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 1

BÍ KÍP LUYỆN RỒNG (Phần 1)

Phiên bản 1.0

TP. HỒ CHÍ MINH, THÁNG 07/2025

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:

- Các cấu trúc rẽ nhánh
- Các cấu trúc lặp
- Mảng 1 chiều và mảng 2 chiều
- Xử lý chuỗi ký tự
- Hàm và lời gọi hàm

2 Dẫn nhập

Bài tập lớn này được phóng tác dựa trên phần đầu tiên của bộ phim How to Train Your Dragon do hãng DreamWorks sản xuất, dựa trên loạt tiểu thuyết nổi tiếng cùng tên của tác giả Cressida Cowell.

Câu chuyện diễn ra tại một hòn đảo xa xôi, lạnh giá và đầy sương mù ở vùng biển Bắc, nơi cư dân Viking đã sinh sống suốt nhiều thế hệ. Ngôi làng ấy được gọi là Berk, một nơi hứng chịu gió bão khắc nghiệt quanh năm. Dân làng Berk nổi tiếng là những chiến binh dũng mãnh, kiên cường, luôn chuẩn bị sẵn sàng để đối phó với các cuộc tấn công bất ngờ từ loài rồng - kẻ thù truyền kiếp của họ.

Nhân vật chính của câu chuyện, Hiccup Horrendous Haddock III, là con trai độc nhất của tù trưởng Stoick the Vast. Dù được mong đợi trở thành một chiến binh mạnh mẽ giống cha, Hiccup lại gầy gò, yếu ớt và thường gây rắc rối bởi những phát minh kỳ lạ của mình. Nhưng chính sự thông minh, sáng tạo và tinh thần khác biệt ấy đã thay đổi tất cả.

Trong một đêm tấn công của rồng, Hiccup đã dùng phát minh riêng của mình để bắn hạ thành công một con rồng bí ẩn và vô cùng nguy hiểm - Night Fury, hay còn được dân làng gọi là "Ác mộng ban đêm". Khi Hiccup tìm thấy nó, thay vì giết chết như truyền thống Viking, cậu lại quyết định cứu chữa và chăm sóc. Dần dần, Hiccup và con rồng - được cậu đặt tên là Toothless (Răng Sún) - đã xây dựng một mối quan hệ dựa trên niềm tin và sự tôn trọng lẫn nhau.

Trong quá trình huấn luyện và tìm hiểu, Hiccup nhận ra rằng loài rồng không hề hung ác như

người Berk vẫn tin. Thực chất, chúng bị ép buộc phải cướp thức ăn để dâng lên một con rồng khổng lồ, đáng sợ hơn rất nhiều. Khi phát hiện sự thật này, Hiccup cùng Toothless đã quyết tâm thay đổi cách nhìn của cả ngôi làng, chứng minh rằng rồng có thể trở thành những người bạn đồng hành trung thành, dũng cảm và không kém phần đáng yêu.

Bên cạnh Hiccup, còn có nhóm bạn thân thiết: Astrid - cô gái mạnh mẽ, quyết đoán; Snotlout - tự tin và liều lĩnh; Fishlegs - hiền lành, am hiểu về rồng; và cặp song sinh tinh nghịch Ruffnut và Tuffnut. Họ không chỉ cùng nhau tham gia khóa huấn luyện khắc nghiệt ở đấu trường Lửa, mà còn dần học cách gắn bó, hợp tác và khám phá sức mạnh tiềm ẩn của bản thân lẫn những chú rồng.

Từ những cuộc đối đầu căng thẳng đến những bài học quý giá về tình bạn, lòng dũng cảm và sự cảm thông, câu chuyện mở ra một hành trình phiêu lưu đầy cảm hứng. Và chính từ hành trình ấy, một tương lai mới cho Berk bắt đầu - nơi mà con người và rồng không còn là kẻ thù, mà trở thành những chiến binh sát cánh bên nhau, cùng bảo vệ ngôi làng thân yêu của mình.

3 Dữ liệu nhập

3.1 Mô tả dữ liệu nhập

Dữ liệu đầu vào của chương trình được lưu trong một tệp văn bản (.txt) theo định dạng dòng của Windows, Linux hoặc macOS. Tên tệp được lưu trong biến `file_input`. Tệp này chứa các thông tin sau, theo thứ tự từng dòng:

```
["Toothless",␣"Stormfly",␣"Hookfang",␣"Meatlug",␣"Barf␣and␣Belch"]
["Night␣Fury",␣"Deadly␣Nadder",␣"Monstrous␣Nightmare",␣"Gronckle",␣"Hideous␣Zippleback"]
[9;␣8;␣7;␣7;␣6]
[3;␣2;␣5;␣1;␣4]
[100;␣80;␣90;␣70;␣95]
["Hiccup",␣"Astrid",␣"Snotlout",␣"Fishlegs",␣"Ruff␣and␣Tuff"]
5
```

Tệp đầu vào thể hiện lần lượt và tuần tự thông tin chi tiết của N con rồng bao gồm: tên, loại rồng, tính khí, số đạn có thể bắn ra và sát thương. Cụ thể như sau

- **Dòng 1 -** `dragon_names`: Danh sách theo thứ tự tên các con rồng dưới dạng chuỗi ký tự.
- **Dòng 2 -** `dragon_types`: Danh sách theo thứ tự loại rồng của từng con rồng dưới dạng chuỗi. Các giá trị hợp lệ:
  - `"Night Fury"` $\rightarrow$ 1

  - `"Deadly Nadder"` → 2
  - `"Monstrous Nightmare"` → 3
  - `"Gronckle"` → 4
  - `"Hideous Zippleback"` → 5

  Nếu tên loại rồng không nằm trong các giá trị trên, ánh xạ mặc định về loại "Night Fury".
- **Dòng 3 -** `dragon_temperament`: Danh sách theo thứ tự tính khí của từng con rồng dưới dạng số nguyên. Mỗi phần tử phải nằm trong khoảng $[0, 10]$. Nếu vượt giới hạn, gán về biên gần nhất.
- **Dòng 4 -** `ammo_counts`: Danh sách theo thứ tự số đạn bắn có thể bắn ra trong 1 lượt nạp của từng con rồng dưới dạng số nguyên. Mỗi phần tử phải nằm trong khoảng $[0, 1000]$. Nếu vượt giới hạn, gán về biên gần nhất.
- **Dòng 5 -** `dragon_damages`: Danh sách theo thứ tự sát thương của từng loại rồng dưới dạng số nguyên. Mỗi phần tử phải nằm trong khoảng $[0, 1000]$. Nếu vượt giới hạn, gán về biên gần nhất. Mảng có đúng 5 phần tử, đại diện sát thương của loại rồng 1 đến 5.
- **Dòng 6 -** `rider_names`: Danh sách theo thứ tự tên chiến binh cưỡi rồng của từng con rồng dưới dạng chuỗi ký tự.
- **Dòng 7 -** `N`: Số lượng con rồng

Lưu ý: Các phần tử trong mảng dưới dạng chuỗi ký tự chỉ được phép chứa các ký tự sau: chữ cái (A–Z, a–z), chữ số (0–9), dấu cách, và 8 ký tự đặc biệt sau `! @ # $ % ^ & *`.

4 Nhiệm vụ

4.1 Nhiệm vụ 1: Đọc dữ liệu đầu vào (3.0 điểm)

Trước khi thực hiện các nhiệm vụ huấn luyện và thi đấu, các huấn luyện viên cần thu thập và xử lý thông tin về các cá thể rồng, loại rồng, số lượng đạn, sát thương và tên chiến binh cưỡi. Nhiệm vụ đầu tiên là xây dựng một hàm đọc và xử lý dữ liệu từ tệp đầu vào.

Mô tả hàm

- **Tên hàm**:

```
int readFile(const string filename,
             Dragon dragons[],
             int dragonDamages[5],
```

```
            int &N);
```

- **Yêu cầu**: Đọc dữ liệu từ tệp đầu vào và lưu thông tin về từng cá thể rồng vào mảng `dragons`, cũng như đọc mảng sát thương `dragonDamages`. Biến `N` sẽ được gán với số lượng rồng thực tế sau khi đọc file.
- **Cấu trúc dữ liệu**:

```
struct Dragon {
    string name;
    int type;
    int temperament;
    int ammoCount;
    string riderName;
};
```

- **Tham số đầu vào**:
  - `filename`: Tên tệp chứa dữ liệu đầu vào.
  - `dragons[]`: Mảng chứa thông tin các cá thể rồng (tên, loại, tính khí, số đạn, và chiến binh cưỡi).
  - `dragonDamages[5]`: Mảng chứa sát thương của 5 loại rồng (từ loại 1 đến loại 5).
  - `N`: Sẽ được gán là số lượng rồng thực tế sau khi đọc file thành công.
- **Giá trị trả về**: Mã lỗi hoặc thành công, theo bảng sau:
  1. `1`: Thành công
  2. `2`: Tệp không có đuôi `.txt`
  3. `3`: Tệp không tồn tại
  4. `4`: Thiếu dòng
  5. `5`: Mảng `dragonNames[]` thiếu hoặc dư phần tử
  6. `6`: Mảng `dragonTypes[]` thiếu hoặc dư phần tử
  7. `7`: Mảng `dragonTemperament[]` thiếu hoặc dư phần tử
  8. `8`: Mảng `ammoCounts[]` thiếu hoặc dư phần tử
  9. `9`: Mảng `dragonDamages[5]` không có đúng 5 phần tử
  10. `10`: Mảng `riderNames[]` thiếu hoặc dư phần tử
  11. `100 + i`: `i` là index của phần tử thứ 2 trong `dragonNames[]` chứa ký tự đặc biệt.
  12. `500 + k`: `k` là tổng số ký tự đặc biệt trong toàn bộ `dragonTypes[]`.

13. `900 + i`: `i` là index của phần tử đầu tiên trong `riderNames[]` có từ 2 dấu cách liên tiếp trở lên.
14. `1000 + k`: `k` là tổng số dấu cách dư (>=2 liên tiếp) trong toàn bộ `dragonNames[]`.

- **Giả định**: File đầu vào nếu có lỗi thì chỉ có 1 lỗi duy nhất xuất hiện trong các lỗi đã được liệt kê ở phần "Giá trị trả về" bên trên. Không xem xét các trường hợp lỗi khác ngoài danh sách đó.
- **Lưu ý**: Nếu sinh viên thực hiện sai nhiệm vụ này, dẫn đến dữ liệu rồng bị sai hoặc không đọc đúng định dạng, thì toàn bộ các nhiệm vụ tiếp theo sẽ bị trừ **50% số điểm** nếu có sử dụng các dữ liệu đó.

Ví dụ 4.1

Ví dụ thành công (Trả về 1)

Dữ liệu đầu vào:

```
["Toothless",␣"Stormfly",␣"Hookfang",␣"Meatlug",␣"Barf␣and␣Belch"]
["Night␣Fury",␣"Deadly␣Nadder",␣"Monstrous␣Nightmare",␣"Gronckle",␣"Hideous␣Zippleback"]
[9;␣8;␣7;␣7;␣6]
[3;␣2;␣5;␣1;␣4]
[100;␣80;␣90;␣70;␣95]
["Hiccup",␣"Astrid",␣"Snotlout",␣"Fishlegs",␣"Ruff␣and␣Tuff"]
5
```

Kết quả trả về: 1

Ví dụ 4.2

Ví dụ lỗi thiếu dòng (Trả về 4)

Dữ liệu đầu vào:

```
["Toothless",␣"Stormfly",␣"Hookfang",␣"Meatlug",␣"Barf␣and␣Belch"]
["Night␣Fury",␣"Deadly␣Nadder",␣"Monstrous␣Nightmare",␣"Gronckle",␣"Hideous␣Zippleback"]
[9;␣8;␣7;␣7;␣6]
[3;␣2;␣5;␣1;␣4]
[100;␣80;␣90;␣70;␣95]
["Hiccup",␣"Astrid",␣"Snotlout",␣"Fishlegs",␣"Ruff␣and␣Tuff"]
```

Kết quả trả về: 4

Ví dụ 4.3

Ví dụ lỗi thiếu phần tử trong dragonNames (Trả về 5)

Dữ liệu đầu vào:

```
["Toothless", "Stormfly", "Hookfang", "Meatlug"]
["Night Fury", "Deadly Nadder", "Monstrous Nightmare", "Gronckle", "Hideous Zippleback"]
[9; 8; 7; 7; 6]
[3; 2; 5; 1; 4]
[100; 80; 90; 70; 95]
["Hiccup", "Astrid", "Snotlout", "Fishlegs", "Ruff and Tuff"]
5
```

Kết quả trả về: 5

Ví dụ 4.4

Ví dụ lỗi dragonDamages không có đúng 5 phần tử (Trả về 9)

Dữ liệu đầu vào:

```
["Toothless", "Stormfly", "Hookfang", "Meatlug", "Barf and Belch"]
["Night Fury", "Deadly Nadder", "Monstrous Nightmare", "Gronckle", "Hideous Zippleback"]
[9; 8; 7; 7; 6]
[3; 2; 5; 1; 4]
[100; 80; 90; 70]
["Hiccup", "Astrid", "Snotlout", "Fishlegs", "Ruff and Tuff"]
5
```

Kết quả trả về: 9

Ví dụ 4.5

Ví dụ lỗi ký tự đặc biệt trong phần tử thứ 2 của dragonNames (Trả về 101)

Dữ liệu đầu vào:

```
["Toothless", "Storm@fly", "Hookfang", "Meatlug", "Barf and Belch"]
["Night Fury", "Deadly Nadder", "Monstrous Nightmare", "Gronckle", "Hideous Zippleback"]
[9; 8; 7; 7; 6]
[3; 2; 5; 1; 4]
[100; 80; 90; 70; 95]
["Hiccup", "Astrid", "Snotlout", "Fishlegs", "Ruff and Tuff"]
5
```

Kết quả trả về: 101

Ví dụ 4.6

Ví dụ lỗi có tổng 3 ký tự đặc biệt trong dragonTypes (Trả về 503)

Dữ liệu đầu vào:

```
["Toothless", "Stormfly", "Hookfang", "Meatlug", "Barf and Belch"]
["Night\$ Fury", "Deadly! Nadder", "Monstrous Nightmare", "Gr@onckle", "Hideous Zippleback"]
[9; 8; 7; 7; 6]
[3; 2; 5; 1; 4]
[100; 80; 90; 70; 95]
["Hiccup", "Astrid", "Snotlout", "Fishlegs", "Ruff and Tuff"]
5
```

Kết quả trả về: 503

Ví dụ 4.7

Ví dụ lỗi riderNames có phần tử chứa 2 dấu cách liên tiếp (Trả về 904)

Dữ liệu đầu vào:

```
["Toothless", "Stormfly", "Hookfang", "Meatlug", "Barf and Belch"]
["Night Fury", "Deadly Nadder", "Monstrous Nightmare", "Gronckle", "Hideous Zippleback"]
[9; 8; 7; 7; 6]
[3; 2; 5; 1; 4]
[100; 80; 90; 70; 95]
["Hiccup", "Astrid", "Snotlout", "Fishlegs", "Ruff  and Tuff"]
5
```

Kết quả trả về: 904

Ví dụ 4.8

Ví dụ lỗi tổng cộng 3 dấu cách dư trong dragonNames (Trả về 1003)

Dữ liệu đầu vào:

```
["Toothless", "Stormfly", "Hookfang", "Meat  lug", "Barf   and Belch"]
["Night Fury", "Deadly Nadder", "Monstrous Nightmare", "Gronckle", "Hideous Zippleback"]
[9; 8; 7; 7; 6]
[3; 2; 5; 1; 4]
[100; 80; 90; 70; 95]
["Hiccup", "Astrid", "Snotlout", "Fishlegs", "Ruff and Tuff"]
5
```

Kết quả trả về: 1003

4.2 Nhiệm vụ 2: Tìm rồng mạnh nhất có tính khí vượt ngưỡng (1.0 điểm)

Trong nhiệm vụ này, sinh viên cần xây dựng một hàm để xác định tên của con rồng có chỉ số **sức mạnh cao nhất** trong số các rồng có **tính khí** lớn hơn hoặc bằng một ngưỡng nhất định `T`. Thông tin về các rồng được lưu trữ trong mảng cấu trúc `Dragon`, bao gồm loại rồng, số lượng đạn, tính khí và tên chiến binh cưỡi.

Mô tả hàm

- **Tên hàm**:

```
string findKthStrongestDragon(Dragon dragons[],
                              int dragonDamages[5],
                              int N,
                              int T);
```

- **Tham số đầu vào**:
  - `dragons[]`: Mảng chứa các cá thể rồng.
  - `dragonDamages[5]`: Mảng chứa sát thương tương ứng với từng loại rồng (loại 1 đến 5).
  - `N`: Số lượng rồng trong mảng.
  - `T`: Ngưỡng tối thiểu của chỉ số tính khí.
- **Giá trị trả về**:
  - Tên của con rồng có chỉ số sức mạnh cao nhất trong số các rồng có `tính khí` $\geq$ `T`.
  - Nếu không có rồng nào thỏa điều kiện, trả về chuỗi `"None"`.

Công thức tính sức mạnh (power)

- **Night Fury** (type = 1):
  `power = (ammo × damage) + temperament × 3`
- **Deadly Nadder** (type = 2):
  `power = (ammo × damage) + temperament × 2`
- **Monstrous Nightmare** (type = 3):
  `power = (ammo × damage) + temperament`
- **Gronckle** (type = 4):
  `power = (ammo × damage) + temperament / 2`

- **Hideous Zippleback** (type = 5):
  `power = (ammo × damage × 0.9) + temperament × 1.5`

Lưu ý

- Nếu có nhiều con rồng có cùng chỉ số sức mạnh cao nhất, ưu tiên con rồng xuất hiện trước trong mảng ban đầu.
- Chỉ trả về tên rồng là chuỗi duy nhất.

Ví dụ 4.9

Ví dụ 1: Có rồng thỏa điều kiện, trả về rồng mạnh nhất

Dữ liệu đầu vào:

```
["Toothless",␣"Stormfly",␣"Hookfang",␣"Meatlug",␣"Barf␣and␣Belch"]
[1;␣2;␣3;␣4;␣5]
[9;␣8;␣7;␣4;␣6]
[3;␣2;␣5;␣1;␣4]
[100;␣80;␣90;␣70;␣95]
["Hiccup",␣"Astrid",␣"Snotlout",␣"Fishlegs",␣"Ruff␣and␣Tuff"]
5
```

Với `T = 6`, các rồng thỏa điều kiện có sức mạnh:

- Toothless (type 1): $(3100) + 93 = 327$
- Stormfly (type 2): $(280) + 82 = 192$
- Hookfang (type 3): $(590) + 7 = 457$
- Barf and Belch (type 5): $(4950.9) + 61.5 = 375$

Meatlug bị loại vì tính khí < 6.

Kết quả trả về: `"Hookfang"`

Ví dụ 4.10

Ví dụ 2: Không có rồng nào thỏa tính khí $\geq$ T

Dữ liệu đầu vào:

```
["Toothless",␣"Stormfly",␣"Hookfang"]
[1;␣2;␣4]
[3;␣2;␣4]
[1;␣2;␣3]
[100;␣80;␣70;␣60;␣50]
["Hiccup",␣"Astrid",␣"Snotlout"]
3
```

Với `T = 6`, không có con rồng nào thỏa điều kiện.

Kết quả trả về: `"None"`

4.3 Nhiệm vụ 3.1: Đánh giá độ tương thích (1 điểm)

Tại Đấu trường Lửa, các chiến binh phải chuẩn bị tổ đội rồng thật mạnh để thi đấu. Tuy nhiên, không phải cứ rồng mạnh là có thể thuần phục. Mỗi con rồng có tính khí riêng (nóng nảy, lạnh lùng, khôn ngoan...), còn chiến binh thì có kỹ năng điều khiển. Nếu điểm tương thích quá thấp, rồng sẽ phản kháng và không nghe lời.

4.3.1 Mô tả hàm

- **Tên hàm**:

```
void compatibilityCheck(Dragon dragons[], string warriorName,
                        int warriorSkill);
```

- **Tham số đầu vào**:
  - `dragons[]`: Mảng chứa thông tin các cá thể rồng (tên, loại, tính khí, số đạn, và chiến binh cưỡi).
  - `warriorName`: Tên chiến binh.
  - `warriorSkill`: Điểm kỹ năng của chiến binh.
- **Kết quả trả về**: In ra bảng Mức độ tương thích của chiến binh với tất cả các cá thể rồng hiện có.

4.3.2 Quy tắc phân loại

Công thức:

```
compatibility = (10 - | dragon.temperament - warrior.skill |) / 2
```

- **Not Compatible:** compatibility ≤ 4
- **Compatible:** compatibility > 4

4.3.3 Định dạng in ra

Ví dụ 4.11

Chương trình cần in ra theo format sau:

```
Warrior     Dragon      Compatibility     Review
Hiccup      Toothless   5.00              Compatible
Hiccup      Stormfly    3.95              Not Compatible
...
```

Sinh viên sẽ được cung cấp hàm printCompatibilityTable() để in ra format bảng trên.

4.4 Nhiệm vụ 3.2: Tìm đồng đội (1 điểm)

Nhóm Warriors bắt đầu tìm kiếm những chú rồng phù hợp cho từng thành viên. Để có thể cùng nhau tham gia trận chiến, mỗi chiến binh cần tìm một chú rồng có độ tương thích cao nhất với mình. Tuy nhiên, không phải rồng nào cũng có thể được chọn nhiều lần: mỗi chú rồng chỉ có thể được ghép với duy nhất một chiến binh.

4.4.1 Mô tả hàm

- **Tên hàm**:

```
void buddyMatching(Dragon dragons[], string warriors[][3]);
```

- **Tham số đầu vào**:

  - `dragons[]`: Mảng chứa thông tin các cá thể rồng (tên, loại, tính khí, số đạn, và chiến binh cưỡi).
  - `warriors`: Danh sách chiến binh. Mỗi chiến binh bao gồm 3 thông tin: Tên, điểm kỹ năng, ID.

- **Kết quả trả về**: In ra bảng kết quả ghép cặp Chiến binh - Rồng.

4.4.2 Mô tả chi tiết

Quy trình:

- Tính điểm tương thích giữa mỗi chiến binh và từng chú rồng. Sinh viên sử dụng hàm compatibilityCheck() để hỗ trợ cho nhiệm vụ này.
- Với mỗi chiến binh (xét lần lượt theo thứ tự xuất hiện trong danh sách warriors), chọn chú rồng có điểm tương thích cao nhất trong danh sách rồng khả thi.
- Một khi chú rồng đã được chọn bởi một chiến binh, nó không thể được chọn lại cho các chiến binh sau.

Điều kiện chọn:

- Chỉ được gán mỗi chú rồng cho tối đa một chiến binh.
- Ưu tiên chọn rồng có điểm tương thích cao nhất.
- Nếu có nhiều rồng cùng điểm cao nhất, ưu tiên rồng đứng trước trong danh sách.

4.4.3 Định dạng in ra

Ví dụ 4.12

Chương trình cần in ra theo format sau:

```
Warrior      Dragon       Compatibility      Review
Hiccup       Toothless    5.00               Compatible
Astrid       Stormfly     4.95               Compatible
Snotlout     Hookfang     5.00               Compatible
Fishlegs     Meatlug      4.75               Compatible
RuffNTuff    BarfBelch    4.60               Compatible
```

Sinh viên sẽ được cung cấp hàm printCompatibilityTable() để in ra format bảng trên.

4.5 Nhiệm vụ 4: VÒNG 1 - THỬ THÁCH ĂN Ý (1 điểm)

Sau khi mỗi chiến binh đã chọn được chú rồng phù hợp, nhóm Warriors chính thức bước vào vòng 1 tại đấu trường lửa. Ở vòng này, các cặp chiến binh - rồng sẽ cùng nhau tham gia thử thách nhằm kiểm tra sự ăn ý trước khi được phép bước vào vòng tiếp theo.

Mục tiêu: Viết hàm tính tổng thời gian mà mỗi chiến binh cần để hoàn thành thử thách (bao gồm thời gian đi, thời gian quay về, và thời gian di chuyển giữa các ô).

4.5.1 Mô tả hàm

- **Tên hàm**:

```
void computeChallengeTime(string warriors[][3], int map[10][10]);
```

- **Tham số đầu vào**:
  - `warriors`: Danh sách chiến binh. Mỗi chiến binh bao gồm 3 thông tin: Tên, điểm kỹ năng, ID.
  - `map`: Bản đồ của thử thách.
- **Kết quả trả về**: In ra bảng Mức độ tương thích của chiến binh với tất cả các cá thể rồng hiện có.

4.5.2 Mô tả chi tiết

Bối cảnh thử thách: Các cặp sẽ phải bay qua bản đồ kho báu được xây dựng trong khu rừng sau núi làng Berk. Bản đồ được chia thành 100 ô (10 x 10), mỗi ô đại diện cho một khu vực nhỏ.

Quy định về bản đồ:

- Mỗi ô có thể chứa một vật phẩm có mã số trùng với một chiến binh cụ thể (mã số từ 0 đến 4), hoặc không chứa vật phẩm dành cho ai cả.
- Mỗi ô có một ID được tính bằng công thức:

```
ID = (row + col) % 5
```

- Chiến binh chỉ được phép thu thập vật phẩm có mã số trùng với mã số của mình.

- Ngoài ra, chiến binh không thể bay vào ô có ID bằng mã số của họ, dù ô đó có vật phẩm hay không.

Quy trình lấy vật phẩm:

- Mỗi khi nhặt xong một vật phẩm, chiến binh phải quay lại vị trí xuất phát để thả vật phẩm xuống trước khi quay lại tiếp tục.
- Nếu ô hiện tại không chứa vật phẩm, chiến binh chỉ bay sang ô kế tiếp, và không cần quay lại.

Cách tính thời gian di chuyển:

- Thời gian từ vị trí xuất phát đến ô chứa vật phẩm (giây):

```
time = [1 + (row + col * 2)] * 5
```

- Thời gian bay về từ ô đó về lại vị trí xuất phát:

```
time = [1 + (row + col * 2)] * 5
```

- Nếu không nhặt vật phẩm, chỉ mất 5 giây để di chuyển sang ô kế tiếp.

Ví dụ 4.13

Ví dụ bản đồ (ID trong mỗi ô):

```
0 1 2 3 4 0 1 2 3 4
1 2 3 4 0 1 2 3 4 0
2 3 4 0 1 2 3 4 0 1
3 4 0 1 2 3 4 0 1 2
4 0 1 2 3 4 0 1 2 3
0 1 2 3 4 0 1 2 3 4
1 2 3 4 0 1 2 3 4 0
2 3 4 0 1 2 3 4 0 1
3 4 0 1 2 3 4 0 1 2
4 0 1 2 3 4 0 1 2 3
```

4.5.3 Định dạng in ra

- Một bảng thống kê thời gian hoàn thành của tất cả chiến binh.
- Chọn ra 4 chiến binh nhanh nhất (loại người chậm nhất).
- In bảng này, sắp xếp theo thời gian từ ít đến nhiều.

Ví dụ 4.14

```
Warrior       Total time (secs)
Hiccup        123
Fishlegs      130
Astrid        150
Snotlout      175
```

4.6 Nhiệm vụ 5.1: Đánh giá sát thương gây ra (1 điểm)

Sau khi hoàn thành vòng Synergy Test, các chiến binh và rồng bước vào vòng 2 để kiểm tra khả năng tấn công. Mỗi chiến binh sẽ phối hợp cùng rồng của mình để tính toán lượng sát thương tối đa có thể gây ra cho đối thủ. Điểm sát thương được tính dựa trên sát thương cơ bản của rồng, số lượng đòn tấn công, và kỹ năng của chiến binh. Đây là yếu tố quan trọng để đánh giá khả năng chiến đấu thực sự của từng cặp đôi.

4.6.1 Mô tả hàm

- **Tên hàm**:

```
void fighterDamage(Dragon dragons[], string warriors[][3], int teamsDamage[]);
```

- **Tham số đầu vào**:
  - `dragons[]`: Mảng chứa thông tin các cá thể rồng (tên, loại, tính khí, số đạn, và chiến binh cưỡi).
  - `warriors`: Danh sách chiến binh. Mỗi chiến binh bao gồm 3 thông tin: Tên, điểm kỹ năng, ID.
  - `teamsDamage`: Danh sách sức mạnh của các chiến binh.
- **Kết quả trả về**: Lưu thông tin sát thương có thể gây ra của từng nhóm (cặp) Chiến binh - Rồng vào mảng `teamsDamage`. Đồng thời in ra theo format bên dưới.

4.6.2 Công thức:

```
damage = (dragon.damage * dragon.ammoCount) + (warrior.skill * 5)
```

4.6.3 Định dạng in ra

Ví dụ 4.15

Chương trình cần in ra theo format sau:

```
Hiccup-Toothless: damage = 125
Astrid-Stormfly: damage = 98
...
```

4.7 Nhiệm vụ 5.2: VÒNG 2 - TÌM KIẾM DI SẢN CỦA BERK (2 điểm)

Sau khi vượt qua thử thách đầu tiên và tìm được người bạn rồng đồng hành, các chiến binh tiến vào vòng tiếp theo: Tìm kiếm di sản của Berk.

Tương truyền, sâu trong khu rừng cổ xưa của làng Berk tồn tại những kho báu vô giá, được các trưởng lão cất giấu để bảo vệ tinh hoa của bộ tộc. Mỗi kho báu chứa đựng biểu tượng danh dự riêng biệt dành cho từng chiến binh - lá cờ khắc tên họ.

Để chạm đến kho báu, các chiến binh phải vượt qua mê cung được canh giữ bởi hàng trăm con rồng khác. Mỗi bước đi là một trận chiến cân não: đối đầu với những chú rồng, đấu trí và đấu lực để bảo toàn điểm số danh dự. Không chỉ đòi hỏi sức mạnh, mà còn cần sự khéo léo và quyết tâm sắt đá để tiến đến đích nhanh nhất.

Liệu ai sẽ là người khẳng định bản lĩnh, giành được di sản và vinh danh tên tuổi mình trên bầu trời Berk? Cuộc truy tìm kho báu đã bắt đầu - và chỉ những chiến binh dũng cảm nhất mới có thể bước qua thử thách này!

4.7.1 Tìm địa điểm kho báu (0.5 điểm)

- **Tên hàm**:

```
void findHeritageLocation(int map[N][N], int &heritageX, int &heritageY);
```

- **Yêu cầu**: Kho báu được đặt tại vị trí duy nhất gọi là điểm **Saddle Point** trên bản đồ. Một Saddle Point là phần tử trong ma trận thỏa mãn điều kiện: nó là giá trị nhỏ nhất trong hàng của nó, nhưng đồng thời là giá trị lớn nhất trong cột của nó.
- **Tham số đầu vào**:
  - `map`: Bản đồ của thử thách.
  - `heritageX`: Toạ độ X của kho báu.
  - `heritageY`: Toạ độ Y của kho báu.
- **Đầu ra**: Giá trị của `heritageX` và `heritageY` sẽ được cập nhật trực tiếp.

4.7.2 Tìm địa điểm chìa (0.5 điểm)

- **Tên hàm**:

```
void findKeyLocation(int map[N][N], int &keyX, int &keyY);
```

- **Yêu cầu**: Để mở được kho báu, chiến binh trước tiên phải tìm và nhặt được chiếc chìa khóa. Kho báu được giấu tại khu vực rồng chiến thần, được xác định là ô trung tâm của ma trận con 3x3 có tổng giá trị lớn nhất trong toàn bộ bản đồ.
- **Tham số đầu vào**:
  - `map`: Bản đồ của thử thách.
  - `keyX`: Toạ độ X của chìa khoá.
  - `keyY`: Toạ độ Y của chìa khoá.
- **Đầu ra**: Giá trị của `keyX` và `keyY` sẽ được cập nhật trực tiếp.

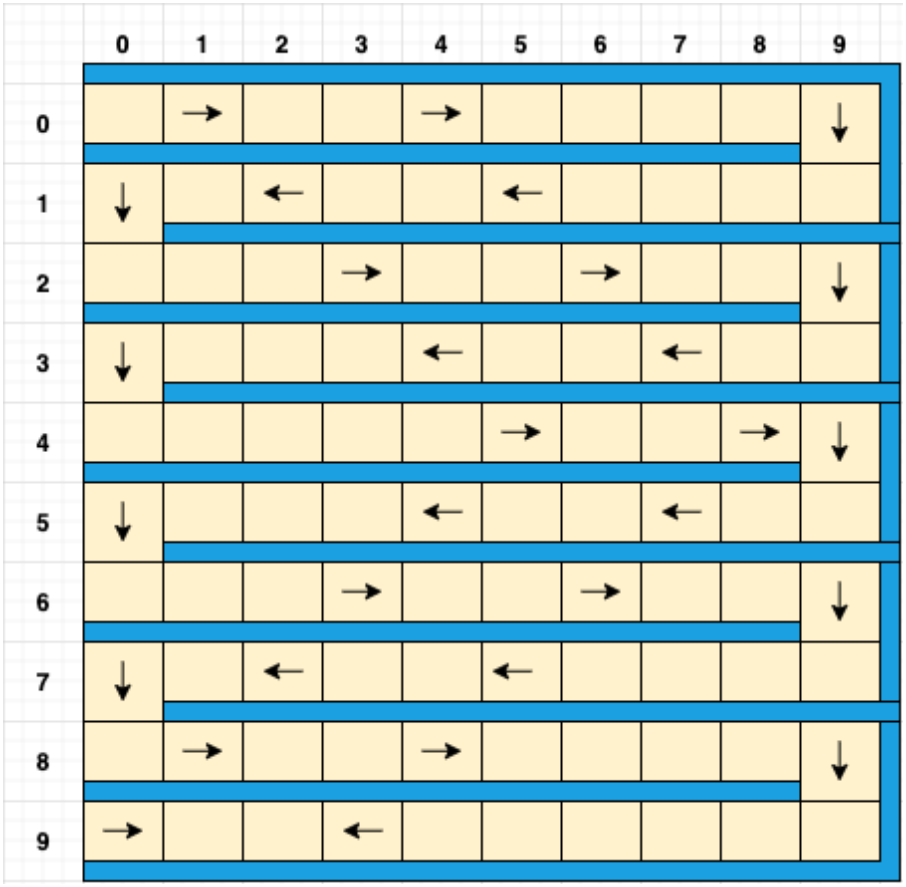
4.7.3 Thời gian hoàn thành thử thách (1 điểm)

- **Tên hàm**:

```
void totalTime(int map[N][N], int warriorDamage, int HP);
```

- **Yêu cầu**: Tính tổng thời gian hoàn thành thử thách của chiến binh.
- **Tham số đầu vào**:
  - `map`: Bản đồ của thử thách.
  - `warriorDamage`: Sức mạnh của chiến binh.
  - `HP`: Số máu (HP) của chiến binh. Khi HP = 0, chiến binh sẽ bị loại và thử thách lập tức kết thúc.

- **Bản đồ**:
  - Bản đồ được chia thành ma trận 2 chiều kích thước 10×10.
  - Mỗi ô trên bản đồ có thể là ô di chuyển, ô rồng con, ô đặc biệt, vị trí kho báu hoặc vị trí chìa khóa.
  - Các chiến binh chỉ được phép di chuyển qua các ô trên bản đồ theo kiểu zig-zag, không được phép nhảy hoặc vượt qua tường. Ví dụ minh họa xem hình bên dưới:



Hình 1: Mô tả ảnh minh hoạ

- **Luật di chuyển (theo bản đồ)**:
  - Luôn bắt đầu từ ô [0][0], di chuyển sang phải đến [0][9], sau đó xuống [1][9], tiếp tục sang trái đến [1][0], rồi xuống [2][0], và cứ thế tiếp tục cho đến hàng cuối cùng [9][0]. Nếu đến [9][0] vẫn chưa hoàn thành thử thách, người chơi cần đảo chiều di chuyển để tiếp tục thử thách.
  - Nếu vị trí của chìa khóa nằm xa hơn kho báu (so với vị trí xuất phát [0][0]), sau khi nhặt được chìa khóa, người chơi cần đảo chiều di chuyển để quay lại kho báu. Khi đó, quãng đường quay lại sẽ nhanh hơn vì các ô đã trở thành ô đất thường.
  - Để thích ứng với bất kỳ sự thay đổi nào do các ô đặc biệt gây ra, chương trình cần

có cơ chế kiểm tra và so sánh vị trí hiện tại của chiến binh với vị trí mục tiêu (chìa khóa hoặc kho báu). Từ đó, chương trình có thể tự động điều chỉnh hướng đi tại mỗi bước, đảm bảo người chơi luôn tìm được đường di chuyển tối ưu và hợp lệ.

- **Các loại Ô trên bản đồ**:
  - **Ô đất thường**
    - Biểu diễn bằng số 0.
    - Thời gian di chuyển (Cost): 2 giây.
  - **Ô kho báu:** Vị trí được xác định bằng hàm findHeritageLocation().
  - **Ô chìa khóa:** Vị trí được xác định bằng hàm findKeyLocation().
  - **Ô rồng con**
    - Biểu diễn bằng số 1–200 (giá trị này cũng chính là sát thương (ST) của rồng).
    - Nếu Sát thương người chơi $\geq$ ST rồng con $\rightarrow$ thắng, được tiếp tục di chuyển.
    - Nếu Sát thương người chơi $<$ ST rồng con $\rightarrow$ thua, bị trừ 1 HP.
    - Thời gian di chuyển (Cost): 5 giây.
    - Rồng con chỉ tồn tại sau một lần chiến đấu duy nhất. Những lần đi qua sau sẽ được xem như ô đất thường.
  - **Ô đặc biệt - Rồng ảo thuật thời gian**
    - Biểu diễn bằng số 1–200 (giá trị này cũng chính là sát thương (ST) của rồng).
    - Chỉ xuất hiện trên các ô biên và là ô có giá trị lớn nhất trên biên.
    - Xuất hiện tối đa 1 ô trên bản đồ.
    - Nếu Sát thương người chơi $\geq$ ST rồng $\rightarrow$ thắng, tiếp tục di chuyển.
    - Nếu Sát thương người chơi $<$ ST rồng $\rightarrow$ thua, bị trừ 2 HP. Người chơi bị đẩy lùi lên trên một hàng, giữ nguyên cột. Nếu người chơi đang ở hàng 0, quay về vị trí [0][0]. Các ô đã đi qua không tái tạo lại (vẫn giữ trạng thái ô đất thường).
    - Ô đặc biệt chỉ hiệu lực một lần duy nhất. Những lần đi qua sau sẽ được xem như ô đất thường.
    - Thời gian di chuyển (Cost): 10 giây.
  - **Ô đặc biệt - Rồng đảo lộn trật tự**
    - Biểu diễn bằng số 1–200 (giá trị này cũng chính là sát thương (ST) của rồng).
    - Vị trí được xác định là đỉnh núi phép thuật, nghĩa là:
      - Giá trị cao hơn giá trị của 4 ô liền kề (trên, dưới, trái, phải).
      - Không xuất hiện ở đường biên.
      - Nếu có nhiều đỉnh núi, chọn đỉnh núi xuất hiện cuối cùng khi duyệt bản đồ.
    - Xuất hiện tối đa 1 ô trên bản đồ.

* Nếu Sát thương người chơi $\geq$ ST rồng $\rightarrow$ thắng, giữ nguyên trật tự bản đồ, tiếp tục di chuyển.
* Nếu Sát thương người chơi $<$ ST rồng $\rightarrow$ thua, bị trừ 2 HP. Vị trí của chiến binh bị đảo lộn, tức: `[row][col] → [col][row]`.
* Ô đặc biệt chỉ hiệu lực một lần duy nhất. Những lần đi qua sau sẽ được xem như ô đất thường.
* Thời gian di chuyển (Cost): 10 giây.

– **Định dạng in kết quả đầu ra**:

* Tính tổng thời gian di chuyển. In ra tổng thời gian khi thử thách kết thúc.
* In ra số HP còn lại sau khi thử thách kết thúc. Nếu chiến binh hết HP trước khi hoàn thành thử thách, in ra **"Warrior defeated! Challenge failed!"**.
* Lưu trữ và in lộ trình di chuyển đầy đủ. In ra danh sách các tọa độ đã đi qua theo định dạng:

```
(x_i0,y_i0)(x_i1,y_i1)(x_i2,y_i2).................(x_in,y_in)
```

trong đó mỗi cặp (x_i, y_i) biểu diễn vị trí mà chiến binh đã đặt chân lên theo đúng thứ tự di chuyển.

4.8 Ví dụ minh họa:

Ví dụ 4.16

Bản đồ:

```
{  10,  20,  30,  40,  50,  60,  70,  80,  90, 100 },
{  15,  25,  35,  45,  55,  65,  75,  85,  95, 105 },
{  12,  22,  32,  42,  52,  62,  72,  82,  92, 175 },
{ 118, 128, 138,  50,  58,  68,  78,  88,  98, 108 },
{  14,  24,  34,  44,  54,  64,  74,  84,  94, 104 },
{  19,  29,  39,  49,  59,  69,  79,  89,  99, 109 },
{  11,  21,  31,  41,  51,  61,  71,  81,  91, 101 },
{  16,  26,  36,  46,  56,  66,  76, 186,  96, 106 },
{  17,  27,  37,  47,  57,  67, 196,  87,  90, 107 },
{  13,  23,  33,  43,  53,  63,  73, 153,  93, 170 };
```

- Chiến binh có sát thương = 150 và HP = 20.
- Vị trí kho báu: (3,3).
- Vị trí chìa khoá: (8,8).
- Vị trí rồng ảo thuật thời gian: (2,9).
- Vị trí rồng đảo lộn trật tự: (8,6).
- Quãng đường chiến binh đã di chuyển: Start → Rồng ảo thuật thời gian (THUA) → Rồng đảo lộn trật tự (THUA) → Chìa khoá → Kho báu (Kết thúc).

Output

```
Total time: 613 (sec)
Remaining HP: 15
Path: (0,0)(0,1)(0,2)(0,3)(0,4)(0,5)(0,6)(0,7)(0,8)(0,9)(1,9)(1,8)
(1,7)(1,6)(1,5).........(4,1)(4,0)(3,0)(3,1)(3,2)(3,3)
```

4.9 Tạm kết

Sau nhiều ngày luyện tập gian khổ và các trận đấu căng thẳng, cuối cùng những chiến binh trẻ của làng Berk đã tìm được những chú rồng đồng hành mạnh mẽ và trung thành nhất. Cùng nhau, họ không chỉ vượt qua các bài kiểm tra sức mạnh, kỹ năng điều khiển và tinh thần đồng

đội, mà còn chứng minh được ý chí thép và lòng dũng cảm của một chiến binh Viking thực thụ. Sự kết hợp hoàn hảo giữa chiến binh và rồng đã tạo nên những màn bay lượn ngoạn mục, những đòn tấn công đầy uy lực và tinh thần đoàn kết không gì phá vỡ được. Tuy nhiên, hành trình thực sự mới chỉ bắt đầu. Phía trước họ là những thử thách khắc nghiệt hơn, những đối thủ bí ẩn từ phương Bắc lạnh giá và những sinh vật lạ chưa từng xuất hiện. Liệu họ có đủ mạnh mẽ để bảo vệ bộ tộc, khám phá những vùng đất mới và viết tiếp huyền thoại của Berk? Hãy sẵn sàng tinh thần và đón chờ phần 2 của bài tập lớn để cùng nhau khám phá những cuộc phiêu lưu đầy bất ngờ và kịch tính đang chờ đợi phía trước!

Chúc các bạn làm Bài tập lớn vui vẻ!!!

5 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, dragon.h, dragon.cpp, và các file dữ liệu đọc mẫu. Sinh viên phải nộp 2 tập tin là dragon.h và dragon.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 dragon.cpp -I . -std=c++11
   Sinh viên sử dụng câu lệnh sau để chạy chương trình:
   ./main tnc_tc_01_input.txt
   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 dragon.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 dragon.h, dragon.cpp để hoàn thành bài tập lớn này và đảm bảo hai yêu cầu sau:
   - Chỉ có một lệnh **include** trong tập tin `dragon.h` là:

     #include "main.h"
   - Trong tập tin `dragon.cpp`, chỉ có một lệnh include:

     #include "dragon.h"
   - Ngoài hai include trên, không cho phép có bất kỳ lệnh **#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 hàm để hoàn thành BTL này.

6 Nộp bài

Sinh viên chỉ nộp 2 tập tin: dragon.h và dragon.cpp, trước thời hạn được đưa ra trong đường dẫn "Assignment 1 - 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.

7 Một số quy định khác

1. 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.
2. Mọi quyết định của giảng viên phụ trách bài tập lớn là quyết định cuối cùng.
3. Sinh viên không được cung cấp testcase sau khi chấm bài.
4. Nội dung Bài tập lớn sẽ được Harmony với một số câu hỏi trong bài Kiểm tra Cuối kỳ với nội dung tương tự.
5. Nếu sinh viên sử dụng mã nguồn từ các chương trình AI hoặc công cụ tự động tạo mã mà không hiểu rõ về cách hoạt động và ý nghĩa của mã đó, điểm BTL này sẽ là 0 điểm.

8 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 này.

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.**

9 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.

10 Thay đổi so với phiên bản trước

...