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

CUỘC TỔNG TIẾN CÔNG VÀ NỔI DẬY MÙA XUÂN NĂM 1975
Phần 2: Chiến dịch Hồ Chí Minh

Phiên bản 1.3

TP. HỒ CHÍ MINH, THÁNG 02/2025

ĐẶC TẢ BÀI TẬP LỚN

Phiên bản 1.3

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 lấy cảm hứng nhân kỷ niệm 50 năm thành công của cuộc tổng tiến công và nổi dậy mùa Xuân 1975 của quân và dân ta, mang đến chiến thắng giải phóng hoàn toàn miền Nam, thống nhất đất nước.

Trong phần 1, ta đã tái hiện lại chiến dịch Tây Nguyên, chiến dịch mở màn của cuộc tổng tiến công và nổi dậy mùa Xuân năm 1975. Ngay sau đó (cuối tháng 3/1975), quân và dân ta tiếp tục giành thắng lợi trong chiến dịch giải phóng Huế - Đà Nẵng, làm thay đổi hoàn toàn cục diện cuộc chiến.

Trước diễn biến "một ngày bằng hai mươi năm", ngày 14/4/1975, Bộ Chính trị quyết định Chiến dịch giải phóng Sài Gòn - Gia Định mang tên Chiến dịch Hồ Chí Minh. Quyết định này một lần nữa thể hiện tầm nhìn chiến lược của Bộ Chính trị, Quân ủy Trung ương và Bộ Tổng Tư lệnh trong lãnh đạo, chỉ đạo và nghệ thuật điều hành kết thúc cuộc chiến tranh cách mạng.

Trong bài tập lớn này, các bạn được yêu cầu hiện thực các lớp (class) để mô tả lại quá trình diễn biến của Chiến dịch. 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 tái hiện chiến dịch Hồ Chí Minh. Các đối tượng trong BTL được biểu diễn thông qua class và được mô tả như bên dưới.

Lưu ý:

- Sinh viên cần đọc kĩ đề để xác định các lớp kế thừa (derivative class), đề sẽ không mô tả tường minh.
- Sinh viên cần 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.
- Trong khuôn khổ BTL này, khi nhắc về khoảng cách, sinh viên cần hiểu là khoảng cách theo độ đo Euclid.
- Trong khuôn khổ BTL này, khi các giá trị tính toán cần làm tròn về số nguyên, SV cần làm tròn lên.
- Trong khuôn khổ BTL này, bất cứ giá trị nào vượt ngưỡng (nếu đề có xác định khoảng giá trị), làm tròn đến ngưỡng gần nhất.

3.1 Đơn vị Quân sự

Mỗi quân lực tham gia cuộc chiến sẽ có các đơn vị quân sự (Unit) khác nhau. Có 2 loại đơn vị quân sự chính là Phương tiện chiến đấu (Vehicle) và lực lượng quân đội (Infantry).

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

1. Các thuộc tính protected:
   - quantity: kiểu int, số lượng của loại đơn vị quân sự này trong quân lực
   - weight: kiểu int, trọng số đóng góp vào các loại chỉ số năng lực của quân đội khác nhau
   - pos: kiểu Position, vị trí của loại đơn vị này trên chiến trận
2. : Phương thức khởi tạo constructor (public) với các tham số quantity, weight, pos có ý nghĩa giống với các thuộc tính cùng tên. Phương thức gán giá trị của tham số cho thuộ tính cùng tên.

```
Unit(int quantity, int weight, const Position pos)
```

3. Phương thức huỷ ảo (virtual destructor) với quyền truy cập public

4. Phương thức ảo thuần tố (pure virtual method) **getAttackScore** trả về một số int trả về lượng đóng góp của loại đơn vị quân sự cho một loại chỉ số năng lực nhất định (chỉ số năng lực là chỉ số **LF** và **EXP** của quân quân đội đó).

```
virtual int getAttackScore() =0;
```

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

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

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

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

3.2 Phương tiện chiến đấu

Mỗi quân đội sẽ có một số lượng các loại phương tiện chiến đấu (Vehicle) khác nhau. Một loại phương tiện là một đơn vị quân sự (Unit).

Ta định nghĩa trước enum **VehicleType** như sau:

```
enum VehicleType {TRUCK, MORTAR, ANTIAIRCRAFT, ARMOREDCAR, APC, ARTILLERY,
    TANK}
```

Trong đó:

- TANK - Xe tăng (ví dụ T-54, M41)
- ARTILLERY - Pháo binh (ví dụ 105mm, 130mm)
- ARMOREDCAR - Xe thiết giáp (ví dụ BTR-152, M113)
- APC - Xe chở quân bọc thép (ví dụ BTR-60, M113)
- TRUCK - Xe tải quân sự (ví dụ ZIL-157, GMC CCKW)
- MORTAR - Súng cối di động (ví dụ 82mm, 120mm)
- ANTIAIRCRAFT - Phòng không (ví dụ ZSU-23-4, DShK)

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

1. Các thuộc tính private:
   - vehicleType: kiểu VehicleType, thể loại phương tiện
2. Constructor (public) được khai báo như bên dưới. Ngoài các tham số như đã được nêu trong **Unit**, có thêm các tham số thuộc tính riêng của **Vehilce** với tên gọi và ý nghĩa như đã nêu ở phần thuộc tính trên

```
Vehicle(int quantity, int weight, const Position pos, VehicleType
    vehicleType)
```

3. Phương thức **getAttackScore** (public) trả về giá trị theo công thức tính số điểm đóng góp của một loại phương tiện cụ thể như sau:

$$score = \frac{typeValue * 304 + quantity * weight}{30}$$

Trong đó:

- $typeValue$ là giá trị enum tương ứng loại phương tiện trong VehicleType
- $quantity, weight$ lần lượt là giá trị quantity và weight tương ứng của phương tiện đó

4. Phương thức **str** (public): Trả về chuỗi thông tin của loại phương tiện theo cú pháp:

```
Vehicle[attr_name=<attr_value>]
```

Với attr_name và attr_value lần lượt là tên và giá trị của thuộc tính của loại phương tiện tương ứng. Trong đó thứ tự các thuộc tính để in ra tương ứng là vehicleType, quantity, weight, pos và cách nhau bởi dấu phẩy (**","**).

3.3 Lực lượng bộ binh

Mỗi quân đội sẽ có các loại bộ binh (Infantry) phục vụ trong lực lượng quân đội khác nhau. Một loại bộ binh là một đơn vị quân sự (Unit).

Ta định nghĩa trước enum **InfantryType** như sau:

```
enum InfantryType {SNIPER, ANTIAIRCRAFTSQUAD, MORTARSQUAD, ENGINEER,
    SPECIALFORCES, REGULARINFANTRY}
```

Trong đó:

- SNIPER – Xạ thủ bắn tỉa
- ANTIAIRCRAFTSQUAD – Đội phòng không
- MORTARSQUAD – Đội súng cối
- ENGINEER – Công binh
- SPECIALFORCES – Lực lượng đặc biệt
- REGULARINFANTRY – Bộ binh chủ lực

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

1. Các thuộc tính private:
   - infantryType: kiểu InfantryType, thể hiện loại bộ binh
2. Constructor (public) được khai báo như bên dưới. Ngoài các tham số như đã được nêu trong **Unit**, có thêm các tham số thuộc tính riêng của **Infantry** với tên gọi và ý nghĩa như đã nêu ở phần thuộc tính trên

```
Infantry(int quantity, int weight, const Position pos, InfantryType
    infantryType)
```

3. Phương thức **getAttackScore** (public) trả về giá trị theo công thức tính số điểm đóng góp của một loại bộ binh cụ thể như sau:

$$score = typeValue * 56 + quantity * weight$$

   Trong đó:
   - $typeValue$ là giá trị enum tương ứng loại phương tiện trong VehicleType
   - $quantity, weight$ lần lượt là giá trị quantity và weight tương ứng của phương tiện đó

   Đặc biệt:
   - Nếu loại lính là lực lượng đặc biệt (SPECIALFORCES) và trọng số (weight) là một số chính phương thì đó là đặc công, chỉ số đóng góp cộng thêm 75 điểm.
   - Ta định nghĩa: Số cá nhân của một số nguyên không âm tại năm Y là số có một chữ số được tính bằng tổng các chữ số của số đó và các chữ số của Y, tiếp tục cộng các chữ số của tổng vừa tính ra cho đến khi chỉ kết quả tổng là số có một chữ số.
     Trong trường hợp này, năm chiến đấu là năm 1975, sau khi xem xét trường hợp lính đặc công, nếu *score* của loại quân có số cá nhân đạt trên 7 tức là sẽ được chi viện thêm 20% số lượng đang có. Ngược lại, nếu số cá nhân nhỏ hơn 3, sẽ có 10% số lượng hiện có đào ngũ ngay tại thời điểm đó. Điều chỉnh giá trị thuộc tính *quantity* và tính lại *score* tương ứng.
4. Phương thức **str** (public): Trả về chuỗi thông tin của loại bộ binh theo cú pháp:

```
Infantry[attr_name=<attr_value>]
```

   Với attr_name và attr_value lần lượt là tên và giá trị của thuộc tính của loại phương tiện tương ứng. Trong đó thứ tự các thuộc tính để in ra tương ứng là infantryType, quantity, weight, pos và cách nhau bởi dấu phẩy (",").

3.4 Quân đội

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

1. Các thuộc tính protected:
   - **LF**: kiểu int, giá trị nằm trong đoạn [0, 1000]. Giá trị này được tính bằng tổng *score* từ tất cả các phương tiện quân sự (Vehicle) mà quân đội này có.
   - **EXP**: kiểu int, giá trị này trong đoạn [0, 500]. Giá trị này được tính bằng tổng *score* từ tất cả các lực lượng bộ binh (Infantry) mà quân đội này có.
   - **name**: kiểu string, tên gọi của quân đội
   - **unitList**: kiểu UnitList*, danh sách đơn vị quân sự mà quân đội này có. Mô tả cụ thể sẽ được nêu ở những phần sau
   - **battleField**: kiểu BattleField*, trận địa mà quân đội này chiến đấu
2. Constructor (public) nhận vào một mảng các đơn vị quân sự, kích thước của mảng này, tên của quân đội và trận địa mà quân đội này chiến đấu. Nhiệm vụ của constructor cần tính ra giá trị thích hợp cho **LF**, **EXP** và thêm các đơn vị váo unitList cho phù hợp

```
Army(const Unit** unitArray, int size, string name, BattleField*
    battleField)
```

3. Phương thức giao tranh (fight) là một pure virtual function nhận vào một Army* (enemy) là đối thủ của quân đội hiện tại và tham số defense kiểu bool, mô tả là đợt giao tranh này là tấn công hay phòng thủ, giá trị mặc định là false (tương ứng là đợt giao tranh thuộc về tấn công). Kiểu dữ liệu trả về là void.

```
virtual void fight(Army* enemy, bool defense = false) = 0;
```

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

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

3.4.1 Quân Giải phóng

Trong Chiến dịch này, bộ phận quân ta bao gồm Quân đội Nhân dân Việt Nam và Quân Giải phóng miền Nam Việt Nam (thành lập năm 1961), BTL này gọi chung là Quân Giải phóng. Để mô tả cho Quân Giải phóng, ta cần định nghĩa một lớp mang tên LiberationArmy

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

1. Constructor có dạng như sau

```
LiberationArmy(const Unit** unitArray, int size, string name,
    BattleField* battleField)
```

2. Phương thức giao tranh (fight)

```
void fight(Army* enemy, bool defense = false)
```

- **Trường hợp tấn công (defense = false):**
  - Vì ở thế chủ động tấn công nên **LF** và **EXP** được nhân 1.5 lần.
  - Chọn phương án kết hợp đánh: Để tối ưu hoá phương án đánh, quân đội sẽ chọn ra tổ hợp các loại bộ binh có tổng số điểm nhỏ nhất lớn hơn **EXP** của đối phương, gọi tổ hợp này là tổ hợp A. Bên cạnh đó chọn ra tổ hợp các loại phương tiện có tổng số điểm nhỏ nhất lớn hơn **LF** của đối thủ, gọi tổ hợp này là tổ hợp B.

    Trường hợp tìm thấy cả 2 tổ hợp A và B phù hợp, tức là chiến thắng, xoá các tổ hợp thoả mãn này khỏi danh sách các đơn vị quân sự.

    Trường hợp chỉ tìm được một trong hai tổ hợp thoả mãn, tức là hoà hoãn, nhưng vì đang có lợi thế tấn công, nếu chỉ số của toàn quân đội tương ứng với loại tổ hợp không thoả mãn lớn hơn chỉ số ấy của đối phương thì vẫn sẽ chiến thắng, vẫn cập nhật chỉ số, xoá tổ hợp thoả mãn và tất cả các đơn vị thuộc tổ hợp không thoả mãn. Nếu chỉ số ấy nhỏ hơn, giao tranh sẽ không xảy ra.

    Trường hợp không tìm được tổ hợp nào thoả mãn, giao tranh không xảy ra.

    Nếu giao tranh chiến thắng, các loại đơn vị quân sự (kể cả các lục lượng bộ binh) của đối phương sẽ bị tịch thu và trưng dụng. Sau khi xoá các đơn vị thuộc tổ hợp thoả mãn ra khỏi danh sách, thêm vào danh sách đơn vị quân sự (nếu có thể thêm, còn nếu đã tồn tại, cập nhật lại số lượng). Sau đó cập nhật lại các chỉ số **LF** và **EXP** tương ứng.

    Nếu giao tranh không xảy ra, mỗi đơn vị quân sự trong danh sách của quân đội sẽ bị mất 10% trọng số (*weight*). Lúc này cập nhật lại các chỉ số sức mạnh của quân đội.

    Gợi ý: Nên viết các phương thức hỗ trợ, đề xuất modifier phù hợp cho các phương thức hỗ trợ này để đảm bảo các tính chất của OOP. Ngoài ra, có rất nhiều trường hợp cần cập nhật các chỉ số sức mạnh, nên có phương án phù hợp.
- **Trường hợp phòng thủ (denfense = true):**
  - Quân Giải phóng luôn trong tư thế sẵn sàng chiến đấu nên dù bị tấn công nhưng vẫn chủ động, các chỉ số sức mạnh được nhân 1.3 lần.
  - So sánh 2 chỉ số chiến đấu tương ứng của 2 phía quân đội. Nếu cả 2 chỉ số Quân Giải phóng đều không nhỏ hơn, chiến thắng thuộc về Quân Giải phóng. Nếu có

1 trong 2 thông số Quân Giải phóng nhỏ hơn, mặc dù không thất bại nhưng sẽ bị mất đi ở mỗi đơn vị quân sự 10% số lượng (*quantity*). Nếu cả 2 chỉ số đều nhỏ hơn, Quân Giải phóng sẽ cần chi viện. Cập nhật mỗi đơn vị quân sự tăng số lượng lên **số Fibonanci** gần nhất. Sau khi cập nhật lại 2 chỉ số chiến đấu tiếp tục xét lại như đã đề cập.

3. Phương thức **str**: Trả về chuỗi thông tin của Quân Giải phóng

```
LiberationArmy[attr_name=<attr_value>]
```

Thứ tự các thuộc tính lần lượt là name, LF, EXP, unitList và battleField và cách nhau bởi dấu phẩy (","").

3.4.2 Quân đội chính quyền Sài Gòn

Nguỵ quân, nguỵ quyền thời điểm bấy giờ không còn Mỹ tham chiến trực tiếp, tuy nhiên Mỹ vẫn viện trợ tài chính, vũ khí, hậu cần và cử chuyên gia quân sự tư vấn.

Yêu cầu: **Hiện thực** class **ARVN** (Army of Republic of Vietnamm) với các mô tả sau:

1. Constructor có dạng như sau

```
ARVN(const Unit** unitArray, int size, string name, BattleField*
    battleField)
```

2. Phương thức giao tranh (fight)

```
void fight(Army* enemy, bool defense = false)
```

- **Trường hợp tấn công (defense = false):**
  - Mặc dù ở thế chủ động tấn công, nhưng quân lực Việt Nam Cộng hoà phải đối mặt với sự rạn nứt nội bộ, mất tinh thần chiến đấu và thiếu sự hỗ trợ thực tế từ Mỹ nên các chỉ số chiến đấu không đổi cả khi tấn công hay phòng thủ.
  - Trường hợp Quân đội Sài Gòn tấn công, tức là Quân Giải phóng phòng thủ. Trong mô tả này, Quân Giải phóng không thất bại ngay cả khi ở thế phỏng thủ, Quân đội Sài Gòn sẽ thất bại và cập nhật lại mất đi 20% số lượng (*quantity*) ở mỗi đơn vị quân sự. Cập nhật lại chỉ số chiến đấu tương ứng. Nếu đơn vị quân sự nào có số lượng (*quantity*) chỉ còn 1, xoá đơn vị quân sự đó khỏi danhh sách và cập nhật lại các chỉ số.

- **Trường hợp phòng thủ (denfense = true):** Trong trường hợp này, tức là Quân Giải phóng đang ở thế tấn công. Theo mô tả trên, Quân Giải phóng có thể sẽ giao tranh và giành chiến thắng, lúc này, Quân đội Sài Gòn sẽ bị tịch thu và trưng dụng các đơn vị quân sự (như mô tả ở phần trên), đồng nghĩa việc cần xoá các đơn vị này khỏi danh sách của Quân đội Sài Gòn. Do thất bại nên mỗi đơn vị quân sự còn lại bị mất đi 20% trọng số (*weight*). Trong trường hợp giao tranh không xảy ra, lực lượng của Quân đội Sài Gòn được bảo toàn. tịch thu từ cuối lên đầu danh sách cho đến khi unitList của bên kia đầy

3. Phương thức **str**: Trả về chuỗi thông tin của Quân đội chính quyền Sài Gòn

```
ARVN[attr_name=<attr_value>]
```

Thứ tự các thuộc tính lần lượt là name, LF, EXP, unitList và battleField và cách nhau bởi dấu phẩy (`","`).

3.5 Danh sách các đơn vị quân sự

Để quản lý các đơn vị quân sự của một quân đội, chúng ta sẽ biểu diễn thành một **danh sách liên kết đơn**. Các hành động thực hiện với danh sách này bao gồm:

- Thêm một đơn vị vào túi đồ (phương thức **insert**)): Nếu đơn vị là một phương tiện (Vehicle) thì thêm vào cuối danh sách. Nếu đơn vị này là một loại quân lính thì thêm vào đầu danh sách. Trong trường hợp đơn vị này đã tồn tại trong danh sách, cập nhật lại số lượng (quantity) của đơn vị tương ứng.
- Kiểm tra xem một đơn vị quân sự có tồn tại trong danh sách hay không (phương thức isContain). Phương thức nhận vào một giá trị **VehicleType** hoặc **InfantryType** và trả về True nếu tìm thấy, ngược lại trả về False.

Ta định nghĩa một số được coi là **số đặc biệt** nếu như số đó có thể phân tích thành tổng các lũy thừa cơ số k riêng biệt. Ví dụ ***10 là số đặc biệt cơ số 3*** vì $10 = 3^2 + 3^0$.

Mỗi quân đội sẽ có một số lượng đơn vị có thể chứa tối đa trong danh sách. Con số này phụ thuộc vào tổng $S = \textbf{LF} + \textbf{EXP}$ của quân đội đó:

- Nếu S là một **số đặc biệt** của ít nhất một trong các **cơ số nguyên tố lẻ nhỏ hơn 10** thì sức chứa tối đa của danh sách này là 12.
- Tất cả các trường hợp khác: Sức chứa là 8.

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

1. Các thuộc tính: Sinh viên tự định nghĩa các thuộc tính, trong đó có 1 thuộc tính bắt buộc là **capacity** (kiểu **int**) (private). Tạo ra **constructor** tương ứng để khởi tạo trong các class quân đội trên. Trong đó, đảm bảo khi khởi tạo: Nếu đơn vị là một phương tiện (Vehicle) thì thêm vào cuối danh sách. Nếu đơn vị này là một loại quân lính thì thêm vào đầu danh sách.
2. Các method như đã mô tả ở trên, bao gồm:

```
bool insert (Unit* unit); //return true if insert successfully
bool isContain(VehicleType vehicleType); //return true if it exists
bool isContain(InfantryType infantryType); //return true if it exists
string str() const;
```

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

```
UnitList[count_vehicle=<v_c>;count_infantry=<i_c>;<unit_list>]
```

Trong đó:

- <v_c>, <i_c>: lần lượt là số lượng đơn vị là phương tiện và đơn vị là bộ binh đang có trong danh sách
- <unit_list>: là một chuỗi biểu diễn các đơn vị từ đầu đến cuối danh sách liên kết, mỗi đơn vị được in theo định dạng của phương thức str() tương ứng của **Unit**, ngăn cách nhau bởi dấu phẩy (,).

3.6 Vị trí

Class **Position** biểu diễn vị trí trong chương trình. Position có thể được sử dụng để lưu trữ vị trí của các đơn vị quân sự.

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

3.7 Các yếu tố địa hình của trận địa

Mỗi trận địa chiến đấu có những yếu tố địa hình ảnh hưởng đến khả năng chiến đấu của các quân đội.

Các yếu tố địa hình bao gồm:

- Đường mòn (ROAD)
- Rừng núi (FOREST)
- Sông sâu (RIVER)
- Chiến hào (FORTIFICATION)
- Khu dân cư (URBAN)
- Khu vực phi quân sự (chùa, nhà thờ, bệnh viện,...) (SPECIAL ZONE)

Mỗi yếu tố địa hình sẽ có những ảnh hưởng khác nhau đến các chỉ số năng lực của quân đội.

Yêu cầu: Hiện thực class TerrainElement, với các mô tả:

1. Constructor, destructor của class
2. Phương thức thuần ảo **getEffect** để xác định ảnh hưởng của yếu tố địa hình đến các chỉ số năng lực của quân đội.

```
    virtual void getEffect(Army* army) = 0;
```

Yêu cầu: Hiện thực các class **Road, Mountain, River, Urban, Fortification và SpecialZone** để cụ thể hoá các yếu tố địa hình. Trong đó hiện thực phương thức **getEffect** với mô tả cụ thể như bảng sau:

Bảng 1: Ảnh hưởng của địa hình đến các đơn vị quân sự

STT	**Địa hình**	**Ảnh hưởng đến Quân Giải phóng**	**Ảnh hưởng đến Quân đội CQ Sài Gòn**
1	Đường mòn	Không gây ảnh hưởng	Không gây ảnh hưởng
2	Rừng núi	Trong khi các lực lượng bộ binh của ta rất thông thạo chiến đấu trong địa hình rừng núi thì các phương tiện quân sự lại không phù hợp di chuyển. Trong bán kính 2 đơn vị khoảng cách giữa vị trí địa hình rừng núi đến vị trí của từng đơn vị quân sự: • **EXP** sẽ được cộng thêm 30% mỗi *attackScore* của mỗi lực lượng bộ binh nằm trong bán kính ảnh hưởng. • **LF** sẽ bị trừ đi 10% mỗi *attackScore* của mỗi phương tiện chiến đấu nằm trong bán kính ảnh hưởng.	Do đây là địa bàn đóng quân của Quân đội Sài Gòn nên phạm vi ảnh hưởng sẽ rộng hơn mặc dù mức độ ảnh hưởng sẽ thấp hơn. Trong bán kính 4 đơn vị khoảng cách giữa vị trí địa hình rừng núi đến vị trí của từng đơn vị quân sự: • **EXP** sẽ được cộng thêm 20% mỗi *attackScore* của mỗi lực lượng bộ binh nằm trong bán kính ảnh hưởng. • **LF** sẽ bị trừ đi 5% mỗi *attackScore* của mỗi phương tiện chiến đấu nằm trong bán kính ảnh hưởng.

STT	Địa hình	Ảnh hưởng đến Quân Giải phóng	Ảnh hưởng đến Quân đội CQ Sài Gòn
3	Sông sâu	Bộ binh bị giới hạn tốc độ hành quân. *attackScore* của các lực lượng bộ binh trong bán kính 2 đơn vị khoảng cách sẽ bị suy giảm 10%.	*attackScore* của các lực lượng bộ binh trong bán kính 2 đơn vị khoảng cách sẽ bị suy giảm 10%.

STT	Địa hình	Ảnh hưởng đến Quân Giải phóng	Ảnh hưởng đến Quân đội CQ Sài Gòn
4	Khu dân cư	• Nếu lực lượng bộ binh đó là lực lượng đặc biệt (**SPECIALFORCES**) hoặc bộ binh chủ lực (**REGULARINFANTRY**) và trong bán kính 5 đơn vị khoảng cách, mỗi *attackScore* của mỗi loại bộ binh này sẽ được tăng thêm một lượng bằng: $\frac{2 * attackScore}{D}$ Trong đó: D là khoảng cách giữa vị trí khu dân cư và loại bộ binh tương ứng. Còn các lực lượng bộ binh khác không có ảnh hưởng. • Các phương tiện chiến đấu thuộc loại pháo binh (**ARTILLERY**) sẽ bị giảm 50% *attackScore* nếu nằm trong bán kính 2 đơn vị khoảng cách.	• Nếu lực lượng bộ binh chủ lực (**REGULARINFANTRY**) trong bán kính 3 đơn vị khoảng cách, mỗi *attackScore* của mỗi loại bộ binh này sẽ được tăng thêm một lượng bằng: $\frac{3 * attackScore}{2 * D}$ Trong đó: D là khoảng cách giữa vị trí khu dân cư và loại bộ binh chủ lực thoả mãn. Còn các lực lượng bộ binh khác không có ảnh hưởng. • Các phương tiện chiến đấu của Quân đội Chính quyền Sài Gòn không bị ảnh hưởng trong các khu dân cư

STT	Địa hình	Ảnh hưởng đến Quân Giải phóng	Ảnh hưởng đến Quân đội CQ Sài Gòn
5	Chiến hào	Tạo thế khó cho Quân Giải phóng có thể tiến công, tất cả các đơn vị quân sự trong bán kính 2 đơn vị khoảng cách của Quân Giải phóng đều bị trừ 20% *attackScore*	Quân đội Sài Gòn chủ động hơn tại các vị trí này nên được cộng thêm 20% *attackScore* mỗi đơn vị quân sự xung quanh bán kinhh1 2 đơn vị khoảng cách
6	Khu vực phi quân sự	Không được diễn ra chiến sự, các đơn vị quân sự trong bán kính 1 đơn vị khoảng cách xung quanh khu vực này đều bị giảm *attackScore* xuống 0	

3.8 Trận địa

Chiến dịch Hồ Chí Minh diễn ra trên 05 mặt trận, mỗi mặt trận sẽ diễn ra các trận đánh khác nhau. Do vậy cần định nghĩa cho trận địa mặt trận ấy là một lớp. Giả sử mỗi trận địa được biểu diễn bằng một mảng 2 chiều.

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

1. Thuộc tính private **n_rows** và **n_cols** (kiểu int), lần lượt là số hàng và số cột của bản đồ trận địa
2. Thuộc tính private **terrain** để biểu diễn một mảng 2 chiều, mỗi phần tử của mảng là một TerreainElement. Sinh viên lựa chọn kiểu dữ liệu phù hợp cho thuộc tính **terrain**
3. Constructor có dạng:

```
BattleField(int n_rows, int n_cols, vector<Position*> arrayForest,
    vector<Position*> arrayRiver, vector<Position*> arrayFortification,
    vector<Position*> arrayUrban, vector<Position*> arraySpecialZone)
```

   Trong đó:

   - n_rows, n_cols: Số hảng và số cột của bản đồ trận địa
   - Các mảng có tên định dạng **array<Element>** là các vị trí của các yếu tố địa hình trong bản đồ. Các vị trí còn lại ngoài các vị trí trong các mảng đều sẽ là đường mòn (ROAD).

4. Phương thức hủy Destructor (public) thu hồi các vùng nhớ được cấp phát động.
5. Phương thức **str**: Trả về chuỗi thông tin của trận địa

```
BattleField[attr_name=<attr_value>]
```

Thứ tự các thuộc tính lần lượt là n_rows, n_cols và cách nhau bởi dấu phẩy (",").

6. Một số phương thức khác được yêu cầu trong các mục sau.

3.9 Thiết lập

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. NUM_ROWS=<nr>
2. NUM_COLS=<nc>
3. ARRAY_FOREST=[<pos1>,<pos2>,..]
4. ARRAY_RIVER=[<pos1>,<pos2>,..]
5. ARRAY_FORTIFICATION=[<pos1>,<pos2>,..]
6. ARRAY_URBAN=[<pos1>,<pos2>,..]
7. ARRAY_SPECIAL_ZONE=[<pos1>,<pos2>,..]
8. UNIT_LIST=[<unit1>,<unit2>,...]
9. EVENT_CODE=<ec>

Trong đó:

1. <nr> là số hàng của bản đồ trận địa, ví dụ:
   NUM_ROWS=10
2. <nc> là số cột của bản đồ trận địa, ví dụ:
   NUM_COLS=10
3. [<pos1>,<pos2>,..] là vị trí của các yếu tố địa hình trên trận địa. Ví dụ:
   ARRAY_FOREST=[(1,2),(2,3),(3,4)]
   là vị trí các yếu tố rừng rậm (FOREST) trong bản đổ trận địa.
4. [<unit1>,<unit2>,...] là danh sách các đơn vị quân sự của cả 02 quân đội. Trong đó mỗi unit sẽ có định dạng kiểu
   UNIT_NAME(quantity,weight,position,armyBelong Với armyBelong=0 tương ứng với Quân Giải phóng, armyBelong=1 tương ứng với Quân đội chính quyền Sài Gòn
   UNIT_LIST=[TANK(5,2,(1,2),0), TANK(5,2,(3,2),1), REGULARINFANTRY(5,2,(1,1),0), REGULARINFANTRY(5,2,(3,3),1)]

5. <ec> là mã sự kiện để xác định quân đội nào đang trong thế tấn công hay phòng thủ ở trận địa hiện tại

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:
   - **num_rows, num_cols** lần lượt là số hàng và cột của bản đồ trận địa
   - **arrayForest, arrayRiver, arrayFortification, arrayUrban, arraySpecialZone**: kiểu *vector<Position*>* là vị trí của các yếu tố địa hình trên bản đồ trận địa
   - **liberationUnits**: kiểu *Unit** là mảng các đơn vị quân sự của Quân Giải phóng
   - **ARVNUnits**: kiểu *Unit** là mảng các đơn vị quân sự của Quân đội chính quyền Sài Gòn
   - **eventCode**: kiểu *int* là mã sự kiện để xác định quân đội nào đang trong thế chủ động tấn công. Mã sự kiện này nằm trong khoảng [00,99]. Nếu số nhận vào lớn hơn 99 thì lấy 2 chữ số cuối của số nhận vào làm mã sự kiện. Trường hợp số nhận vào nhỏ hơn 00 thì mã sự kiện là 00.
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>...
]
```

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. Đối với mỗi đơn vị quân sự, việc in sẽ được thực hiện gọi phương thức **str** tương ứng của đơn vị quân sự đó.

3.10 class HCMCampaign

Cuộc chiến tại một trận địa được diễn ra theo hình thức gọi đến phương thức fight tương ứng của quân đội theo mã sự kiện. Sau giao tranh giữa 2 đơn vị quân sự, nếu đơn vị quân sự có $attackScore \leq 5$ sẽ bị xoá khỏi danh sách đơn vị quân sự của quân đội đó.

Trong đó cụ thể:

- Nếu mã sự kiện **nhỏ hơn 75**: Quân Giải phóng trong thế tấn công và Quân đội Sài Gòn trong thế phòng thủ
- Nếu mã sự kiện **từ 75**: Quân đội Sài Gòn trong thế chủ động tấn công. Sau khi Quân đội Sài Gòn tấn công kết thúc, Quân Giải phóng sẽ phản công ngay lập tức.

Yêu cầu: Sinh viên tự đề xuất và hiện thực class HCMCampaign theo các yêu cầu sau:

1. Các thuộc tính:
   - config (Kiểu Configuration*)
   - battleField (Kiểu BattleField*)
   - liberationArmy (Kiểu LiberationArmy*)
   - ARVN (Kiểu ARVN*)
2. 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.

```
HCMCampaign(const string & config_file_path)
```

3. Phương thức **run**: Không có tham số đầu vào. Phương thức thực hiện trận chiến giữa 2 quân đội tại trận địa cụ thể theo đúng luồng mô tả như trên. Trong đó, khi bắt đầu, các yếu tố địa hình sẽ ảnh hưởng đến các chỉ số chiến đấu rồi mới xét đến giao tranh.
4. Phương thức **printResult** (kiểu trả về: string): Không có tham số đầu vào. In ra các chỉ số chiến đấu của cả 2 quân đội sau giao tranh, theo định dạng:

```
LIBERATIONARMY[LF=<LF>,EXP=<EXP>]-ARVN[LF=<LF>,EXP=<EXP>]
```

Với <LF>, <EXP> lần lượt là **LF** và **EXP** của quân đội tương ứng.

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, hcmcampaign.h, hcmcampaign.cpp, và các file dữ liệu đọc mẫu. Sinh viên phải nộp 2 tập tin là hcmcampaign.h và hcmcampaign.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 hcmcampaign.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 hcmcampaign.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 LMS 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 LMS để có đúng kết quả khi chấm thực tế.
5. Sửa đổi các file hcmcapaign.h, hcmcapaign.cpp để hoàn thành bài tập lớn này và đảm bảo hai yêu cầu sau:

   - Chỉ có 1 lệnh **include** trong tập tin hcmcampaign.h là **#include "main.h"** và một include trong tập tin hcmcampaign.cpp là **#include "hcmcampaign.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.
   - Hoán thiện các class để đảm bảo các tính chất cơ bản của lập trình hướng đối tượng

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: hcmcapaign.h và hcmcapaign.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 Harmony cho Bài tập lớn

Bài kiểm tra cuối kì của môn học sẽ có một số câu hỏi Harmony với nội dung của BTL.

Sinh viên phải giải quyết BTL bằng khả năng của chính mình. Nếu sinh viên gian lận trong BTL, sinh viên sẽ không thể trả lời câu hỏi Harmony và nhận điểm 0 cho BTL.

Sinh viên **phải** chú ý làm câu hỏi Harmony trong bài kiểm tra cuối kỳ. Các trường hợp không làm sẽ tính là 0 điểm cho BTL, và bị không đạt cho môn học. **Không chấp nhận giải thích và không có ngoại lệ.**

8 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 GIẢI THÍCH NÀO VÀ KHÔNG CÓ NGOẠI LỆ!

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.

9 Thay đổi so với phiên bản trước

- Thêm yêu cầu về làm tròn về ngưỡng khi vượt ngưỡng
- Các phương thức **str**: Bổ sung mô tả các thuộc tính được cách nhau bởi dấu phẩy
- Bổ sung phương thức **str** cho class BattleField
- Thay đổi quy định về capacity của class UnitList
- Điều chỉnh mô tả của hàm **getEffect** của địa hình sông sâu
- Điều chỉnh công thức tính *attackScore* của các phương tiện
- Điều chỉnh mô tả class **HCMCampaign**

10 Tiểu kết

Cuộc tổng tiến công và nổi dậy mùa Xuân năm 1975 đã thành công rực rỡ. Ngày 30/04/1975, chính quyền Sài Gòn tuyên bố đầu hàng vô điều kiện, miền Nam được giải phóng, đất nước hoàn toàn thống nhất, mở ra kỷ nguyên mới cho dân tộc, kỷ nguyên độc lập, hoà bình, tự chủ,

tự cường.

Nhân kỷ niệm 50 năm giải phóng miền Nam, thống nhất đất nước (30/04/1975 - 30/04/2025), hy vọng qua Bài tập lớn này các bạn đã được ôn lại lịch sử hào hùng của dân tộc, qua đó hun đúc tinh thần yêu nước để tiếp tục hành trình kiến thiết nước nhà, dựng xây quê hương trong giai đoạn cả đất nước đang từng bước tiến vào kỷ nguyên vươn mình của dân tộc.

CHÚC CÁC BẠN LÀM BÀI TỐT