20

good

```
//iTunes Music Library
#include <iostream>
#include <string>
#include <sstream>
#include<algorithm>

using namespace std;

class iTunes
{
private:
   string name;
   string artist;
   int bitRate;
   int totalTime;

   static bool validRate(int bitRate);
   static bool validArtist(string artist);
   static bool validTime(int totalTime);
   static bool validSong(string name);
   static bool top(iTunes list[], int top);
   static void change(iTunes &first, iTunes &next);

public:
   string toString();

   iTunes();
   iTunes(string artist, string name, int totalTime, int bitRate);

   static const int MIN_BITRATE = 64;
   static const int MAX_BITRATE = 705;
   static const int MIN_STR_LENGTH = 1;
   static const int MAX_STR_LENGTH = 80;
   static const int MIN_TIME_PLAY = 5000;
   static const long MAX_TIME_PLAY = 1000 * 60 * 60;
   static const int DEFAULT_BITRATE = 64;
   static const int DEFAULT_PLAYTIME = 1000;
   static const string DEFAULT_STRING;

   static void printArray(string data, iTunes list[], int arraySize);
   static void sortArray(iTunes[], int arraySize);
   static int compare(iTunes first, iTunes second);

   string getArtist() { return artist; }
   string getName() { return name; }
   int getTime() { return totalTime; }
   int getRate() { return bitRate; }

   bool setName(string song);
   bool setTime(int time);
   bool setRate(int rate);
   bool setArtist(string artName);

};
const string iTunes::DEFAULT_STRING = "undefined";
int main()
{
   iTunes library[] =
   {
      iTunes("Starboys", "The Weekend", 210000, 120),
      iTunes("Main Chick", "Kid Ink", 270000, 150),
      iTunes("So Sick", "Ne-Yo", 195000, 90),
      iTunes("Waiting on the Rain", "JBoog", 196800, 100)
   };
   int arraySize = sizeof(library) / sizeof(library[0]);

   iTunes::printArray("Original: ", library, arraySize);
   iTunes::sortArray(library, arraySize);
   iTunes::printArray("Modified: ", library, arraySize);
   iTunes::printArray("Reset: ", library, arraySize);
}
iTunes::iTunes(string artist, string name, int totalTime, int bitRate)
{
   if (!setTime(totalTime))
      totalTime = DEFAULT_PLAYTIME;
   if (!setRate(bitRate))
      bitRate = DEFAULT_BITRATE;
   if (!setArtist(artist))
      artist = DEFAULT_STRING;
   if (!setName(name))
      name = DEFAULT_STRING;
}
bool iTunes::setTime(int time)
{
   if (!validTime(time))
      return false;
   totalTime = time;
   return true;
}
bool iTunes::setRate(int rate)
{
   if (!validRate(rate))
      return false;
   bitRate = rate;
   return true;
}
bool iTunes::setName(string songs)
{
   if (!validSong(songs))
      return false;
   name = songs;
   return true;
}
bool iTunes::setArtist(string artName)
{
   if (!validArtist(artName))
      return false;
   artist = artName;
   return true;
}
bool iTunes::validRate(int rate)
{
   if (rate >= MIN_BITRATE && rate <= MAX_BITRATE)
      return true;
   return false;
}
bool iTunes::validTime(int time)
{
   if (time >= MIN_TIME_PLAY && time <= MAX_TIME_PLAY)
      return true;
   return false;
}
bool iTunes::validArtist(string length)
{
   if (length.length() >= MIN_STR_LENGTH && length.length() <= MAX_STR_LENGTH)
      return true;
   return false;
}
bool iTunes::validSong(string length)
{
   if (length.length() >= MIN_STR_LENGTH && length.length() <= MAX_STR_LENGTH)
      return true;
   return false;
}
int iTunes::compare(iTunes first, iTunes second)
{
   int different;

   different = first.name.compare(second.name);

   return different;
}
string iTunes::toString()
{
   string results;
   ostringstream convrtName, convrtArtist, convrtTotalTime, convrtBit;

   convrtName << name;
   convrtArtist << artist;
   convrtTotalTime << totalTime;
   convrtBit << bitRate;

   results =
      "Artist: " + convrtName.str() + " / Title: "
      + convrtArtist.str() + " / Play Time: " + convrtTotalTime.str()
      + " Milliseconds / Bit Rate: " + convrtBit.str() + "k\n";

   return results;
}
void iTunes::printArray(string data, iTunes list[], int arraySize)
{
   string output = "";

   cout << data << "\n";
   for (int i = 0; i < arraySize; i++)
      output += " " + list[i].toString();

   cout << output << "\n";
}
void iTunes::sortArray(iTunes array[], int arraySize)
{
   for (int k = 0; k < arraySize; k++)
      if (!top(array, arraySize - 1 - k))
         return;
}

bool iTunes::top(iTunes list[], int top)
{
   bool changed = false;

   for (int i = 0; i < top; i++)
      if (iTunes::compare(list[i], list[i + 1]) > 0)
      {
         swap(list[i], list[i + 1]);
         changed = true;
      }
```

use blank lines to separate methods

```
   return changed;
}

// I couldn't figure out how to reset it. Sorry it's not complete

/*---------------------------- Posted Run # 1 ---------------------------------
-----

Original:
Artist: The Weekend / Title: Starboys / Play Time: 210000 Milliseconds / Bit Rat
e: 120k
Artist: Kid Ink / Title: Main Chick / Play Time: 270000 Milliseconds / Bit Rate:
 150k
Artist: Ne-Yo / Title: So Sick / Play Time: 195000 Milliseconds / Bit Rate: 90k
Artist: JBoog / Title: Waiting on the Rain / Play Time: 196800 Milliseconds / Bi
t Rate: 100k

Modified:
Artist: JBoog / Title: Waiting on the Rain / Play Time: 196800 Milliseconds / Bi
t Rate: 100k
Artist: Kid Ink / Title: Main Chick / Play Time: 270000 Milliseconds / Bit Rate:
 150k
Artist: Ne-Yo / Title: So Sick / Play Time: 195000 Milliseconds / Bit Rate: 90k
Artist: The Weekend / Title: Starboys / Play Time: 210000 Milliseconds / Bit Rat
e: 120k

Reset:
Artist: JBoog / Title: Waiting on the Rain / Play Time: 196800 Milliseconds / Bi
t Rate: 100k
Artist: Kid Ink / Title: Main Chick / Play Time: 270000 Milliseconds / Bit Rate:
 150k
Artist: Ne-Yo / Title: So Sick / Play Time: 195000 Milliseconds / Bit Rate: 90k
Artist: The Weekend / Title: Starboys / Play Time: 210000 Milliseconds / Bit Rat
e: 120k

Press any key to continue . . .


--------------------------------------------------------------------------------
-*/
```

good job but why Sort?

```
// Lab 07 - Instructor Solution:
// Original - Prof. Loceff, Updates, Edits, Annotations: &
//
//Notes:
//- Correct access qualifiers (private/public)
//- Correct use of getters/setters
//- Correct use of global consts
//- Use of symbolic consts rather than literals (magics)
//- No output in interior methods
//-
//- Faithfulness to spec

#include <iostream>
#include <string>
#include <sstream>

using namespace std;

class iTunes {
private:
    string name;
    string artist;
    int bitrate;
    int totalTime;

public:
    static const int MIN_BITRATE = 64;
    static const int MAX_BITRATE = 705;
    static const int MIN_STR_LENGTH = 1;
    static const int MAX_STR_LENGTH = 128;
    static const int MIN_PLAY_TIME = 5000;     // 5s
    static const int MAX_PLAY_TIME = 3600000;  // 1h

    static const int DEFAULT_BITRATE = 64;
    static const int DEFAULT_PLAY_TIME = MIN_PLAY_TIME;
    static const string DEFAULT_STRING;

    iTunes();
    iTunes(const string& nm, const string& art, int btrt, int tTime);
    bool setName(const string& nm);
    bool setArtist(const string& art);
    bool setBitRate(int btrt);
    bool setTotalTime(int tTime);

    string getName() const { return name; }
    string getArtist() const { return artist; }
    int getBitRate() const { return bitrate; }
    int getTotalTime() const { return totalTime; }

    void display() const;
    string toString() const;
    void setDefaults();
};
```

```
// out-of-line defs for non-primitive static constants
string const iTunes::DEFAULT_STRING = " (undefined) ";

// Implementation  -----------------------------------------------
iTunes::iTunes() {
    setDefaults();
}

iTunes::iTunes(const string& nm, const string& art, int btrt, int tTime) {
    if (!setName(nm))
        name = DEFAULT_STRING;
    if (!setArtist(art))
        artist = DEFAULT_STRING;
    if (!setBitRate(btrt))
        bitrate = DEFAULT_BITRATE;
    if (!setTotalTime(tTime))
        totalTime = DEFAULT_PLAY_TIME;
}

bool iTunes::setName(const string& nm) {
    if (nm.length() < MIN_STR_LENGTH || nm.length() > MAX_STR_LENGTH)
        return false;
    name = nm;
    return true;
}

bool iTunes::setArtist(const string& art) {
    if (art.length() < MIN_STR_LENGTH || art.length() > MAX_STR_LENGTH)
        return false;
    artist = art;
    return true;
}

bool iTunes::setBitRate(int btrt) {
    if (btrt < MIN_BITRATE || btrt > MAX_BITRATE)
        return false;
    bitrate = btrt;
    return true;
}

bool iTunes::setTotalTime(int tTime) {
    if (tTime < MIN_PLAY_TIME || tTime > MAX_PLAY_TIME)
        return false;
    totalTime = tTime;
    return true;
}

string iTunes::toString() const {
    ostringstream cnvrt;

    cnvrt << "\"" << name << "\", by " << artist
          << "\n Duration: " << totalTime / 1000
          << " seconds, Bit Rate: " << bitrate;

    return cnvrt.str();
}

void iTunes::display() const {
    cout << "\niTunes Song ---------:\n" << toString() << endl;
}

void iTunes::setDefaults() {
    name = DEFAULT_STRING;
    artist = DEFAULT_STRING;
    totalTime = DEFAULT_PLAY_TIME;
    bitrate = DEFAULT_BITRATE;
}

// client ----------------------------------------------
int main()
{
    iTunes tune1, tune2,
    tune3("Hobo Blues", "John Lee Hooker", 128, 182000),
    tune4("Give It All U Got", "Lil Jon", 128, 218000);

    tune1.display();
    tune2.display();
    tune3.display();
    tune4.display();

    // mutate tune1:
    tune1.setArtist("Steely Dan");
    tune1.setName("Black Cow");
    tune1.setBitRate(256);
    tune1.setTotalTime(310 * 1000);  // 310 seconds

    // mutate others:
    tune2.setBitRate(512);
    tune3.setBitRate(512);
    tune4.setBitRate(512);

    cout << "\nAll tunes after mutation\n";
    tune1.display();
    tune2.display();
    tune3.display();
    tune4.display();

    // reset to defaults and test
    tune1.setDefaults();
    tune2.setDefaults();
    tune3.setDefaults();
    tune4.setDefaults();

    cout << "\nsetDefaults Tests ----------- \n";
    tune1.display();
    tune2.display();
    tune3.display();
    tune4.display();

    // mutator tests
    cout << "\nMutator Tests ----------- \n";
    if (!tune2.setArtist(""))
        cout << "\n Correctly rejected blank string\n";

    if (!tune2.setBitRate(999))
        cout << "\n Correctly rejected out-of-range bit rate\n";

    // accessor tests
    cout << "\nAccessor Tests ----------- \n";
    cout << "tune1 artist: " << tune1.getArtist() << endl;
    cout << "tune3 total time (ms): " << tune3.getTotalTime() << endl;

    return 0;
}

/* ----------------------- run -----------------------------


 iTunes Song ---------:
 " (undefined) ", by  (undefined)
 Duration: 5 seconds, Bit Rate: 64

 iTunes Song ---------:
 " (undefined) ", by  (undefined)
 Duration: 5 seconds, Bit Rate: 64

 iTunes Song ---------:
 "Hobo Blues", by John Lee Hooker
 Duration: 182 seconds, Bit Rate: 128

 iTunes Song ---------:
 "Give It All U Got", by Lil Jon
 Duration: 218 seconds, Bit Rate: 128

 All tunes after mutation

 iTunes Song ---------:
 "Black Cow", by Steely Dan
 Duration: 310 seconds, Bit Rate: 256

 iTunes Song ---------:
 " (undefined) ", by  (undefined)
 Duration: 5 seconds, Bit Rate: 512

 iTunes Song ---------:
 "Hobo Blues", by John Lee Hooker
 Duration: 182 seconds, Bit Rate: 512
```

```
iTunes Song ---------:
"Give It All U Got", by Lil Jon
Duration: 218 seconds, Bit Rate: 512

setDefaults Tests -----------

iTunes Song ---------:
" (undefined) ", by  (undefined)
Duration: 5 seconds, Bit Rate: 64

iTunes Song ---------:
" (undefined) ", by  (undefined)
Duration: 5 seconds, Bit Rate: 64

iTunes Song ---------:
" (undefined) ", by  (undefined)
Duration: 5 seconds, Bit Rate: 64

iTunes Song ---------:
" (undefined) ", by  (undefined)
Duration: 5 seconds, Bit Rate: 64

Mutator Tests -----------

Correctly rejected blank string

Correctly rejected out-of-range bit rate

Accessor Tests -----------
tune1 artist:  (undefined)
tune3 total time (ms): 5000
Program ended with exit code: 0

-------------------------------------------------------------- */
```