ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

Kỹ thuật lập trình - CO1027

Bài tập lớn 2

SHERLOCK
A STUDY IN PINK - Phần 3

TP. HỒ CHÍ MINH, THÁNG 03/2022

ĐẶC TẢ BÀI TẬP LỚN

Phiên bản 1.0

1 Chuẩn đầu ra

Sau khi hoàn thành bài tập lớn này, sinh viên ôn lại và sử dụng thành thục:

- Con trỏ
- Lập trình hướng đối tượng
- Danh sách liên kết đơn

2 Dẫn nhập

Bài tập lớn (BTL) này được phóng tác dựa trên tập 1 mùa 1 của bộ phim Sherlock của đài BBC. Bộ phim này cũng được thực hiện dựa trên cuốn tiểu thuyết Sherlock Holmes của tác giả Sir Arthur Conan Doyle.

Cuối phần 2, một tài xế taxi xuất hiện trước căn hộ số 221B đường Baker, tài xế mời Sherlock làm khách trên chuyến taxi tiếp theo. Sherlock biết rằng vị tài xế này là tên tội phạm, nhưng, Sherlock vẫn chưa thể hiểu, tại sao nạn nhân sau một chuyến taxi này lại tự tử. Sherlock đã lựa chọn tham gia chuyến xe nguy hiểm này.

Mặt khác, một thời gian sau khi Sherlock rời khỏi phòng, Watson xem lại tín hiệu định vị trên laptop. Anh thấy rằng vị trí điện thoại đang dần di chuyển khỏi căn hộ. Watson cũng bắt một chuyến xe khác đi theo vị trí định vị của điện thoại.

Trong bài tập lớn này, các bạn được yêu cầu hiện thực các lớp (class) sau:

1. **class Person**: class thể hiện cho một nhân vật, ví dụ: tên tội phạm, Sherlock, Watson.
2. **class Path**: class để lưu trữ các điểm mà một nhân vật đi qua, như vậy trong class Person sẽ có một biến kiểu Path để lưu trữ các điểm này.
3. **class Node**: class thể hiện một Node trong class Path, class Path sẽ được hiện thực dựa trên *danh sách liên kết đơn*, mỗi phần tử là một Node gồm 2 thành phần là dữ liệu cần lưu trữ, và một liên kết đến Node tiếp theo.
4. **class Point**: class thể hiện một điểm (vị trí) trong quá trình các nhân vật di chuyển. Điểm này là thành phần cần lưu trữ trong Node.

Chi tiết của các class cần hiện thực sẽ được nêu chi tiết trong các nhiệm vụ bên dưới. Chúng ta sẽ hiện thực các class theo mức độ phức tạp tăng dần, nên các nhiệm vụ bên dưới sẽ yêu cầu hiện thực các class theo thứ tự ngược lại với mô tả trên: Point, Node, Path, Person.

3 Yêu cầu

Sinh viên được yêu cầu xây dựng một chương trình giả tưởng trên ngôn ngữ C++ để mô phỏng lại quá trình giải quyết vụ án đầu tiên của Sherlock và Watson: A study in Pink, thông qua các nhiệm vụ được mô tả bên dưới. Mỗi nhiệm vụ được yêu cầu hiện thực các phương thức (methods) thuộc class tương ứng, các tham số của cho hàm này sẽ được cho trong mô tả của yêu cầu nhiệm vụ.

3.1 Yêu cầu 1: Hiện thực class Point (1.5 điểm)

Định nghĩa class được cho như sau:

```
class Point {
private:
    int x;
    int y;
public:
    Point(int x=0, int y=0);
    string toString() const;

    int distanceTo(const Point & otherPoint) const;
};
```

Trong đó: thuộc tính x, y lần lượt là vị trí theo trục Ox và Oy trong mặt phẳng toạ độ. Chúng ta không cần quan tâm đến hướng cụ thể của 2 trục trong BTL này.

1. Hiện thực phương thức khởi tạo đối tượng (Constructor):

   ```
   Point(int x=0, int y=0);
   ```

   Phương thức này gán giá trị của các tham số x, y vào các thuộc tính x, y tương ứng của class. Tham số x, y được truyền có sử dụng giá trị mặc định.
2. Hiện thực phương thức **toString**:

   ```
   string toString() const;
   ```

Phương thức này trả về chuỗi buổi diễn diễn giá trị bên trong của đối tượng hiện tại. Xem ví dụ bên dưới về cách biểu diễn chuỗi:

Ví dụ 3.1

Ví dụ về chuỗi biểu diễn cho class Point. Cho đoạn code sau:

```
1    Point p1(0, 5);
2    cout << p1.toString() << endl;
```

Kết quả in ra màn hình:

```
<Point[0,5]>
```

Ví dụ 3.2

Ví dụ về chuỗi biểu diễn cho class Point. Cho đoạn code sau:

```
1    Point p2(9, 3);
2    cout << p2.toString() << endl;
```

Kết quả in ra màn hình:

```
<Point[9,3]>
```

3. Hiện thực phương thức **distanceTo**:

```
1    int distanceTo(const Point & otherPoint) const;
```

Phương thức trả về khoảng cách Euclid từ đối tượng hiện tại đến đối tượng **otherPoint**. Khoảng cách Euclid giữa 2 điểm p1(x1,y1) và p2(x2,y2) được tính bằng công thức:

$$d = \sqrt{(x2 - x1)^2 + (y2 - y1)^2}$$

Nếu d là một số lẻ thập phân thì ta thực hiện làm tròn lên giá trị của d. Phương thức distanceTo trả về giá trị d này.

Ví dụ 3.3

Cho đoạn code sau:

```
1    Point p3(0, 1);
2    Point p4(2, 2);
3    cout << p3.distanceTo(p4) << endl;
```

Kết quả in ra màn hình:

```
3
```

Giải thích:

$$d = \sqrt{2^2 + 1^2} = \sqrt{5} \approx 2.24 \xrightarrow{\text{Làm tròn lên}} 3$$

3.2 Yêu cầu 2: Hiện thực class Node (1.5 điểm)

Định nghĩa class Node được cho như sau:

```
1  class Node {
2  private:
3      Point point;
4      Node * next;
5
6      friend class Path;
7
8  public:
9      Node(const Point & point=Point(0,0), Node * next=NULL);
10     string toString() const;
11 };
```

Trong đó:

- Thuộc tính **point**: chứa dữ liệu mà Node này lưu trữ, dữ liệu này có kiểu Point.
- Thuộc tính **next**: con trỏ chứa địa chỉ của Node tiếp theo trong Path.
- Dòng 6: khai báo class Path là friend với class Node để giúp ta có thể dễ dàng truy cập 2 thuộc tính của Node khi đang ở trong Path.

Sinh viên được yêu cầu hiện thực các phương thức sau của class Node:

1. Phương thức khởi tạo **Node**:

```
1      Node(const Point & point=Point(0,0), Node * next=NULL);
```

- Tham số đầu vào:
  - **point**: chứa giá trị của điểm cần gán vào thuộc tính *point* của đối tượng khởi tạo.
  - **next**: chứa giá trị cần gán vào thuộc tính *next* của đối tượng khởi tạo.
- Yêu cầu: Khởi tạo các thuộc tính tương ứng như mô tả trên *Tham số đầu vào*.

2. Phương thức **toString**:

```
    string toString() const;
```

- Tham số đầu vào: không.
- Kết quả trả về: Chuỗi biểu diễn giá trị bên trong của đối tượng thuộc lớp Node. Tham khảo các ví dụ bên dưới về cách biểu diễn chuỗi:

Ví dụ 3.4

Cho đoạn code sau:

```
    Node node1(Point(1, 2));
    Node node2(Point(2,3), &node1);
    cout << "node 1:" << node1.toString() << endl;
    cout << "node 2:" << node2.toString() << endl;
```

Kết quả in ra màn hình:

```
node 1:<Node[<Point[1,2]>]>
node 2:<Node[<Point[2,3]>]>
```

Chú ý: Các bạn chú ý đến chuỗi biểu diễn cho Point trong ví dụ trên, có thể sử dụng lại hàm toString của class Point trong lúc hiện thực hàm này không?

3.3 Yêu cầu 3: Hiện thực class Path (3 điểm)

Định nghĩa class Path được cho như sau:

```

class Path {
private:
    Node * head;
    Node * tail;
    int count;
    int length;

```

```
public:
    Path();
    ~Path();

    void addPoint(int x, int y);
    string toString() const;
    Point getLastPoint() const;
};
```

Trong đó:

- Thuộc tính **head**: kiểu con trỏ chứa địa chỉ của Node đầu tiên trong Path. Nếu đối tượng thuộc lớp Path chưa có Node nào thì **head** có giá trị là **NULL**.
- Thuộc tính **tail**: kiểu con trỏ chứa địa chỉ của Node cuối cùng trong Path. Nếu đối tượng thuộc lớp Path chưa có Node nào thì **tail** có giá trị là **NULL**.
- Thuộc tính **count**: lưu trữ số lượng Node hiện có trong Path.
- Thuộc tính **length**: lưu trữ chiều dài hiện tại của Path. Nếu Path không có Node nào thì **length** có giá trị có -1. Nếu Path chỉ có 1 Node thì **length** có giá trị là 0. Nếu Path có k Node với $k >= 2$, gọi dữ liệu mà k Node này đang lưu trữ là k điểm: $p_1, p_2, ..., p_k$. Khi đó **length** được tính như sau:

$$\text{length} = \lceil d(p_1, p_2) \rceil + \lceil d(p_2, p_3) \rceil + ... + \lceil d(p_{k-1}, p_k) \rceil$$

  Với:

  - $\lceil x \rceil$: là phép làm tròn x lên số nguyên gần nó nhất (bao gồm chính nó).
  - $d(p_m, p_n)$: là khoảng cách Euclid giữa 2 điểm p_m và p_n như đã nêu trong Nhiệm vụ 1.

Sinh viên được yêu cầu hiện thực các phương thức sau của class Path:

1. Phương thức khởi tạo Path:

```
    Path();
```

   - Tham số đầu vào: không.
   - Yêu cầu: khởi tạo giá trị cho các thuộc tính như mô tả ở trên.

2. Phương thức huỷ Path:

```
    ~Path();
```

- Tham số đầu vào: không.
- Yêu cầu: thu hồi tất cả vùng nhớ từng được cấp phát động (nếu có) bởi đối tượng này.

3. Phương thức **addPoint**:

```
1    void addPoint(int x, int y);
```

- Tham số đầu vào:
  - **x**: vị trí theo trục Ox của điểm cần thêm vào.
  - **y**: vị trí theo trục Oy của điểm cần thêm vào.
- Yêu cầu:
  Tạo một điểm mới có vị trí tương ứng là **x, y** và thêm nó vào cuối danh sách đang được duy trì bởi **head**, **tail** trong class Path. Lưu ý: các thuộc tính của Path đều phải cập nhật sao cho phù hợp với những gì đã mô tả ở trên.
- Kết quả trả về: không.

4. Phương thức **toString**:

```
1    string toString() const;
```

- Tham số đầu vào: không.
- Kết quả trả về: Chuỗi biểu diễn giá trị bên trong của đối tượng thuộc lớp Path. Tham khảo các ví dụ bên dưới về cách biểu diễn chuỗi.

Ví dụ 3.5

Cho đoạn code sau:

```
Path p1;
cout << p1.toString() << endl;
p1.addPoint(0, 1);
cout << p1.toString() << endl;
p1.addPoint(1, 3);
cout << p1.toString() << endl;
```

Kết quả in ra màn hình:

```
<Path[count:0,length:-1,[]]>
<Path[count:1,length:0,[<Node[<Point[0,1]>]>]]>
<Path[count:2,length:3,[<Node[<Point[0,1]>]>,<Node[<Point[1,3]>]>]]>
```

Chú ý: Chuỗi biểu diễn cho Node có thể được sử dụng lại trong hàm **toString** của class Path này không?

5. Phương thức **getLastPoint**:

```
Point getLastPoint() const;
```

- Tham số đầu vào: không.
- Kết quả trả về: trả về đối tượng Point tương ứng với dữ liệu nằm ở Node cuối cùng của Path. (Testcase sẽ đảm bảo Path luôn có ít nhất 1 Node trước khi gọi phương thức này.)

Ví dụ 3.6

Cho đoạn code sau:

```
    Path p2;
    p2.addPoint(0, 1);
    Point pt1 = p2.getLastPoint();
    cout << pt1.toString() << endl;

    p2.addPoint(5, 6);
    Point pt2 = p2.getLastPoint();
    cout << pt2.toString() << endl;
```

Kết quả in ra màn hình:

```
<Point[0,1]>
<Point[5,6]>
```

3.4 Yêu cầu 4: Hiện thực class Character (3 điểm)

Class Character biểu diễn cho một nhân vật trong BTL này. Một nhân vật được quan tâm bởi tên nhân vật và các điểm mà họ di chuyển. Chúng ta sẽ lưu trữ các điểm di chuyển này trong một đối tượng thuộc class Path. Định nghĩa class Character được cho như sau:

```
class Character {
private:
    string name;
    Path * path;

public:
    Character(const string & name="");
    ~Character();

    string getName() const;
    void setName(const string & name);

    void moveToPoint(int x, int y);
    string toString() const;
};
```

Trong đó:

- Thuộc tính **name**: chuỗi biểu diễn tên của nhân vật.

- Thuộc tính **path**: con trỏ kiểu Path, chứa địa chỉ của một đối tượng thuộc class Path. Thuộc tính này dùng để lưu trữ các điểm mà nhân vật này đi qua.

Sinh viên được yêu cầu hiện thực các phương thức sau của class Character:

1. Phương thức khởi tạo Character:

```
Character(const string & name="");
```

   - Tham số đầu vào:
     - **name**: chuỗi biểu diễn tên sẽ được gán vào thuộc tính **name** cho đối tượng.
   - Yêu cầu:
     - Khởi tạo giá trị thuộc tính **name** bằng giá trị của biến **name**.
     - Khởi tạo một đối tượng thuộc class Path (bằng cách cấp phát động) và gán giá trị của đối tượng này vào thuộc tính **path**.

2. Phương thức huỷ Character:

```
~Character();
```

   - Tham số đầu vào: không.
   - Yêu cầu: thu hồi tất cả vùng nhớ từng được cấp phát động (nếu có) bởi đối tượng Character này.

3. Phương thức **getName**:

```
string getName() const;
```

   - Tham số đầu vào: không.
   - Kết quả trả về: trả về chuỗi tương ứng với tên của đối tương Character đang lưu trữ.

4. Phương thức **setName**:

```
void setName(const string & name);
```

   - Tham số đầu vào:
     - **name**: chuỗi chứa tên sẽ gán cho thuộc tính **name** của đối tượng Character.
   - Yêu cầu: gán tên cho thuộc tính **name** của đối tượng Character với giá trị được truyền vào.
   - Kết quả trả về: không.

Ví dụ 3.7

Cho đoạn code sau:

```
Character chWatson("Watson");
cout << chWatson.getName() << endl;
chWatson.setName("John Watson");
cout << chWatson.getName() << endl;
```

Kết quả in ra màn hình:

```
Watson
John Watson
```

5. Phương thức **moveToPoint**

```
void moveToPoint(int x, int y);
```

- Tham số đầu vào:
  - **x**: vị trí theo trục Ox của điểm mà đối tượng Character này sẽ di chuyển tới.
  - **y**: vị trí theo trục Oy của điểm mà đối tượng Character này sẽ di chuyển tới.
- Yêu cầu: thêm một điểm mới vào cuối thuộc tính **path** với vị trí x, y tương ứng với giá trị truyền vào. Quá trình này thể hiện đối tượng Character này đã di chuyển đến 1 điểm mới và điểm này được lưu trữ ở cuối của thuộc tính **path**.
- Kết quả trả về: không.

6. Phương thức **toString**:

```
string toString() const;
```

- Tham số đầu vào: không.
- Kết quả trả về: chuỗi biểu diễn giá trị bên trong của đối tượng Character. Tham khảo ví dụ bên dưới về cách biểu diễn chuỗi.

Ví dụ 3.8

Cho đoạn code sau:

```
Character chWatson("Watson");
cout << chWatson.toString() << endl;
chWatson.moveToPoint(2, 7);
cout << chWatson.toString() << endl;
```

Kết quả in ra màn hình:

```
<Character[name:Watson,path:<Path[count:0,length:-1,[]]>]>
<Character[name:Watson,path:<Path[count:1,length:0,[<Node[<Point[2,
↪ 7]>]>]]>]>
```

3.5 Yêu cầu 5: Giải cứu Sherlock (1 điểm)

Sau khi đưa Sherlock lên Taxi, tên tội phạm chở anh qua nhiều con phố, cuối cùng dừng lại tại một trường học. Hắn tiết lộ cách thức thực hiện các vụ án trước cho Sherlock. Tên tội phạm sẽ lấy ra 2 lọ thuốc, một lọ là thuốc độc, lọ còn lại là thuốc bình thường. Nạn nhân phải chọn một lọ thuốc và tên tội phạm sẽ chọn lọ còn lại. Sherlock cho rằng, đây đơn giản chỉ là sự may mắn tình cờ. Nhưng tên tội phạm không cho là vậy, đã có 4 lần lựa chọn lọ thuốc xảy ra, và hắn đã thắng cả 4 lần. Với hắn, đây là tài năng dự đoán và thao túng nạn nhân phải chọn trúng lọ thuốc độc. Lần này, tên tội phạm mời cả Sherlock cùng tham gia trò chơi, vị thám tử luôn tự tin với khả năng suy luận logic tuyệt vời của mình.

Mặt khác, Watson cũng đang đi theo vị trí của định vị nhưng con đường này không giống với Sherlock. Watson có thể bị đi một con đường xa hơn, nếu con đường này quá xa, anh sẽ không kịp đến chỗ Sherlock. Nếu con đường này đưa anh đến một vị trí đủ gần với Sherlock, Watson có thể sử dụng sở trường bắn súng của mình và bắn vào tên tội phạm. Sherlock cũng bị tiếng bắn súng làm giật mình và không uống viên thuốc đã chọn.

Sinh viên được yêu cầu viết hàm sau để mô tả quá trình trên:

- Tên hàm: **rescueSherlock**
- Khai báo hàm:

```
bool rescueSherlock(
const Character & chMurderer,
const Character & chWatson,
```

```
4         int maxLength,
5         int maxDistance,
6         int & outDistance
7         );
8
```

- Tham số đầu vào:
  - **chMurderer**: đối tượng Character đại diện cho tên tội phạm.
  - **chWatson**: đối tượng Character đại diện cho Watson.
  - **maxLength**: chiều dài tối đa mà chiều dài path của Watson được phép vượt quá chiều dài path của tên tội phạm.
  - **maxDistance**: khoảng cách tối đa giữa vị trí của Watson và tên tội phạm.
  - **outDistance**: biến được truyền tham khảo để trả về khoảng cách giữa Watson và tên tội phạm.
- Yêu cầu: gọi l_1, l_2 lần lượt là chiều dài path của Watson và của tên tội phạm.
  - Nếu $l_1 - l_2$ <= maxLength: gán **outDistance** bằng khoảng cách giữa Watson và tên tội phạm. Nếu khoảng cách này không vượt quá **maxDistance** thì Watson bắn trúng tên tội phạm và giải cứu Sherlock thành công. Ngược lại, Watson không giải cứu được Sherlock.
  - Nếu $l_1 - l_2 >$ maxLength: gán **outDistance** bằng -1 và Watson không giải cứu được Sherlock trong trường hợp này.
- Kết quả trả về: trả về **true** nếu Watson giải cứu Sherlock thành công. Ngược lại, trả về **false.**

Chú ý: Các bạn có thể cần viết thêm các phương thức mới trong các class đã cho để hoàn thành nhiệm vụ này.

3.6 Yêu cầu 6:

Lưu ý:

- Đây là yêu cầu dành riêng và bắt buộc cho các SV thuộc Khoá học **Đồ án Kỹ thuật lập trình**, các SV thuộc Khoá học **Kỹ thuật lập trình** vui lòng bỏ qua yêu cầu này.
- Cách tính điểm đối với các SV thuộc Khoá học **Đồ án Kỹ thuật lập trình**:

$$\text{Điểm BTL} = (\text{Điểm các nhiệm vụ 1-5}) * 0.7 + (\text{Điểm nhiệm vụ 6}) * 0.3$$

SV được yêu cầu hiện thực thêm 2 phương thức sau thuộc class **Path**:

1. Phương thức **removeLast**:

```
bool removeLast();
```

- Tham số đầu vào: không.
- Yêu cầu: xoá đi Node cuối cùng của đối tượng Path, SV cần chú ý cập nhật các thuộc tính của đối tượng với giá trị phù hợp với mô tả đã nêu.
- Kết quả trả về: trả về **true** nếu có thể xoá Node cuối. Ngược lại, trả về **false**.

Ví dụ 3.9

Cho đoạn Code sau:

```
Path p1;
p1.addPoint(0, 1);
p1.addPoint(1, 2);
cout << p1.toString() << endl;
p1.addPoint(3, 5);
p1.removeLast();
cout << p1.toString() << endl;
```

Kết quả in ra màn hình:

```
<Path[count:2,length:2,[<Node[<Point[0,1]>]>,<Node[<Point[1,2]>]>]]>
<Path[count:2,length:2,[<Node[<Point[0,1]>]>,<Node[<Point[1,2]>]>]]>
```

2. Quá tải toán tử cộng (+) để có thể thực hiện phép cộng lên 2 đối tượng Path.

```
Path operator+ (const Path & other);
```

- Tham số đầu vào: (tự tìm hiểu).
- Yêu cầu: khởi tạo một đối tượng Path mới bằng cách nối 2 đối tượng Path với nhau.
- Kết quả trả về: trả về một đối tượng Path mới. Chú ý rằng đối tượng này cần có vùng nhớ riêng cho từng thuộc tính của nó.

Ví dụ 3.10

Cho đoạn Code sau:

```
Path p1;
p1.addPoint(0, 1);
p1.addPoint(1, 2);

Path p2;
p2.addPoint(3, 4);

Path p3 = p1 + p2;

cout << "p1: " << p1.toString() << endl;
cout << "p2: " << p2.toString() << endl;
cout << "p3: " << p3.toString() << endl;

Path p4;
p4.addPoint(0, 1);
p4.addPoint(1, 2);
p4.addPoint(3, 4);
cout << "p4: " << p4.toString() << endl;
```

Kết quả in ra màn hình:

```
p1: <Path[count:2,length:2,[<Node[<Point[0,1]>]>,<Node[<Point[1,2]>]>]]>
p2: <Path[count:1,length:0,[<Node[<Point[3,4]>]>]]>
p3: <Path[count:3,length:5,[<Node[<Point[0,1]>]>,<Node[<Point[1,2]>]>,<Node[<Point[3,4]>]>]]>
p4: <Path[count:3,length:5,[<Node[<Point[0,1]>]>,<Node[<Point[1,2]>]>,<Node[<Point[3,4]>]>]]>
```

3.7 Đoạn kết

Watson đến nơi kịp lúc và thực hiện cú bắn từ xa để ngăn Sherlock uống viên thuốc. Sherlock xem qua dấu vết của viên đạn, anh nhìn ra được sự chính xác cùng kinh nghiệm của một người từ chiến trường về. Sherlock nhìn về phía người cộng sự Watson đang đi tới. Anh nói cảnh sát không cần điều tra về người bắn súng nữa. Sherlock đã biết được, anh vừa được ai giúp đỡ rồi!!

Cảm ơn các bạn đã đồng hành cùng Sherlock và Watson trong Bài tập lớn học kỳ này.

Chúc các bạn làm bài tốt và hoàn thành tốt môn học!!!

4 Nộp bài

Sinh viên nộp 1 tập tin: **studyInPink3a.h** trong site "Kỹ thuật lập trình (CO1027)_HK212_ALL". Sinh viên Khoá học Đồ án Kỹ thuật lập trình nộp 1 tập tin: **studyInPink3b.h**.

Thời hạn nộp bài được công bố tại nơi nộp bài trong site nêu trên. Đến thời hạn nộp bài, đường liên kết sẽ tự động khoá nên sinh viên sẽ không thể nộp chậm. Để tránh các rủi ro có thể xảy ra vào thời điểm nộp bài, sinh viên PHẢI nộp bài trước thời hạn quy định ít nhất **một giờ**.

5 Xử lý gian lận

Bài tập lớn phải được sinh viên TỰ LÀM. Sinh viên sẽ bị coi là gian lận nếu:

- Có sự giống nhau bất thường giữa mã nguồn của các bài nộp. Trong trường hợp này, TẤT CẢ các bài nộp đều bị coi là gian lận. Do vậy sinh viên phải bảo vệ mã nguồn bài tập lớn của mình.
- Sinh viên không hiểu mã nguồn do chính mình viết, trừ những phần mã được cung cấp sẵn trong chương trình khởi tạo. Sinh viên có thể tham khảo từ bất kỳ nguồn tài liệu nào, tuy nhiên phải đảm bảo rằng mình hiểu rõ ý nghĩa của tất cả những dòng lệnh mà mình viết. Trong trường hợp không hiểu rõ mã nguồn của nơi mình tham khảo, sinh viên được đặc biệt cảnh báo là KHÔNG ĐƯỢC sử dụng mã nguồn này; thay vào đó nên sử dụng những gì đã được học để viết chương trình.
- Nộp nhầm bài của sinh viên khác trên tài khoản cá nhân của mình.

Trong trường hợp bị kết luận là gian lận, sinh viên sẽ bị điểm 0 cho toàn bộ môn học (không chỉ bài tập lớn).

KHÔNG CHẤP NHẬN BẤT KỲ GIẢI THÍCH NÀO VÀ KHÔNG CÓ BẤT KỲ NGOẠI LỆ NÀO!

Sau mỗi bài tập lớn được nộp, sẽ có một số sinh viên được gọi phỏng vấn ngẫu nhiên để chứng minh rằng bài tập lớn vừa được nộp là do chính mình làm.

6 Thay đổi so với phiên bản trước

Tài liệu

[1] A Study in Pink, Wikipedia, https://en.wikipedia.org/wiki/A_Study_in_Pink

[2] Sherlock, Season 1 - Episode 1: A Study in Pink, Netflix, https://www.netflix.com/watch/70174779?trackId=13752289.

——————————HẾT——————————