString(),min,max);
	current->setLabel(HQplayer->current_song);
	range->setMax(HQplayer->get_playlist()->count);
	// update the sound playing system:
	ofSoundUpdate();
	// (5) grab the fft, and put in into a "smoothed" array,
	//		by taking maximums, as peaks and then smoothing downward
	float * val = ofSoundGetSpectrum(nBandsToGet);		// request 128 values for fft
	for (int i = 0;i < nBandsToGet; i++){

		// let the smoothed calue sink to zero:
		fftSmoothed[i] *= 0.96f;

		// take the max, either the smoothed or the incoming:
		if (fftSmoothed[i] < val[i]) fftSmoothed[i] = val[i];

	}


}

//--------------------------------------------------------------
void ofApp::draw(){
	ofBackground(red, green, blue, 255);
	ofPushStyle();
	ofEnableBlendMode(OF_BLENDMODE_ALPHA);
	//ofSetColor(255,255,255,100);
	img->draw(ofGetWidth()/2,ofGetHeight()/2);
```

```
		equalizer0->setValue(fftSmoothed[0] * 100);
		equalizer1->setValue(fftSmoothed[5] * 300);
		equalizer2->setValue(fftSmoothed[10] * 300);
		equalizer3->setValue(fftSmoothed[15] * 500);
		equalizer4->setValue(fftSmoothed[20] * 500);
		equalizer5->setValue(fftSmoothed[25] * 700);
		equalizer6->setValue(fftSmoothed[30] * 700);
		equalizer7->setValue(fftSmoothed[35] * 700);
		equalizer8->setValue(fftSmoothed[40] * 700);
		equalizer9->setValue(fftSmoothed[53] * 1000);
		equalizer10->setValue(fftSmoothed[61] * 1000);
		equalizer11->setValue(fftSmoothed[68] * 1000);
		equalizer12->setValue(fftSmoothed[74] * 1000);
		equalizer13->setValue(fftSmoothed[80] * 1000);
		beats->setValue(fftSmoothed[0] * 80);




		listplay->drawString(playlist,0,20);
		if(bdrawGrid)
		{
			ofSetColor(255, 255, 255, 25);
			drawGrid(8,8);
		}

		ofPopStyle();

		ofSetRectMode(OF_RECTMODE_CENTER);
}

void ofApp::guiEvent(ofxUIEventArgs &e)
{
		string name = e.getName();
		int kind = e.getKind();
		cout << "got event from: " << name << endl;

		if(name == "ADD")
		{

			ofxUITextInput *is = (ofxUITextInput *) e.widget;
			ofDirectory dir(is->getTextString());
			dir.allowExt("mp3");
			int num = dir.listDir();
			for(unsigned int i=0;i<num;i++)
			{
				song* newSong=new song(0,dir.getName(i),dir.getPath(i));
				HQplayer->get_playlist()->append(newSong);
				HQplayer->insert_trie(newSong);
			}
		}
		else if(name == "Stop")
		{
			HQplayer->set_stop();
		}
		else if(name == "Pause")
		{
			HQplayer->set_pause();
		}
		else if(name == "SEARCH")
		{
			ofxUITextInput* input = (ofxUITextInput*)e.getButton();
			searchedIndex = ofToInt(input->getTextString());
			if (selection==2)
				searchedSong=HQplayer->get_favourites()->search(searchedIndex);
			else if (selection==3)
				searchedSong=HQplayer->get_recent()->search(searchedIndex);
			else if (selection==4)
				searchedSong=mostPlayed->search(searchedIndex);
			else if (selection==5)
				searchedSong=mostPlayed->neversearch(searchedIndex);
			else
				searchedSong=HQplayer->search(searchedIndex);
```

```
		deleteSearch->setVisible(true);
		markfavSearch->setVisible(true);
		queueSearch->setVisible(true);
		playSearch->setVisible(true);
	}
	else if(name == "SEARCH 2")
	{
		ofxUITextInput* input = (ofxUITextInput*)e.getButton();
		searchedSong=HQplayer->search(input->getTextString());
		deleteSearch->setVisible(true);
		markfavSearch->setVisible(true);
		queueSearch->setVisible(true);
		playSearch->setVisible(true);
		selection =6;
	}
	else if(name == "Play")
	{
		HQplayer->play_from_start();
	}
	else if(name == "Mute")
	{
		HQplayer->mute();
	}
	else if(name == "Next")
	{

		current->setLabel(HQplayer->next());
	}
	else if(name == "Previous")
	{

		HQplayer->previous();
	}
	else if(name == "Shuffle")
	{
		HQplayer->shuffle();
	}

	else if(name == "Repeat one")
	{
		HQplayer->repeat_one();
	}
	else if(name == "Repeat all")
	{
		HQplayer->repeat_all();
	}
	else if(name == "No repeat")
	{
		HQplayer->no_repeat();
	}
	else if(name == "Playlist")
	{
		selection=1;
	}
	else if(name == "Track")
	{
		ofxUISlider* sli = (ofxUISlider*) e.widget;
		HQplayer->set_position(sli->getValue()/100);
	}
	else if(name == "Volume")
	{
		ofxUIRotarySlider *vol = (ofxUIRotarySlider *) e.widget;
		float n=(float) vol->getValue()/100;
		HQplayer->set_volume(n);
	}
	else if(name == "Fav")
	{
		HQplayer->mark_favourite(searchedSong);
		deleteSearch->setVisible(false);
		markfavSearch->setVisible(false);
		queueSearch->setVisible(false);
		playSearch->setVisible(false);

	}
	else if(name == "Favourites")
```

```
		{
			selection=2;
		}
		else if(name == "Recently Played")
		{
			selection =3;
		}
		else if(name == "Most Played")
		{
			selection=4;
		}
		else if(name == "Never Played")
		{
			selection=5;
		}
		else if(name == "Playlist Scroll")
		{
			ofxUIRangeSlider* scroll = (ofxUIRangeSlider *) e.widget;
			min=(int) scroll->getValueLow();
			max=(int) scroll->getValueHigh();
		}
		else if(name == "Play Searched")
		{
			HQplayer->play(searchedSong);
			deleteSearch->setVisible(false);
			markfavSearch->setVisible(false);
			queueSearch->setVisible(false);
			playSearch->setVisible(false);
		}
		else if(name == "Queue Searched")
		{
			HQplayer->queue_song(searchedSong);
			deleteSearch->setVisible(false);
			markfavSearch->setVisible(false);
			queueSearch->setVisible(false);
			playSearch->setVisible(false);

		}
		else if(name == "Delete Searched")
		{
			HQplayer->delete_song(searchedIndex,searchedSong);
			deleteSearch->setVisible(false);
			markfavSearch->setVisible(false);
			queueSearch->setVisible(false);
			playSearch->setVisible(false);

		}
		else if(name == "TEXT INPUT")
		{
			ofxUITextInput *ti = (ofxUITextInput *) e.widget;
			if(ti->getInputTriggerType() == OFX_UI_TEXTINPUT_ON_ENTER)
			{
				cout << "ON ENTER: ";
			}
			else if(ti->getInputTriggerType() == OFX_UI_TEXTINPUT_ON_FOCUS)
			{
				cout << "ON FOCUS: ";
			}
			else if(ti->getInputTriggerType() == OFX_UI_TEXTINPUT_ON_UNFOCUS)
			{
				cout << "ON BLUR: ";
			}
			string output = ti->getTextString();
			cout << output << endl;
		}

		//HQplayer->no_repeat();
}

//------------------------------------------------------------
void ofApp::exit()
{

		gui3->saveSettings("gui3Settings.xml");
		ofstream outfile("playlist.txt");
```

```
	if(!outfile)
		cout<<"Couldn't load the Playlist file"<<endl;
	else
	{
		HQplayer->get_playlist()->move_to_start();
		for(int i=0;i<HQplayer->get_playlist()->count;i++)
		{
			outfile<<(*HQplayer->get_playlist()->get_song());
			HQplayer->get_playlist()->next();
		}
	}
	outfile.close();

	outfile.open("favourites.txt");
	if(!outfile)
		cout<<"Couldn't load the favourites file"<<endl;
	else
	{
		HQplayer->get_favourites()->move_to_start();
		for(int i=0;i<HQplayer->get_favourites()->get_count();i++)
		{
			outfile<<(*HQplayer->get_favourites()->get_data());
			HQplayer->get_favourites()->next();
		}
	}
	outfile.close();

	delete HQplayer;
	delete listplay;
	delete gui3;
	delete img;
}

//--------------------------------------------------------------
void ofApp::keyPressed(int key){

	switch (key)
	{

	case ']':
		{
			HQplayer->next();
			break;
		}
	case '[':
		{
			HQplayer->previous();
			break;
		}


	case '`':
		{
			ofToggleFullscreen();
			break;
		}
	case '.':
		{
			HQplayer->shuffle();
			break;
		}
	default:
		break;
	}
}

void ofApp::drawGrid(float x, float y)
{
	float w = ofGetWidth();
	float h = ofGetHeight();

	for(int i = 0; i < h; i+=y)
	{
		ofLine(0,i,w,i);
	}
```

```
		for(int j = 0; j < w; j+=x)
		{
			ofLine(j,0,j,h);
		}
}


void ofApp::setGUI()
{
		gui3 = new ofxUISuperCanvas("HQ MEDIA PLAYER");
		gui3->setWidgetPosition(OFX_UI_WIDGET_POSITION_RIGHT);
		textInput2=gui3->addTextInput("ADD", "Add Song");
		textInput2->setTriggerOnClick(false);
		textInput2->setAutoClear(true);
		textInput2->setAutoUnfocus(true);
		gui3->addSpacer(150,0)->setColorBack(ofxUIColor(red, green, blue, 255));
		textInput = gui3->addTextInput("SEARCH", "Search by song number");
		textInput->setTriggerOnClick(false);
		textInput->setAutoClear(true);
		textInput->setAutoUnfocus(true);
		textInput->setOnlyNumericInput(true);
		textInput1=gui3->addTextInput("SEARCH 2", "Search by song name");
		textInput1->setTriggerOnClick(false);
		textInput1->setAutoClear(true);
		textInput1->setAutoUnfocus(true);
		gui3->setWidgetFontSize(OFX_UI_FONT_MEDIUM);
		gui3->setWidgetPosition(OFX_UI_WIDGET_POSITION_DOWN);

		gui3->addSpacer();

		vector<string> names;
		names.push_back("No repeat");
		names.push_back("Repeat all");
		names.push_back("Repeat one");
		gui3->setWidgetPosition(OFX_UI_WIDGET_POSITION_RIGHT);
		gui3->addRadio("LOOPING", names, OFX_UI_ORIENTATION_HORIZONTAL);
		gui3->addSpacer(200,0);
		markfavSearch=gui3->addToggle("Fav",false);
		markfavSearch->setVisible(false);
		gui3->setWidgetPosition(OFX_UI_WIDGET_POSITION_DOWN);
		gui3->addRotarySlider("Volume", 0.0, 100.0, 10.0,100,0.0,0.0,1);
		gui3->setWidgetPosition(OFX_UI_WIDGET_POSITION_RIGHT);
		equalizer0=gui3->addSlider("0", 0.0, 255.0, 150, 34, 200);
		gui3->setWidgetPosition(OFX_UI_WIDGET_POSITION_RIGHT);
		equalizer1=gui3->addSlider("1", 0.0, 250.0, 150, 34, 200);
		equalizer2=gui3->addSlider("2", 0.0, 250.0, 150, 34, 200);
		equalizer3=gui3->addSlider("3", 0.0, 250.0, 150, 34, 200);
		equalizer4=gui3->addSlider("4", 0.0, 250.0, 150, 34, 200);
		equalizer5=gui3->addSlider("5", 0.0, 250.0, 150, 34, 200);
		equalizer6=gui3->addSlider("6", 0.0, 250.0, 150, 34, 200);
		equalizer7=gui3->addSlider("7", 0.0, 250.0, 150, 34, 200);
		equalizer8=gui3->addSlider("8", 0.0, 250.0, 150, 34, 200);
		equalizer9=gui3->addSlider("9", 0.0, 250.0, 150, 34, 200);
		equalizer10=gui3->addSlider("10", 0.0, 250.0, 150, 34, 200);
		equalizer11=gui3->addSlider("11", 0.0, 250.0, 150, 34, 200);
		equalizer12=gui3->addSlider("12", 0.0, 250.0, 150, 34, 200);
		equalizer13=gui3->addSlider("13", 0.0, 250.0, 150, 34, 200);
		beats= gui3->addCircleSlider("Beats",0,250.0,150);

		gui3->setWidgetPosition(OFX_UI_WIDGET_POSITION_DOWN);
		gui3->setWidgetFontSize(OFX_UI_FONT_SMALL);
		gui3->addLabelButton("Playlist",false);
		gui3->setWidgetPosition(OFX_UI_WIDGET_POSITION_RIGHT);
		current= gui3->addLabel("Current", "Song Title", OFX_UI_FONT_SMALL);
		gui3->addSpacer(482,0);
		queueSearch=gui3->addLabelButton("Queue Searched",false);
		queueSearch->setVisible(false);
		gui3->setWidgetPosition(OFX_UI_WIDGET_POSITION_DOWN);
		gui3->addLabelButton("Favourites",false);
		gui3->setWidgetPosition(OFX_UI_WIDGET_POSITION_RIGHT);
		track=gui3->addSlider("Track",0.0,100.0,0.0,550,5);
		playSearch=gui3->addLabelButton("Play Searched",false);
		playSearch->setVisible(false);
		gui3->setWidgetPosition(OFX_UI_WIDGET_POSITION_DOWN);
```

```
		gui3->addLabelButton("Recently Played",false);
		gui3->setWidgetFontSize(OFX_UI_FONT_SMALL);
		gui3->setWidgetPosition(OFX_UI_WIDGET_POSITION_RIGHT);
		gui3->addSpacer(50,0);
		gui3-
>addImageButton("Shuffle","F:\\Downloads\\of_v0.8.4_vs_release\\of_v0.8.4_vs_release\\apps\\myApps\\HQMediaPlayer\\src\\shuffle.
png",true,60.0,60.0);
		gui3-
>addImageButton("Stop","F:\\Downloads\\of_v0.8.4_vs_release\\of_v0.8.4_vs_release\\apps\\myApps\\HQMediaPlayer\\src\\stop.png",t
rue,60.0,60.0);
		gui3-
>addImageButton("Previous","F:\\Downloads\\of_v0.8.4_vs_release\\of_v0.8.4_vs_release\\apps\\myApps\\HQMediaPlayer\\src\\previo
us.png",true,60.0,60.0);
		gui3-
>addImageButton("Play","F:\\Downloads\\of_v0.8.4_vs_release\\of_v0.8.4_vs_release\\apps\\myApps\\HQMediaPlayer\\src\\play.png",t
rue,60.0,60.0);
		gui3-
>addImageButton("Next","F:\\Downloads\\of_v0.8.4_vs_release\\of_v0.8.4_vs_release\\apps\\myApps\\HQMediaPlayer\\src\\next.png",
true,60.0,60.0);
		gui3-
>addImageToggle("Pause","F:\\Downloads\\of_v0.8.4_vs_release\\of_v0.8.4_vs_release\\apps\\myApps\\HQMediaPlayer\\src\\pause.pn
g",true,60.0,60.0);
		gui3-
>addImageButton("Mute","F:\\Downloads\\of_v0.8.4_vs_release\\of_v0.8.4_vs_release\\apps\\myApps\\HQMediaPlayer\\src\\mute.png
",true,60.0,60.0);
		gui3->addSpacer(32,0);
		deleteSearch=gui3->addLabelButton("Delete Searched",false);
		deleteSearch->setVisible(false);
		gui3->setWidgetPosition(OFX_UI_WIDGET_POSITION_DOWN);
		gui3->addLabelButton("Most Played",false);
		gui3->addLabelButton("Never Played",false);

		range=gui3->addRangeSlider("Playlist Scroll",0.0,1000.0,0.0,100.0,950,12);
		range->setLabelPrecision(100);

		gui3->setDrawPadding(false);
		gui3->setGlobalButtonDimension(OFX_UI_GLOBAL_BUTTON_DIMENSION);
		gui3->setPosition(212*2, 0);
		gui3->autoSizeToFitWidgets();

		ofAddListener(gui3->newGUIEvent,this,&ofApp::guiEvent);
}


//--------------------------------------------------------------
void ofApp::keyReleased(int key){

}

//--------------------------------------------------------------
void ofApp::mouseMoved(int x, int y ){

}

//--------------------------------------------------------------
void ofApp::mouseDragged(int x, int y, int button){

}

//--------------------------------------------------------------
void ofApp::mousePressed(int x, int y, int button){

}

//--------------------------------------------------------------
void ofApp::mouseReleased(int x, int y, int button){

}

//--------------------------------------------------------------
void ofApp::windowResized(int w, int h){

}

//--------------------------------------------------------------
```

```
void ofApp::gotMessage(ofMessage msg){

}

//--------------------------------------------------------------
void ofApp::dragEvent(ofDragInfo dragInfo){

}
```

HQPlayer.h

```
#pragma once
#include"Llist.h"
#include"trie.h"
#include"Queue.h"
#include"dblist.h"
#include"ofSoundPlayer.h"
#include"Stack.h"
#include"Heap.h"

class HQPlayer
{
private:
		bool pause;
		Queue* queue;
		dblist* playlist;
		LList* favourites;
		Trie* trie;
		bool repeat;
		Stack* recent;
public:
		string current_song;
		ofSoundPlayer* player;
		string play(song* temp);
		string next();
		string previous();
		void repeat_all();//careful about the display function now
		void repeat_one();//repeats the currently playing song when next or previous is pressed
		void no_repeat(); //playling loop ends with the end of playlist
		void shuffle();//shuffle using fisher yates algorithm
		void add_song();
		void delete_song();
		void edit_song();
		string display(int min, int max); //returns string containing songs  between index min and max returned from scroll bar
		void mark_favourite(song* key); //appends into favourites
		void queue_song(song* key); //queues the song into queue
		void delete_song(int n,song* key); //delete song from the playlist, trie, recent played, never played, mostly played if exists
		void backwards_display();//displays playlist backwards
		void search();
		song* search(int); //searches and returns the song by song number in playlist
		song* search(string);//searches and returns the song by song name in playlist
		string get_current_song_name(); //returns current song name
		void insert_trie(song* key); //inserts to trie
		node* get_next(); //returns next song name
		void mute(); //mutes the player
		string play_from_start(); //plays from start
		dblist* get_playlist(); // retuns playlist
		LList* get_favourites();
		Stack* get_recent();
		Trie* get_trie();
		string display_favourites(int min,int max);
		void set_position(float n); //set track postition
		void set_volume(float n); //sets volume
		float get_position(); //returns track position
		void set_stop(); //stops the player
		void set_pause(); //pauses the track
		HQPlayer(); //constructor
};
```

HQPlayer.cpp

```
#include"HQPlayer.h"
#include<time.h>

string HQPlayer::play(song* temp){
		if(temp==NULL) return "\0";
```

```
		player->loadSound(temp->get_directory());
		player->play();
		recent->push(temp);
		temp->count++;
		return current_song=temp->get_name();
}


void HQPlayer::set_position(float n)
{
		player->setPosition(n);
}

float HQPlayer::get_position()
{
		return player->getPosition();
}
void HQPlayer::set_volume(float n)
{
		player->setVolume(n);
}

node* HQPlayer:: get_next()
{
		return playlist->curr->next;

}
string HQPlayer::next(){
		if(repeat==true)
		{
				player->play();
				recent->peek()->count++;
				return recent->peek()->get_name();
		}
		if(!queue->isEmpty()){
				song* temp =queue->dequeue();
				return current_song=play(temp);
		}
		if (playlist->next()){
				return current_song=play(playlist->get_song());
		}
		return current_song="No Song Loaded";
}

string HQPlayer::get_current_song_name()
{
		return current_song=playlist->get_song()->get_name();
}
string HQPlayer::previous(){
		if(repeat==true)
		{
				player->play();
				recent->peek()->count++;
				return recent->peek()->get_name();
		}

		if (playlist->previous())
				return current_song=play(playlist->get_song());
		return current_song="No song loaded";
}

void HQPlayer::repeat_all()
{
		repeat=false;
		playlist->get_end()->next = playlist->get_start();
		playlist->get_start()->prev=playlist->get_end();
}

void HQPlayer::no_repeat()
{
		repeat=false;
		playlist->get_start()->prev = NULL;
		playlist->get_end()->next = NULL;
}
```

```
void HQPlayer::repeat_one()
{
		repeat=true;
		player->setLoop(true);

}
void HQPlayer::shuffle(){
		srand(time(NULL));
		for (int i = playlist->count - 1; i > 0; i--)
		{
			int j = rand() % i;
			playlist->move_to_pos(i);
			node* temp1 = playlist->get_current();
			playlist->move_to_pos(j);
			node* temp2 = playlist->get_current();
			playlist->swap(temp1, temp2);
		}
}


void HQPlayer::insert_trie(song* key)
{
		trie->insert(key);
}



void HQPlayer::delete_song(){
		playlist->delnode();//delete from trie
}


void HQPlayer::mute()
{
		player->setVolume(0.0);
}

string HQPlayer::display(int min, int max)
{
		return playlist->display(min,max);
}

dblist* HQPlayer::get_playlist(){
		return playlist;
}

Stack* HQPlayer::get_recent()
{
		return recent;
}


void HQPlayer::backwards_display(){
		playlist->show();
}
song* HQPlayer::search(int n)
{
		if (n <= 0 ||n > playlist->count)
		{
			cout<<"Invalid input!"<<endl;
			return NULL;
		}
		return playlist->search(n)->data;
}

song* HQPlayer::search(string key)
{
		song* keySong = trie->search(key);
		if (keySong == NULL)
		{
			cout << "No song exists of that name" << endl;
			return NULL;
		}
		else
```

```
                        return keySong;
}

string HQPlayer::play_from_start(){
        if (playlist->get_start() != NULL)
                return play(playlist->get_start()->data);
        else
                cout << "There is no song in the playlist" << endl;
}


LList* HQPlayer::get_favourites()
{
        return favourites;
}
Trie* HQPlayer::get_trie()
{
        return trie;
}

void HQPlayer::set_pause(){
        if(pause==true)
        {
                pause=false;
                player->setPaused(pause);
        }
        else
        {
                pause=true;
                player->setPaused(pause);
        }
}

void HQPlayer::set_stop()
{
        player->stop();
}

string HQPlayer::display_favourites(int min, int max){
        if (favourites->ptr != NULL)
                return favourites->display_list(min,max);
        else
                return "There is no song in favourites";
}
HQPlayer::HQPlayer()
{
        repeat=false;
        playlist = new dblist();
        trie = new Trie();
        favourites = new LList();
        player= new ofSoundPlayer();
        pause=false;
        queue= new Queue();
        recent= new Stack();
        current_song="\0";
}

void HQPlayer::mark_favourite(song* key)
{
        if(key==NULL) return;
        else if (!favourites->find(key))
                favourites->append(key);
        else
                return;
}

void HQPlayer::queue_song(song* key)
{
        if(key==NULL) return;
        queue->enqueue(key);

}

void HQPlayer::delete_song(int n,song* key)
{
```

```
		if(key==NULL) return;
		else if(key==playlist->curr->data) return;
		trie->_delete(key->get_name());
		favourites->_delete(key);
		playlist->delnode(n);

}
```

Heap.h

```
#pragma once

#include"song.h"
class heapNode
{

public:
		song* data;
		bool flag;
		heapNode(){
			data = NULL;
			flag = false;
		}
};

class Heap
{
private:

		int count;
		int size;
		heapNode* heap;
public:

		Heap(int n);

		void sort();

		void swap(heapNode& a, heapNode& b);

		void ReheapUp(int root, int bottom);

		bool insert(song* temp);

		bool remove(song* temp);

		bool IsEmpty();

		song* search(int key);

		song* neversearch(int key);

		bool IsFull();

		string display(int min,int max);

		string neverdisplay(int min,int max);



};
```

Heap.cpp

```
#include"Heap.h"
//constructor
Heap::Heap(int n)
{
		count = 0;
		size = n;
		heap = new heapNode[size]();
}
//swap node data of two nodes
void Heap:: swap(heapNode& a, heapNode& b)
{
```

```
		song* temp = a.data;
		a.data= b.data;
		b.data = temp;
		bool flag=a.flag;
		a.flag=b.flag;
		b.flag=flag;
}
//reheaps the array given the upper and lower index (Max Heap)
void Heap::ReheapUp(int root, int bottom)
{
		int parent;
		int temp;

		// Check base case in recursive calls.  If bottom's index is greater
		// than the root index we have not finished recursively reheaping.
		if (bottom > root)
		{
			parent = (bottom - 1) / 2;
			if (heap[parent].data->count < heap[bottom].data->count)
			{
				// Swap these two elements
				swap(heap[parent],heap[bottom]);

				if (bottom % 2==0 && heap[bottom - 1].data->count < heap[bottom].data->count)
				{
					// Swap these two elements
					swap(heap[bottom-1],heap[bottom]);
				}
			}
			ReheapUp(root, parent);

		}
}
//bubble sort using flag O(n^2)
void Heap::sort()
{
		int i,j,temp,flag=0;
		//Big O(count) in best case
		for(i=0;i<count;i++)
		{

			flag = 0;
			for(j=0;j<count-1;j++)
			{
				if (heap[j].data->count < heap[j+1].data->count)
				{
					swap(heap[j],heap[j+1]);
					flag = 1;
				}
			}
			if(flag==0)
				break;
		}
}
//inserts a song into heap tree BigO(logN)
bool Heap::insert(song* temp)
{
		if (count < size)
		{

			heap[count].data = temp;
			heap[count].flag = true;
			ReheapUp(0, count);
			count++;
			return true;
		}
		else
			return false;
}
//to remove a song from the heap tree
bool Heap::remove(song* temp)
{
		for( int i=0;i<count;i++)
		{
			if(heap[i].data==temp)
```

```
		{
			heap[i].flag=false;
			return true;
		}
	}
	return false;

}

//return true if heap is empty
bool Heap::IsEmpty()
{
	if (count == 0)
		return true;
	else
		return false;

}


//searches and returns the song given the index no in the most played list
song* Heap::search(int key)
{
	if(key>count)
		return NULL;
	if(heap[key-1].data->count==0)
		return NULL;
	else
		return heap[key-1].data;

}
//searches and returns the song given the index no in the never played list
song* Heap::neversearch(int key)
{
	if(key>count)
		return NULL;
	if(heap[key-1].data->count!=0)
		return NULL;
	else
		return heap[key-1].data;

}
//returns true if heap is full
bool Heap::IsFull(){
	if (count == size)
		return true;
	else
		return false;
}

//displays the songs present in the heap tree (most played only )
string Heap::display(int min,int max)
{
	string listplay;
	if(max>count)max=count;
	if(IsEmpty())
		return "There is no song in most played list";
	for (int i = min; i < max; i++)
	{
		if(heap[i].flag&&heap[i].data->count!=0)
			listplay+= song::int_to_char(i+1)+" - "+ "(Times= "+song::int_to_char(heap[i].data->count)+")
"+heap[i].data->get_name() + "\n";
	}
	return listplay;
}

//displays the songs present in the heap tree (never played only )
string Heap::neverdisplay(int min,int max)
{
	string listplay;
	if(max>count)max=count;
	if(IsEmpty())
		return "There is no song in most played list";
	for (int i = min; i < max; i++)
	{
```

```
		if(heap[i].flag&&heap[i].data->count==0)
			listplay+= song::int_to_char(i+1)+" - "+ "(Times= "+song::int_to_char(heap[i].data->count)+")
"+heap[i].data->get_name() + "\n";
	}
	return listplay;
}
```

Dblist.h

```
//dblist.h
#pragma once

#include"song.h"

class node
{
public:
	song *data;
	node *next;
	node *prev;
	int index;
	node(song* s = NULL,int i=0, node* p = NULL, node* n = NULL) : next(n), prev(p),index(i)
	{
		if(s==NULL)
			data= new song();
		else
			data=s;
	}
	void display(){
		cout << "Song name: " << data->get_name().c_str() << endl
			<< "Directory: " << data->get_directory().c_str() << endl;
	}
};
class dblist{
public:
	string addnode();//adds node into dblist
	string append(song* s); //appends song into doubly liinked list
	void delnode(); //delete nodes from double linked list
	void delnode(int num); //deletes node at the place num
	string display(int min, int max); //returns string containing song names between index min and max
	void show(); //displays the linked list in console
	node* search(int p); //searches and returns the searched node at place p if exists
	bool has_next(); //return true if curr->next!=NULL
	bool next();
	bool previous();
	void move_to_start(); //moves current pointer to the start
	song* get_song();
	node* get_current();
	void move_to_end();
	void move_to_pos(int a);
	void swap(node* node1, node* node2); //swaps two nodes
	node* get_start();
	node* get_end();
	dblist();
	static int count;
	node* temp1;
	node* curr;
	node* end;
	node* get_next();
private:
	node *start, *temp2, *temp3;
	static int id;

	friend std::ofstream& operator<< (std::ofstream& stream, const dblist& s);//writing doubly linked list node into file
	friend std::ifstream& operator>> (std::ifstream& stream, dblist& s);//reading doubly linked list node into file
};
```

Dblist.h

```
//dblist.h
```

```
#pragma once

#include"song.h"

class node
{
public:
	song *data;
	node *next;
	node *prev;
	int index;
	node(song* s = NULL,int i=0, node* p = NULL, node* n = NULL) : next(n), prev(p),index(i)
	{
		if(s==NULL)
			data= new song();
		else
			data=s;
	}
	void display(){
		cout << "Song name: " << data->get_name().c_str() << endl
			<< "Directory: " << data->get_directory().c_str() << endl;
	}
};
class dblist{
public:
	string addnode();//adds node into dblist
	string append(song* s); //appends song into doubly liinked list
	void delnode(); //delete nodes from double linked list
	void delnode(int num); //deletes node at the place num
	string display(int min, int max); //returns string containing song names between index min and max
	void show(); //displays the linked list in console
	node* search(int p); //searches and returns the searched node at place p if exists
	bool has_next(); //return true if curr->next!=NULL
	bool next();
	bool previous();
	void move_to_start(); //moves current pointer to the start
	song* get_song();
	node* get_current();
	void move_to_end();
	void move_to_pos(int a);
	void swap(node* node1, node* node2); //swaps two nodes
	node* get_start();
	node* get_end();
	dblist();
	static int count;
	node* temp1;
	node* curr;
	node* end;
	node* get_next();
private:
	node *start, *temp2, *temp3;
	static int id;

	friend std::ofstream& operator<< (std::ofstream& stream, const dblist& s);//writing doubly linked list node into file
	friend std::ifstream& operator>> (std::ifstream& stream, dblist& s);//reading doubly linked list node into file
};
```

Dblist.cpp

```
#include "dblist.h"


dblist::dblist(){
	start = NULL;
	curr = NULL;
	end=NULL;
}
int dblist::id = 0;
int dblist::count = 0;


bool dblist::has_next(){
	if (curr->next != NULL)
		return true;
	else
		return false;
```

```
}

bool dblist::next(){
	if (has_next()){
		curr = curr->next;
		return true;
	}
	else
	{
		cout << "There is no next song" << endl;
		return false;
	}
}

bool dblist::previous(){
	if (curr->prev != NULL){
		curr = curr->prev;
		return true;
	}
	else{
		cout << "There is no previous song" << endl;
		return false;
	}
}

void dblist::move_to_start(){
	curr = start;
}

song* dblist::get_song(){
	return curr->data;
}

node* dblist::get_current(){
	return curr;
}

node* dblist::get_next()
{
	if(has_next())
		return curr->next;
	else
		return NULL;
}
void dblist::move_to_pos(int a)
{
	if (a >= count)
		cout << "The position does not exists" << endl;
	else{
		move_to_start();
		for (int i = 0; i < a; i++)
			next();
	}
}

void dblist::swap(node* node1, node* node2) {
	song* temp_data = node1->data;
	node1->data = node2->data;
	node2->data = temp_data;

	int temp = node1->index;
	node1->index = node2->index;
	node2->index = temp;
}

node* dblist::get_start(){
	return start;
}

node* dblist::get_end(){
	return end;
}

void dblist::move_to_end(){
	curr=end;
```

```
}

string dblist::addnode()        //adding node
{
        id++;
        count++;
        char r;
        song* new_song = new song();
        new_song->new_song();
        cout << "press 's' to add in start,'m' for midd , 'e' for end" << endl;
        cin >> r;
        switch (r)
        {
        case's':            //add start
                if (start == NULL)
                {
                        end=curr=start = new node(new_song, id, NULL, NULL);
                }
                else
                {
                        start = start->prev = new node(new_song, id, NULL, start);
                }
                break;
        case'e':            //add end
                if (start == NULL)
                {
                        end=curr=start = new node(new_song, id, NULL, NULL);
                }
                else
                {
                        end=end->next = new node(new_song, id, end, NULL);
                }
                break;
        case'm':            //add mid
                int num;
                cout << "enter song number after which you want to enter" << endl;
                cin >> num;
                if (num >= count)
                {
                        cout << "No song exists at that number" << endl;
                        throw -2;
                }
                temp2 = start;
                for (int i = 0; i < num; i++)
                {
                        if (start == NULL)
                                cout << "given song not found" << endl;
                        else
                        {
                                temp3 = temp2;
                                temp2 = temp2->next;

                        }
                }

                temp2->prev = temp3->next = new node(new_song, id, temp3, temp2);
                break;
        }
        return new_song->get_name();

}


string dblist::append(song* s)        //adding node
{
        id++;
        count++;
        if (start == NULL)
        {
                end=curr=start = new node(s, id, NULL, NULL);
        }
        else
        {
                end->next = new node(s, id, end, NULL);
                end=end->next;
```

```
		}
		return s->get_name();

}
string dblist::display(int min, int max)	//displaying
{
		string listplay;
		if (start == NULL)
			return "No songs in the playlist\n";
		temp3 = start;
		int i = min+1;
		for(int j=1;j<i;j++)
			temp3=temp3->next;
		if(max>count)max=count;
		if (start == NULL)
			listplay= "No songs in the playlist\n";
		else
		{
			for (; i < max;i++)
			{
				listplay +=  song::int_to_char(i);
				listplay+= " - ";
				listplay+=temp3->data->get_name();
				listplay+= "\n";
				temp3 = temp3->next;
			}
			listplay+= song::int_to_char(i) + " - " + temp3->data->get_name() + "\n";
		}
		return listplay;
}

node* dblist::search(int temp)		//searching
{

		temp1 = start;
		for (int i = 1; i < temp; i++)
			temp1 = temp1->next;
		return temp1;

}
void dblist::delnode()		//deleting
{

		char d;
		cout << "press 's' to delete from start,'m' for midd , 'e' for end" << endl;
		cin >> d;
		switch (d)
		{
		case's':		//delete start
			if (start == NULL)
			{
				cout << "no song to delete" << endl;
			}
			else
			{
				temp1 = start;
				start = start->next;
				start->prev = NULL;
				delete temp1;
				count--;
			}
			break;
		case'e':		//delete end
			if (start == NULL)
			{
				cout << "no song to delete" << endl;
			}
			else
			{
				temp1 = start;
				while (temp1->next != NULL)
				{
					temp2 = temp1;
					temp1 = temp1->next;
				}
```

```
                        delete temp1;
                        temp2->next = NULL;
                        count--;
                }
                break;
        case'm':        //delete mid
                int num;
                cout << "enter song you want to delete from playlist" << endl;
                cin >> num;
                if (num >= count)
                {
                        cout << "No song exists at that number" << endl;
                        return;
                }
                temp1 = start;
                for (int i = 1; i < num; i++)
                {
                        if (start == NULL)
                                cout << "given song does not exist" << endl;
                        else
                        {
                                temp2 = temp1;
                                temp1 = temp1->next;
                        }
                }
                temp3 = temp1->next;
                temp2->next = temp3;
                temp3->prev = temp2;
                delete temp1;
                break;
        }
}

void dblist::delnode(int num)
{
        if (num >= count)
        {
                cout << "No song exists at that number" << endl;
                return;
        }
        temp1 = start;
        for (int i = 1; i < num; i++)
        {
                if (start == NULL)
                        cout << "given song does not exist" << endl;
                else
                {
                        temp2 = temp1;
                        temp1 = temp1->next;
                }
        }
        temp3 = temp1->next;
        temp2->next = temp3;
        temp3->prev = temp2;
        delete temp1;
}

void dblist::show()             //backward display
{
        cout << "backward display" << endl;
        temp3 = start;
        int i = 0;
        if (start == NULL)
                cout << "no song to display" << endl;
        else
        {
                while (temp3->next != NULL)
                {
                        temp3 = temp3->next;
                        i++;
                }
                while (temp3->prev != NULL)
                {
                        cout << i + 1 << " - " << temp3->data->get_name().c_str() << endl;
                        temp3 = temp3->prev;
```

```
                                        i--;
                                }
                                cout << "1 - " << temp3->data->get_name().c_str() << endl;
                }
}

std::ofstream& operator<< (std::ofstream& stream, const dblist& s)
{
                node* temp=s.start;
                for(int i=0;i<s.count;i++){
                                stream<<*(temp->data);
                                stream<<temp->index<<endl;
                                temp=temp->next;
                }
                return stream;
}
std::ifstream& operator>> (std::ifstream& stream, dblist& s)
{
                s.start = new node();
                stream>>(*(s.start->data));
                stream>>s.start->index;
                stream.ignore();
                dblist::id=s.start->index;
                dblist::count=1;
                node* temp =s.curr= s.start;
                while(!stream.eof())
                {
                                temp->next= new node();
                                stream>>*(temp->next->data);
                                stream>>temp->next->index;
                                temp->next->prev=temp;
                                stream.ignore();
                                dblist::id=temp->next->index;
                                dblist::count++;
                                if(!stream.eof())
                                                break;
                }
                return stream;
}
```

Llist.h

```
#pragma once

#include"song.h"

class llist_node{
public:

                song* data;
                llist_node* next;
                llist_node(song* a = 0, llist_node* ptr = NULL) :data(a), next(ptr){}
                song* get_data(){ return data; }
};


class List
{
public:
                virtual void append(song* a) = 0;
};


class LList :public List
{
private:
                llist_node* curr;
                int count;

public:
                llist_node* ptr;
                LList();//constructor for the linked list
                void append(song* a);//appends a song in the linked list
                string display_list(int min,int max);//displays list of song along with their numbers between places min and max.
                bool isempty();//returns true if the list is empty
                int get_count();//returns number of songs in the list
```

```
	song* get_data();//returns song from current node
	void next();//move to next song
	void move_to_start();//moves to the first song
	song* search(int a);//search songs according to their numbers
	bool find(song* a);//returns true if the song name matches
	bool _delete(song* a);//deletes song and returns true otherwise false.

};
```

Llist.cpp

```
#include"Llist.h"


LList::LList(){
	ptr = NULL;
	curr = NULL;
	count = 0;
}

void LList::append(song* a){
	if (ptr == NULL){
		curr = ptr = new llist_node(a, NULL);
	}
	else {
		llist_node* temp = ptr;
		while (temp->next != NULL){
			temp = temp->next;
		}
		temp->next = new llist_node(a, NULL);
	}
	count++;
}

song* LList::search(int n)
{
	llist_node* temp=ptr;
	if(n>0&&n<=count){
		for(int i=1;i<n;i++)
			temp=temp->next;
		return temp->data;
	}
	else
	{
		cout<<"No song exists of that number"<<endl;
		return NULL;
	}

}


bool LList::_delete(song* a)
{
	llist_node* temp = ptr;
	if (ptr == NULL){
		cout << "No element exists" << endl;
		return false;
	}

	if ((ptr->get_data()->get_directory() == a->get_directory())){
		llist_node* itemp = ptr;
		cout << "Deleted element " << a->get_name().c_str() << " from the playlist" << endl;
		if (ptr->next != NULL)
			ptr = ptr->next;

		else
			ptr = NULL;
		delete itemp;
		count--;
		return true;
	}

	else {
		temp = ptr;
		while (temp->next != NULL){
```

```
                        if ((temp->next->data->get_directory() == a->get_directory())
                                && (temp->next->data->get_name() == a->get_name())){
                                        llist_node* itemp = temp->next;
                                        temp->next = temp->next->next;
                                        delete itemp;
                                        cout << "Deleted element " << a->get_name().c_str() << " from the playlist" <<
endl;

                                        count--;
                                        return true;
                        }
                        else if (temp->next != NULL)
                                temp = temp->next;
                        else
                                return false;
                }
        }

}
bool LList::find(song* a)
{
        llist_node* temp = ptr;
        if (ptr == NULL){
                cout << "No element exists" << endl;
                return false;
        }

        if ((ptr->get_data()->get_directory() == a->get_directory())){
                return true;
        }

        else {
                temp = ptr;
                while (temp->next != NULL){
                        if ((temp->next->data->get_directory() == a->get_directory())
                                && (temp->next->data->get_name() == a->get_name())){
                                        return true;
                        }
                        else if (temp->next != NULL)
                                temp = temp->next;
                        else
                                return false;
                }
        }
        return false;

}
string LList::display_list(int min,int max){
        llist_node* temp = ptr;
        string listplay;
        int i=1;
        if(max>count)max=count;
        if(min>count) min=0;
        for(;i<min+1;i++)
                temp=temp->next;
        if(max>count)max=count;
        cout << endl << endl;
        if (ptr == NULL)
                return "No song exists to display" ;
        else{
                while (temp != NULL&& i<=max){
                        listplay+= song::int_to_char(i++)+" - "+ temp->data->get_name()+ "\n";
                        temp = temp->next;
                }
                return listplay;

        }
}
bool LList::isempty(){
        if (ptr == NULL)
                return true;
        else
                return false;

}
```

```
int LList::get_count(){
		return count;
}

song* LList::get_data()
{
		return curr->data;
}
void LList::next(){
		if (curr->next != NULL)
			curr = curr->next;
}
void LList::move_to_start(){
		curr = ptr;
}
```

ofSoundPlayer.h

```
#include "ofSoundPlayer.h"
#include "ofUtils.h"

// these are global functions, that affect every sound / channel:
// ------------------------------------------------------------
// ------------------------------------------------------------

//--------------------
void ofSoundStopAll(){
		#ifdef OF_SOUND_PLAYER_FMOD
			ofFmodSoundStopAll();
		#endif
}

//--------------------
void ofSoundSetVolume(float vol){
		#ifdef OF_SOUND_PLAYER_FMOD
			ofFmodSoundSetVolume(vol);
		#endif
}

//--------------------
void ofSoundUpdate(){
		#ifdef OF_SOUND_PLAYER_FMOD
			ofFmodSoundUpdate();
		#endif
}

#if !defined(TARGET_ANDROID) && !defined(TARGET_LINUX_ARM)
//--------------------
void ofSoundShutdown(){
		#ifdef OF_SOUND_PLAYER_FMOD
			ofFmodSoundPlayer::closeFmod();
		#endif
}
#endif

//--------------------
float * ofSoundGetSpectrum(int nBands){
		#ifdef OF_SOUND_PLAYER_FMOD
			return ofFmodSoundGetSpectrum(nBands);
		#elif defined(OF_SOUND_PLAYER_OPENAL)
			return ofOpenALSoundPlayer::getSystemSpectrum(nBands);
		#else
			ofLogError("ofSoundPlayer") << "ofSoundGetSpectrum(): not implemented, returning NULL";
			return NULL;
		#endif
}



#include "ofSoundPlayer.h"
//------------------------------------------------------------------------------
ofSoundPlayer::ofSoundPlayer (){
		player	= ofPtr<OF_SOUND_PLAYER_TYPE>(new OF_SOUND_PLAYER_TYPE);
}
```

```
//---------------------------------------------------------------------------
void ofSoundPlayer::setPlayer(ofPtr<ofBaseSoundPlayer> newPlayer){
		player = newPlayer;
}

//---------------------------------------------------------------------------
ofPtr<ofBaseSoundPlayer> ofSoundPlayer::getPlayer(){
		return player;
}

//---------------------------------------------------------------------------
bool ofSoundPlayer::loadSound(string fileName, bool stream){
		if( player != NULL ){
				return player->loadSound(fileName, stream);
		}
		return false;
}

//---------------------------------------------------------------------------
void ofSoundPlayer::unloadSound(){
		if( player != NULL ){
				player->unloadSound();
		}
}

//---------------------------------------------------------------------------
void ofSoundPlayer::play(){
		if( player != NULL ){
				player->play();
		}
}

//---------------------------------------------------------------------------
void ofSoundPlayer::stop(){
		if( player != NULL ){
				player->stop();
		}
}

//---------------------------------------------------------------------------
void ofSoundPlayer::setVolume(float vol){
		if( player != NULL &&player->isLoaded()){
				player->setVolume(vol);
		}
}

//---------------------------------------------------------------------------
void ofSoundPlayer::setPan(float pan){
		if( player != NULL ){
				player->setPan(CLAMP(pan,-1.0f,1.0f));
		}
}

//---------------------------------------------------------------------------
void ofSoundPlayer::setSpeed(float spd){
		if( player != NULL ){
				player->setSpeed(spd);
		}
}

//---------------------------------------------------------------------------
void ofSoundPlayer::setPaused(bool bP){
		if( player != NULL ){
				player->setPaused(bP);
		}
}

//---------------------------------------------------------------------------
void ofSoundPlayer::setLoop(bool bLp){
		if( player != NULL ){
				player->setLoop(bLp);
		}
}

//---------------------------------------------------------------------------
```

```
void ofSoundPlayer::setMultiPlay(bool bMp){
		if( player != NULL ){
			player->setMultiPlay(bMp);
		}
}

//----------------------------------------------------------------------
void ofSoundPlayer::setPosition(float pct){
		if( player != NULL ){
			player->setPosition(pct);
		}
}

//----------------------------------------------------------------------
void ofSoundPlayer::setPositionMS(int ms){
		if( player != NULL ){
			player->setPositionMS(ms);
		}
}

//----------------------------------------------------------------------
float ofSoundPlayer::getPosition(){
		if( player != NULL ){
			return player->getPosition();
		} else {
			return 0;
		}
}

//----------------------------------------------------------------------
int ofSoundPlayer::getPositionMS(){
		if( player != NULL ){
			return player->getPositionMS();
		} else {
			return 0;
		}
}

//----------------------------------------------------------------------
bool ofSoundPlayer::getIsPlaying(){
		if( player != NULL ){
			return player->getIsPlaying();
		} else {
			return false;
		}
}

//----------------------------------------------------------------------
bool ofSoundPlayer::isLoaded(){
		if( player != NULL ){
			return player->isLoaded();
		} else {
			return false;
		}
}

//----------------------------------------------------------------------
float ofSoundPlayer::getSpeed(){
		if( player != NULL ){
			return player->getSpeed();
		} else {
			return 0;
		}
}

//----------------------------------------------------------------------
float ofSoundPlayer::getPan(){
		if( player != NULL ){
			return player->getPan();
		} else {
			return 0;
		}
}

//----------------------------------------------------------------------
```

```
float ofSoundPlayer::getVolume(){
	if( player != NULL ){
		return player->getVolume();
	} else {
		return 0;
	}
}
```

ofSoundPlayer.cpp

```
#pragma once

#include "ofConstants.h"
#include "ofTypes.h"

/// \todo: FIX THIS!!!!!!
/// #warning FIX THIS.

/// \brief Stops all active sound players on FMOD-based systems (windows, osx).
void ofSoundStopAll();

/// \brief Cleans up FMOD (windows, osx).
void ofSoundShutdown();

/// \brief Sets global volume for FMOD-based sound players (windows, osx).
/// \param vol range is 0 to 1.
void ofSoundSetVolume(float vol);

/// \brief Call in your app's update() to update FMOD-based sound players.
void ofSoundUpdate();

/// \brief Gets a frequency spectrum sample, taking all current sound players into account.
///
/// Each band will be represented as a float between 0 and 1.
///
/// \warning This isn't implemented on mobile & embedded platforms.
/// \param nBands number of spectrum bands to return, max 512.
/// \return pointer to an FFT sample, sample size is equal to the nBands parameter.
float * ofSoundGetSpectrum(int nBands);

#include "ofBaseTypes.h"
#include "ofBaseSoundPlayer.h"


#ifdef OF_SOUND_PLAYER_QUICKTIME
#include "ofQuicktimeSoundPlayer.h"
#define OF_SOUND_PLAYER_TYPE ofQuicktimeSoundPlayer
#endif

#ifdef OF_SOUND_PLAYER_FMOD
#include "ofFmodSoundPlayer.h"
#define OF_SOUND_PLAYER_TYPE ofFmodSoundPlayer
#endif

#ifdef OF_SOUND_PLAYER_OPENAL
#include "ofOpenALSoundPlayer.h"
#define OF_SOUND_PLAYER_TYPE ofOpenALSoundPlayer
#endif

#ifdef TARGET_OF_IOS
#include "ofxiOSSoundPlayer.h"
#define OF_SOUND_PLAYER_TYPE ofxiOSSoundPlayer
#endif

#ifdef TARGET_ANDROID
#include "ofxAndroidSoundPlayer.h"
#define OF_SOUND_PLAYER_TYPE ofxAndroidSoundPlayer
inline void ofSoundShutdown(){}
#endif

#ifdef TARGET_LINUX_ARM
inline void ofSoundShutdown(){}
#endif

/// \class ofSoundPlayer
/// \brief Plays sound files.
```

```
///
/// ofSoundPlayer handles simple playback of sound files, with controls for
/// volume, pan, speed, seeking and multiplay.  This is a common cross-platform
/// sound player interface which is inherited by each of the platform-specific
/// sound player implementations.
class ofSoundPlayer : public ofBaseSoundPlayer {
public:
    ofSoundPlayer();

    void setPlayer(ofPtr<ofBaseSoundPlayer> newPlayer);
    ofPtr<ofBaseSoundPlayer> getPlayer();

    /// \brief Tells the sound player which file to play.
    ///
    /// Codec support varies by platform but wav, aif, and mp3 are safe.
    ///
    /// \param fileName Path to the sound file, relative to your app's data folder.
    /// \param stream set "true" to enable streaming from disk (for large files).
    bool loadSound(string fileName, bool stream = false);

    /// \brief Stops and unloads the current sound.
    void unloadSound();

    /// \brief Starts playback.
    void play();

    /// \brief Stops playback.
    void stop();

    /// \brief Sets playback volume.
    /// \param vol range is 0 to 1.
    void setVolume(float vol);

    /// \brief Sets stereo pan.
    /// \param pan range is -1 to 1 (-1 is full left, 1 is full right).
    void setPan(float pan);

    /// \brief Sets playback speed.
    /// \param speed set > 1 for faster playback, < 1 for slower playback.
    void setSpeed(float speed);

    /// \brief Enables pause / resume.
    /// \param paused "true" to pause, "false" to resume.
    void setPaused(bool paused);

    /// \brief Sets whether to loop once the end of the file is reached.
    /// \param loop "true" to loop, default is false.
    void setLoop(bool loop);

    /// \brief Enables playing multiple simultaneous copies of the sound.
    /// \param multiplay "true" to enable, default is false.
    void setMultiPlay(bool multiplay);

    /// \brief Sets position of the playhead within the file (aka "seeking").
    /// \param percent range is 0 (beginning of file) to 1 (end of file).
    void setPosition(float percent);

    /// \brief Sets position of the playhead within the file (aka "seeking").
    /// \param ms number of milliseconds from the start of the file.
    void setPositionMS(int ms);

    /// \brief Gets position of the playhead.
    /// \return playhead position in milliseconds.
    int getPositionMS();

    /// \brief Gets position of the playhead.
    /// \return playhead position as a float between 0 and 1.
    float getPosition();

    /// \brief Gets current playback state.
    /// \return true if the player is currently playing a file.
    bool getIsPlaying();

    /// \brief Gets playback speed.
    /// \return playback speed (see ofSoundPlayer::setSpeed()).
```

```
    float getSpeed();

    /// \brief Gets stereo pan.
    /// \return stereo pan in the range -1 to 1.
    float getPan();

    /// \brief Gets current volume.
    /// \return current volume in the range 0 to 1.
    float getVolume();

    /// \brief Queries the player to see if its file was loaded successfully.
    /// \return whether or not the player is ready to begin playback.
    bool isLoaded();

protected:
    ofPtr<ofBaseSoundPlayer> player;

};
```

Queue.h

```
#pragma once
#include"song.h"
//node of queue class
class q_node
{
public:
	song* data;
	q_node* next;
	q_node(song* s=NULL, q_node* n= NULL):next(n)
	{
		if(s==NULL)
			data= new song();
		else
			data=s;
	}
};

class Queue
{
private:
	q_node* head;
	q_node* tail;
public:
	Queue();//constructor
	bool isEmpty();//returns true  if queue is empty
	void enqueue(song* a); //enqueues a song into queue
	song* dequeue(); //returns dequeued song
};
```

Queue.cpp

```
#include"Queue.h"

Queue::Queue()
{
	head = NULL;
	tail = NULL;
}
bool Queue::isEmpty()
{
	return tail == NULL && head == NULL;
}

void Queue::enqueue(song* a)
{
	q_node* q= new q_node(a,NULL);
	if (tail == NULL)
	{
		head = q;
		tail = q;
	}
	else
		tail->next = q;
	tail = q;
}
```

```
song* Queue:: dequeue()
{
	song* a;
	if (isEmpty())
	{
		//cout << "Empty" << endl;
		return NULL;
	}
	if (head == tail)
	{
		a= head->data;
		q_node*p = head;
		head = NULL;
		tail = NULL;
		delete p;
	}
	else
	{
		a = head->data;
		q_node*p = head;
		head = head->next;

		delete p;
	}
	return a;

}
```

Song.h

```
#pragma once
#include<iostream>
#include<string>
#include<fstream>

using namespace std;

class song{
private:
	string name;
	string directory;
public:
	int count;
	static string int_to_char(int c); //converts integer into corresonding string like (int)5 = "005"(string)
	song(int a,string n, string d);
	song(song* temp);
	song();
	string get_name();
	string get_directory();
	void new_song();
	void play();
	void display();

	friend std::ofstream& operator << (std::ofstream& out, const song& s);//friend function writing song class object into file
	friend std::ifstream& operator >> (std::ifstream& in, song& s);//friend function reading song from a file
};
```

Song.cpp

```
#include"song.h"

song::song(int a,string n, string d) :name(n), directory(d),count(a){}
string song::get_name(){
	return name;
}

string song::get_directory(){
	return directory;
}

string song::int_to_char(int c)
{
	char unit = 48+ c%10;
	char ten= 48+ (c/10)%10;
	char hundred=48 + (c/100)%10;
	string tem;
	tem+=hundred;
```

```
		tem+=ten;
		tem+=unit;
		return tem;
}

void song::display(){
		cout<<"Song Name: "<<name<<endl
			<<"Directory: "<<directory<<endl;
}

song::song()
{
		name=" ";
		directory=" ";
		count=0;

}

void song::new_song()
{
		string filename, extension;
		cout << "Enter the directory of the song:";
		fflush(stdin);
		getline(cin, filename);

		//to ensure that user only enters the mp3 files
		for (int i = filename.size() - 4; i < filename.size(); i++){
			extension += filename[i];
		}
		if (extension != ".mp3")
			throw - 1;

		//to get the name of the song for searching
		int i = filename.size() - 4;
		for (; filename[i] != '\\'; i--);i++;
		name="";
		for (; i < filename.size() - 4; i++)
			name += filename[i];
		directory = filename;
}
void song::play(){
		system(directory.c_str());
}

song::song(song* temp)
{
		name=temp->name;
		directory=temp->directory;
		count=temp->count;
}

std::ofstream& operator << (std::ofstream& out, const song& s)
{
		out<<s.name<<endl;
		out<<s.directory<<endl;
		out<<s.count<<endl;
		return out;
}
std::ifstream& operator >> (std::ifstream& in, song& s)
{
		string n,d,line;
		int c;
		getline(in,n);
		getline(in,d);
		in>>c;
		std::getline(in,line);
		s.directory=d;
		s.name=n;
		s.count=c;
		return in;
}
```

Source.cpp

```
#include<iostream>
using namespace std;
```

```
#include<string>
#include"HQPlayer.h"


void Menu()
{
	HQPlayer player;
	char ch;
	do
	{
		char i;
		cout << "Press a to add song" << endl
			<< "Press d to delete" << endl
			<< "Press s for search" << endl
			<< "Press v for display" << endl
			<< "Press e for backward display" << endl
			<< "Press n for Next Song" << endl
			<< "Press p for previous song"<<endl
			<<"Press r for repeat all"<<endl
			<<"Press t for no repeat"<<endl
			<<"Press u for shuffle"<<endl
			<<"Press l for play"<<endl;
		cin >> i;
		switch (i)
		{
		case'a':
			player.add_song();
			break;
		case'd':
			player.delete_song();
			break;
		case'v':
			player.display();
			break;
		case's':
			player.search();
			break;
		case'e':
			player.backwards_display();
			break;
		case'n':
			player.next();
			break;
		case'p':
			player.previous();
			break;
		case'r':
			player.repeat_all();
			break;
		case't':
			player.no_repeat();
			break;
		case'u':
			player.shuffle();
			break;
		case'l':
			player.play_from_start();
		default:
			cout << "Bad input" << endl;
			break;
		}
		cout << "want to process more y/n" << endl;
		cin >> ch;
	} while (ch != 'n');
}
int main()
{
	try{
		Menu();
		cin.get();
		return 0;
	}
	catch (int n){
		if (n == -1){
			cout << "Extension not valid" << endl;
```

```
		}
	}
}
```

Stack.h

```
#pragma once
#include"song.h"
//node of Stack class
class stack_node{
	public:
		stack_node* next;
		song* data;
		stack_node(song* d, stack_node* temp = NULL){
			next = temp;
			data=d;
		}
	};

class Stack
{
private:

	stack_node* top;
	int count;
public:
	Stack();
	void push( song* d);
	song* search(int n);
	string display(int min,int max);//returns string of song names inbetween the place min and max
	song* pop(); //pops the song in top node
	song* peek(); //returns the song in top node
	int get_count();
	bool isEmpty();

};
```

Stack.cpp

```
#include"Stack.h"

Stack::Stack(){
	top  = NULL;
	count = 0;
}

void Stack::push(song* d){
	if (top == NULL)
		top = new stack_node(d);
	else{
		top = new stack_node(d, top);
	}
	count++;
}

string Stack :: display(int min,int max)
{
	int n=0;
	string listplay;
	if(top==NULL)
	{
		return "There is no song in the recenly played songs";
	}
	else
	{
		stack_node* temp = top;
		int i;
		for(i=0;i<min+1;i++)
			temp=temp->next;

		while(temp!=NULL&&i<max)
		{
			listplay+=song::int_to_char(i++)+" - "+temp->data->get_name()+"\n";
			temp=temp->next;
		}
		return listplay;
```

```
		}

}
song* Stack::search(int n)
{
		stack_node* temp=top;
		if(n>0&&n<=count){
			for(int i=1;i<n;i++)
				temp=temp->next;
			return temp->data;
		}
		else
		{
			cout<<"No song exists of that number"<<endl;
			return NULL;
		}
}
song* Stack::pop()
{
		if (top != NULL){
			song* temp = top->data;
			stack_node* temp2=top;
			top = top->next;
			count--;
			delete temp2;
			return temp;
		}
		else
		{
			cout<<"No song exists to pop"<<endl;
			return NULL;
		}

}
bool Stack::isEmpty(){
		return top == NULL;
}

song* Stack::peek(){
		if (top != NULL)
			return top->data;
		else{
			cout<<"No element exists to peek"<<endl;
			return NULL;
		}
}

int Stack::get_count(){
		return count;
}
```

Trie.h

```
#pragma once
#include"Llist.h"
const int ALPHABET_SIZE = 26;

class Trie_node
{
public:
		LList* words;
		song* data;
		Trie_node* children[ALPHABET_SIZE];
		Trie_node(){
			data = NULL;
			for (int i = 0; i < ALPHABET_SIZE; i++)
				children[i] = NULL;
			words = new LList();
		}
};


class Trie
{
private:
		Trie_node* root;
```

```
	Trie_node* curr;
	int count;
public:
	Trie(); //constructor
	int char_to_index(char c); //gives ascii value
	void move_to_start(); //moves current pointer to start
	void insert(song* key); //inserts song into trie
	void _delete(string key); //deletes a song from trie
	song* search(string key);  //searches a song and returns it
	string remove_spaces(string key); //remove spaces from a string ( used in inserting song name containing spaces)
	string display(string key, int min, int max);
	//returns a string consisting of all song names (in response to string entered by user i.e auto suggestion..entered between min
and max index no)
};
```

Trie.cpp

```
#include"trie.h"

Trie::Trie(){
	count = 0;
	curr = root = new Trie_node();
}

void Trie::move_to_start(){
	curr = root;

}

int Trie::char_to_index(char c){
	if (c >= 'a'&&c <= 'z')
		return (int)c - 'a';
	else if (c >= 'A' && c <= 'Z')
		return (int)c - 'A';
	else
		return -1;
}

string Trie::remove_spaces(string key)
{
	string temp;
	int j=0;
	for(int i=0;i<key.size();i++)
	{
		if(key[i]!=' ')
			temp.append(1,key[i]);
	}
	return temp;
}
void Trie::insert(song* key)
{
	string keyName= remove_spaces(key->get_name());
	move_to_start();
	count++;
	for (int i = 0; i < keyName.length()-4; i++){
		int index = char_to_index(keyName[i]);
		if(index==-1)continue;
		if (!curr->children[index])
			curr->children[index] = new Trie_node();
		curr->words->append(key);
		curr = curr->children[index];
	}
	curr->data=key;
}

void Trie::_delete(string key)
{
	key=remove_spaces(key);
	move_to_start();
	count++;
	for (int i = 0; i < key.length(); i++){
		int index = char_to_index(key[i]);
		if (!curr->children[index])
			curr->children[index] = new Trie_node();
		curr = curr->children[index];
	}
```

```
		curr->data = NULL;
}

song* Trie::search(string key)
{
		key= remove_spaces(key);
		move_to_start();
		for (int i = 0; i < key.length(); i++){
			int index = char_to_index(key[i]);
			if(index==-1) continue;
			if (!curr->children[index])
				return NULL;
			curr = curr->children[index];
		}
		if (curr != NULL){
			return curr->data;
		}
		else
			return NULL;
}


string Trie::display(string key,int min,int max)
{
		key= remove_spaces(key);
		move_to_start();
		for (int i = 0; i < key.length(); i++){
			int index = char_to_index(key[i]);
			if(index==-1) continue;
			if (!curr->children[index])
				return "\0";
			curr = curr->children[index];
		}
		if (curr != NULL ){
			return curr->words->display_list(min,max);
		}
		else
			return "\0";
}
```