COMP 2012 Midterm Exam - Spring 2018 - HKUST

Date: March 24, 2018 (Saturday)
Time Allowed: 2 hours, 1–3 pm
Instructions:

1. This is a closed-book, closed-notes examination.
2. There are **7** questions on **24** pages (including this cover page and 3 blank pages at the end).
3. Write your answers in the space provided in black/blue ink. *NO pencil please, otherwise you are not allowed to appeal for any grading disagreements.*
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.

Student Name	Solution & Marking Scheme
Student ID	
Email Address	
Venue & Seat Number	

For T.A. Use Only

Problem	Score
1	/ 10
2	/ 8
3	/ 10
4	/ 8
5	/ 9
6	/ 25
7	/ 30
Total	/ 100

Problem 1 [10 points] True or false

Indicate whether the following statements are *true* or *false* by circling **T** or **F**. You get 1.0 point for each correct answer, −0.5 for each wrong answer, and 0.0 if you do not answer.

T **F** (a) There is compilation error in the following code segment.

```
class A {
  public:
    int& getA() const {
      return a;
    }
    int calc(int b) const {
      return getA() + b;
    }
  private:
    const int a{10};
};
```

T **F** (b) For a const data member of a class, if it is already initialized by a default member initializer in the class definition, it cannot be initialized again by any constructor of the class.

T **F** (c) Default argument for a given parameter of a function should NOT be specified in both .h and .cpp files.

T **F** (d) The compiler always generates a default constructor for a class if no such constructor has been implemented.

T **F** (e) The initialization order of non-static data members can be controlled by the order of their appearance in the member initialization list.

T **F** (f) If an object of a user-defined type is returned by a function by value, the returned object cannot be further modified. In contrast, if the object is returned by reference, it can be further modified.

T **F** (g) The output of the following program is `~C~A~B`.

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

class A {
  public:
    ~A() { cout << "~A"; }
};
class B {
  public:
    ~B() { cout << "~B"; }
};
class C : public B {
  public:
    ~C() { cout << "~C"; }
    A a;
};

int main() {
  C* c = new C;
  delete c;
}
```

T **F** (h) There is NO compilation error in the following program.

```
class Base {
  private: int a;
  public: Base(int a) { }
};

class Derived : public Base {
  private: int b;
  public: Derived(int a, int b) : b(b), Base(a) {}
};

int main() {
  Derived obj(1, 2);
}
```

T **F** (i) With public inheritance, one may always assign a pointer of a base class `B` to an object of its derived class `D`:

```
D d;
B* bp = &d;
```

regardless of whether `B` is a direct base class, an indirect base class, or an abstract base class of `D`.

T **F** (j) C++ does not allow an abstract base class to be derived from another abstract base class.

Problem 2 [8 points] Classes and Objects

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

class Word
{
  private:
    int freq = 0; char* str = nullptr;  // You are NOT allowed to modify this line

  public:
    Word() = default;        // Default constructor

    Word(char* s, int f = 1) // Initialize with the given C string s and frequency f
    {
        freq = f;
        str = s;
        cout << "conversion: "; print();
    }

    ~Word() // You are NOT allowed to modify the destructor function
    {
        cout << "destructor: "; print();
        delete [] str;
    }

    void print() const // Print out the stored C string and frequency
    {
        cout << str << " ; " << freq << endl;
    }
};

int main() // You are NOT allowed to modify the main function
{
    char s[] = "stress";
    Word x, y, z {s};
    x = z;
    return 0;
}
```

The program above will compile but it runs with 3 run-time errors during the destruction of the 3 `Word` objects, `z`, `y`, and `x` in that order. *Some remarks:*

- *You are NOT allowed to change the meaning of any given member function.*
- *You are NOT allowed to modify the main function nor the destructor function of the Word class in your answers to this question.*
- *You don't get ANY marks if you only give the line numbers without correction for parts (a) and (b), or explanation for part (c).*

(a) [3 points] Identify the statement, by giving its line number in the program, that causes the run-time error when `z` is destructed. Re-write the concerned statement to eliminate this error.

Answer:

(b) [3 points] However, even after you fix the error in part (a), it still will run into another run-time error when `y` is destructed. Again, identify the statement, by giving its line number, that causes the error, and then re-write the concerned statement to eliminate this 2nd error.

Answer:

(c) [2 points] To your dismay, even after you fix the 2 errors in part (a) and (b), your program still will run into the 3rd run-time error when `x` is destructed. Identify, for the last time, the statement, by giving its line number, that causes this last error. This time, you only need to explain how the error is produced and you don't need to fix it.

Answer:

Solution:

(a) Line #16. When z is destructed, the destructor tries to delete the memory for z.str which is pointing to a non-dynamic array on the stack.
Fix: rewrite the conversion constructor to perform deep copy.

(b) Line #28. Word y is created with the default constructor and its str is a nullptr. When y is destructed, y.print() tries to print a nullptr and gives a run-time error.
Fix: modify the print function, check str and print it only if it is not a nullptr.

(c) Line #36. With the default assignment operator function doing the shallow copy, the destruction of x will delete z.str again which has been returned to the heap during z's destruction.
Fix (not required): rewrite the Word::operator= function to do deep copy.

Scheme:

- For all 3 parts, no marks are given if no explanations are given.
- Otherwise, 1 point for the line number as far as some reasonable explanation is given.
- 2 points if the line number is correct and the reason is correct. For parts (a) and (b), 1 point for the fixing code.

Here is the complete program after fixing the 3 errors.

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

class Word
{
  private:
    int freq = 0; char* str = nullptr;  // You are NOT allowed to modify this line

  public:
    Word() = default;          // Default constructor

    Word(char* s, int f = 1)  // Initialize with the given C string s and frequency f
    {
        freq = f;
        str = new char [strlen(s)+1]; strcpy(str, s); // Part (a)
        cout << "conversion: "; print();
    }

    ~Word() // You are NOT allowed to modify the destructor function
    {
        cout << "destructor: "; print();
        delete [] str;
    }

    void print() const // Print out the stored C string and frequency
    {
        cout << (str ? str : "") << " ; " << freq << endl; // Part (b)
    }

    const Word& operator=(const Word& w) // Part (c)
    {
        if (this != &w)
        {
            freq = w.freq;
            delete [] str;
            str = new char [strlen(w.str)+1]; strcpy(str, w.str);
        }
        return (*this);
    }
};

int main() // You are NOT allowed to modify the main function
{
    char s[] = "stress";
    Word x, y, z {s};
    x = z;
    return 0;
}
```

Problem 3 [10 points] Const-ness

In the following program, for the 10 statements ending with the following comment:

```
/* Error:   Yes   /   No   /   Don't know   */
```

decide whether the statement is syntactically INCORRECT - that is, it will produce compilation error(s). Circle "Yes" if it will give compilation error and "No" otherwise.
You get 1 point for each correct answer, -0.5 for each wrong answer, and 0.0 if you do not answer by circling "Don't know".

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

int main() {
  // A pointer to a pointer to a const int
  const int** ptr1 = new int*;                  /* Error:  Yes  */
  // A const pointer to a pointer to an int
  int** const ptr2 = new int*;                  /* Error:  No   */
  // A pointer to a pointer to a const int
  const int** ptr3 = new const int*;            /* Error:  No   */
  // A pointer to a pointer to a const int
  const int** ptr4 = new int const*;            /* Error:  No   */
  // A pointer to a pointer to a const int
  const int** ptr5 = new int* const;            /* Error:  Yes  */

  // To evaluate the correctness of the following statements, you may
  // assume that the incorrect statements (if any) are fixed so that all
  // pointer variables declared above are correctly allocated.

  *ptr1 = new int;
  **ptr1 = 1;                                   /* Error:  Yes  */
  *ptr2 = new int;
  **ptr2 = 2;                                   /* Error:  No   */
  *ptr3 = new int;
  **ptr3 = 3;                                   /* Error:  Yes  */
  *ptr4 = new int;
  **ptr4 = 4;                                   /* Error:  Yes  */
  *ptr5 = new int;
  **ptr5 = 5;                                   /* Error:  Yes  */

  // Assume memory de-allocations are purposely not done here.
  return 0;
}
```

Problem 4 [8 points] Order of Constructions and Destructions

Write down the output of the following program when it is run.

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

class Furniture
{
  public:
    Furniture() { cout << "F "; }
    ~Furniture() { cout << "~F "; }
};

class Seat : public Furniture
{
  public:
    Seat() { cout << "S0 "; }
    Seat(int n) { cout << "Sn "; }
    ~Seat() { cout << "~S "; }
};

class Display
{
  public:
    Display() { cout << "D "; }
    ~Display() { cout << "~D "; }
};

class Machine
{
  public:
    Machine() { cout << "M "; }
    ~Machine() { cout << "~M "; }
};

class Vehicle : public Machine
{
    Display display;
  public:
    Vehicle() { cout << "V "; }
    ~Vehicle() { cout << "~V "; }
};

class Car : public Vehicle
{
    Seat seat;
  public:
    Car() { cout << "C "; seat = Seat(2) ; }
    ~Car() { cout << "~C "; }
};

int main() { Car x; cout << endl; return 0; }
```

Answer:

M D V F S0 C F Sn ~S ~F

~C ~S ~F ~V ~D ~M

Scheme:

- Scan the output from left to right, once for the constructions and once for the destructions
- If the printout is not exactly correct, try to find the LONGEST matched sub-sequence of correctly printed tokens
- Give 0.5 point for each correctly printed token in the LONGEST matched sub-sequence
- 2 points if all expected tokens are printed but they are in wrong order

Problem 5 [9 points] Inheritance

The following program contains 6 ERRORS (syntax errors, logical errors, etc.). Study the program carefully, identify each error by writing down the line number where the error occurs, and **explain** why it is an error. You don't need to write codes to fix the errors.

Remark: You don't get ANY marks if you only give the line numbers without explanation.

```
1  #include <iostream>
2  using namespace std;
3
4  class Base {
5    private:
6      int* p { nullptr };
7    public:
8      Base() = default;
9      Base(int n = 0) { // **** Assume line #9 has NO error. ****
10       p = (n) ? new int[n] : new int;
11     }
12     ~Base() {
13       delete [] p;    // **** Assume line #13 has NO error. ****
14     }
15 };
16
17 class Derived : public Base {
18   private:
19     int d(1);
20     int* p = nullptr;
21   public:
22     Derived() : p(new int) {}
23     Derived(int n, int d) : Base(n), d(d), p(new int) {}
24     Derived(const Derived& d) = delete;  // **** Assume line #24 has NO error. ****
25     Derived& func(Derived& d) {
26       d.d = 10;
27       return *this;
28     }
29     ~Derived() { delete p; }
30 };
31
32 int main() {
33   Base b { 2 };
34   Derived d1(1, 2);
35   Derived d2(d1);
36   Derived d3(2, 4);
37   Derived d4 = d3.func(d2);
38   Base* bPtr = new Derived;
39   delete bPtr;
40   return 0;
41 }
```

Error#	Line#	Explanation
1	8	As a default constructor has been defined at line 9, getting the implicitly defined default constructor from the compiler would introduce conflict.
2	10	If n = 0, p = new int, and this causes problem for de-allocation in the destructor.
3	12	The Base's destructor is not made virtual and because of this, derived class destructor will never be called, which lead to a memory leak.
4	19	Cannot initialize d using parenthesis notation.
5	35	Cannot construct d2 using d1 as there is no default copy constructor of Derived class.
6	37	d2 can be passed to func as it takes on reference. However, the return of (*this) is used to copy construct d4, which is an error as there is no default copy constructor of Derived class.

Scheme:

0.5 point for each correct line number, 1 point per correct explanation

Problem 6 [25 points] Classes and Objects

This question is about a special implementation of linked list. Its implementation involves 3 classes, namely '`Data`', '`Node`' and '`SpecialList`'. Below are the header files of the 3 classes:

```
/* File: Data.h */

#ifndef DATA_H
#define DATA_H

#include <iostream>
using namespace std;

class Data {
  public:
    Data(int v) : value{v} {
      cout << "Construct data object " << value << endl;
    }

    ~Data() { cout << "Destroy data object " << value << endl; }

    int getValue() const { return value; }

  private:
    int value; // The value stored
};

#endif /* DATA_H */
```

```
/* File: Node.h */

#ifndef NODE_H
#define NODE_H

#include "Data.h"

class Node {
  public:
    Node(const Data& d) : data{d} {
      cout << "Construct node object " << data.getValue() << endl;
    }

    ~Node() { cout << "Destroy node object " << data.getValue() << endl; }

    Data getData() const { return data; }
    Node* getNext() const { return next; }

    void setNext(Node* next) { this->next = next; }

  private:
    Data data;                // The data associated with this node
    Node* next { nullptr };   // The pointer pointing to the next node
};

#endif /* NODE_H */
```

```
/* File: SpecialList.h */

#ifndef SPECIALLIST_H
#define SPECIALLIST_H

#include "Node.h"

class SpecialList {
  public:
    SpecialList() { cout << "Construct special list" << endl; }

    // TODO: (c)(i)  Implement the destructor
    ~SpecialList();

    // TODO: (c)(ii) Implement the copy constructor ** deep copy is required **
    SpecialList(const SpecialList& another);

    void add(const Data &data) {
      Node* newNode = new Node(data);
      if (head == nullptr) {
        head = newNode;
        newNode->setNext(newNode);
      } else {
        Node* cur = head;
        while (cur->getNext() != head)
          cur = cur->getNext();
        cur->setNext(newNode);
        newNode->setNext(head);
      }
    }

    void printHead() const {
      if (head == nullptr) return;
      cout << head->getData().getValue() << endl;
    }

  private:
    Node* head { nullptr };
};

#endif /* SPECIALLIST_H */
```

The testing program is called "`test-speciallist.cpp`", given below:

```
#include "SpecialList.h" /* File: test-speciallist.cpp */
using namespace std;

int main() {
  SpecialList* list = new SpecialList;

  Data data1(1);
  Data data2(2);
  Data data3(3);
  list->add(data1);
  list->add(data2);
  list->add(data3);
  list->printHead();

  delete list;
  return 0;
}
```

Based on the given information, complete the following questions.

(a) [5 points] Write down the outputs of the testing program up to and including Line 13.

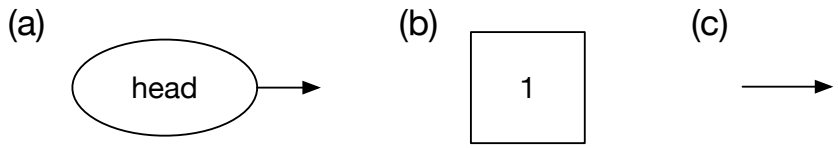
Answer:

```
Construct special list
Construct data object 1
Construct data object 2
Construct data object 3
Construct node object 1
Construct node object 2
Construct node object 3
1
Destroy data object 1
```

Scheme (5 points in total):

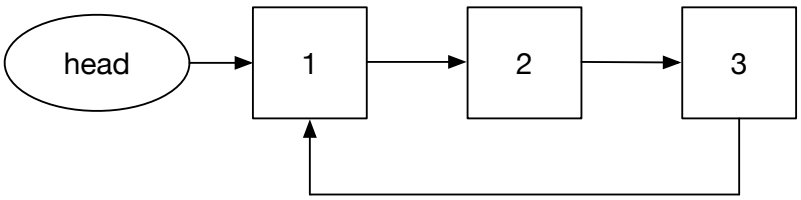
- 1 point for "Construct special list" (line 1);
- 1 point for "Construct data object X" (line 2 - 4);
- 1 point for "Construct node object X" (line 5 - 7);
- 1 point for "1" (line 8);
- 1 point for "Destroy data object 1" (line 9).

(b) [4 points] Using the symbol representation below, draw the resultant structure of the list created by the testing program.



- Use (a) to indicate the pointer called head, which points to the first node of the list.
- Use (b) to represent the node with data value 1.
- Use (c) to show the pointer pointing from one node to another node.

Answer:



Scheme (4 points in total):

- 1 point for drawing "head $\rightarrow$ 1";
- 1 point for drawing "1 $\rightarrow$ 2 $\rightarrow$ 3";
- 2 points for drawing "3 $\rightarrow$ 1".

(c) [16 points] Below is a partial implementation of class '`SpecialList`' in a separate file called "`SpecialList.cpp`". Complete the implementation of its

(i) destructor and

(ii) copy constructor (** deep copy is required **).

Note that you must use recursion to perform deep copy in the copy constructor. You may add any **non-member** helper function to achieve this. If you do not use recursion, you can at most get half of the full marks.

(i) Destructor (6 points)

Answer:

```
SpecialList::~SpecialList() {
    if (head == nullptr)
        return;
    Node* cur = head;
    while (cur->getNext() != head) {
        Node* tmp = cur->getNext();
        delete cur;
        cur = tmp;
    }
    delete cur;
}
```

Scheme (6 points in total):

- 1 point for correctly handling the case when head == nullptr.
- 1 point for the correct deletion on the first and last node.
- 1 point for the correct use of loop.
- 1 point for the correct loop condition.
- 2 points for the correct deletion of the current node and getting the next node. Be careful about the order of node deletion and getting the next pointer. If student try to get the next pointer from the deleted node, he/she can only 1 point.

(ii) Copy constructor (10 points)

Answer:

```
/************************** Alternative answer 1 **************************/

Node* duplicateList(Node* cur, Node* head, Node* dupHead) {
    if (cur == head)
        return dupHead;
    // copy cur
    Node* dupNode = new Node(cur->getData());
    dupNode->setNext(duplicateList(cur->getNext(), head, dupHead));
    return dupNode;
}

SpecialList::SpecialList(const SpecialList& another) {
    if (another.head) {
        head = new Node(another.head->getData());
        head->setNext(duplicateList(another.head->getNext(), another.head, head));
    }
}

/******************************** CODE ENDS ********************************/


/************************** Alternative answer 2 **************************/

Node* duplicateList(Node* currentHead, Node* originalHead, Node* firstNewNode) {
    Node* newNode = new Node (currentHead->getData());
    Node* nextHead = currentHead->getNext();

    if (!firstNewNode)
        firstNewNode = newNode;        // Need this for set up the last node's next

    if (nextHead == originalHead) // End of the list
        newNode->setNext(firstNewNode);
    else // Recursively create the sub-list
        newNode->setNext(duplicateList(nextHead, originalHead, firstNewNode));

    return newNode;
}

SpecialList::SpecialList(const SpecialList& another) {
    if (another.head)
        head = duplicateList(another.head, another.head, nullptr);
}

/******************************** CODE ENDS ********************************/
```

```
/************************** Alternative answer 3 **************************/

Node *duplicateList(Node *head, Node *currentHead) {
    Node *newNode = new Node(currentHead->getData());
    Node *nextHead = currentHead->getNext();

    if (nextHead == head) // There is only 1 node
        newNode->setNext(newNode);
    else                  // Recursively create the sub-list
        newNode->setNext(duplicateList(head, nextHead));

    return newNode;
}

SpecialList::SpecialList(const SpecialList &another) {
    if (another.head) {
        head = duplicateList(another.head, another.head);
        // Find the last node and point it to the actual head
        Node *node = head;
        while (node->getNext() != node)
            node = node->getNext();
        node->setNext(head);
    }
}

/******************************** CODE ENDS ********************************/
```

Scheme (10 points in total):

We provide 3 possible answers above. Answer 1 and 2 are "dedicated" solution for the circular linked list. They apply recursion to directly copy the whole list. Answer 3 takes two steps. It first treats the structure as a normal linked list, and then modifies the "next pointer" in the tail node to the first one. We also accept this version.

The general grading scheme is listed:

- 1 point for correctly handling the case when head == nullptr.
- 3 points for the correct use of help function/recursion.
- 2 points for the correct pointer from the last node to the first node.
- 2 points for the correct copy of each node.
- 2 points for the correct operation when there is only one node in the list.

If students did not use recursion in the copy constructor, they can get at most 5 points. Normally a loop should be used. The grading scheme for this situation is listed:

- 1 point for correctly handling the case when head == nullptr.
- 1 points for the correct use of the loop.
- 1 points for the correct pointer from the last node to the first node.
- 1 points for the correct copy of each node.
- 1 points for the correct when there is only one node in the list.

Problem 7 [30 points] Inheritance, Polymorphism and Dynamic Binding

This problem involves 3 classes called 'Document', 'Book' (derived from 'Document' using public inheritance), and 'Email' (also derived from 'Document' using public inheritance). Below are the header files of the 3 classes.

```
/* File: Document.h */

#ifndef DOCUMENT_H
#define DOCUMENT_H

#include <iostream>
using namespace std;

class Document {
    string* authors;         // A pointer which points to an array of strings
                             // representing the authors of the document.
    unsigned int numAuthors; // The number of authors in the array pointed by authors.

  public:
    /* ----- Functions to implement ----- */

    // Construct a document with the given array of authors and the number of authors in
    // the array. A dynamic array should be constructed and initialized according to the
    // given parameters, and the resulting dynamic array should be pointed by authors.
    // Note: The array needs to be of size numAuthors.
    Document(const string* authors, unsigned int numAuthors);

    // Copy constructor - Perform deep copy.
    Document(const Document& d);

    // Destructor - De-allocate all dynamically allocated memory to avoid any memory leak
    // as the program finishes.
    virtual ~Document();

    // Add a new author to the end of the dynamic array. A new array of size
    // (numAuthors + 1) should be created, copy all the authors in the existing array over,
    // and replace the existing array pointed by "authors" by the new array.
    // Also, the numAuthors should be increased by 1.
    // Note: You need to make sure that there is no memory leak after adding a new author.
    void addAuthor(const string& name);

    // Print all the data of document.
    virtual void print() const;
};

#endif /* DOCUMENT_H */
```

```
/* File: Book.h */

#ifndef BOOK_H
#define BOOK_H

#include "Document.h"

class Book : public Document {
    string title;          // The title of book.
    unsigned int edition;  // The edition of book.

  public:
    /* ----- Functions to implement ----- */

    // Construct a book with the given title, edition, and the list of authors.
    Book(const string& title, unsigned int edition,
         const string authors[], unsigned int numAuthors);

    // Print all the data of Book.
    void print() const;
};

#endif /* BOOK_H */

/* File: Email.h */

#ifndef EMAIL_H
#define EMAIL_H

#include "Document.h"

class Email : public Document {
    string subject;  // The subject of email.
    string toList;   // The list of email recipients.

  public:
    /* ----- Functions to implement ----- */

    // Construct an email with the given subject, array of authors (also with
    // number of authors), and a string representing all the email recipients.
    Email(const string& subject, const string* authors,
          unsigned int numAuthors, const string& toList);

    // Copy constructor - Perform deep copy.
    Email(const Email& e);

    // Add the new recipient by concatenating it to the end of the existing list.
    // Hint: Operator + can be used to concatenate two strings.
    void addTo(const string& name);

    // Print all the data of email.
    void print() const;
};

#endif /* EMAIL_H */
```

Your task is to implement all the missing member functions in the three given classes 'Document', Book, and 'Email'. The following is the test program "test-document.cpp" for the 3 classes.

```
/* File: test-document.cpp */

#include <iostream>
#include <typeinfo>
#include "Document.h"
#include "Book.h"
#include "Email.h"
using namespace std;

int main() {
  Document* documents[2];

  /* ----- Construct a Book ----- */
  string bookAuthors[] = { "Paul J. Deitel", "Harvey Deitel" };
  documents[0] = new Book("C++ How to Program", 10, bookAuthors, 2);

  /* ----- Construct an Email ----- */
  string emailAuthors[] = { "Brian", "Gary", "Desmond" };
  Email* email = new Email("COMP 2012 Midterm Exam", emailAuthors, 3, "Wallace");
  email->addTo("Kevin");
  email->addTo("Yingke");
  documents[1] = email;

  /* ----- Print documents ----- */
  for(int i=0; i<2; i++) {
    cout << ((typeid(*documents[i]) == typeid(Book)) ? "Book\n====" : "Email\n=====") << endl;
    documents[i]->print();
    cout << endl;
  }

  /* ----- De-allocate documents ----- */
  for(int i=0; i<2; i++)
    delete documents[i];

  return 0;
}
```

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

```
Book
====
Title: C++ How to Program
Edition: 10
Authors: Paul J. Deitel, Harvey Deitel


Email
=====
Subject: COMP 2012 Midterm Exam
To: Wallace, Kevin, Yingke
Authors: Brian, Gary, Desmond
```

(a) [16 points] Implement the following missing member functions of the class 'Document' in a separate file called "Document.cpp".

- `Document(const string* authors, unsigned int numAuthors)`
- `Document(const Document& d)`
- `~Document()`
- `void addAuthor(const string& name)`
- `void print() const`

Answer: `/* File "Document.cpp" */`

```
#include "Document.h"

Document::Document(const string* authors, unsigned int numAuthors) { // 0.5 point
  this->authors = nullptr;                                            // 0.5 point
  this->numAuthors = 0;                                               // 0.5 point
  for(int i=0; i<numAuthors; i++)                                     // 0.5 point
    addAuthor(authors[i]);                                            // 1 point
}

Document::Document(const Document& d) {                              // 0.5 point
  numAuthors = d.numAuthors;                                          // 0.5 point
  authors = new string[d.numAuthors];                                 // 1 point
  for(int i=0; i<d.numAuthors; ++i)                                   // 0.5 point
    authors[i] = d.authors[i];                                        // 0.5 point
}

Document::~Document() {                                               // 0.5 point
  delete [] authors;                                                  // 1 point
}

void Document::addAuthor(const string& name) {                        // 0.5 point
  string* authorsNew = new string[numAuthors + 1];                    // 1 point
  for(int i=0; i<numAuthors; ++i)                                     // 0.5 point
    authorsNew[i] = authors[i];                                       // 0.5 point
  authorsNew[numAuthors] = name;                                      // 1 point
  delete [] authors;                                                  // 1 point
  authors = authorsNew;                                               // 0.5 point
  ++numAuthors;                                                       // 0.5 point
}

void Document::print() const {                                        // 0.5 point
  cout << "Authors: ";                                                // 0.5 point
  for(unsigned int i=0; i < numAuthors; ++i) {                        // 0.5 point
    cout << authors[i];                                               // 0.5 point
    if(i != numAuthors-1)                                             // 0.5 point
      cout << ", ";                                                   // 0.5 point
  }
  cout << endl;
}
```

(b) [5 points] Implement the following missing member functions of the class 'Book' in a separate file called "Book.cpp".

- `Book(const string& title, unsigned int edition, const string authors[], unsigned int numAuthors)`
- `void print() const`

Answer:

```
/* File "Book.cpp" */
#include "Book.h"

Book::Book(const string& title, unsigned int edition,
           const string authors[], unsigned int numAuthors)          // 0.5 point
           : Document(authors, numAuthors),                          // 1 point
             title(title), edition(edition) {                        // 1 point
}

void Book::print() const {                                           // 0.5 point
  cout << "Title: " << title << endl;                                // 0.5 point
  cout << "Edition: " << edition << endl;                            // 0.5 point
  Document::print();                                                 // 1 point
}
```

(c) [9 points] Implement the following missing member functions of the class 'Email' in a separate file called "Email.cpp".

- `Email(const string& subject, const string* authors, unsigned int numAuthors, const string& toList)`
- `Email(const Email& e)`
- `void addTo(const string& name)`
- `void print() const`

Answer:

```
/* File "Email.cpp" */
#include "Email.h"

Email::Email(const string& subject, const string* authors,
             unsigned int numAuthors, const string& toList)          // 0.5 point
             : Document(authors, numAuthors),                        // 1 point
               subject { subject },                                  // 0.5 point
               toList { toList }                                     // 0.5 point
{ }

Email::Email(const Email& e)                                         // 0.5 point
      : Document(e) {                                                // 1 point
  subject = e.subject;                                               // 0.5 point
  toList = e.toList;                                                 // 0.5 point
}

void Email::addTo(const string& name) {                              // 0.5 point
  toList += (", " + name);                                           // 1 point
}

void Email::print() const {                                          // 0.5 point
  cout << "Subject: " << subject << endl;                            // 0.5 point
  cout << "To: " << toList << endl;                                  // 0.5 point
  Document::print();                                                 // 1 point
}
```

---------- END OF PAPER ----------