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 2

Phiên bản 1.0

TP. HỒ CHÍ MINH, THÁNG 03/2024

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:

- Hàm và lời gọi hàm.
- Các thao tác đọc/ghi tập tin.
- Con trỏ và cấp phát động.
- 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 1, 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 người 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. Sau cùng, Sherlock được đưa đến một mê cung cùng với lời thách thức rằng Sherlock có thể đuổi kịp người tài xế.

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 và đến được mê cung. Anh cùng Sherlock sẽ cùng nhau truy bắt tên tội phạm trong mê cung với nhiều cạm bẫy.

Trong bài tập lớn này, các bạn được yêu cầu hiện thực các lớp (class) để mô tả lại quá trình đuổi bắt tội phạm của Sherlock và Watson. Chi tiết của các class cần hiện thực sẽ được nêu chi tiết trong các nhiệm vụ bên dưới.

3 Các lớp trong chương trình

Bài tập lớn này sử dụng Lập trình Hướng đối tượng để mô tả quá trình mà Sherlock và Watson truy bắt tên tội phạm. Các đối tượng trong BTL được biểu diễn thông qua class và được mô tả như bên dưới.

3.1 Thành phần của bản đồ

Mê cung mà Sherlock, Watson đuổi bắt tên tội phảm được biểu diễn bởi một bản đồ có kích thước là (nr, nc). Bản độ này là một mảng 2 chiều có kích thước nr hàng và nc cột. Mỗi phần tử của bản đồ được biểu diễn bằng class **MapElement**. Bản đồ có 2 loại phần tử:

- **Path**: biểu diễn lối đi, các đối tượng có thể di chuyển trên phần tử này.
- **Wall**: biểu diễn bức tường, các đối tượng không được di chuyển trên phần tử này.
- **FakeWall**: biểu diễn một bức tường giả, tên tội phạm vì là người tạo ra mê cung nên nhận biết được tường giả, còn Sherlock bằng khả năng quan sát của mình thì có thể phát hiện được tường giả này. Đối với Watson, FakeWall sẽ bị phát hiện (và di chuyển qua được) nếu Watson có EXP lớn hơn EXP yêu cầu của FakeWall.

Bên cạnh đó, các đối tượng chỉ được di chuyển bên trong bản đồ.

Cho trước định nghĩa của enum **ElementType như sau**:

```
enum ElementType { PATH, WALL, FAKE_WALL };
```

Yêu cầu: Hiện thực class **MapElement** với các mô tả sau:

1. Thuộc tính protected tên **type** có kiểu là **ElementType** biểu diễn kiểu của thành phần bản đồ.
2. Phương thức khởi tạo (public) có 1 tham số truyền vào kiểu ElementType. Phương thức khởi tạo gán giá trị của tham số cho thuộc tính **type**.
   ```
   MapElement(ElementType in_type);
   ```
3. Phương thức hủy ảo (virtual destructor) với quyền truy cập public.
4. Phương thức **getType** (public) được định nghĩa bên trong class như bên dưới:
   ```
   virtual ElementType getType() const;
   ```

Yêu cầu: Hiện thực các lớp cụ thể (concrete class) **Path**, **Wall**, **FakeWall kế thừa** class **MapElement** với mô tả sau:

1. Class **FakeWall** có thuộc tính private tên **req_exp** biểu diễn EXP tối thiểu mà Watson cần có hơn để phát hiện ra bức tường.
2. Mỗi class có các phương thức khởi tạo (public) được khai báo như bên dưới. Phần hiện hiện thực của mỗi phương thức khởi tạo cần gọi phương thức khởi tạo của class **MapElement** và gán các giá trị phù hợp. Class **FakeWall** còn cần gán giá trị của **in_req_exp** cho **req_exp**. Class **FakeWall** có phương thức **getReqExp** trả về giá trị của thuộc tính **req_exp**. **in_req_exp** được tính bằng $(r * 257 + c * 139 + 89)\%900 + 1$ với r và c lần lượt là vị trí theo hàng và cột của **FakeWall**.

```
// Constructor of class Path
Path();
// Constructor of class Wall
Wall();
// Constructor of class FakeWall
FakeWall(int in_req_exp);
// Method getReqExp
int getReqExp() const;
```

3.2 Bản đồ

Yêu cầu: **Hiện thực** Class **Map** để biểu diễn bản đồ theo các mô tả sau:

1. Thuộc tính private **num_rows** và **num_cols** đều có kiểu int, lần lượt lưu trữ số hàng và số cột của bản đồ.
2. Thuộc tính private **map** để biểu diễn một mảng 2 chiều, mỗi phần tử của mảng là một MapElement. SV lựa chọn kiểu dữ liệu phù hợp cho **map** để thể hiện được tính đa hình (polymorphism). Gợi ý: cân nhắc giữa các kiểu dữ liệu: **MapElement**** và **MapElement*****.
3. Phương thức khởi tạo Constructor (public) với khai báo như sau:

```
Map(int num_rows, int num_cols, int num_walls, Position * array_walls,
    int num_fake_walls, Position * array_fake_walls);
```

Trong đó:

- **num_rows**, **num_cols** lần lượt là số hàng và số cột
- **num_walls** là số lượng đối tượng Wall
- **array_walls** là mảng các Position, biểu diễn một mảng các vị trí của các Wall. Mảng này có **num_walls** phần tử. Thông tin về class **Position** sẽ được mô tả trong các mục sau.

- **num_fake_walls** là số lượng đối tượng FakeWall
- **array_fake_walls** là mảng các Position, biểu diễn một mảng các vị trí của các FakeWall. Mảng này có **num_fake_walls** phần tử.

Constructor cần tạo ra một mảng 2 chiều mà mỗi phần tử là các đối tượng phù hợp. Nếu phần tử có vị trí nằm trong mảng **array_walls** thì phần tử đó là đối tượng Wall. Nếu phần tử có vị trí nằm trong mảng **array_fake_walls** thì phần tử đó là đối tượng FakeWall. Các phần tử còn lại là đối tượng Path.

4. Phương thức hủy Destructor (public) thu hồi các vùng nhớ được cấp phát động.
5. Một số phương thức khác được yêu cầu trong các mục sau.

3.3 Vị trí

Class **Position** biểu diễn vị trí trong chương trình. Position có thể được sử dụng để lưu trữ vị trí của thành phần bản đồ (như vị trí của Wall, FakeWall) hoặc vị trí của các đối tượng di chuyển trên bản đồ.

Yêu cầu: **Hiện thực** class **Position** với các mô tả sau:

1. Hai thuộc tính private có tên **r** và **c** đều có kiểu int, lần lượt mô tả vị trí theo hàng và theo cột.
2. Phương thức khởi tạo Constructor (public) với hai tham số **r** và **c** lần lượt gán cho hai thuộc tính hàng và cột. Hai tham số này có giá trị mặc định là 0.

```
Position(int r=0, int c=0);
```

3. Phương thức khởi tạo Constructor (public) với 1 tham số **str_pos** biểu diễn một vị trí ở dạng chuỗi. Định dạng của **str_pos** là `"(<r>,<c>)"` với <r> và <c> lần lượt là giá trị cho hàng và cột.

```
Position(const string & str_pos); // Example: str_pos = "(1,15)"
```

4. 4 phương thức get, set cho 2 thuộc tính hàng và cột với khai báo như sau:

```
int getRow() const ;
int getCol() const ;
void setRow(int r) ;
void setCol(int c) ;
```

5. Phương thức **str** trả về một chuỗi biểu diễn thông tin vị trí. Định dạng trả về giống như định dạng của chuỗi **str_pos** trong Constructor. Ví dụ với $r = 1, c = 15$ thì **str** trả về giá trị **"(1,15)"**.

```
string str() const
```

6. Phương thức **isEqual** có hai tham số truyền vào **in_r** và **in_c** biểu diễn cho một vị trí. **isEqual** trả về giá trị **true** nếu vị trí truyền vào trùng với vị trí của đối tượng này. Ngược lại, **isEqual** trả về **false**.

```
bool isEqual(int in_r, int in_c) const
```

7. Một số phương thức khác có thể được yêu cầu trong các mục sau.

3.4 Đối tượng di chuyển

Abstract class **MovingObject** được sử dụng để biểu diễn các đối tượng di chuyển trong chương trình như Sherlock, Watson, tên tội phạm.

Yêu cầu: **Hiện thực** abstract class **MovingObject** với các mô tả sau:

1. Các thuộc tính protected:
   - **index**: kiểu *int*, vị trí của đối tượng di chuyển trong mảng các đối tượng di chuyển, mảng này sẽ được mô tả sau.
   - **pos**: kiểu *Position*, vị trí hiện tại của đối tượng di chuyển.
   - **map**: kiểu *Map **, bản đồ cho đối tượng này di chuyển trong đó.
   - **name**: kiểu *string*, tên của đối tượng di chuyển.
2. Phương thức khởi tạo Constructor (public) với các tham số **index**, **pos**, **map**, **name** có ý nghĩa giống với thuộc tính có cùng tên. Phương thức gán giá trị của tham số cho thuộc tính cùng tên. Riêng tham số **name** có giá trị mặc định là "".

```
MovingObject(int index, const Position pos, Map * map, const string &
    name="")
```

3. Phương thức hủy ảo (virtual destructor) với quyền truy cập public.
4. Phương thức ảo thuần tố (pure virtual method) **getNextPosition** trả về Position tiếp theo mà đối tượng này di chuyển đến. Mặt khác, trong trường hợp không có Position nào để đối tượng di chuyển đến, ta định nghĩa một giá trị để trả về cho phương thức này và lưu trong biến **npos** của class **Position**. Khi không có Position để di chuyển đến thì phương thức trả về **npos**.

```
virtual Position getNextPosition() = 0;
```

5. Method **getCurrentPosition** trả về Position hiện tại của đối tượng di chuyển.

```
Position getCurrentPosition() const;
```

6. Method **move** thực hiện 1 bước di chuyển của đối tượng.

```
virtual void move() = 0;
```

7. Pure virtual method **str** trả về chuỗi biểu diễn thông tin của đối tượng.

```
virtual string str() const = 0;
```

Yêu cầu: Định nghĩa biến **npos** (not position) trong class **Position** biểu diễn rằng không có vị trí nào để đối tượng di chuyển đến. Biến **npos** có $r = -1$ và $c = -1$. Khai báo của biến như sau:

```
static const Position npos;
```

Yêu cầu: Trong class **Map**, định nghĩa phương thức **isValid** kiểm tra vị trí **pos** có phải là một vị trí hợp lệ cho đối tượng **mv_obj** di chuyển đến. Một vị trí hợp lệ cho việc di chuyển phải phụ thuộc vào đối tượng di chuyển là gì và thành phần bản đồ. Ví dụ, Sherlock có thể di chuyển trên FakeWall nhưng Watson thì cần thỏa yêu cầu về EXP. SV cần tìm đọc mô tả trong BTL này để hiện thực phương thức cho đúng.

```
bool isValid(const Position & pos, MovingObject * mv_obj) const;
```

3.5 Sherlock

Class **Sherlock** biểu diễn cho nhân vật Sherlock trong chương trình. Class **Sherlock** nhận class **MovingObject** là lớp tổ tiên (ancestor class). Do đó, class Sherlock phải hiện thực các pure virtual method của class MovingObject.

Yêu cầu: SV hiện thực class Sherlock với yêu cầu bên dưới. **SV có thể tự đề xuất thêm các thuộc tính, các phương thức hoặc các class khác để hỗ trợ cho việc hiện thực các class trong BTL này.**

1. Constructor (public) được khai báo như bên dưới. Constructor này có thêm một số tham số khác bên cạnh tham số đã có trong **MovingObject**:

```
Sherlock(int index, const string & moving_rule, const Position &
    init_pos, Map * map, int init_hp, int init_exp)
```

   - **moving_rule**: mô tả cách thức mà Sherlock di chuyển. Đây là một chuỗi mà các ký tự chỉ có thể là một trong 4 giá trị: 'L' (Left - đi sang trái), 'R' (Right - đi sang phải), 'U' (Up - đi lên trên), 'D' (Down - đi xuống dưới). Ví dụ về **moving_rule** là "LU".

- **init_hp**: HP ban đầu của Sherlock. HP nằm trong khoảng $[0, 500]$. Nếu HP vượt quá 500 thì được cài đặt về 500, nếu HP bằng 0 thì coi như Sherlock đã hết thể lực và không thể di chuyển tiếp trong mê cung. Nếu HP của cả Sherlock và Watson đều bằng 0 thì Sherlock và Watson bị thua trong cuộc truy đuổi với tên tội phạm.
- **init_exp**: EXP ban đầu của Sherlock. EXP nằm trong khoảng $[0, 900]$. Nếu HP vượt quá 900 thì cài đặt về 900, nếu EXP bằng 0 thì Sherlock cũng sẽ không di chuyển tiếp trong mê cung.
- Tham số **name** của Constructor MovingObject được truyền vào giá trị "Sherlock".
- Sherlock có thêm các thuộc tính **hp** và **exp**.

2. Phương thức **getNextPosition** (public) trả về vị trí di chuyển tiếp theo của Sherlock. Sherlock di chuyển dựa theo **moving_rule**. Mỗi lần gọi phương thức, một ký tự tiếp theo được sử dụng để làm hướng di chuyển. Lần đầu tiên gọi phương thức thì ký tự đầu tiên sẽ được sử dụng. Khi ký tự cuối cùng được sử dụng thì sẽ quay lại bắt đầu quá trình này từ ký tự đầu tiên. Ví dụ với **moving_rule** = "LR" thì thứ tự các ký tự được sử dụng là: 'L', 'R', 'L', 'R', 'L', 'R',... Nếu Position được trả ra không phải là một vị trí hợp lệ cho đối tượng này di chuyển thì trả về **npos** thuộc class **Position**.
3. Phương thức **move** (public) thực hiện một bước di chuyển của Sherlock. Trong **move** có lời gọi đến **getNextPosition**, nếu nhận được giá trị trả về khác **npos** là bước đi hợp lệ thì Sherlock sẽ di chuyển đến đó. Nếu không phải bước đi hợp lệ thì Sherlock sẽ đứng im.
4. Phương thức **str** trả về chuỗi có định dạng như sau:

```
Sherlock[index=<index>;pos=<pos>;moving_rule=<moving_rule>]
```

Trong đó:

- <index> là giá trị của thuộc tính *index*.
- <pos> là chuỗi biểu diễn của thuộc tính *pos*.
- <moving_rule> là giá trị của thuộc tính *moving_rule*.

3.6 Watson

Class **Watson** biểu diễn cho nhân vật Watson trong chương trình. Class **Watson** nhận class **MovingObject** làm lớp tổ tiên (ancestor class).

Yêu cầu: SV hiện thực class **Watson** tương tự như class **Sherlock**. Tuy nhiên, class **Watson** có sự thay đổi như sau:

1. Tham số **name** của Constructor MovingObject được truyền vào giá trị "Watson".

2. Phương thức **str** trả về chuỗi có định dạng như sau:

```
Watson[index=<index>;pos=<pos>;moving_rule=<moving_rule>]
```

Các thành phần có ý nghĩa tương tự như trong class **Sherlock**.

3.7 Tên tội phạm

Class **Criminal** biểu diễn cho nhân vật tên tội phạm trong chương trình. Class **Criminal** nhận class **MovingObject** làm lớp tổ tiên (ancestor class). Tên tội phạm có camera theo dõi cả Sherlock và Watson trong mê cung này. Do đó, khác với cách di chuyển của cặp đôi thám tử, tên tội phạm sẽ lựa chọn vị trí di chuyển tiếp theo là vị trí hợp lệ có tổng khoảng cách đến Sherlock và Watson là lớn nhất. Trong BTL này, khi nói đến khoảng cách, ta đang sử dụng khoảng cách **Manhattan**. Khoảng cách Manhattan giữa 2 điểm P1 có tọa độ $(x1, y1)$ và P2 có tọa độ $(x2, y2)$ là:

$$|x1 - x2| + |y1 - y2|$$

Trong trường hợp có nhiều hơn 1 vị trí đều có tổng khoảng cách đến Sherlock và Wáton là lớn nhất thì ưu tiên chọn vị trí theo thứ tự các hướng đi 'U', 'L', 'D', 'R'.

Yêu cầu: Hiện thực class **Criminal** tương tự như class **Sherlock** với các sự thay đổi như sau:

1. Constructor (public) được khai báo như bên dưới. Một số tham số có ý nghĩa tương tự như class **Sherlock**. Một số điểm khác nhau là:

```
Criminal(int index, const Position & init_pos, Map * map, Sherlock *
    sherlock, Watson * watson);
```

- **sherlock**, **watson** lần lượt là con trỏ đến đối tượng Sherlock và Watson. Thông qua 2 con trỏ, ta có thể lấy được vị trí hiện tại của hai nhân vật này.
- Tham số **name** của Constructor MovingObject được truyền vào giá trị "Criminal".

2. Phương thức **str** trả về chuỗi có định dạng như sau:

```
Criminal[index=<index>;pos=<pos>]
```

3.8 Mảng các đối tượng di chuyển

Class **ArrayMovingObject** biểu diễn một mảng các đối tượng di chuyển. Khi chương trình chạy, mảng này được duyệt từ đầu đến cuối và gọi phương thức **move** của mỗi phần tử để mỗi đối tượng thực hiện 1 bước đi.

Yêu cầu: Hiện thực class **ArrayMovingObject** với các yêu cầu sau:

1. Các thuộc tính private:
   - **arr_mv_objs**: mảng các đối tượng di chuyển (MovingObject). Mỗi phần tử trong mảng cần thể hiện được tính đa hình. SV tự đề xuất kiểu dữ liệu cho biến.
   - **count**: kiểu *int*, số lượng phần tử hiện tại của mảng.
   - **capacity**: kiểu *int*, số lượng phần tử tối đa của mảng.
2. Các phương thức public:
   - Constructor **ArrayMovingObject** nhận vào một tham số để khởi tạo cho thuộc tính **capacity**. Đồng thời phương thức cần thực hiện cấp phát cho phù hợp.
   - Destructor của class cần thu hồi vùng nhớ được cấp phát động.
   - Phương thức **isFull** trả về giá trị **true** nếu mảng đã đầy, ngược lại thì trả về **false**. Mảng đã đầy nếu số lượng phần tử hiện tại bằng số lượng phần tử tối đa của mảng.

     ```
     bool isFull() const;
     ```

   - Phương thức **add** thêm một đối tượng di chuyển mới vào cuối mảng nếu mảng chưa đầy, sau đó trả về **true**. Ngược lại, phương thức sẽ không thêm đối tượng mới vào và trả về **false**.

     ```
     bool add(MovingObject * mv_obj)
     ```

   - Phương thức **str** trả về chuỗi biểu diễn thông tin cho **ArrayMovingObject**.

     ```
     string str() const
     ```

     Định dạng của chuỗi trả về là:

     ```
     ArrayMovingObject[count=<count>;capacity=<capacity>;<MovingObject1>;...]
     ```

     Trong đó:
     - count, capacity: Lần lượt là số lượng và số phần tử tối đa của đối tượng **ArrayMovingObject**
     - MovingObject1,...: Lần lượt là các MovingObject có trong mãng. Mỗi MovingObject được in theo định dạng tương ứng của loại đối tượng đó.

Ví dụ về chuỗi trả về của phương thức **str** của **ArrayMovingObject**:

```
ArrayMovingObject[count=3;capacity=10;Criminal[index=0;pos=(8,9)];
    ↪ Sherlock[index=1;pos(1,4);moving_rule=RUU];Watson[index=2;pos
    ↪ =(2,1);moving_rule=LU]]
```

3.9 Cấu hình cho chương trình

Một tập tin được sử dụng để chứa cấu hình cho chương trình. Tập tin gồm các dòng, mỗi dòng có thể là một trong các định dạng như bên dưới. Lưu ý rằng thứ tự các dòng có thể thay đổi.

1. MAP_NUM_ROWS=<nr>
2. MAP_NUM_COLS=<nc>
3. MAX_NUM_MOVING_OBJECTS=<mnmo>
4. ARRAY_WALLS=<aw>
5. ARRAY_FAKE_WALLS=<afw>
6. SHERLOCK_MOVING_RULE=<smr>
7. SHERLOCK_INIT_POS=<sip>
8. SHERLOCK_INIT_HP=<sih>
9. SHERLOCK_INIT_EXP=<exp>
10. WATSON_MOVING_RULE=<wmr>
11. WATSON_INIT_POS=<wip>
12. WATSON_INIT_HP=<wih>
13. WATSON_INIT_EXP=<wie>
14. CRIMINAL_INIT_POS=<cip>
15. NUM_STEPS=<ns>

Trong đó:

1. <nr> là số hàng của bản đồ, ví dụ:
   MAP_NUM_ROWS=10
2. <nc> là số cột của bản đồ, ví dụ:
   MAP_NUM_COLS=10
3. <mnmo> là số lượng phần tử tối đa của mảng các đối tượng di chuyển, ví dụ:
   MAX_NUM_MOVING_OBJECTS=10
4. <aw> là danh sách các vị trí của các Wall. Danh sách được đặt trong một cặp dấu "[]", gồm các vị trí được phân cách bởi 1 dấu ';'. Mỗi vị trí là một cặp mở đóng ngoặc "()",

bên trong là số hàng và số cột phân cách bởi 1 dấu phẩy. Ví dụ:
`ARRAY_WALLS=[(1,2);(2,3);(3,4)]`

5. `<afw>` là danh sách các vị trí của các FakeWall. Danh sách này có định dạng giống `<aw>`. Ví dụ:
   `ARRAY_WALLS=[(4,5)]`
6. `<smr>` là chuỗi biểu diễn **moving_rule** của Sherlock. Ví du:
   `SHERLOCK_MOVING_RULE=RUU`
7. `<sip>` là vị trí ban đầu của Sherlock. Ví du:
   `SHERLOCK_INIT_POS=(1,3)`
8. `<wmr>` là chuỗi biểu diễn **moving_rule** của Watson. Ví du:
   `WATSON_MOVING_RULE=LU`
9. `<wip>` là vị trí ban đầu của Watson. Ví du:
   `WATSON_INIT_POS=(2,1)`
10. `<cip>` là vị trí ban đầu của tên tội phạm. Ví du:
    `WATSON_INIT_POS=(7,9)`
11. `<ns>` là số vòng lặp mà chương trình sẽ thực hiện. Trong mỗi vòng lặp, chương trình sẽ duyệt qua toàn bộ các MovingObject và để các đối tượng này thực hiện 1 bước di chuyển. Ví dụ:
    `NUM_STEPS=100`
12. <sih>, <sie> lần lượt là HP và EXP ban đầu của Sherlock.
13. <wih>, <wie> lần lượt là HP và EXP ban đầu của Watson.

Yêu cầu: Hiện thực class **Configuration** biểu diễn cho một cấu hình của chương trình thông qua việc đọc tập tin cấu hình. Class **configuration** có các mô tả sau:

1. Các thuộc tính private:
   - **map_num_rows**, **map_num_cols** lần lượt là số hàng và số cột của bản đồ.
   - **max_num_moving_objects**: kiểu *int*, tương ứng với `<mnmo>`.
   - **num_walls**: kiểu *int*, số lượng đối tượng Wall.
   - **arr_walls**: kiểu *Position**, tương ứng với `<aw>`.
   - **num_fake_walls**: kiểu *int*, số lượng đối tượng FakeWall.
   - **arr_fake_walls**: kiểu *Position**, tương ứng với `<afw>`.
   - **sherlock_moving_rule**: kiểu *string*, tương ứng với `<smr>`.
   - **sherlock_init_pos**: kiểu *Position*, tương ứng với `<sip>`.
   - **sherlock_init_hp**: kiểu *int*, tương ứng với `<sih>`.
   - **sherlock_init_exp**: kiểu *int*, tương ứng với `<sie>`.

- **watson_moving_rule**: kiểu *string*, tương ứng với `<wmr>`.
- **watson_init_pos**: kiểu *Position*, tương ứng với `<wip>`.
- **watson_init_hp**: kiểu *int*, tương ứng với `<wih>`.
- **watson_init_exp**: kiểu *int*, tương ứng với `<wie>`.
- **criminal_init_pos**: kiểu *Position*, tương ứng với `<cip>`.
- **num_steps**: kiểu *int*, tương ứng với `<ns>`.

2. Constructor **Configuration** được khai báo như bên dưới. Constructor nhận vào **filepath** là chuỗi chứa đường dẫn đến tập tin cấu hình. Constructor khởi tạo các thuộc tính cho phù hợp với các mô tả trên.

```
Configuration(const string & filepath);
```

3. Destructor cần thu hồi các vùng nhớ được cấp phát động.
4. Phương thức **str** trả về chuỗi biểu diễn cho Configuration.

```
string str() const;
```

Định dạng của chuỗi này là:

```
Configuration[
<attribute_name1>=<attribute_value1>...
]
```

Ví dụ về một chuỗi trả về cho phương thức **str** của **Configuration**:

```
Configuration[
MAP_NUM_ROWS=10
MAP_NUM_COLS=10
MAX_NUM_MOVING_OBJECTS=10
NUM_WALLS=3
ARRAY_WALLS=[(1,2);(2,3);(3,4)]
NUM_FAKE_WALLS=1
ARRAY_FAKE_WALLS=[(4,5)]
SHERLOCK_MOVING_RULE=RUU
SHERLOCK_INIT_POS=(1,3)
SHERLOCK_INIT_HP=250
SHERLOCK_INIT_EXP=500
WATSON_MOVING_RULE=LU
WATSON_INIT_POS=(2,1)
WATSON_INIT_HP=300
WATSON_INIT_EXP=350
CRIMINAL_INIT_POS=(7,9)
NUM_STEPS=100
]
```

Trong đó mỗi thuộc tính và giá trị thuộc tính tương ứng sẽ được in theo thứ tự như đã liệt kê tại phần "Các thuộc tính private" của class này.

3.10 Robot

Trong quá trình di chuyển của tội phạm, cứ sau mỗi **3 bước** mà tội phạm di chuyển, một robot sẽ được tạo ra. Lưu ý rằng khi phương thức **getNextPosition** của **Criminal** không trả về được một vị trí di chuyển hợp lệ, phương thức **move** không thực hiện bước di chuyển, ta không tính đó là 1 bước di chuyển. Mỗi loại robot sẽ là một nhận **MovingObject** làm ancestor class. Mỗi loại robot đều có thể di chuyển trên **Path** hoặc **FakeWall**, nhưng không thể di chuyển trên **Wall**. Sau khi được tạo ra, robot sẽ được thêm vào mảng các đối tượng di chuyển (**ArrayMovingObject**) thông qua phương thức **add** của lớp **ArrayMovingObject**. Trong trường hợp số lượng của mảng này đã đầy, thì robot không được tạo ra.

Sau mỗi **3 bước** đi của tên tội phạm một robot sẽ được tạo ra tại **vị trí trước đó mà tên tội phạm đang đứng**. Các loại robot và điều kiện tạo ra tương ứng:

- Nếu là robot đầu tiên được tạo ra trên bản đồ, đó sẽ là loại robot **RobotC**. Nếu không, ta xét khoảng cách từ Robot đến Sherlock và Watson:
- Nếu khoảng cách đến Sherlock gần hơn: Tạo ra loại robot **RobotS**
- Khoảng cách đến Watson gần hơn: Tạo ra loại robot **RobotW**
- Khoảng cách đến Sherlock và Watson là bằng nhau: Tạo ra loại robot **RobotSW**

Các loại robot được định nghĩa thành enum **RobotType** như sau:

```
enum RobotType { C, S, W, SW} // C stands for type RobotC,...
```

Yêu cầu: SV hiện thực các class cần thiết liên quan đến các đối tượng là robot theo các yêu cầu như các nhân vật khác, có một số thay đổi như sau:

1. Các loại robot có thuộc tính giống nhau bao gồm:
   - Thuộc tính **robot_type** mang giá trị là loại robot có kiểu **RobotType**
   - Thuộc tính **item** có kiểu **BaseItem *** mang giá trị là loại thuộc tính nếu chiến thắng nhân vật sẽ nhận được. Chi tiết về các vật phẩm sẽ được trình bày tại phần sau.
2. Các thuộc tính riêng của một số loại robot:
   - **RobotC**: Thuộc tính **Criminal* criminal** là con trỏ trỏ đến tên tội phạm
   - **RobotS**: Thuộc tính **Criminal* criminal** như **RobotC** và thuộc tính **Sherlock* sherlock** là con trỏ trỏ đến Sherlock
   - **RobotW**: Thuộc tính **Criminal* criminal** như **RobotC** và thuộc tính **Watson* watson** là con trỏ trỏ đến Watson
   - **RobotSW**: Thuộc tính **Criminal* criminal** như **RobotC** và 2 thuộc tính **Sherlock* sherlock** và **Watson* watson**
3. Constructor (public) tương ứng nhận vào các tham số liên quan đến **MovingObject** và các tham số riêng cho từng loại robot. Cụ thể constructor của từng loại được gọi như sau:

```
RobotC(int index, const Position & init_pos, Map * map, Criminal*
criminal);

RobotS(int index, const Position & init_pos, Map * map, Criminal*
criminal, Sherlock* sherlock);

RobotW(int index, const Position & init_pos, Map * map, Criminal*
criminal, Watson* watson);

```

```
7     RobotSW(int index, const Position & init_pos, Map * map, Criminal*
      criminal, Sherlock* sherlock, Watson* watson);
8
```

Trong đó:

- **criminal** là con trỏ đến đối tượng Criminal tương ứng có thể lấy được vị trí hiện tại của tên tội phạm.
- **Sherlock* sherlock** là con trỏ trỏ tới Sherlock. Từ đó có thể lấy được vị trí hiện tại của Sherlock.
- **Watson* watson** là con trỏ trỏ tới Watson. Từ đó có thể lấy được vị trí hiện tại của Watson.
- Các tham số còn lại có ý nghĩa giống như các phần trước.

4. Phương thức **getNextPosition** (public): Đối với mỗi loại robot, quy tắc di chuyển cũng khác nhau, cụ thể:
   - **RobotC:** Di chuyển đến vị trí tiếp theo cùng vị trí với tên tội phạm
   - **RobotS:** Di chuyển đến vị trí tiếp theo cách ban đầu 1 đơn vị và gần nhất với vị trí hiện tại của Sherlock. Lưu ý khi nói vị trí cách vị trí ban đầu 1 đơn vị thì khoảng cách được nói đến là khoảng cách Manhattan. Nếu có nhiều vị trí gần nhất thì thứ tự lựa chọn ***theo chiều quay của kim đồng hồ*** , bắt đầu từ hướng ***đi lên***, và lựa chọn vị trí đầu tiên.
   - **RobotW:** Di chuyển đến vị trí tiếp theo cách ban đầu 1 đơn vị và gần nhất với vị trí tiếp theo của Watson. Nếu có nhiều vị trí phù hợp thì thứ tự lựa chọn như nếu trong **RobotS**.
   - **RobotSW:** Di chuyển đến vị trí tiếp theo cách ban đầu 2 đơn vị và có tổng khoảng cách đến cả Sherlock và Watson là gần nhất. Nếu có nhiều vị trí phù hợp thì thứ tự lựa chọn như nếu trong **RobotS**.
   - Nếu vị trí đó không hợp lệ, trả về giá trị **npos** của **Position**
5. Phương thức **move** (public): Nếu vị trí tiếp theo không phải là **npos** thì di chuyển đến vị trí này.
6. Phương thức **getDistance** (public): Đối với các loại robot S, W hay SW, phương thức này trả về giá trị khoảng cách tương ứng của đối tượng robot đó đến Sherlock, Watson hay tổng của cả 2. Riêng robot loại C cung cấp 2 phương thức tính toán khoảng cách đến Sherlock hoặc đến Watson dựa trên 1 tham số đầu vào là con trỏ **Sherlock* sherlock** hay **Watson* watson**.
7. Phương thức **str** (public): Định dạng chuỗi trả về như sau:

```
Robot[pos=<pos>;type=<robot_type>;dist=<dist>]
```

<**pos**> in ra vị trí hiện tại của Robot
robot_type in ra giá trị có thể là C, S, W hay SW
dist in ra khoảng cách đến Sherlock, Watson hay tổng cả 2 tuỳ vào loại robot là S, W hay SW. Nếu loại robot là RobotC thì in ra chuỗi rỗng.

3.11 Vật phẩm

Class **BaseItem** là một abstract class biểu diễn thông tin cho 1 vật phẩm. SV tự định nghĩa class **Character** biểu diễn cho một nhân vật như Sherlock, Watson hay tên tội phạm. **Character** cần thỏa điều kiện: **Character** nhận **MovingObject** là ancestor class, và **Criminal**, **Sherlock**, **Watson** nhận **Character** là ancestor class. SV tự định nghĩa class **Robot** biểu diễn có một loại Robot nào đó. **Robot** cần thỏa mãn điều kiện: **Robot** nhận **MovingObject** lá ancestor class; và **RobotC**, **RobotS**, **RobotW**, **RobotSW** nhận **Robot** làm ancestor class. Class có hai public pure virtual method là:

- **canUse**

```
virtual bool canUse(Character* obj, Robot * robot) = 0;
```

Method trả về `true` nếu Sherlock hoặc Watson có thể dùng được vật phẩm này, ngược lại trả về `false`.

- **use**

```
virtual void use(Character* obj, Robot * robot) = 0;
```

Method thực hiện tác động lên Sherlock hoặc Watson nhằm thay đổi các thông số của họ cho phù hợp với tác dụng của vật phẩm. Tham số **robot** biểu diễn cho việc nhân vật gặp một Robot và sẽ thực hiện việc sử dụng vật phẩm sao cho phù hợp. Nếu nhân vật không gặp robot thì tham số **robot** bằng **NULL**.

Mỗi khi Sherlock hoặc Watson gặp một loại Robot nào đó, mỗi người sẽ có 2 thời điểm tìm kiếm **Vật phẩm** trong **Túi đồ** (sẽ mô tả ở Mục sau). Thời điểm sử dụng đầu tiên là khi vừa gặp **Robot**, họ sẽ tìm thử trong túi đồ có vật phẩm nào giúp vượt qua thử thách của Robot hay không. Khi đó, tham số **robot** cần truyền vào đối tượng **Robot**. Thời điểm đầu chỉ có thể sử dụng **ExcemptionCard** hoặc **PassingCard**. Thời điểm sử dụng thứ 2 là ở cuối lúc gặp Robot (sau khi thực hiện thử thách nếu thử thách có xảy ra), Sherlock hoặc Watson sẽ lại tìm trong túi đồ thử có vật phẩm nào có thể hồi phục được **hp** hoặc **exp** của họ hay không. Khi

đó, tham số **robot** có giá trị là **NULL**. Thời điểm thứ hai chỉ có thể sử dụng **MagicBook**, **EnergyDrink** hoặc **FirstAid**

Các class MagicBook, EnergyDrink, FirstAid, ExcemptionCard, PassingCard lần lượt biểu diễn cho các vật phẩm: Nước tăng lực, túi cứu thương hồi phục **HP**, thẻ miễn trừ ảnh hưởng của chướng ngại, thẻ miễn trừ thực hiện thử thách. Riêng đối với thẻ miễn trừ thực hiện thử thách có thuộc tính loại thẻ để thể hiện loại thử thách mà thẻ có thể miễn trừ. Các class này kế thừa từ class BaseItem và cần phải định nghĩa lại 2 method canUse và use sao cho phù hợp.

Tác dụng của từng loại vật phẩm được mô tả như sau:

Vật phẩm	**Tác dụng**	**Điều kiện sử dụng**
MagicBook	Chứa kiến thức ma thuật cổ đại giúp Sherlock và Watson tăng cường kiến thức, kinh nghiệm và trải nghiệm cho họ một cách nhanh chóng nên dễ dàng hồi phục **exp** tăng thêm 25% khi sử dụng.	**exp** ≤ 350
EnergyDrink	Nước tăng lực khi được sử dụng sẽ giúp nhận vật hồi phục **hp** tăng thêm 20% khi sử dụng.	**hp** ≤ 100
FirstAid	Túi đồ cứu thương khi sử dụng sẽ giúp nhân vật hồi phục **hp** tăng thêm 50% khi sử dụng.	**hp** ≤ 100 hoặc **exp** ≤ 350
ExcemptionCard	Thẻ miễn trừ ảnh hưởng có tác dụng giúp nhân vật miễn trừ **hp, exp** khi không vượt qua các thử thách tại một vị trí đi đến.	Chỉ có Sherlock sử dụng được thẻ này và khi **hp** của Sherlock là số lẻ
PassingCard	Khi sử dụng thẻ PassingCard thực hiện thử thách giúp nhân vật không cần thực hiện thử thách tại một vị trí đi đến. Thẻ có một thuộc tính là **challenge** (kiểu chuỗi) là tên của một thử thách (ví dụ loại RobotS là thẻ để vượt qua thử thách của RobotS đặt ra mà không cần thực hiện). Nếu loại của thẻ là **all** thẻ có thể sử dụng cho bất kì loại nào mà không cần quan tâm thử thách gặp phải là gì. Nếu không, khi sử dụng phương thức **use** để thực hiện hành động tác động lên Sherlock và Watson hay không thì cần phải kiểm tra loại thẻ trùng với loại hành động thử thách mà nhân vật gặp phải. Trong trường hợp này, nếu không trùng thể loại thẻ thì sẽ khiến **exp** của nhân vật bị trừ đi **50 EXP** mặc dù tác dụng vẫn thực hiện được.	Chỉ có Watson sử dụng được thẻ này và khi **hp** của Watson là số chẵn

Các vật phẩm này được chứa trong các robot. Điều kiện tạo ra như sau:

Gọi vị trí tạo ra robot có toạ độ là (i,j) với i là chỉ số hàng, j là chỉ số cột.

Với $p = i * j$. Gọi s số chủ đạo của p . Ta định nghĩa số chủ đạo của một số là giá trị tổng các chữ số, cho đến khi giá trị tổng đó là số có 1 chữ số. Ta có:

- Nếu s nằm trong đoạn [0, 1] thì sẽ tạo ra **MagicBook**
- Nếu s nằm trong đoạn [2, 3] thì sẽ tạo ra **EnergyDrink**
- Nếu s nằm trong đoạn [4, 5] thì sẽ tạo ra **FirstAid**
- Nếu s nằm trong đoạn [6,7] thì sẽ tạo ra **ExcemptionCard**
- Nếu s nằm trong đoạn [8, 9] thì sẽ tạo ra **PassingCard**. Đặt $t = (i * 11 + j)\%4$. Thuộc tính **challenge** của **PassingCard** được khởi tạo như sau:
  - $t = 0$: **challenge** = "RobotS"
  - $t = 1$: **challenge** = "RobotC"
  - $t = 2$: **challenge** = "RobotSW"
  - $t = 3$: **challenge** = "all"

Yêu cầu: Sinh viên hiện thực các class được mô tả như trên.

3.12 Túi đồ

Sherlock và Watson mỗi người được trang bị một túi đồ để chứa vật phẩm thu thập được trên đường di chuyển. Túi đồ được hiện thực dưới hình thức một danh sách liên kết đơn. Các hành động thực hiện với túi bao gồm:

- Thêm một món đồ vào túi đồ (phương thức **insert)**): Thêm món đồ vào đầu danh sách
- Sử dụng một món đồ bất kì (phương thức **get**): Tìm món đồ trong túi có thể sử dụng được và gần đầu túi nhất. Món đồ này sẽ được đảo với món đồ đầu tiên trong danh sách và sau đó được xóa ra khỏi danh sách.
- Sử dụng một món đồ cụ thể (phương thức **get(ItemType)**: Nếu trong túi đồ có món đồ có kiểu cần sử dụng, nhân vật có thể sử dụng món đồ này bằng cách đảo vị trí nó với món đồ đầu tiên (nếu đó không phải là món đồ đầu tiên) và xoá nó khỏi danh sách. Nếu có nhiều món đồ thì món đồ ở gần đầu túi nhất sẽ được thực hiện như trên.

Mỗi nhân vật sẽ có một số lượng món đồ tối đa có thể chứa trong túi. Túi có Sherlock có thể chứa tối đa **13 món đồ**, trong khi Watson có thể chứa tối đa **15 món đồ**.

Trong trường hợp Sherlock và Watson gặp nhau, Sherlock tặng Watson thẻ **PassingCard** nếu có và ngược lại Watson sẽ tặng thẻ **ExcepmtionCard** nếu có. Nếu có nhiều hơn một thẻ trong túi đồ, họ sẽ tặng cho đối phương tất cả. Hành động tặng tức là việc xoá các thẻ ấy ở túi đồ của mình và thêm vào đầu của túi đồ người kia. Hành động trao đổi này sẽ diễn ra theo thứ tự Sherlock trước rồi mới đến Watson. Trong trường hợp một trong hai không có hoặc cả hai đều không có loại vật phẩm ấy thì hành động trao đổi sẽ không diễn ra. Thứ tự nhân vật tặng

vật phẩm là Sherlock tặng Watson trước rồi mới đến Watson tặng Sherlock. Thứ tự việc tặng vật phẩm của một nhân vật là nhân vật sẽ tìm từ đầu đến cuối của túi thông qua phương thức **get**. Mỗi lần tìm thấy 1 vật phẩm có thể tặng được, nhân vật sẽ xóa (thao tác xóa như mô tả trong phần *sử dụng một món đồ*) vật phẩm khỏi túi của mình và thêm (thông qua phương thức **insert**) vào túi của nhân vật kia.

Yêu cầu: Sinh viên cần hiện thực các thông tin biểu diễn túi đồ bao gồm:

Class **BaseBag** là một class biểu diễn túi đồ. Class có một thuộc tính **obj** kiểu **Character*** biểu diễn nhân vật đang sở hữu túi. Ngoài ra class còn có các public method cơ bản như mô tả, bao gồm:

```
1    virtual bool insert (BaseItem* item); //return true if insert
    successfully
2    virtual BaseItem* get(); //return the item as described above, if not
    found, return NULL
3    virtual BaseItem* get(ItemType itemType); //return the item as described
     above, if not found, return NULL
4    virtual string str() const;
```

Đối với method str(), định dạng chuỗi trả về là:

```
Bag[count=<c>;<list_items>]
```

Trong đó:

- <c>: là số lượng item hiện tại mà túi đồ đang có.
- <list_items>: là một chuỗi biểu diễn các vật phẩm từ đầu đến cuối của danh sách liên kết, mỗi vật phẩm được biểu diễn bằng tên kiểu của vật phẩm, các vật phẩm được ngăn cách nhau bởi 1 dấu phẩy. Tên kiểu của các vật phẩm như tên class đã mô tả ở trên.

Yêu cầu: Sinh viên tự đề xuất các class kế thừa từ class BaseBag và biểu diễn các loại túi đồ khác nhau cho Sherlock và Watson. Tên của hai class này lần lượt là SherlockBag và WatsonBag. Constructor của hai class này chỉ có 1 tham số truyền vào lần lượt là **Sherlock *** (biểu diễn cho Sherlock) và **Watson *** (biểu diễn cho Watson).

3.13 StudyInPinkProgram

Nếu trong quá trình di chuyển Sherlock hoặc Watson đụng độ với các loại robot hay với tên tội phạm sẽ có các vấn đề xảy ra. Các trường hợp bao gồm:

1. Nếu Sherlock gặp:
    - RobotS: Sherlock cần giải quyết một bài toán để có thể chiến thắng RobotS. Nếu **EXP** của Sherlock lúc này **lớn hơn 400**, Sherlock sẽ giải quyết được và nhận về vật phẩm mà robot này nắm giữ. Nếu không, EXP của Sherlock sẽ mất đi 10%.
    - RobotW: Sherlock sẽ vượt qua và nhận vật phẩm mà không cần phải chiến đấu.
    - RobotSW: Sherlock chỉ có thể thắng RobotSW khi **EXP** của Sherlock **lớn hơn 300** và **HP** của Sherlock **lớn hơn 335**. Nếu chiến thắng, Sherlock nhận được vật phẩm mà robot này nắm giữ. Nếu không, Sherlock sẽ bị mất 15% HP và 15% EXP.
    - RobotC: Sherlock gặp RobotC tức là đã gặp được vị trí liền kề tên tội phạm. Lúc này, nếu **EXP** của Sherlock lớn hơn **500**, Sherlock sẽ chiến thắng tên robot, bắt được tên tội phạm (không nhận vật phẩm mà robot này nắm giữ). Ngược lại, Sherlock sẽ để tội phạm chạy thoát. Dù vậy vẫn sẽ tiêu diệt được robot và nhận về vật phẩm của robot này nắm giữ.
    - FakeWall: Như mô tả phía trên
2. Nếu Watson gặp:
    - RobotS: Watson sẽ không thực hiện hành động gì với robot và cũng không nhận được vật phẩm của robot này nắm giữ.
    - RobotW: Watson cần đối đầu với Robot này và chỉ chiến thắng khi có **HP lớn hơn 350**. Nếu chiến thắng thì Watson sẽ nhận được vật phẩm mà robot nắm giữ, nếu thua thì HP của Watson bị giảm 5%.
    - RobotSW: Watson chỉ có thể thắng RobotSW khi **EXP** của Watson **lớn hơn 600** và **HP** của Watson **lớn hơn 165**. Nếu chiến thắng, Watson nhận được vật phẩm mà robot này nắm giữ. Nếu không, Watson sẽ bị mất 15% HP và 15% EXP.
    - RobotC: Watson gặp RobotC tức là đã gặp được vị trí liền kề tên tội phạm. Watson chưa thể bắt được tên tội phạm vì bị giữ chân bởi RobotC. Tuy vậy, Watson vẫn sẽ tiêu diệt được robot và nhận về vật phẩm của robot này nắm giữ.
    - FakeWall: Như mô tả phía trên

    Trước và sau mỗi sự kiện đụng độ, nhân vật sẽ kiểm tra và sử dụng các vật dụng có thể trong túi của mình

    Yêu cầu: SV tự đề xuất và hiện thực class **StudyInPinkProgram** theo các yêu cầu sau:

    (a) Class có các thuộc tính: config (kiểu **Configuration***), sherlock (kiểu **Sherlock***), watson (kiểu **Watson***), criminal (kiểu **Criminal***), map (kiểu **Map***), arr_mv_objs (kiểu **ArrayMovingObject**.

(b) Constructor nhận 1 tham số là đường dẫn đến tập tin cấu hình cho chương trình. Constructor cần khởi tạo các thông tin cần thiết cho class. Riêng **arr_mv_objs** sau khi khởi tạo thì lần lượt thêm vào *criminal*, *sherlock*, *watson* bằng phương thức **add**.

```
StudyPinkProgram(const string & config_file_path)
```

(c) Phương thức **isStop** trả về **true** nếu chương trình dừng lại, ngược lại trả về **false**. Chương trình dừng khi một trong các điều kiện sau xảy ra: **hp** của Sherlock bằng 0; **hp** của Watson bằng 0; Sherlock bắt được tên tội phạm; Watson bắt được tên tội phạm.

```
bool isStop() const;
```

(d) Phương thức **printResult** và **printStep** in ra thông tin của chương trình. Các phương thức này được hiện thực sẵn, SV không thay đổi các phương thức này.

(e) Phương thức **run** có 1 tham số là **verbose**. Nếu **verbose** bằng **true** thì cần in ra thông tin của mỗi MovingObject trong ArrayMovingObject sau khi thực hiện một bước di chuyển và các cập nhật sau đó nếu có (ví dụ như Watson gặp một Robot và thực hiện thử thách với Robot). SV tham khảo thêm initial code về vị trí của hàm **printStep** trong **run**. Phương thức **run** chạy tối đa **num_steps** (lấy từ tập tin cấu hình). Sau mỗi **step** nếu chương trình thỏa điều kiện dừng nêu trong phương thức **isStop** thì chương trình sẽ dừng lại. Mỗi **step**, chương trình sẽ lần lượt chạy từ đầu đến cuối một ArrayMovingObject và gọi **move**. Sau đó thực hiện các công việc như khi 2 đối tượng gặp nhau và tạo ra một robot mới.

```
void run(bool verbose)
```

3.14 TestStudyInPink

TestStudyInPink là một class dùng để kiểm tra các thuộc tính của các class trong BTL này. SV phải khai báo **TestStudyInPink** là friend class khi định nghĩa tất cả các class trên.

4 Yêu cầu

Để hoàn thành bài tập lớn này, sinh viên phải:

1. Đọc toàn bộ tập tin mô tả này.

2. Tải xuống tập tin initial.zip và giải nén nó. Sau khi giải nén, sinh viên sẽ nhận được các tập tin: main.cpp, main.h, study_in_pink2.h, study_in_pink2.cpp, và các file dữ liệu đọc mẫu. Sinh viên phải nộp 2 tập tin là study_in_pink2.h và study_in_pink2.cpp nên không được sửa đổi tập tin main.h khi chạy thử chương trình.
3. Sinh viên sử dụng câu lệnh sau để biên dịch:
   g++ -o main main.cpp study_in_pink2.cpp -I . -std=c++11
   Sinh viên sử dụng câu lệnh sau để chạy chương trình:
   ./main
   Các câu lệnh trên được dùng trong command prompt/terminal để biên dịch và chạy chương trình. Nếu sinh viên dùng IDE để chạy chương trình, sinh viên cần chú ý: thêm đầy đủ các tập tin vào project/workspace của IDE; thay đổi lệnh biên dịch của IDE cho phù hợp. IDE thường cung cấp các nút (button) cho việc biên dịch (Build) và chạy chương trình (Run). Khi nhấn Build IDE sẽ chạy một câu lệnh biên dịch tương ứng, thông thường câu lệnh chỉ biên dịch file main.cpp. Sinh viên cần tìm cách cấu hình trên IDE để thay đổi lệnh biên dịch: thêm file study_in_pink2.cpp, thêm option -std=c++11, -I .
4. Chương trình sẽ được chấm trên nền tảng Unix. Nền tảng chấm và trình biên dịch của sinh viên có thể khác với nơi chấm thực tế. Nơi nộp bài trên BKeL được cài đặt để giống với nơi chấm thực tế. Sinh viên phải chạy thử chương trình trên nơi nộp bài và phải sửa tất cả các lỗi xảy ra ở nơi nộp bài BKeL để có đúng kết quả khi chấm thực tế.
5. Sửa đổi các file study_in_pink2.h, study_in_pink2.cpp để hoàn thành bài tập lớn này và đảm bảo hai yêu cầu sau:
   - Chỉ có 1 lệnh **include** trong tập tin study_in_pink2.h là **#include "main.h"** và một include trong tập tin study_in_pink1.cpp là **#include "study_in_pink2.h"**. Ngoài ra, không cho phép có một **#include** nào khác trong các tập tin này.
   - Hiện thực các hàm được mô tả ở các Nhiệm vụ trong BTL này.
6. Sinh viên được khuyến khích viết thêm các class và các hàm để hoàn thành BTL này.
7. Sinh viên bắt buộc phải cung cấp phần hiện thực cho tất cả các class và phương thức được liệt kê trong BTL này. Nếu không thì SV sẽ bị 0 điểm. Nếu đó là class hoặc phương thức mà SV chưa thể giải quyết được, thì SV phải cung cấp phần hiện thực rỗng chỉ gồm cặp dấu mở đóng ngoặc nhọn {}.

5 Nộp bài

Sinh viên chỉ nộp 2 tập tin: study_in_pink2.h và study_in_pink2.cpp, trước thời hạn được đưa ra trong đường dẫn "Assignment 2 - Submission". Có một số testcase đơn giản được sử dụng để kiểm tra bài làm của sinh viên nhằm đảm bảo rằng kết quả của sinh viên có thể biên dịch và chạy được. Sinh viên có thể nộp bài bao nhiêu lần tùy ý nhưng chỉ có bài nộp cuối cùng được tính điểm. Vì hệ thống không thể chịu tải khi quá nhiều sinh viên nộp bài cùng một lúc, vì vậy sinh viên nên nộp bài càng sớm càng tốt. Sinh viên sẽ tự chịu rủi ro nếu nộp bài sát hạn chót (trong vòng 1 tiếng cho đến hạn chót). Khi quá thời hạn nộp bài, hệ thống sẽ đóng nên sinh viên sẽ không thể nộp nữa. Bài nộp qua các phương thức khác đều không được chấp nhận.

6 Một số quy định khác

- Sinh viên phải tự mình hoàn thành bài tập lớn này và phải ngăn không cho người khác đánh cắp kết quả của mình. Nếu không, sinh viên sẽ bị xử lý theo quy định của trường vì gian lận.
- Mọi quyết định của giảng viên phụ trách bài tập lớn là quyết định cuối cùng.
- Sinh viên không được cung cấp testcase sau khi chấm bài, sinh viên sẽ được cung cấp phân bố điểm của BTL.
- Nội dung Bài tập lớn sẽ được Harmony với một câu hỏi trong bài Kiểm tra Cuối kỳ với nội dung tương tự.

7 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.
- Bài của sinh viên bị sinh viên khác nộp lên.
- 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.
- Sử dụng mã nguồn từ các công cụ có khả năng tạo ra mã nguồn mà không hiểu ý nghĩa.

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.

8 Thay đổi so với phiên bản trước

- Thay đổi các phương thức từ private thành public ở mục 3.8
- Thay đổi định dạng trả về của phương thức str ở mục 3.9
- Chỉnh sửa tên hàm **getNextMovePosition** thành **getNextPosition** ở mục 3.5
- Mục **4. Yêu cầu** thêm các phần 6 và 7.
- Mục 3.1, cập nhật mô tả FakeWall, Robot có thể đi được trên FakeWall.
- Mục 3.10, yêu cầu thứ 4, chỉnh sửa quy tắc di chuyển của RobotS.
- Mục 3.10, yêu cầu thứ 3, chỉnh sửa lại constructor của các Robot (loại bỏ thuộc tính RobotType).
- Mục 3.1, yêu cầu 1, chỉnh sửa class MapElement
- Mục 3.4, yêu cầu 1, chỉnh sửa method getCurrentPosition