COMP 2012 Final Exam - Spring 2020 - HKUST

Date: June 2, 2020 (Tuesday)
Time Allowed: 3 hours, 6:30–9:30 pm
Instructions:

1. This is a closed-book, closed-notes examination.
2. There are **5** questions delivered in **3** separate parts A, B and C.
3. Type your answers in the space provided on Canvas.
4. All programming codes in your answers must be written in the ANSI C++ version as taught in the class.
5. For programming questions, unless otherwise stated, you are **NOT** allowed to define additional structures, classes, helper functions and use global variables, **auto**, nor any library functions not mentioned in the questions.

COMP 2012 Final Exam - Spring 2020 - HKUST: Part A

Time Allowed: 40 minutes

Instructions:

1. There are **3** short questions in this part.
2. This is a closed-book, closed-notes examination.
3. Type your answers in the space provided on Canvas.
4. All programming codes in the questions are written in the ANSI C++ version as taught in the class.

Problem	**Topic**	**Score**
1	Hashing	/ 12
2	AVL	/ 12
3	Inheritance and Move Semantics	/ 12
	Total	/ 36

Problem 1 [12 points] Hashing

Insert the following numbers,

$$21, 47, 18, 31, 44, 70$$

in order (from left to right) into a hash table using double hashing with the first hash function being

$$hash(k) = k \bmod 13$$

and the second hash function being

$$hash_2(k) = 11 - (k \bmod 11) \ .$$

The table size is 13 and it is initially empty. Fill in the following table to show the result of the six insertions.

Solution:

Each correct insertion gives 2 points. If a student makes a mistake at one insertion but the next insertions are "consistent" with this mistake, the they should only lose one point for the original mistake. Run the accompanying program hashing.cpp to get the correct grade for a sequence of chosen positions.

Position	Add 21	Add 47	Add 18	Add 31	Add 44	Add 70
0						
1					44	44
2						
3		47	47	47	47	47
4						
5			18	18	18	18
6						
7				31	31	31
8	21	21	21	21	21	21
9						
10						
11						
12						70

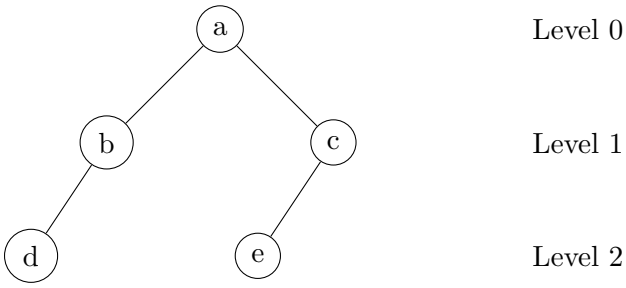
Problem 2 [12 points] AVL Tree

Insert the following keys,

11, 21, 16, 26, 31, 10, 9, 7, 8

in order (from left to right), into an initially empty AVL tree. Write the level order traversal of the AVL tree that results from successive key insertions in the table below.

Note: Level order traversal prints the nodes of a tree level by level. That is, all nodes of level 0 are printed first, followed by nodes of level 1, and so on. All nodes of any level are printed from left to right. For example, the level order traversal of the following tree prints out: `a, b, c, d, e`.



To illustrate what you need to do, the insertion of the first three keys has been done for you.
Solution:

	Level order traversal output of the AVL tree that results from successive key insertions
After the insertion of 11, 21 & 16	16, 11, 21
After further insertion of 26 & 31	16, 11, 26, 21, 31
After further insertion of 10 & 9	16, 10, 26, 9, 11, 21, 31
After further insertion of 7 & 8	16, 10, 26, 8, 11, 21, 31, 7, 9

Marking Scheme:

- For each line of output,
  - give 4 points if the output sequence is exactly the same as the solution.
  - give 3 points if the output sequence is not the same, but it forms an AVL tree (use the given program to check whether the sequence forms an AVL tree).

Problem 3 [12 points] Inheritance and Move Semantics

Below is the definition of the `Word` class in the file `word-move.h` as described in our lecture, except for the addition of the `get_word()` member functions.

```
class Word              /* File: word-move.h */
{
  private:
    int freq = 0; char* str = nullptr;

  public:
    Word() { cout << "default constructor" << endl; }

    Word(const char* s, int f = 1) : freq(f), str(new char [strlen(s)+1])
        { strcpy(str, s); cout << "Conversion: "; print(); }

    Word(const Word& w) : freq(w.freq), str(new char [strlen(w.str)+1])
        { strcpy(str, w.str); cout << "Copy: "; print(); }

    Word(Word&& w) : freq(w.freq), str(w.str)
        { w.freq = 0; w.str = nullptr; cout << "Move: "; print(); }

    ~Word() { cout << "D: "; print(); delete [] str; }

    Word to_upper_case() const
    {
        Word x(*this);
        for (char* p = x.str; *p != '\0'; p++) *p += 'A' - 'a';
        return (x);
    }

    Word& get_word() { return *this; }
    const Word& get_word() const { return *this; }

    void print() const
        { cout << (str ? str : "null") << " ; " << freq << endl; }
};
```

Let's derive a new class `WordX` in the file `wordx.h` from `Word`. WordX has another `Word` object called 'word2' that is the same as the `Word` object from its parent except that all characters of 'word2' are written in upper case. We assume that the word from `WordX`'s parent `Word` class is ALWAYS written with the 26 English letters in lower case.

```
1  #include "word-move.h"  /* File: wordx.h */
2
3  class WordX: public Word
4  {
5    private:
6      Word word2;
7
8    public:
9      WordX() { cout << "-- Default --" << endl; print(); }
10
11     WordX(const Word& w) : Word(w), word2(w.to_upper_case())
12         { cout << "-- Conversion --" << endl; print(); }
13
14     WordX(const WordX& wx) : Word(wx.get_word()), word2(wx.word2)
15         { cout << "-- Deep Copy --" << endl; print(); }
16
17     WordX(WordX&& wx) :  Word(std::move(wx.get_word())), word2(std::move(wx.word2))
18         { cout << "-- Shallow Copy --" << endl; print(); }
19
20     void print() const
21     {
22         cout << "word = "; Word::print();
23         cout << "word2 = "; word2.print();
24     }
25 };
```

Write down the output of the program in the file `wordx.cpp` below after it is compiled with the following command:

```
g++ -std=c++11 -fno-elide-constructors wordx.cpp
```

Note that the above compilation command will NOT use RVO (return value optimization).

Moreover, the outputs from lines #8, #11, #14 and #17 carry no points by themselves, but they serve as partitions for the outputs which help us give you partial credits; there will be penalty if you miss them.

```
1  #include <iostream>     /* File: wordx.cpp */
2  #include <cstring>
3  using namespace std;
4  #include "wordx.h"
5
6  int main()
7  {
8      cout << "< pet >" << endl;
9      Word pet { "cat" };
10
11     cout << "\n< petX >" << endl;
12     WordX petX { pet };
13
14     cout << "\n< petY >" << endl;
15     WordX petY { std::move(petX) };
16
17     cout << "\n*** End Game ***" << endl;
18     return 0;
19 } /* compilation command: g++ -std=c++11 -fno-elide-constructors wordx.cpp */
```

Solution:

```
< pet >                    // Call conversion constructor of Word
Conversion: cat ; 1        // 0.5

< petX >                   // Call conversion constructor of WordX
Copy: cat ; 1              // petX(1): 0.5 : copy construction of parent Word
Copy: cat ; 1              // petX(2): 0.5 : copy *this to x in to_upper_case()
Move: CAT ; 1              //          1.0 : create temp. obj. by move during return
D: null ; 0                //          0.5 : destruction of x in to_upper_case()
Move: CAT ; 1              // petX(3): 1.0 : create petX by move from the temp. obj.
D: null ; 0                //          0.5 : destruction of temp. obj.
-- Conversion --           // petX(4): 0.5
word = cat ; 1             //          0.5 : WordX::print()
word2 = CAT ; 1            //          0.5

< petY >                   // Call move constructor of WordX
Move: cat ; 1              // petY(1): 1.0 : move construction of parent Word
Move: CAT ; 1              //          1.0 : move construction of member word2
--  Shallow Copy --        // petY(2): 0.5
word = cat ; 1             //          0.5 : WordX::print()
word2 = CAT ; 1            //          0.5

*** End Game ***
D: CAT ; 1                 // 0.5 : destruction of petY
D: cat ; 1                 // 0.5
D: null ; 0                // 0.5 : destruction of petX
D: null ; 0                // 0.5
D: cat ; 1                 // 0.5 : destruction of pet
```

Scheme:

- No points for the blank lines and no penalty for missing them.
- No points for the partition lines with `< .... >` or `***` , but there is a penalty for missing them: -0.25 point for each missing line.
- Simply put, 0.5 point for each meaningful output line, except (1) the blank lines and partition lines, and (2) those lines with "move" which are worth 1.0 point each.
- Points are given based on 8 concepts in the Canvas grading rubric.
- The outputs have to be given in the order of the concepts.
- Partial credits are given based on identifying the concepts from top to bottom as much as we can.
- Penalty for extra lines: -0.25 each, but the min. for each concept is 0.
- No partial credits for inexact outputs from petX(4) and petY(1) and End Game.

---------- END OF PART A ----------

COMP 2012 Final Exam - Spring 2020 - HKUST: Part B

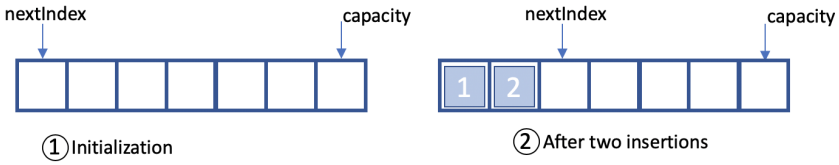
Time Allowed: 40 minutes

Instructions:

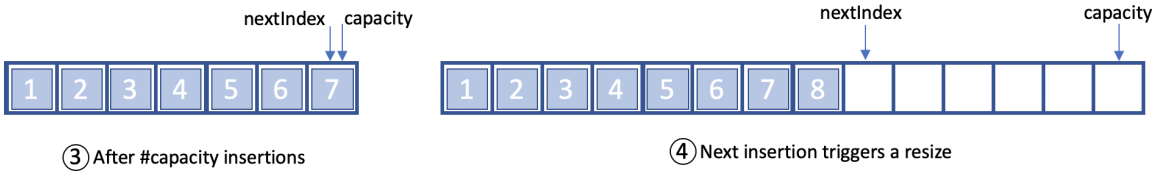
1. There is **1** long question in this part.
2. This is a closed-book, closed-notes examination.
3. Type your answers in the space provided on Canvas.
4. All programming codes in your answers must be written in the ANSI C++ version as taught in the class.
5. For programming questions, unless otherwise stated, you are **NOT** allowed to define additional structures, classes, helper functions and use global variables, **auto**, nor any library functions not mentioned in the questions.

Problem 4 [32 points] Generic Programming

A *flexible array* is an array-based data structure whose allocated size can grow, in contrast to a regular array. Each flexible array has a `capacity` chosen upon initialization and items can only be inserted at the first empty position.



When a new item cannot be inserted because the array is full, a "resize" operation takes place; i.e., a new chunk of memory of size two times that of the previous capacity is allocated. The previous items are copied to the starting memory locations and the new item is inserted after them.



For this question, you will implement a template class `FlexArray<T>` for flexible arrays. The class `FlexArray` has the following private data members:

- `T* data` is a pointer to the first memory location of the array
- `int capacity` is the chosen capacity
- `int nextIndex` is the index where the next element will be inserted (initially 0, incremented after every insertion)

It also has a public accessor function:

- `int getNext() const {return nextIndex;}` that returns the next available position.

You must write the *entire class implementation* in a single `FlexArray.h` file below. Specifically, you need to implement the following methods:

(i) Write a conversion constructor that takes input `int n` for capacity and allocates consecutive memory for `n` values of type `T`; if no input is given, it works as a default constructor with capacity 10.

(ii) Write a copy constructor that performs standard deep copy.

(iii) Write a destructor that correctly releases all allocated memory.

(iv) Overload the subscript operator `[]` that takes input `int i`. For `FlexArray arr`, the expression `arr[i]` returns the i-th position item in read-only form. Since items should only

be inserted at the first available position, it should not be possible to use the subscript operator to write to a position of the `FlexArray`. E.g., code of the form `arr[i] = x` should return a compilation error.

(v) Overload the `+=` operator that takes input `T val` and returns a non-constant reference to the calling `FlexArray` object. It inserts `val` to the first available position of the array. If the array has reached capacity and there is no space to do the insertion, new memory of size 2∗`capacity` should be allocated. All previous items should be copied starting from the beginning and the new item should be added after them.

(vi) Overload the `-=` operator that takes input `T val` and returns a non-constant reference to the calling `FlexArray` object. It removes *all occurrences* of `T val` from the array. After removing them, all remaining items at the array must be stored at *consecutive* memory locations starting from the beginning, in the same relative order as before.

(vii) Overload the stream insertion operator `<<` that takes as input an `ostream` object by non-constant reference and a `FlexArray` object by constant reference, and returns the `ostream` object by non-constant reference. The operator prints every existing item on a separate line starting from position 0. It should be compatible with standard syntax; e.g., `cout << arr` should work as described.

Note:

- All member functions should be defined inside the class definition.
- Operators `[]`,`+=`,`-=` must be overloaded as public class member functions.
- Operator `<<` must be overloaded as global function.
- Methods that do not modify their arguments should be declared constant.
- If necessary, you may write additional private helper functions.
- You may assume that type `T` has well-defined copy assignment `=`, stream insertion `<<`, and is-equal `==` operators, as well as a default constructor and proper destructor.
- Make sure that your code does not cause any memory leakage.

Solution: The distribution of points is as follows:

Preamble Inclusion of iostream (optional using namespace std; if not they need to declare the std scope later when using cout), class declaration, correct declaration of the three private data members and the getIndex function. 3 pts

Part (i) 2 pts

Part (ii) 2 pts

Part (iii) 2 pts

Part (iv) 2 pts

Part (v) 8 pts

Part (vi) 9 pts

Part (vii) 4 pts

In the comments of the following code there is sample point value for each line. This is meant as a guideline to you and you do not need to follow it exactly. E.g., students may have implemented things slightly differently. As long as their implementation works, they should receive the points. As an example, if students choose to do the resizing operation as soon as the array is full (and not during the following insertion) and they have implemented this correctly, they should get full points.

```
/* Syntax errors: -0.5 point each, max 2 points; same error counts only once. */

//PREABLE
#include <iostream>     //0.5 pts
using namespace std;    //0.5 pts

template <class T>      //1 pts
class FlexArray
{
private:
    T *data;            //0.5pt for correct declaration
    int capacity;       //of all three private
    int nextIndex;      // data members

public:
    int getNext() const { return nextIndex; } //0.5 pts
//PART (i)
    FlexArray(int n =10) : data(new T[n]), capacity(n), nextIndex(0){ } //2 pts
//PART (ii)
    FlexArray(const FlexArray& arr) : FlexArray(arr.capacity){    //1 pts
        for (int i = 0; i < capacity; i++) data[i] = arr.data[i]; //0.5 pts
        nextIndex = arr.nextIndex;                                //0.5 pts
    }
//PART (iii)
    ~FlexArray() { delete [] data; } //2 pts

//PART (iv)
    const T& operator[](int index) const { return data[index]; } //2 pts

//PART (v)
    FlexArray<T>& operator+=(T val){                               //0.5 pts
        if (nextIndex == capacity) {                               //1 pts
            capacity = 2*capacity;                                 //0.5 pts
            T *pnew = new T[capacity];                             //1 pts
            for (int i = 0; i < nextIndex; i++) pnew[i] = data[i]; //1 pts
            delete [] data;                                        //1 pts
            data = pnew;                                           //1 pts
```

```
        }
        data[nextIndex++] = val; //1 pts
        return *this;            //1 pts
    }

//PART (vi)
    FlexArray<T>& operator-=(T val){         //0.5 pts
        T *pnew = new T[capacity];           //1 pts
        int pos = 0;                         //0.5 pts
        for (int i = 0; i < nextIndex; i++){ //1 pts
            if (!(data[i] == val)) {         //1 pts
                pnew[pos] = data[i];         //0.5 pts
                pos++;                       //0.5 pts
            }
        }
        delete [] data;  //1 pts
        data = pnew;     //1 pts
        nextIndex = pos; //1 pts
        return *this;    //1 pts
    }

};

//PART (vii)
template <class T>                                          //1 pts
ostream& operator<<(ostream& os, const FlexArray<T>& arr) //1 pts
{
    for (int i = 0; i < arr.getNext(); i++) os << arr[i] << endl; //1 pts
    return os;                                                    //1 pts
}
```

---------- END OF PART B ----------

COMP 2012 Final Exam - Spring 2020 - HKUST: Part C

Time Allowed: 40 minutes

Instructions:

1. There is **1** long question in this part.
2. This is a closed-book, closed-notes examination.
3. Type your answers in the space provided on Canvas.
4. All programming codes in your answers must be written in the ANSI C++ version as taught in the class.
5. For programming questions, unless otherwise stated, you are **NOT** allowed to define additional structures, classes, helper functions and use global variables, **auto**, nor any library functions not mentioned in the questions.

Problem 5 [32 points] Abstract Base Class, Inheritance & Polymorphism

This problem involves 4 classes called '`Media`', '`Song`', '`Video`', and '`Album`'. Below are the header files of the 4 classes.

```
/* File: Media.h */
// An Abstract Base Class

class Media {
  private:
    string name; // The name of the Media

  public:
    // Conversion constructor
    Media(string n) : name(n) {}

    // Virtual destructor
    virtual ~Media() { }

    // Accessor of the data member name
    string getName() const { return name; }

    // A pure virtual function, which is supposed to return the length of the media
    virtual int getLength() const = 0;

    // A pure virtual function, which prints all the details of the media
    virtual void print() const = 0;
};


/* File: Song.h */

class Song : public Media {
  private:
    string performer;   // The performer of the song
    int duration;       // The duration of the song in seconds

  public:
    // Construct a song with the given name, performer, and duration
    Song(string n, string p, int d) : Media(n), performer(p), duration(d) {}

    // Override the pure virtual function getLength inherited from the Media class,
    // which returns the duration of the song
    virtual int getLength() const override { return duration; }

    // Override the pure virtual function print inherited from the Media class,
    // which prints all the details of the song
    virtual void print() const override {
      cout << getName() << ", " << performer << ", " << duration << endl;
    }
};
```

```
/* File: Video.h */

class Video : public Media {
  private:
    int fps;         // The number of frames in the video per second
    int numFrames;   // The number of frames in the video
    bool interlaced; // Whether it is an interlaced video

  public:
    // Construct a video with the given name, frames per second,
    // number of frames, and a boolean indicating if it is interlaced
    Video(string n, int f, int nf, bool i) : Media(n), fps(f),
                                             numFrames(nf), interlaced(i) {}

    // Override the pure virtual function getLength inherited from the Media class,
    // which returns the length of the video in seconds
    virtual int getLength() const override { return numFrames / fps; }

    // Override the pure virtual function print inherited from the Media class,
    // which prints all the details of the Video
    virtual void print() const override {
      cout << boolalpha << getName() << ", " << fps << ", "
           << numFrames << ", " << interlaced << endl;
    }
};
```

```cpp
/* File: Album.h */
#include <iostream>
using namespace std;
#include "Media.h"
#include "Song.h"
#include "Video.h"

class Album {
  private:
    string name;             // The name of the album
    string* artist;          // A pointer to an array of strings representing
                             // the artists of the album
    unsigned int numArtist; // The number of artists in the array pointed by artist
    Media** mediaList;       // A pointer to an array of pointers of Media type
    unsigned int numMedia;  // The number of media in the album. It refers to the
                             // size of the pointer array pointed by mediaList

  public:
    /*** IMPLEMENTED ***/
    // Default / Conversion constructor, which constructs an album with the given name
    Album(string n = "") : name(n), artist(nullptr), numArtist(0),
                           mediaList(nullptr), numMedia(0) {}

    /*** FUNCTION TO BE IMPLEMENTED in Album.cpp ***/
    // Copy constructor - Perform deep copy
    // Note: Two different types of objects will be pointed by the array of pointers.
    //       Create Song object when the object to be duplicated is a type of Song.
    //       Create Video object when the object to be duplicated is a type of Video.
    // Hint: Use typeid(<type>).name() and typeid(<expression>).name()
    Album(const Album& album);

    /*** FUNCTION TO BE IMPLEMENTED in Album.cpp ***/
    // Move constructor
    Album(Album&& album);

    /*** FUNCTION TO BE IMPLEMENTED in Album.cpp ***/
    // Destructor, which deallocates all dynmaically allocated memory to avoid any
    // memory leak as the program finishes
    ~Album();

    /*** FUNCTION TO BE IMPLEMENTED in Album.cpp ***/
    // "Expand" the mediaList array by 1, dynamically allocate a Song object
    // with name, performer, duration, and add it to the end of the array
    void addSong(string name, string performer, int duration);

    /*** FUNCTION TO BE IMPLEMENTED in Album.cpp ***/
    // Print the album according to output as specified in the output
    void print() const;
```

```
    /*** IMPLEMENTED ***/
    // Accessor of the data member name
    string getName() const { return name; }

    /*** IMPLEMENTED in Album-partial.cpp ***/
    // "Expand" the artist array by 1 and add the artist named n to the end of the array
    void addArtist(string n);

    /*** IMPLEMENTED in Album-partial.cpp ***/
    // "Expand" the mediaList array by 1; dynamically allocate a Video object
    // with name, frames per second, number of frames, interlaced, and add it to the
    // end of the array
    void addVideo(string name, int fps, int numFrames, bool interlaced);
};
```

The following is the test program "`test-album.cpp`" for the classes.

```
#include "Album.h"
#include "Album-partial.cpp"

int main() {
  Album album1("Soul of HKUST");
  album1.addArtist("Brian");
  album1.addArtist("Desmond");
  album1.addArtist("Dimitris");
  album1.addSong("OOP is cool", "Brian", 150);
  album1.addSong("I love UST", "Desmond", 180);
  album1.addSong("Legend of COMP 2012", "Dimitris", 184);
  album1.addVideo("Deadline fighter", 30, 6000, true);
  album1.addVideo("Free of free rider", 40, 9000, false);

  cout << "album1" << endl;
  cout << "======" << endl;
  album1.print();
  cout << endl;

  Album album2(album1);
  album2.addVideo("Be confident", 40, 5000, false);
  cout << "album2" << endl;
  cout << "======" << endl;
  album2.print();

  return 0;
}
```

A sample run of the test program is given as follows:

```
album1
======
Album: Soul of HKUST
Artists:
Brian Desmond Dimitris
Media:
OOP is cool, Brian, 150
I love UST, Desmond, 180
Legend of COMP 2012, Dimitris, 184
Deadline fighter, 30, 6000, true
Free of free rider, 40, 9000, false

album2
======
Album: Soul of HKUST
Artists:
Brian Desmond Dimitris
Media:
OOP is cool, Brian, 150
I love UST, Desmond, 180
Legend of COMP 2012, Dimitris, 184
Deadline fighter, 30, 6000, true
Free of free rider, 40, 9000, false
Be confident, 40, 5000, false
```

Implement the following member functions of the class '`Album`' in the given order in a seperate file "`Album.cpp`" according to the details given in "`Album.h`".

- `Album(const Album& album)`
- `Album(Album&& album)`
- `~Album()`
- `void addSong(string name, string performer, int duruation)`
- `virtual void print() const;`

Note:

- The artist and medalist arrays should be always just big enough to hold all their existing elements.
- Make sure that your code does not cause any memory leakage.

Solution:

```
/* Syntax errors: -0.5 point each, max 2 points; same error counts only once. */
#include <typeinfo>   // 0.5 point
#include "Album.h"    // 0.5 point

/* Total: 10 points */
Album::Album(const Album& album) {
  name = album.name;                                                    // 0.5 point

  artist = new string[album.numArtist];                                 // 1 point
  for(unsigned int i = 0; i < album.numArtist; ++i)                     // 0.5 point
    artist[i] = album.artist[i];                                        // 0.5 point
  numArtist = album.numArtist;                                          // 0.5 point

  mediaList = new Media*[album.numMedia];                               // 1 point
  for(unsigned int  i = 0; i < album.numMedia; ++i) {                   // 0.5 point
    if(typeid(*album.mediaList[i]) == typeid(Song))                     // 1 point
      mediaList[i] = new Song(*dynamic_cast<Song*>(album.mediaList[i])); // 2 points
    else
      mediaList[i] = new Video(*dynamic_cast<Video*>(album.mediaList[i])); // 2 points
  }
  numMedia = album.numMedia;                                            // 0.5 point
}

/* Total: 5.5 points */
Album::Album(Album&& album) {
  name = album.name;                                    // 0.5 point
  artist = album.artist;                                // 0.5 point
  numArtist = album.numArtist;                          // 0.5 point
  mediaList = album.mediaList;                          // 0.5 point
  numMedia = album.numMedia;                            // 0.5 point

  album.artist = nullptr;                               // 1 point
  album.numArtist = 0;
  album.mediaList = nullptr;                            // 1 point
  album.numMedia = 0;                                   // 1 point*
  // Remark:
  // 1 point for album.numMedia = 0; can be also given if students
  // do some checking before using numMedia in the destructor.
}

/* Total: 4 points */
Album::~Album() {
  delete [] artist;                                     // 1 point
  for(unsigned int  i = 0; i < numMedia; ++i)           // 1 point
    delete mediaList[i];                                // 1 point
  delete [] mediaList;                                  // 1 point
}
```

```
/* Total: 7 points */
void Album::addSong(string name, string performer, int duration) {
  Media** temp = new Media*[numMedia + 1];                  // 2 points
  for(unsigned int  i = 0; i < numMedia; ++i)               // 0.5 point
    temp[i] = mediaList[i];                                 // 0.5 point
  temp[numMedia++] = new Song(name, performer, duration);   // 2 points*
  // Remark:
  // 1 point for temp[numMedia] = new Song(name, performer, duration);
  // 1 point for numMedia++
  delete [] mediaList;                                      // 1 point
  mediaList = temp;                                         // 1 point
}

/* Total: 4.5 points */
void Album::print() const {
  cout << "Album: " << name << endl;                        // 0.5 point
  cout << "Artists: " << endl;                              // 0.5 point
  for(unsigned int  i = 0; i < numArtist; ++i)              // 0.5 point
    cout << artist[i] << " ";                               // 0.5 point
  cout << endl;                                             // 0.5 point
  cout << "Media: " << endl;                                // 0.5 point
  for(unsigned int  i = 0; i < numMedia; ++i)               // 0.5 point
    mediaList[i]->print();                                  // 1 point
}

// Remarks:
// 0.5 point is given if students put delete[] instead of delete.
// For any part that is worth more than 1 point, we can give 0 mark,
// full points, or half point for partially correct answer.
```

---------- END OF PART C ----------