COMP 2012 Midterm Exam - Fall 2021 - HKUST

Date: October 23, 2021 (Saturday)
Time Allowed: 2 hours, 10:00–12:00 nn
Instructions:

1. This is a closed-book, closed-notes examination.
2. There are **6** questions on **31** pages (including this cover page and 2 blank pages at the end).
3. Write your answers in the space provided.
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, nor any library functions not mentioned in the questions.

Student Name	SOLUTIONS AND MARKING SCHEME
Student ID	
Email Address	
Venue and Seat Number	

For T.A. Use Only

Problem	Topic	Score
1	True/False Questions	/ 10
2	Const-ness and Lambda Expressions	/ 16
3	Order of Constructions and Destructions	/ 15
4	Class and Object	/ 8
5	Inheritance	/ 12
6	Classes and Objects	/ 39
	Total	/ 100

Problem 1 [10 points] True/False Questions

Indicate whether the following statements are true or false by circling **T** or **F**. You get 1.0 point for each correct answer.

T **F** (a) The following program has NO compilation error.

```
int main() {
  const int& a = 50;
  int b = a;
}
```

T **F** (b) The following program will NOT cause compilation error NOR runtime error for certain.

```
int& f() {
  int x = 1;
  return x;
}

int main() {
  int& x = f();
  x = 2;
}
```

T **F** (c) The following program has NO compilation error.

```
int main() {
  int const* const p = new const int;
  // ...
  delete p;
  return 0;
}
```

T **F** (d) If NO copy consturctor is defined in a class, the compiler will automatically supply a default copy constructor for it and it performs memberwise assignment of each data member.

T **F** (e) The following program has NO compilation error.

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

class Point {
  int x, y;

  public:
    Point(int i = 0, int j = 0) {
      x = i; y = j;
    }
    int getX() const { return x; }
    int getY() { return y; }
};

int main() {
  const Point t;
  cout << t.getX() << " ";
  cout << t.getY();
  return 0;
}
```

T **F** (f) The output of the following program is `BA`.

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

class A {
  public:
    A(int a) { cout << "A"; }
};

class B {
  public:
    B(int b) { cout << "B"; }
};

class C {
  B bObj;
  A aObj;
  public:
    C(int a, int b) : aObj(a), bObj(b) {}
};

int main() {
  C cObj(1, 2);
  return 0;
}
```

T **F** (g) Assume the program in `test.cpp` is compiled using the command:
`g++ -fno-elide-constructors Test.cpp`
which disables the return value optimization (RVO). The output of the program is `copy ctor`.

```
#include <iostream> /* File: test.cpp */
using namespace std;

class Test {
  int data;

  public:
    Test(int data) : data(data) {
      cout << "conversion ctor" << endl;
    }
    Test(const Test& t) : data(t.data) {
      cout << "copy ctor" << endl;
    }
};

int main() {
  Test first_test(Test(2));
  return 0;
}
```

T **F** (h) The following program has NO compilation error.

```
class A {
  public:
    A() {}
};

class B : public A {
  public:
    B() {}
};

B b;

int main() {
  A* a = &b;
  return 0;
}
```

T **F** (i) The output of the following program is `BAC`.

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

class A {
  public:
    A() { cout << "A"; }
};

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

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

int main() {
  C* c = new C;
  delete c;
  return 0;
}
```

T **F** (j) All the statements ending with the comment `/* <-- */` cause slicing.

```
class A {
  int a;
};

class B : public A {
  int b;
};

int main() {
  A aObj;
  B bObj;
  aObj = bObj;   /* <-- */
  A* aP = &bObj; /* <-- */
  A& aR = bObj;  /* <-- */
  return 0;
}
```

Problem 2 [16 points] Const-ness and Lambda Expressions

In the following program, for the 16 statements ending with or below the following comment:

```
/* Error/Warning:   Yes   /   No   */
```

decide whether the statement is syntatically INCORRECT – that is, it will produce compilation error(s) or warning(s). Circle "Yes" if it will give compilation error or warning and "No" otherwise.

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

int main()
{
  const int* ptr1 = new int;

  int* const ptr2 = new int;

  const int val1 = 4;

  int val2 = 5;

  ptr1 = &val1;                    /* Error/Warning:    Yes    /    No    */

  ptr2 = &val1;                    /* Error/Warning:    Yes    /    No    */

  *ptr1 = 10;                      /* Error/Warning:    Yes    /    No    */

  *ptr2 = 11;                      /* Error/Warning:    Yes    /    No    */

  const int** ptr3 = new const int*;

  const int** ptr4 = new const int*;

  *ptr3 = *ptr4;                   /* Error/Warning:    Yes    /    No    */

  **ptr3 = 2;                      /* Error/Warning:    Yes    /    No    */

  int& ref1 = val1;                /* Error/Warning:    Yes    /    No    */

  const int& ref2 = val2;          /* Error/Warning:    Yes    /    No    */
```

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

  for(int i = 0; i < 4; ++i)
  {
     /* Error/Warning:     Yes     /     No     */

     cout << [=](int x) { ref1 = ref1 * x; return ref1; }(i) << endl;
     // -------------------------------------------------------------------------

     /* Error/Warning:     Yes     /     No     */

     cout << [=](int x) { ref2 = ref2 * x; return ref2; }(i) << endl;
     // -------------------------------------------------------------------------

     /* Error/Warning:     Yes     /     No     */

     cout << [&](int x) { ref1 = ref1 * x; return ref1; }(i) << endl;
     // -------------------------------------------------------------------------

     /* Error/Warning:     Yes     /     No     */

     cout << [&](int x) { ref2 = ref2 * x; return ref2; }(i) << endl;
     // -------------------------------------------------------------------------

     /* Error/Warning:     Yes     /     No     */

     cout << [&ref1, ref1](int x) { ref1 = ref1 * x; return ref1; }(i) << endl;
     // -------------------------------------------------------------------------

     /* Error/Warning:     Yes     /     No     */

     cout << [&ref1, ref2](int x) { ref1 = ref2 * x; return ref2; }(i) << endl;
     // -------------------------------------------------------------------------

     /* Error/Warning:     Yes     /     No     */

     cout << [=, &ref1](int x) { ref1 = ref1 * x; return ref1; }(i) << endl;
     // -------------------------------------------------------------------------

     /* Error/Warning:     Yes     /     No     */

     cout << [&, &ref1](int x) { ref1 = ref1 * x; return ref1; }(i) << endl;
  }

  return 0;
}
```

Answer:

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

int main() {
  const int* ptr1 = new int;
  int* const ptr2 = new int;
  const int val1 = 4;
  int val2 = 5;

  ptr1 = &val1;                          /* Error/Warning:     No     */
  ptr2 = &val1;                          /* Error/Warning:     Yes    */
  *ptr1 = 10;                            /* Error/Warning:     Yes    */
  *ptr2 = 11;                            /* Error/Warning:     No     */

  const int** ptr3 = new const int*;
  const int** ptr4 = new const int*;

  *ptr3 = *ptr4;                         /* Error/Warning:     No     */
  **ptr3 = 2;                            /* Error/Warning:     Yes    */
  int& ref1 = val1;                      /* Error/Warning:     Yes    */
  const int& ref2 = val2;                /* Error/Warning:     No     */

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

  for(int i = 0; i < 4; ++i) {
     /* Error/Warning:     Yes    */
     cout << [=](int x) { ref1 = ref1 * x; return ref1; }(i) << endl;
     /* Error/Warning:     Yes    */
     cout << [=](int x) { ref2 = ref2 * x; return ref2; }(i) << endl;
     /* Error/Warning:     No     */
     cout << [&](int x) { ref1 = ref1 * x; return ref1; }(i) << endl;
     /* Error/Warning:     Yes    */
     cout << [&](int x) { ref2 = ref2 * x; return ref2; }(i) << endl;
     /* Error/Warning:     Yes    */
     cout << [&ref1, ref1](int x) { ref1 = ref1 * x; return ref1; }(i) << endl;
     /* Error/Warning:     No     */
     cout << [&ref1, ref2](int x) { ref1 = ref2 * x; return ref2; }(i) << endl;
     /* Error/Warning:     No     */
     cout << [=, &ref1](int x) { ref1 = ref1 * x; return ref1; }(i) << endl;
     /* Error/Warning:     Yes    */
     cout << [&, &ref1](int x) { ref1 = ref1 * x; return ref1; }(i) << endl;
  }

  return 0;
}
```

Marking scheme:

- 1 point for choosing the correct answer of each question. 16 points in total.

Problem 3 [15 points] Order of Constructions and Destructions

Given the following program:

```
#include <iostream>  /* File: order-constructions-destructions.cpp */
using namespace std;

class X {
  public:
    X()           { cout << "X's def ctor\n";  }
    X(int a)      { cout << "X's conv ctor\n"; }
    X(const X& x) { cout << "X's copy ctor\n"; }
    ~X()          { cout << "X's dtor\n";      }
};

class Y {
    X x_var;
  public:
    Y(): x_var(10) { cout << "Y's def ctor\n";  }
    Y(X x)         { cout << "Y's conv ctor\n"; }
    Y(const Y& y)  { cout << "Y's copy ctor\n"; }
    ~Y()           { cout << "Y's dtor\n";      }
};

class Z {
    X x;
    Y y1, y2;
  public:
    Z() { cout << "Z's def ctor\n"; }
    Z(int q): y2(y1) {
      cout << "----- begin: temp object -----" << endl;
      y1 = Y(q);
      cout << "----- end: temp object -----" << endl;
      cout << "Z's conv ctor\n";
    }
    ~Z() { cout << "Z's dtor\n"; }
};

int main(){
  cout << "----- Making X -----" << endl;
  X test_x;
  cout << "----- Making Y -----" << endl;
  cout << "----- begin: temp object -----" << endl;
  Y test_y(10);
  cout << "----- end: temp object -----" << endl;
  cout << "----- Making Z -----" << endl;
  Z test_z1;
  Z test_z2(10);
  cout << "----- end -----" <<endl;
}
```

Assume the program is compiled using the command:
`g++ -fno-elide-constructors order-constructions-destructions.cpp`
which disables the return value optimization (RVO). Write down the output of the program.

Answer:

Answer:

```
----- Making X -----
X's def ctor
----- Making Y -----
----- begin: temp object -----
X's conv ctor
X's copy ctor
X's def ctor
Y's conv ctor
X's dtor
X's dtor
----- end: temp object -----
----- Making Z -----
X's def ctor
X's conv ctor
Y's def ctor
X's conv ctor
Y's def ctor
Z's def ctor
X's def ctor
X's conv ctor
Y's def ctor
X's def ctor
Y's copy ctor
----- begin: temp object -----
X's conv ctor
X's copy ctor
X's def ctor
Y's conv ctor
Y's dtor
X's dtor
X's dtor
X's dtor
----- end: temp object -----
Z's conv ctor
----- end -----
Z's dtor
Y's dtor
X's dtor
```

```
Y's dtor
X's dtor
X's dtor
Z's dtor
Y's dtor
X's dtor
Y's dtor
X's dtor
X's dtor
Y's dtor
X's dtor
X's dtor
```

Marking scheme:

This question is marked in sections distributed as following:

```
Section1: "----- Making X -----" to "----- Making Y -----" inclusive
Section2: "----- begin: temp object -----" to "----- end: temp object-----" inclusive
Section3: "----- Making Z -----" to "Y's copy ctor" inclusive
Section4: "----- Begin: temp object -----" to "----- end: temp object-----" inclusive
Section5: "Z's conv ctor" to "----- end -----" inclusive
Section6: "Z's dtor" to "X's dtor" inclusive
```

- Within each section, the **sequence with longest** correct statement count is selected for marking and each statement in the sequence is worth **0.3** marks.
- In sections with temporary object creation (Section 2 and Section 4), **immediate** destruction of temporary object is accepted with **no penalty**.
- **No penalty** is given for extra lines of code beyond the accepted sequence of statements.

Problem 4 [8 points] Class and Object

(a) [6 points] The following header file contains **3 errors** (syntax/logical error) in **3 different lines**. Identify each error by writing down the **line number** where it occurs, and **suggest how to correct it** in the given table.

```
1  #ifndef QUEUE_H  /* File: queue.h */
2  #define QUEUE_H
3
4  #include <iostream>
5  using namespace std;
6
7  class Queue
8  {
9    private:
10     int id;
11     int num_elements;
12
13   public:
14     Queue(int id) : num_elements(0) {
15       id = id;
16     }
17     void enqueue() const {
18       ++num_elements;
19     }
20     void dequeue() {
21       if(num_elements > 0)
22         --num_elements;
23     }
24     void concatenate(Queue& q) {
25       num_elements += q->num_elements;
26     }
27     void print_info() const {
28       cout << "#" << id << " queue has " << num_elements << " elements" << endl;
29     }
30 };
31
32 #endif
```

```
#include "queue.h"  /* File: main.cpp */
#include "queue.h"

int main() {
  Queue a(1), b(2);
  a.enqueue();
  a.enqueue();
  b.enqueue();
  a.print_info();
  b.print_info();
  b.concatenate(&a);
  a.print_info();
  b.print_info();
  return 0;
}
```

After fixing the 3 errors, the testing program '`main.cpp`' is expected to produce the following output.

```
#1 queue has 2 elements
#2 queue has 1 elements
#1 queue has 2 elements
#2 queue has 3 elements
```

Error #	Line #	Correction
1		
2		
3		

(b) [2 points] If we remove lines 1, 2 and 32 in '`queue.h`' (given in part (a)), will we be able to successfully compile '`main.cpp`'? Explain why.

Answer:

(a)

Error #	Line #	Correction
1	15	It should be "`this->id = id;`".
2	17	const keyword should be removed, "`void enqueue()`".
3	24	It should be "`void concatenate(Queue* q)`".

(b) No, a compilation will occur if we remove Line 1, 2 and 32 in queue.h. Since we include queue.h twice in main.cpp and so re-definition error occurs.

Marking scheme:

Part (a)

- 1 point for giving the correct line number. 3 points in total.
- 1 point for correcting each error. 3 points in total.

Part (b)

- 1 point for stating compilation error will occur.
- 1 point for giving the correct explanation.

Problem 5 [12 points] Inheritance

A student feels confident to tell the following file, '`dish.h`', contains exactly **4 errors**.

```
1  #include <iostream>  // File: dish.h
2  using namespace std;
3
4  class Dish {
5    public:
6      Dish(string ingredient) : ingredient{ingredient} {}
7      void cook();
8      void fusion(Drink* drink);
9
10   private:
11     string ingredient;
12 };
13
14 class Pasta : public Dish {
15   public:
16     Pasta(string ingredient, string sauce) : ingredient(ingredient), sauce(sauce) {}
17     void cook();
18
19     string ingredient;
20     string sauce;
21 };
22
23 class Drink : public Dish {
24   public:
25     inline Drink(string ingredient) : Dish(ingredient) {}
26
27   private:
28     void cook() override;
29 };
30
31 inline void Dish::cook() {
32   cout << "Add MSG to give " << ingredient << " some flavor." << endl;
33 }
34
35 inline void Dish::fusion(Drink* drink) {
36   cook(); cout << "Pour some " << drink->Dish::ingredient << " over it." << endl;
37 }
38
39 inline void Pasta::cook() {
40   cout << "Mix the " << ingredient << " and " << sauce << " well." << endl;
41 }
42
43 inline void Drink::cook() {
44   cout << "Put some " << ingredient << " in a cup of water." << endl;
45 }
```

(a) [8 points] Consider '`dish.h`' alone. A student claims it is possible to locate **4 errors** that will *definitely* prevent a source file which includes '`dish.h`' from being compiled. To help him justify his claim, identify the 4 errors by writing down the **line numbers** where they occur and **explaining** what they are in the following table.

Error #	Line #	Explanation
1		
2		
3		
4		

Answer:

Error #	Line #	Explanation
1	8	A forward declaration of `Drink` is needed for `Dish::fusion(Drink*)`.
2	16	The default constructor of `Dish` is not available / The `Pasta` class must call the converting constructor of the `Dish` class explicitly.
3	28	The `override` keyword cannot be used with a non-virtual function.
3 (Alernative)	7	The `Dish::cook` should be virtual so that it can be overridden by `Drink::cook()`.
4	44	`Dish::ingredient` is private and hence not accessible to a different class `Drink`.

(b) [4 points] Suppose the student has fixed all errors in '`dish.h`' that will *definitely* cause compilation errors. He goes on to write the following source file.

```
1  #include "dish.h"  // File: main.cpp
2
3  int main()
4  {
5    Dish* dish = new Pasta("macaroni", "cheese");
6    static_cast<Pasta*>(dish)->ingredient = "spaghetti";
7    dish->cook();
8    (*static_cast<Pasta*>(dish)).cook();
9    delete dish;
10
11   return 0;
12 }
```

(i) Will the program compile? If so, write down its output. Otherwise, explain why.

Answer: The program will / won't compile. *(Circle the right word.)*

Output/Explanation:

Answer: The program **will** / won't compile. *(Circle the right word.)*

Output/Explanation:

```
Add MSG to give macaroni some flavor.
Mix the spaghetti and cheese well.
```

Alternative answer:

```
Mix the spaghetti and cheese well.
Mix the spaghetti and cheese well.
```

(ii) Will the program have potential runtime issues? If your answer is yes, provide an explanation.

The program will / won't have potential runtime issues. *(Circle the right word.)*

Explanation (if your answer is yes)

Answer: The program **will** / won't have potential runtime issues. *(Circle the right word.)*

Explanation (if any)
Destructing a derived class (Pasta) object using its base class's (Dish's) non-virtual destructor results in undefined behavior.

Marking Scheme:

Part (a)

- 1 point for giving each correct line number. 4 points in total.
- 1 point for giving each correct explanation. 4 points in total.

Part (b)(i)

- 1 point for "will compile".
- 1 point for giving the correct program output.

Part (b)(ii)

- 1 point for "will have potential runtime issues".
- 1 point for giving the correct explanation.

Problem 6 [39 points] Classes and Objects

This problem involves 2 global functions called "copy" and "equal", and 2 classes called "Movie" and "Collection" in 3 different header files. Based on the information given in the header files, complete part (a), (b) and (c).

```
/* File: global.h */

// Copy the C-string from src (pointer to a const char array) to dst (pointer to
// a char array) you can assume both given arrays have enough sizes to hold the
// C-string content including the null character you can also assume src is a
// proper C-string
void copy(const char* src, char* dst);

// Check if the given C-strings s1 (pointer to a const char array) and s2 (pointer
// to a const char array) have the same string content you can assume both inputs
// are proper C-strings
bool equal(const char* s1, const char* s2);
```

(a) [5 points] According to the information given in '`global.h`', implement the two global functions, "copy" and "equal" in '`global.cpp`'.
Note: You CANNOT use any C-string manipulation functions, e.g., strlen, strcpy, strcmp, etc.. You should include the function headers.

```
// File: global.cpp
// Define and implement the copy function below.
```

Answer:

```
#include "global.h"

void copy(const char* src, char* dst) {
    int i=0;
    do {
        dst[i] = src[i];
    }while(src[i++]);
}


bool equal(const char* s1, const char* s2) {
    int i = 0;
    while(true) {
        if(s1[i] != s2[i])
            return false;
        if(s1[i] == '\0') //both strings reached the end
            return true;
        i++;
    }
}
```

Marking scheme:

copy:

- the loop: 1 mark
- copying the characters (must copy also the null character \0 at the end to dst): 1 mark

Note: sizeof won't be able to give you the length of the C-string because the parameter is a pointer

equal:

- the loop: 1 mark
- return false for the first non-equal character: 1 mark
- return true if we reach the end of the strings: 1 mark

Note: a very common mistake is that the student solution will return incorrectly true for equal("ab", "abc") when the comparison of characters stops prematurely at the \0 in one of the two strings

```
/* File: movie.h */

// Movie is a movie which simply has a movie name
class Movie
{
  public:
    Movie(const char* name); // Construct the Movie object with the given name

    const char* getName() const; // Return the movie name

  private:
    char name[256]; // Movie name
    // Assume the movie name has no more than 256 characters (including the null
    // character) you can also assume all movies names we use in our code
    // (e.g. in main.cpp) won't exceed that limit as well
};
```

(b) [2 points] According to the information given in '`movie.h`', implement the two member functions, the constructor of Movie class and "getName", in '`movie.cpp`'.
Note: You can assume the functions in part (a) are already correctly implemented and usable, but you CANNOT use any C-string manipulation functions, e.g., strlen, strcpy, strcmp, etc.. You should include the function headers.

```
// File: movie.cpp
// Include the required header files here




// Define and implement the constructor of Movie class here




// Define and implement the getName member function here
```

Answer:

```
#include "movie.h"
#include "global.h"

//part (b) code goes below

Movie::Movie(const char* name) {
    copy(name, this->name);
}

const char* Movie::getName() const {
    return name;
}
```

Marking scheme:

- including both header files: 0.5 mark
- correct function headers for both functions: 0.5 mark
- correct constructor implementation: 0.5 mark
- correct getName implementation: 0.5 mark

```
/* File: collection.h */
#include <iostream>
#include "global.h"
#include "movie.h"

using std::cout;
using std::endl;

// Collection is a collection of movies
class Collection {
    Movie** movies; // Pointer that points to a dynamic array of Movie pointers which
                    // point to dynamic Movie objects. The size of the array MUST be
                    // just big enough to hold the actual number of Movie objects

    int size; // The exact size of the movies array

  public:
    Collection(); // Default constructor: set movies to nullptr and size to 0
    ~Collection(); // Destructor

    bool hasMovie(const char* name); // Return true if there is a movie of the given
                                     // name in the movies array. Return false otherwise

    bool add(const char* name); // Try to add a new movie of the given name
                                // If there is already a movie of the same name in the
                                // collection, do nothing and return false. otherwise,
                                // enlarge the movies array, add the new movie to the
                                // beginning of it, and return true. Refer to the sample
                                // output to see how the array is arranged after a new
                                // movie is added

    bool remove(const char* name); // Try to remove a movie of the given name
                                   // If there is no movie of the given name in the
                                   // collection, do nothing and return false.
                                   // Otherwise, remove the movie from the array
                                   // (you need to shrink the array) while keeping the
                                   // relative ordering of all other items, and return
                                   // true. Refer to the sample output to see how the
                                   // array is arranged after a movie is removed
                                   // Note: if the collection becomes empty after the
                                   // removal, set the movies to nullptr

    void print() const { // Print the movie collection; given
      if(size == 0) cout << "No collection. :(" << endl;
      cout << "Collection of " << size << ": ";
      for(int i=0; i<size; i++)
        cout << movies[i]->getName() << (size-1==i?"\n":", ");
    }
};
```

```cpp
/* File: main.cpp */

#include <iostream>
#include "collection.h"

using namespace std;

int main()
{
  char names[][256] = { "Good Will Hunting", "The Shawshank Redemption",
                        "The Godfather", "The Dark Knight" };
  cout << boolalpha; // To print true/false instead of 1/0
  Collection collection;
  for(int i=0; i<3; i++)
  {
    cout << "Add " << names[i] << ": " << collection.add(names[i]) << endl;
    collection.print();
  }
  cout << "====================================" << endl;

  for(int i=0; i<4; i++)
  {
    cout << "Add " << names[i] << ": " << collection.add(names[i]) << endl;
    collection.print();
  }
  cout << "====================================" << endl;

  cout << "Remove " << names[2] << ": " << collection.remove(names[2]) << endl;
  collection.print();
  cout << "====================================" << endl;

  cout << "Remove " << names[2] << ": " << collection.remove(names[2]) << endl;
  collection.print();
  cout << "====================================" << endl;

  for(int i=0; i<4; i++)
    cout << "Has " << names[i] << "? " << collection.hasMovie(names[i]) << endl;
  cout << "====================================" << endl;

  cout << "Be reminded to avoid any memory leak :)" << endl;

  return 0;
}
```

Expected output

```
Add Good Will Hunting: true
Collection of 1: Good Will Hunting
Add The Shawshank Redemption: true
Collection of 2: The Shawshank Redemption, Good Will Hunting
Add The Godfather: true
Collection of 3: The Godfather, The Shawshank Redemption, Good Will Hunting
====================================
Add Good Will Hunting: false
Collection of 3: The Godfather, The Shawshank Redemption, Good Will Hunting
Add The Shawshank Redemption: false
Collection of 3: The Godfather, The Shawshank Redemption, Good Will Hunting
Add The Godfather: false
Collection of 3: The Godfather, The Shawshank Redemption, Good Will Hunting
Add The Dark Knight: true
Collection of 4: The Dark Knight, The Godfather, The Shawshank Redemption, Good Will Hunting
====================================
Remove The Godfather: true
Collection of 3: The Dark Knight, The Shawshank Redemption, Good Will Hunting
====================================
Remove The Godfather: false
Collection of 3: The Dark Knight, The Shawshank Redemption, Good Will Hunting
====================================
Has Good Will Hunting? true
Has The Shawshank Redemption? true
Has The Godfather? false
Has The Dark Knight? true
====================================
Be reminded to avoid any memory leak :)
```

(c) [32 points] According to the information given in collection.h, implement the 5 member functions, constructor and destructor of the Collection class, "hasMovie", "add", and "remove", in '`collection.cpp`'.

Note: You can assume the functions in part (a) are already correctly implemented and usable, but you CANNOT use any C-string manipulation functions, e.g., strlen, strcpy, strcmp, etc.. You should include the function headers. Also, you should implement all the required member functions so that they will work with the testing program '`main.cpp`' to produce the expected output.

```
// File: collection.cpp
// Include the required header files here




// Define and implement the constructor of the Collection class here




// Define and implement the destructor of the Collection class here




// Define and implement the hasMovie member function here
```

```
// File: collection.cpp
// Continued:

// Define and implement the remove member function here
```

Answer:

```
#include "collection.h"
#include "global.h"

Collection::Collection() : movies(nullptr), size(0)
{
}

Collection::~Collection()
{
    for(int i=0; i<size; i++)
        delete movies[i];
    delete [] movies;
}

bool Collection::hasMovie(const char* name)
{
    for(int i=0; i<size; i++)
        if(equal(name, movies[i]->getName()))
            return true;
    return false;
}

bool Collection::add(const char* name)
{
    if(hasMovie(name))
        return false;

    Movie** newMovies = new Movie*[size+1];
    for(int i=0; i<size; i++)
        newMovies[i+1] = movies[i];

    newMovies[0] = new Movie(name);

    delete [] movies;
    movies = newMovies;
    size++;

    return true;
}
```

```
bool Collection::remove(const char* name)
{
    if(!hasMovie(name))
        return false;

    Movie** newMovies = new Movie*[size-1];
    for(int i=0, j=0; i<size; i++)
    {
        if(!equal(movies[i]->getName(), name))
            newMovies[j++] = movies[i];
        else
            delete movies[i];
    }

    delete [] movies;
    movies = newMovies;
    size--;

    if(size == 0) // Note: it is not efficient to check it at the end,
                  // yet it is simple and most students probably do this
    {
        delete [] movies;
        movies = nullptr;
    }

    return true;
}
```

Marking scheme:

including "collection.h": 1 mark

constructor:

- correct function header: 1 mark
- initialize movies: 1 mark
- initialize size: 1 mark

destructor:

- correct function header: 1 mark
- `delete movies[i]`: 1 mark
- `delete [] movies`: 1 mark

hasMovie:

- correct function header:1 mark
- correct use of `equal(name, movies[i]->getName())`: 1 mark
- return true if a match is found: 1 mark
- return false at the end (no match is found): 1 mark

add:

- correct function header: 1 mark
- return false if hasMovie: 1 mark
- `new Movie*[size+1]`: 1 mark
- properly shift/move existing movies: 1 mark
- `newMovies[0] = new Movie(name)`: 1 mark
- `delete [] movies`: 1 mark
- `movies = newMovies`: 1 mark
- `size++`: 1 mark
- `return true`: 1 mark

remove:

- correct function header: 1 mark
- return false if movie doesn't exist: 1 mark
- `new Movie*[size-1]`: 1 mark
- correct use of `!equal(movies[i]->getName(), name)`: 1 mark
- shift/move movies if it is not the movie to be removed: 1 mark
- delete the movie if it is the one to be removed: 1 mark
- `delete [] movies`: 1 mark
- `movies = newMovies`: 1 mark
- `size--`: 1 mark
- `delete [] movies` if size is 0: 1 mark
- set movies to nullptr if size is 0: 1 mark
- return true at the end: 1 mark

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