CS 201, Fall 2018
Homework Assignment 1

Due: 23:59, November 21 (Wednesday), 2018

In this homework, you will implement a music album collection system to store the song names of the music albums in a particular collection. For each music album, you will store its artist, its title, its year, and its list of songs. In your implementation, the collection of music albums and the list of songs in each music album will need to be implemented using **dynamically allocated arrays**. The homework has two parts whose requirements are explained below.

PART A: (30 points)

To take the final exam, you must submit this part and receive at least half of its points.

This part is a simplified version of the entire system. It stores the artists, titles, and years of music albums in a collection without their list of songs. The music albums will be stored in a dynamically allocated array of `MusicAlbum` objects. Thus, you will first implement the `MusicAlbum` class. This class is rather simple for Part A, but will need to be extended for Part B.

1. Below is the required part of the `MusicAlbum` class. The name of the class should be `MusicAlbum`; the interface for the class:

```
#ifndef __SIMPLE_MUSIC_ALBUM_H
#define __SIMPLE_MUSIC_ALBUM_H

#include <string>
using namespace std;

class MusicAlbum {
public:
   MusicAlbum(const string maArtist = "",
              const string maTitle = "",
              const int maYear = 0);
   ~MusicAlbum();
   MusicAlbum(const MusicAlbum &maToCopy);
   void operator=(const MusicAlbum &right);
   string getMusicAlbumArtist();
   string getMusicAlbumTitle();
   int getMusicAlbumYear();

private:
   string artist;
   string title;
   int year;
};
#endif
```

should be coded in a file called `SimpleMusicAlbum.h` and its implementation should be coded in a file called `SimpleMusicAlbum.cpp`.

- The `MusicAlbum` class should have the data members `Artist`, `Title`, and `Year`. You should implement get functions for these data members since they will be used to test your program.
- You should also implement a default constructor that initializes the `Artist`, `Title`, and `Year` data members. Additionally, you should implement your own destructor and copy constructor, and overload the assignment operator. Although you may use the default ones for some of these member functions, you are advised to implement them (although some may have no statements) in Part A so that it will be easier for you to extend them in Part B.
- Do not delete or modify any part of the given data members or member functions. However, you may define additional data members and member functions, if necessary.

2. Below is the required part of the music album collection (`MAC`) class that you will code in Part A of this assignment. The name of the class should be `MAC`; the interface for the class:

```
#ifndef __SIMPLE_MAC_H
#define __SIMPLE_MAC_H

#include <string>
using namespace std;
#include "SimpleMusicAlbum.h"

class MAC{
public:
   MAC();
   ~MAC();
   MAC(const MAC &macToCopy);
   void operator=(const MAC &right);

   bool addMusicAlbum(const string maArtist,
                      const string maTitle,
                      const int maYear);
   bool removeMusicAlbum(const string maArtist,
                         const string maTitle);
   int getMusicAlbums(MusicAlbum *&allMusicAlbums);

private:
   MusicAlbum *musicAlbums;
   int noOfMusicAlbums;
};
#endif
```

should be coded in a file called `SimpleMAC.h` and its implementation should be coded in a file called `SimpleMAC.cpp`.

- Implement the default constructor, which creates an empty album music collection. Also overload the assignment operator and implement the destructor and copy constructor.
- Implement the add and remove music album functions whose details are given below:

  Add a music album: This function adds a music album to the collection. The artist, the title and the year are specified as parameters. In this collection, the pair of artist and title are unique (however, there can be multiple albums of the same artist and multiple albums with the same title). Thus, if the user attempts to add a music album with an already existing artist and title, do not add the music album and return `false`. Do not display any warning messages. Otherwise, if the album does not exist in the collection, add it to the collection and return `true`.

Remove a music album: This function removes a music album from the collection. The artist and the title are specified as parameters. If the given album exists in the collection, remove it from the collection and return `true`. Otherwise, if there is no album with the given artist and title, do not perform any action and return `false`. Likewise, do not display any warning messages.

- Implement the get function for music albums. The `getMusicAlbums` function should return the number of the music albums in the collection using the return value and the music albums in the collection by a pass-by-reference parameter called `allMusicAlbums`. Do not forget to put a deep copy of the `musicAlbums` to the pass-by-reference parameter; otherwise, you may encounter run-time errors.
- Do not delete or modify any part of the given data members or member functions. However, you may define additional functions and data members, if necessary.

3. To test Part A, you should code your own main function in a separate file. Do not forget to test your code for different cases. However, do not submit any file containing the main function; otherwise, you may lose a considerable number of points. We will code our own driver to grade Part A of your assignment. In doing that, we will use the get functions of the `MusicAlbum` and `MAC` classes. Thus, do not forget to implement the get functions of these two classes.

   Below is an example of the test code and its output. In this example code, there is a global function called `displayAllMusicAlbums`. We will use it to display the music albums in the collection and to understand whether you add/remove music albums correctly. If you want, you may use this global function for your tests as well. Notice that the last two lines of this global function are to deallocate the `allMusicAlbums` array. Do not remove these two lines if you get any run-time errors while using the function. If you have such run-time errors, most probably, you made an error in implementing either the `getMusicAlbums` function or one or more of the destructor, copy constructor, and overloaded assignment operator.

```
#include <iostream>
using namespace std;
#include "SimpleMusicAlbum.h"
#include "SimpleMAC.h"

void displayAllMusicAlbums(MAC mac){
   MusicAlbum *allMusicAlbums;
   int noOfMusicAlbums = mac.getMusicAlbums(allMusicAlbums);

   cout <<"No of music albums: "<< noOfMusicAlbums << endl;
   for (int i = 0; i < noOfMusicAlbums; i++){
       cout << allMusicAlbums[i].getMusicAlbumArtist() <<", "
            << allMusicAlbums[i].getMusicAlbumTitle() <<" ("
            << allMusicAlbums[i].getMusicAlbumYear() <<")"
            << endl;
   }

   if (allMusicAlbums != NULL)
      delete [] allMusicAlbums;
}
```

```
int main(){
   MAC m;

   m.addMusicAlbum("John Coltrane", "My Favorite Things",
                   1961);
   if (m.addMusicAlbum("John Coltrane", "A Love Supreme",
                       1965))
      cout << "Successful insertion of John Coltrane, "
           << "A Love Supreme (1965)" << endl;
   else
      cout << "Unsuccessful insertion of John Coltrane, "
           << "A Love Supreme (1965)" << endl;
   m.addMusicAlbum("Jethro Tull", "Thick As A Brick",
                   1972);
   m.addMusicAlbum("Mike Oldfield", "Tubular Bells",
                   1973);
   m.addMusicAlbum("Pink Floyd", "The Dark Side of the Moon",
                   1973);
   displayAllMusicAlbums(m);
   if (m.removeMusicAlbum("John Coltrane", "Giant Steps"))
      cout << "Successful deletion of John Coltrane, "
           << "Giant Steps" << endl;
   else
      cout << "Unsuccessful deletion of John Coltrane, "
           << "Giant Steps" << endl;

   return 0;
}
```

The output should be:

```
Successful insertion of John Coltrane, A Love Supreme (1965)
No of music albums: 5
John Coltrane, My Favorite Things (1961)
John Coltrane, A Love Supreme (1965)
Jethro Tull, Thick As A Brick (1972)
Mike Oldfield, Tubular Bells (1973)
Pink Floyd, The Dark Side of the Moon (1973)
Unsuccessful deletion of John Coltrane, Giant Steps
```

What to submit for Part A?

You should put your `SimpleMusicAlbum.h`, `SimpleMusicAlbum.cpp`, `SimpleMAC.h`, and `SimpleMAC.cpp` files into a folder and zip the folder. In this zip file, there should not be any file containing the `main` function. The name of this zip file needs to be `PartA_secX_Firstname_Lastname_StudentID.zip`, where `X` is your section number. Then you should follow the steps that are explained at the end of this document for the submission of Part A.

What to be careful about implementation and submission for Part A?

You need to read the "notes about implementation" and the "notes about submission" parts that are given at the end of this document.

PART B: (70 points)

Now, Part A is to be extended such that each music album has a list of songs. The full functionality of this extended MAC system is to be provided. In order for this to be done, the `MusicAlbum` class needs to be extended such that it additionally keeps the list of songs contained. The song list of a music album must be kept in a dynamically allocated array of `Song` objects. Note that the number of songs can be different from one music album to another, but is normally fixed for a particular music album. The details of the classes are given below.

1. The requirements of the `Song` class are given below. The name of the class should be `Song`; its interface:

```
#ifndef __SONG_H
#define __SONG_H

#include <string>
using namespace std;

class Song {
public:
   Song(const string sName = "", const int sMins = 0,
        const int sSecs = 0);
   ~Song();
   Song(const Song &songToCopy);
   void operator=(const Song &right);

private:
   string name;
   int mins;
   int secs;
};
```

should be coded in a file called `Song.h` and its implementation should be coded in a file called `Song.cpp`.

- The `Song` class keeps the name of a song and its running length in minutes and seconds. The name should be unique for each music album. That is, a music album cannot have multiple songs with the exact same name but different music albums may have a song with the same name.
- Implement your own default constructor, destructor, and copy constructor, and overload the assignment operator.
- Do not delete or modify any part of the given data members or member functions. However, you may define additional functions and data members, if necessary.

2. The requirements of the `MusicAlbum` class are given below. The name of the class should be `MusicAlbum`. This time, the interface for the class:

```
#ifndef __MUSIC_ALBUM_H
#define __MUSIC_ALBUM_H

#include <string>
using namespace std;
#include "Song.h"

class MusicAlbum {
public:
   MusicAlbum(const string maArtist = "",
              const string maTitle = "",
              const int maYear = 0);
   ~MusicAlbum();
   MusicAlbum(const MusicAlbum &maToCopy);
   void operator=(const MusicAlbum &right);
   string getMusicAlbumArtist();
   string getMusicAlbumTitle();
   int getMusicAlbumYear();
   void calculateMusicAlbumLength(int &minutes, int &seconds);

private:
   string artist;
   string title;
   int year;
   Song *songs;
   int noSongs;
};
#endif
```

should be coded in a file called `MusicAlbum.h` and its implementation should be coded in a file called `MusicAlbum.cpp`.

- The `MusicAlbum` class is similar to the one given in Part A. However, now it should keep a dynamic array of `Song` objects as well. Similar to Part A, you should implement your own default constructor, destructor and copy constructor, and do not forget to overload the assignment operator. Also, you should make sure to implement get functions for the `Artist`, `Title` and `Year` data members since they will be used to test your program.
- Your class should have a public member function called `calculateMusicAlbumLength` which will be used for testing. This function should calculate the running length of the music album upon which it is invoked by summing up the lengths of its songs in minutes and seconds. The running length of the music album should be returned through pass-by-reference parameters called `minutes` and `seconds`. Note that in implementing this function, you should use the fact that 1 minute is 60 seconds.
- Do not delete or modify any part of the given data members or member functions. However, you may define additional functions and data members, if necessary.

3. Below is the required part of the `MAC` class that you will code in Part B of this assignment. The name of the class should be `MAC`; the interface for the class:

```
#ifndef __MAC_H
#define __MAC_H

#include <string>
using namespace std;
#include "MusicAlbum.h"

class MAC{
public:
   MAC();
   ~MAC();
   MAC(const MAC &macToCopy);
   void operator=(const MAC &right);
   bool addMusicAlbum(const string maArtist,
                      const string maTitle,
                      const int maYear);
   bool removeMusicAlbum(const string maArtist,
                         const string maTitle);
   int getMusicAlbums(MusicAlbum *&allMusicAlbums);
   bool addSong(const string maArtist, const string maTitle,
                const string sName, const int sMins,
                const int sSecs);
   bool removeSongs(const string maArtist,
                    const string maTitle);
   void calculateAvgMusicAlbumLength(int &minutes,
                                     int &seconds);

private:
   MusicAlbum *musicAlbums;
   int noOfMusicAlbums;
};
#endif
```

should be coded in a file called `MAC.h` and its implementation should be coded in a file called `MAC.cpp`.

- The `MAC` class is similar to the one given in Part A. However, now it should also keep the list of songs for each music album. Similar to Part A, you should implement your own default constructor, destructor and copy constructor, and overload the assignment operator. Also you should make sure to implement the `getMusicAlbums` function, as explained before.
- Implement the following functions that your extended system should support.

  Add a music album: This function adds a music album to the collection. The artist, the title and the year are specified as parameters. In this function, the song list is not specified; the song(s) will be added later. In the music album collection, the pair of artist and title are unique (however, there can be multiple albums of the same artist and multiple albums with the same title). Thus, if the user attempts to add a music album with an already existing artist and title, do not add the music album and return `false`. Do not display any warning messages. Otherwise, if the album does not exist in the collection, add it to the collection and return `true`. This function is similar to what you will have implemented in Part A. But, in Part B, you should also create an empty song list for the music album when you add it to the collection.

Remove a music album: This function removes a music album from the collection. The artist and the title are specified as parameters. If the given album exists in the collection, remove it from the collection and return `true`. Otherwise, if there is no album with the given artist and title, do not perform any action and return `false`. Likewise, do not display any warning messages. Note that this function should also clear the song list of the specified music album. This function is similar to what you will have implemented in Part A. But now, for Part B, you should also remove the song list when you remove the music album from the collection.

Add a song to the music album: This function adds a song to the song list of a music album in the collection. The artist**,** the title and the year together with the name and the running length of the song in minutes and seconds are specified as parameters. In this function, you should take care of the following issues:

- If the specified music album does not exist in the collection, do not perform any action and return `false`. Do not display any warning messages.
- All song names are unique in a music album. Thus, if the user attempts to add a song with an existing name for the specified music album, do not perform any action and return `false`. Do not display any warning messages. (However, different music albums may have a song with the same name.)
- Otherwise, add the song to the song list of the specified music album and return `true`.

Remove song list from the music album: This function clears the song list of a music album. The artist and the title are given as parameters. If there is no music album with the specified artist and title, or if there are no songs in the song list, do not perform any action and return `false`. Do not display any warning messages. Otherwise, clear the song list of the specified music album and return `true`.

Calculate the average running length of music albums in the collection: This function calculates the average running length of all music albums in the collection in minutes and seconds and returns the result through pass-by-reference parameters called `minutes` and `seconds`. Note that in implementing this function you should resort to the `calculateMusicAlbumLength` function of the `MusicAlbum` class and use the fact that 1 minute is 60 seconds. If there are no music albums in the collection, or all music albums have empty song lists, this function should return 0 minutes and 0 seconds.

- Do not delete or modify any part of the given data members or member functions. However, you may define additional functions and data members, if necessary.

4. To test Part B, you should code your own main function in a separate file. Do not forget to test your code for different cases. However, do not submit any file containing the main function; otherwise, you may lose a considerable number of points. We will code our own driver to grade Part B of your assignment. In doing that, we will use the get functions of the `MusicAlbum` and `MAC` classes. Thus, do not forget to implement the get functions of these two classes.

Below is an example of the test code and its output. In this example code, there are two global functions called `displayAllMusicAlbums` and `displayStatistics`. The former is the same as the one implemented for Part A. The purpose of the latter is to obtain some statistics about music albums of artists. We will use these functions to understand whether you add/remove music

```
#include <iostream>
#include "MusicAlbum.h"
#include "MAC.h"
using namespace std;

void displayAllMusicAlbums(MAC mac){
   MusicAlbum *allMusicAlbums;
   int noOfMusicAlbums = mac.getMusicAlbums(allMusicAlbums);

   cout <<"No of music albums: "<< noOfMusicAlbums << endl;
   for (int i = 0; i < noOfMusicAlbums; i++){
       cout << allMusicAlbums[i].getMusicAlbumArtist() <<", "
            << allMusicAlbums[i].getMusicAlbumTitle() <<" ("
            << allMusicAlbums[i].getMusicAlbumYear() <<")"
            << endl;
   }
   if (allMusicAlbums != NULL)
      delete [] allMusicAlbums;
}

void displayStatistics(MAC mac){
   MusicAlbum *allMusicAlbums;
   int noOfMusicAlbums = mac.getMusicAlbums(allMusicAlbums);
   int count[8] = {0};
   int mins, secs;

   for (int i = 0; i < noOfMusicAlbums; i++) {
       allMusicAlbums[i].calculateMusicAlbumLength(mins,
                                                   secs);
       if (mins < 70)
          count[mins/10]++;
       else
          count[7]++;
   }
   for (int i = 0; i < 7; i++)
       if (count[i] > 0)
          cout <<"Number of albums with running time in [ "
               << i*10 << "," << (i+1)*10 << ") minutes: "
               << count[i] << endl;
   if (count[7] > 0)
      cout << "Number of albums with running time >= 70 "
           << "minutes: " << count[7] << endl;
   mac.calculateAvgMusicAlbumLength(mins, secs);
   cout << "Average album running time: " << mins
        << " minutes, " << secs << " seconds" << endl;

   if (allMusicAlbums != NULL)
      delete [] allMusicAlbums;
}
```

```
int main(){
   MAC m;

   m.addMusicAlbum("John Coltrane", "My Favorite Things",
                   1961);
   m.addSong("John Coltrane", "My Favorite Things",
             "My Favorite Things", 13, 44);
   m.addSong("John Coltrane", "My Favorite Things",
             "Ever'y Time We Say Goodbye", 5, 43);
   m.addSong("John Coltrane", "My Favorite Things",
             "Summertime", 11, 36);
   m.addSong("John Coltrane", "My Favorite Things",
             "But Not For Me", 9, 35);

   m.addMusicAlbum("John Coltrane", "A Love Supreme",
                   1965);
   m.addSong("John Coltrane", "A Love Supreme",
             "Acknowledgement", 7, 48);
   m.addSong("John Coltrane", "A Love Supreme",
             "Resolution", 7, 25);
   m.addSong("John Coltrane", "A Love Supreme",
             "Pursuance", 10, 46);
   m.addSong("John Coltrane", "A Love Supreme",
             "Psalm", 7, 05);

   m.addMusicAlbum("Jethro Tull", "Thick As A Brick",
                   1972);
   m.addSong("Jethro Tull", "Thick As A Brick",
             "Thick As A Brick, Part 1", 22, 45);
   m.addSong("Jethro Tull", "Thick As A Brick",
             "Thick As A Brick, Part 2", 21, 05);

   m.addMusicAlbum("Mike Oldfield", "Tubular Bells",
                   1973);
   m.addSong("Mike Oldfield", "Tubular Bells",
             "Part One", 25, 00);
   m.addSong("Mike Oldfield", "Tubular Bells",
             "Part Two", 23, 50);

   m.removeMusicAlbum("John Coltrane", "My Favorite Things");

   displayAllMusicAlbums(m);
   displayStatistics(m);

   return 0;
}
```

albums and song lists correctly. If you want, you may use these global functions for your tests as well. Again the last two lines are included to deallocate the `allMusicAlbums` array. Do not remove these lines if you get any run-time errors while using these functions. If you have such run-time errors, most probably, you made an error in implementing either the `getMusicAlbums` function or one or more of the destructor, copy constructor, and overloaded assignment operator.

The output should be:

```
No of music albums: 3
John Coltrane, A Love Supreme (1965)
Jethro Tull, Thick As A Brick (1972)
Mike Oldfield, Tubular Bells (1973)
Number of albums with running time in [30,40) minutes: 1
Number of albums with running time in [40,50) minutes: 2
Average album running time: 41 minutes, 54 seconds
```

What to submit for Part B?

You should put your `MusicAlbum.h`, `MusicAlbum.cpp`, `Song.h`, `Song.cpp`, `MAC.h`, and `MAC.cpp` files into a folder and zip the folder. In this zip file, there should not be any file containing the `main` function. The name of this zip file should be `PartB_secX_Firstname_Lastname_StudentID.zip` where `X` is your section number. Then you should follow the steps explained at the end of this document for the submission of Part B.

What to be careful about implementation and submission for Part B?

You need to read the "notes about implementation" and "notes about submission" parts that are given just below.

NOTES ABOUT IMPLEMENTATION (for both Part A and Part B):

1. You must use dynamically allocated arrays. You will receive no points if you use fixed-sized arrays, linked-lists or any other data structures such as `vector/array` from the standard library.
2. You are not allowed to use any global variables or any global functions when implementing the classes. However, you may use global functions to test your program, but should not submit them.
3. Your code must not have any memory leaks. You will lose points if your code has memory leaks even though it produces correct output. To detect memory leaks, you may want to use Valgrind which is available at `http://valgrind.org`.

NOTES ABOUT SUBMISSION (for both Part A and Part B):

1. Conform to the rules given separately for Part A and Part B. That is, the name of classes, the name of `.h` and `.cpp` files, and the name of zip files should conform to the specifications given separately for Part A and Part B. Otherwise, you may lose a considerable number of points.
2. **Before 23:59 on November 21, you need to send an email with a subject line CS 201 HW1 to Furkan Hüseyin, by attaching two zip files (one for Part A and the other for Part B). Read "what to submit for Part A" and "what to submit for Part B" sections very carefully.**
3. No hardcopy submission is needed. The standard rules about late homework submissions apply.
4. Do not submit any files containing the `main` function.
5. You are free to code your programs on Linux or Windows platforms. However, we will test your programs on "`dijkstra.ug.bcc.bilkent.edu.tr`" and we will expect your programs to compile and run on the `dijkstra` machine. If we cannot get your program to properly work on the `dijkstra` machine, you will end up losing a considerable number of points. Therefore, we recommend you to make sure that your program compiles and properly works on "`dijkstra.ug.bcc.bilkent.edu.tr`" before submitting your assignment.
6. This assignment will be graded by your TA **Furkan Hüseyin** (**`furkan.huseyin at bilkent edu tr`**). Thus, you may ask your homework related questions directly to him.