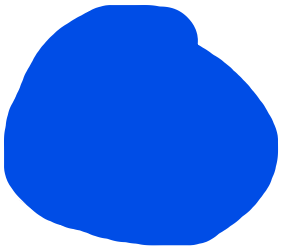


Nhập môn lập trình - CO1003

Bài tập lớn

CỜ CARO

ĐẶC TẢ BÀI TẬP LỚN

Phiên bản 1.1

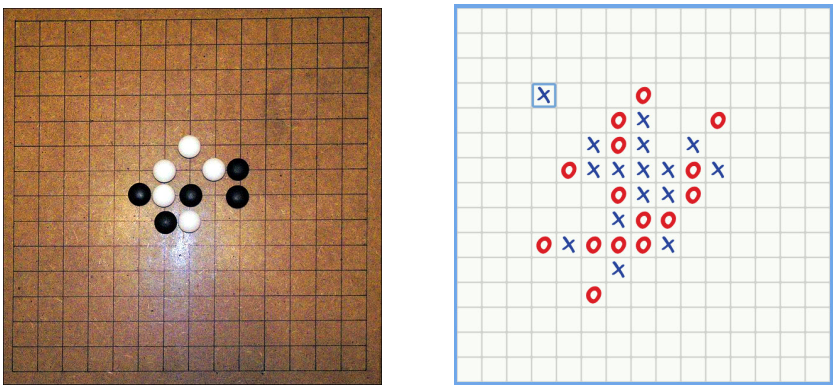
1 Chuẩn đầu ra

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
- Cấu trúc do người dùng tự định nghĩa (struct)

2 Dẫn nhập

Cờ Caro (hay còn được gọi là cờ chéo, Gomoku, Five in a Row, v.v..) là một trò chơi chiến lược trên bàn cờ. Theo truyền thống, cờ Caro được chơi bằng các quân cờ vây (đá đen và trắng) trên bàn cờ vây với kích thước bàn cờ là 15 x 15 ô (Hình 1). Vì các quân cờ thường không có nhu cầu bị di chuyển hay loại khỏi bàn cờ, nên cờ Caro cũng có thể được chơi bằng giấy và bút chì. Trong bài tập lớn này, bạn sẽ hiện thực lại trò chơi cờ Caro với một số luật chơi được cung cấp bên dưới.



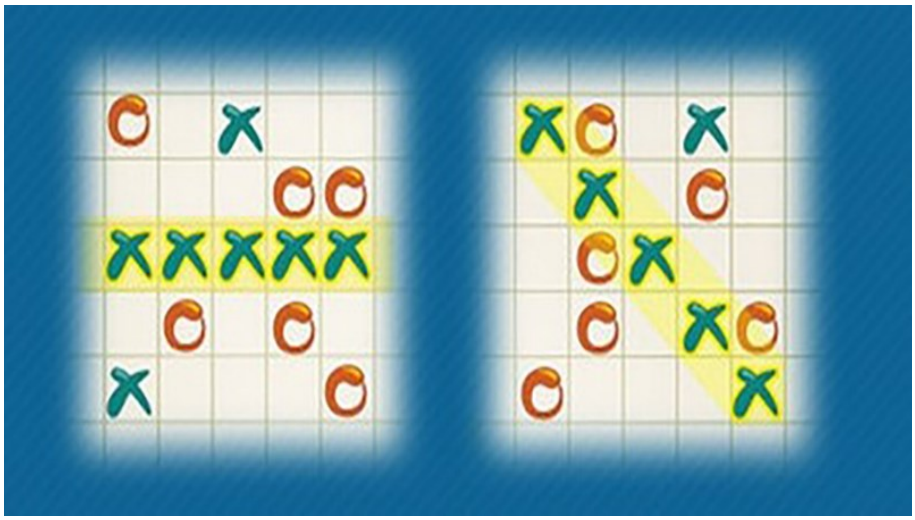
(a) Cờ Caro trên bàn cờ vây (b) Cờ Caro chơi trên giấy

Hình 1: Hình minh họa cho trò chơi cờ Caro

3 Luật chơi

Để đơn giản, luật chơi được nêu trong BTL này sẽ được áp dụng cho phiên bản chơi trên giấy (nước cờ được kí hiệu là X hoặc O).

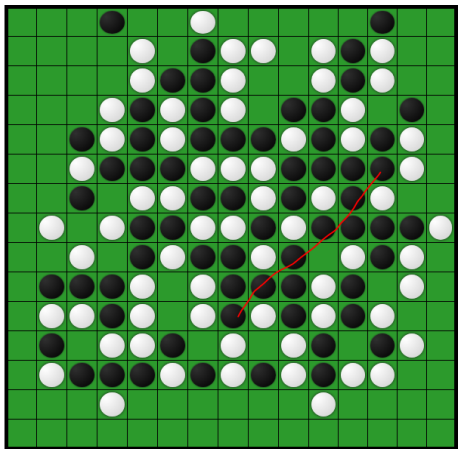
Trong cờ Caro thông thường, hai người chơi sẽ luân phiên đặt một nước cờ lên một ô trống. Người chơi đầu tiên sẽ bắt đầu bằng nước cờ X, người chơi kế tiếp sẽ sử dụng nước cờ O và luân phiên nhau đặt xuống bàn cờ. Người chiến thắng là người chơi đầu tiên tạo thành một chuỗi **năm (5)** nước cờ liên tiếp, không bị đứt đoạn theo chiều ngang, chiều dọc hoặc đường chéo. Hình 2 mô tả một số trạng thái chiến thắng của trò chơi.



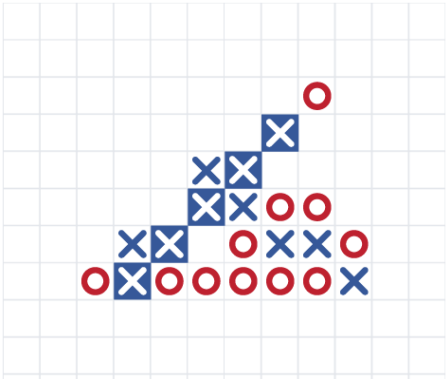
Hình 2: Một số trạng thái thắng của cờ Caro

Ngoài ra, trong BTL này, chúng ta sẽ bổ sung thêm 2 luật cho trò chơi:

- Luật bổ sung 1 (6 nước trở lên): Một chuỗi nước cờ liên tiếp có **nhiều hơn năm (5)** nước cờ sẽ **không tính là chiến thắng**.
- Luật bổ sung 2 (Chặn 2 đầu): Một chuỗi năm (5) nước cờ liên tiếp bị **chặn 2 đầu** bởi 2 nước cờ của đối thủ sẽ **không tính là chiến thắng**.



(a) Hình minh họa cho luật 6 nước trở lên



(b) Hình minh họa cho luật chặn 2 đầu

Hình 3: Luật bổ sung

Ở hình 3(a), đường kẻ màu đỏ chỉ rằng người chơi Đen dù có 6 nước cờ cũng không tính là chiến thắng. Ở hình 3(b), người chơi O không được tính là thắng dù đã có 5 nước cờ O liên tiếp vì đã bị người chơi X chặn bằng nước cờ X ở hai đầu.

4 Mô tả chương trình

Phần này sẽ trình bày các thông tin mà sinh viên cần phải nắm về chương trình trước khi bắt đầu thực hiện các yêu cầu của Bài tập lớn. Trong BTL này, sinh viên được cung cấp trước một đoạn mã nguồn đã được **hiện thực sẵn** bao gồm: hàm **main()**, hàm **startGame()**, hàm **displayBoard()** (cùng với hàm **lineString()** bổ trợ cho displayBoard) hàm **displayBoardSimple()**, kiểu enum **Stone** và một số các biến toàn cục khác phục vụ cho mã nguồn cho trước.

Khi chương trình bắt đầu chạy, màn hình sẽ hiển thị như sau:

```
Welcome to Gomoku!
1. Play game
2. History
3. Exit
Please select mode [1/2/3]:
```

Chương trình sẽ đợi input của người dùng là một số nguyên và với mỗi input tương ứng:

- Nếu input là 1: Chương trình sẽ gọi hàm **startGame()** và bắt đầu chế độ chơi theo lượt

của trò chơi cờ Caro

- Nếu input là 2: Chương trình sẽ gọi hàm **displayHistory()** và bắt đầu chế độ xem lại một ván đấu cờ Caro.
- Nếu input là 3: Chương trình sẽ kết thúc.
- Nếu input không hợp lệ: Chương trình sẽ in ra `Illegal input, please try again:` và yêu cầu người dùng nhập lại.

5 Nhiệm vụ

Trong BTL này, sinh viên cần hiện thực một chương trình giả tưởng trên ngôn ngữ C để mô phỏng lại quá trình chơi trò chơi cờ Caro trên màn hình console thông qua luật chơi được mô tả ở trên và các nhiệm vụ được mô tả bên dưới. Mỗi nhiệm vụ được yêu cầu viết hàm tương ứng, các tham số của cho hàm này sẽ được cho trong mô tả của yêu cầu nhiệm vụ.

Trước khi đọc tiếp phần dưới, sinh viên được yêu cầu phải đọc và hiểu rõ cách hàm **main()** và **startGame()** hoạt động. Việc hiểu rõ nội dung hàm **displayBoard()** và **lineString()** là không khuyến khích.

5.1 Đi một nước cờ (3 điểm)

Khi người dùng chọn chế độ chơi theo lượt, chương trình sẽ gọi hàm **startGame()**. Chương trình kế đến sẽ đợi người dùng nhập một nước đi là vị trí của bàn cờ.

Cờ caro được chơi trên một bảng vuông gồm 15 hàng được đánh số từ 15 đến 1 (từ trên xuống dưới) và 15 cột được đánh thứ tự từ a đến o (từ trái sang phải). Một nước đi của người chơi trên bàn cờ sẽ được ký hiệu là một tổ hợp ***hàngcột*** với giá trị hàng và cột nằm trong khoảng hợp lệ của bàn cờ. Một số ví dụ về nước đi hợp lệ: 1a, 3d, 5o, v.v...

Sinh viên được yêu cầu viết một hàm để xác định nước đi nhập vào của người chơi là hợp lệ hay không và ghi nhận nước đi vào bàn cờ nếu hợp lệ. Cụ thể, mô tả về hàm như sau:

- Tên hàm: **makeMove**.
- Tham số đầu vào:
  - **board[][MAX_SIZE] (kiểu enum Stone)**: Mảng 2 chiều kiểu enum Stone, lưu trữ thông tin về trạng thái hiện tại của bàn cờ.
  - **size (kiểu int)**: Kích thước của bàn cờ.
  - **playerMove (kiểu char*)**: Chuỗi chứa nước đi của người chơi

- **isFirstPlayerTurn (kiểu bool)**: Biến logic dùng để xác định lượt hiện tại là lượt của người đi nước cờ đầu tiên hay không

- Yêu cầu hàm:

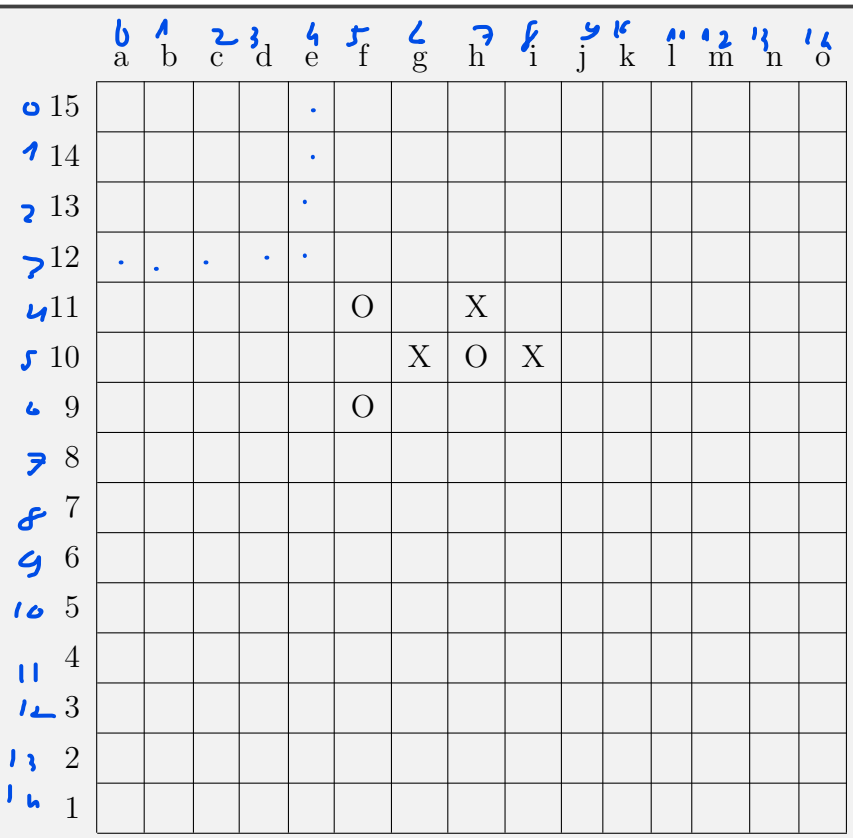
1. Hàm sẽ kiểm tra nước đi của người có phải là một nước đi hợp lệ hay không. Nước đi hợp lệ là nước đi: tuân theo ký hiệu của nước cờ, giá trị hàng và cột nằm trong khoảng hợp lệ, và ô cờ hiện tại đang trống.
2. Nếu nước đi là hợp lệ, hàm sẽ cập nhật một quân cờ vào bàn cờ **board** thành quân cờ của người chơi dựa vào biến **isFirstPlayerTurn**. Ngược lại, hàm sẽ không làm gì cả.
3. Hàm trả về `true` nếu bàn cờ được cập nhật thêm 1 nước cờ mới, ngược lại trả về `false`

Ví dụ 1: Các ví dụ về **playerMove** là nước cờ không hợp lệ:

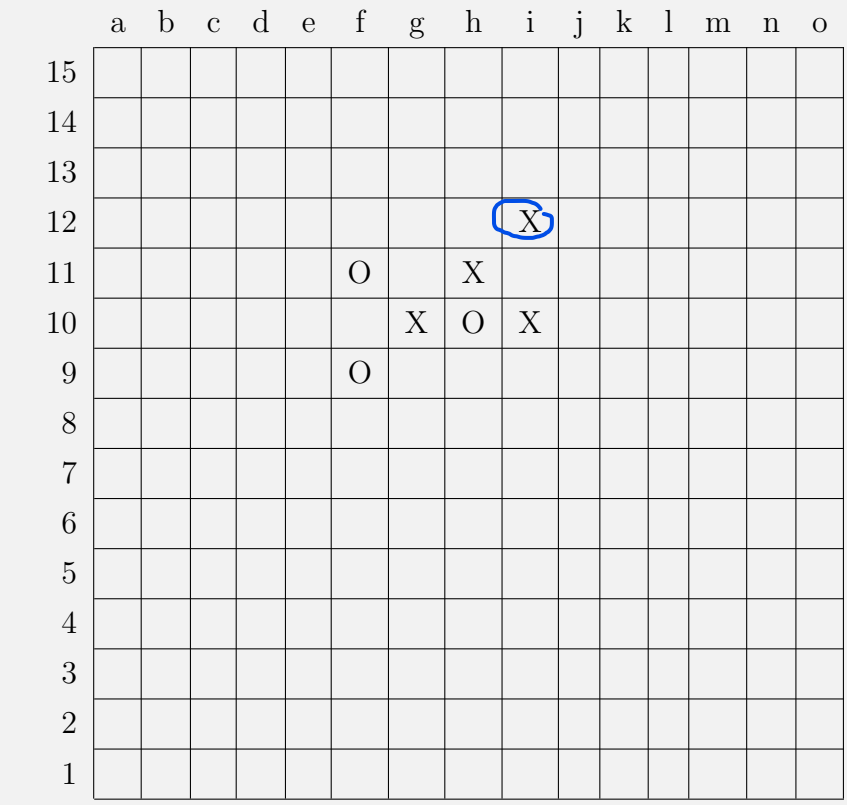
- **32x**: không hợp lệ do có giá trị hàng là 32 và giá trị cột là 'x' nằm ngoài khoảng hợp lệ của bàn cờ
- **t3**: không hợp lệ do không tuân theo cách ký hiệu của một nước cờ.

Hàm trả về giá trị `false` trong 2 trường hợp này.

Ví dụ 2: Cho trạng thái hiện tại của bàn cờ như sau:



Với playerMove = "12i" và isFirstPlayerTurn = true, hàm sẽ cập nhật board thành:



Hàm trả về giá trị `true`.

5.2 Kiểm tra chiến thắng (4 điểm)

Ta tiếp tục quan sát hàm **startGame()**. Sau khi nước đi là hợp lệ và quân cờ đã được lên bàn cờ, chương trình sẽ tiếp tục kiểm tra người chơi vừa đi nước cờ đó đã thắng hay chưa. Điều kiện thắng của trò chơi đã được mô tả rõ trong mục Luật chơi.

Sinh viên được yêu cầu viết hàm sau để mô tả lại quá trình kiểm tra chiến thắng. Thông tin của hàm như sau:

- Tên hàm: **hasWon**.
- Tham số đầu vào:
  - **board[][MAX_SIZE] (kiểu enum Stone)**: Mảng 2 chiều kiểu enum Stone, lưu trữ thông tin về trạng thái hiện tại của bàn cờ.
  - **size (kiểu int)**: Kích thước của bàn cờ.
  - **isFirstPlayerTurn (kiểu bool)**: Biến logic dùng để xác định lượt hiện tại là lượt của người đi nước cờ đầu tiên hay không
- Yêu cầu hàm: Hàm kiểm tra trạng thái hiện tại của bàn cờ **board** có phải là trạng thái chiến thắng của người chơi 1/người chơi 2 (dựa vào biến **isFirstPlayerTurn**) hay không.
- Kết quả trả về: Hàm trả về giá trị `true` nếu bàn cờ **board** đang ở trạng thái chiến thắng. Ngược lại, trả về `false`.

Ví dụ 3: Cho trạng thái hiện tại của bàn cờ như sau:

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
15															
14															
13									O	O					
12									X						
11						O		X	X	X	O				
10							X	O	X		O				
9						O		X	X	X	X	X	O		
8									O		O				
7															
6															
5															
4															
3															
2															
1															

Nhận thấy, người chơi X đã có 5 nước cờ liên tiếp từ 9h đến 9l. Vì vậy nên, người chơi X đã chiến thắng. Hàm trả về kết quả `true`.

Ví dụ 4: Cho trạng thái hiện tại của bàn cờ như sau:

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
15															
14										X					
13									O	O					
12							X		X	O					
11						O		X	X	O					
10							X	O	X	O					
9						O			X	O					
8									O	X					
7															
6															
5															
4															
3															
2															
1															

Mặc dù người chơi O đã có 5 nước cờ liên tiếp (từ 9j đến 13j) nhưng theo luật Chặn 2 đầu, người chơi O chưa chiến thắng vì đã bị người chơi X chặn ở nước 14j và 8j. Vì vậy, hàm trả về `false` vì chưa có ai chiến thắng cả.

Ví dụ 5: Cho trạng thái hiện tại của bàn cờ như sau:

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
15															
14															
13							O	O	O	O	O	O			
12									X			O			
11						O		X	X	X	X	O			
10							X	O	X	X	X	X	X		
9						O			X						
8									O						
7															
6															
5															
4															
3															
2															
1															

Người chơi O không được tính là chiến thắng mặc dù đã có 5 nước cờ liên tiếp (13g đến 13k, hoặc 13h đến 13l) vì những nước cờ đó đã rơi vào luật **6 nước trở lên**. Trong khi đó, người chơi X có 5 nước cờ liên tiếp từ 10i đến 10m nên X là người chiến thắng. Hàm trả về kết quả `true`.

5.3 Xem lịch sử trận đấu (3 điểm)

Mục này yêu cầu sinh viên hiện thực chức năng xem lại một ván đấu cờ Caro thông qua việc hiện thực hàm **displayHistory()**. Biết rằng sau khi một trận đấu kết thúc, lịch sử trận đấu sẽ lưu lại dưới format như sau:

$$a_0a_1a_2...a_i...a_n$$

Trong đó:

- a_i: Ký hiệu của một nước cờ
- n: Số lượng nước cờ của trận đấu

Sau khi chế độ xem lại lịch sử này được chọn, người dùng sẽ nhập vào lịch sử trận đấu để

theo dõi lại trận đấu này. Chương trình sẽ tiếp tục cung cấp cho người dùng 3 cách thức để điều khiển quá trình xem lại bao gồm: Xem nước cờ tiếp theo, Xem nước cờ trước đó và Thoát.

SV được yêu cầu viết hàm sau để cho phép người dùng điều khiển quá trình xem lại trận đấu dựa vào lịch sử do người dùng nhập vào. Thông tin cụ thể của hàm như sau:

- Tên hàm: **displayHistory**.
- Tham số đầu vào:
  - **history (kiểu char*)**: chuỗi chứa lịch sử trận đấu
  - **numOfMoves (kiểu int)**: số lượng nước cờ của trận đấu
- Yêu cầu hàm:
  1. Khai báo một mảng 2 chiều kiểu enum Stone (gọi là game) với kích thước 15x15 để lưu trữ trạng thái trận đấu. Khởi tạo các phần tử của mảng về giá trị NA.
  2. Hiển thị sàn đấu hiện tại thông qua gọi hàm displayBoard với các đối số (argument) cần thiết để hiển thị bàn đấu trống.
  3. In ra màn hình chuỗi `inputCommand`.
  4. Đọc vào input của người dùng là một kí tự kiểu `char`.
     - Nếu người dùng nhập vào kí tự `'n'`: Hiển thị bàn cờ sau khi đánh nước cờ tiếp theo (tiếp tục sử dụng hàm displayBoard). Tuy nhiên, nếu nước cờ hiện tại là nước cờ cuối cùng, in ra màn hình chuỗi `endOfHistory`. Sau đó, quay lại bước 4.
     - Nếu người dùng nhập vào kí tự `'p'`: Hiển thị bàn cờ trước khi đánh nước cờ hiện tại (sử dụng hàm displayBoard). Tuy nhiên, nếu bàn cờ hiện tại đang trống, in ra màn hình `startOfGame`. Sau đó, quay lại bước 4.
     - Nếu người dùng nhập vào kí tự `'s'`: Kết thúc hàm.
     - Nếu người dùng nhập vào kí tự khác với 3 kí tự trên, in ra màn hình chuỗi `invalidInput`. Sau đó, quay lại bước 4.
- Kết quả trả về: Không
- Lưu ý: Trong BTL này, lịch sử trận đấu do người dùng nhập vào được **đảm bảo** là luôn hợp lệ và n $<$ 225.

Ví dụ 6: Đây là một số ví dụ về chuỗi lịch sử trận đấu:

1. 7i7h8f8h9g10h9h9i8j6h6f9f4h6g5f7f10g11f5h5g4g10j11k6i8g11g3h2i4f4e2h1h3f2f3g5e3e3d3i (n = 34)
2. 13n10n12n8e7k9e12m7e6e8n12l12o10k14o12j12k11l9j10l9l9k8k11m13o8j (n = 25)

3. 13n10n12n8e7k9f9h7f9d8f6f7d6c7e7g8c8d6e5e5f4g9b (n = 22)

Ví dụ 7: Ví dụ sau đây là một luồng hoạt động của người dùng tương tác với chương trình sau khi người dùng chọn chế độ xem lịch sử trận đấu:

- Màn hình hiển thị: Please enter number of moves:
- Người dùng nhập vào: 22
- Màn hình hiển thị: Please enter history:
- Người dùng nhập vào: 13n10n12n8e7k9f9h7f9d8f6f7d6c7e7g8c8d6e5e5f4g9b
- Màn hình hiển thị:

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
15															
14															
13															
12															
11															
10															
9															
8															
7															
6															
5															
4															
3															
2															
1															

Previous move/Next move/Stop [p/n/s]:

- Người dùng nhập vào: n
- Màn hình hiển thị:

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
15															
14															
13														X	
12															
11															
10															
9															
8															
7															
6															
5															
4															
3															
2															
1															

Previous move/Next move/Stop [p/n/s]:

- Người dùng nhập vào: n
- Màn hình hiển thị:

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
15															
14															
13														X	
12															
11															
10														O	
9															
8															
7															
6															
5															
4															
3															
2															
1															

Previous move/Next move/Stop [p/n/s]:

- Người dùng nhập vào: s
- Chương trình kết thúc.

Ví dụ 8: Xét lịch sử trận đấu là

`13n10n12n8e7k9e12m7e6e8n12l12o10k14o12j12k11l9j10l9l9k8k11m13o8j`

với n = 25. Sau đây là một trạng thái của bàn cờ trong quá trình xem lại:

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
15															
14															O
13														X	
12										X	O	X	X	X	O
11												X			
10											X			O	
9						O				O					
8						O								O	
7						O					X				
6						X									
5															
4															
3															
2															
1															

Sau khi người dùng nhập vào `p`, chương trình sẽ in ra:

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
15															
14															O
13														X	
12										X	O	X	X	X	O
11												X			
10											X			O	
9						O									
8						O								O	
7						O					X				
6						X									
5															
4															
3															
2															
1															

Nếu người dùng nhập vào `a`, chương trình sẽ in ra: `Illegal input, please try again:`

Người dùng tiếp tục nhập vào `s`, chương trình kết thúc.

5.4 Cập nhật hàm displayBoard (bắt buộc)

Bởi vì kích thước hiển thị của hàm **displayBoard** là rất lớn và hệ thống sẽ chấm bài của sinh viên bằng cách so trùng testcase, sinh viên bắt buộc phải thay thế tất cả các lời gọi hàm **displayBoard** sang lời gọi hàm **displayBoardSimple** trước khi nộp bài lên hệ thống. Thay vì in ra toàn bộ bàn cờ như hàm **displayBoard**, hàm **displayBoardSimple** sẽ mã hóa bàn cờ một chuỗi số để tăng tốc độ so trùng và giảm kích thước của testcase. Hàm **displayBoardSimple** đã được hiện thực sẵn.

6 Nộp bài

Sinh viên download file các file sau từ trang Web của môn học:

gomoku.c	Mã nguồn khởi tạo
[NMLT]Gomoku - Assignment.pdf	File mô tả nội dung bài tập lớn

File gomoku.c là mã nguồn khởi tạo. Sinh viên **phải** sử dụng mã nguồn này để viết tiếp.

Khi nộp bài, sinh viên nộp bài trên site e-Learning của môn học. Sinh viên điền code bài tập lớn giống như các bài thực hành khác. Nội dung điền vào sẽ là toàn bộ file gomoku.c sau khi đã được hiện thực của sinh viên. Sinh viên được cung cấp 4 nơi nộp bài:

- **Nơi nộp bài thử**: Sinh viên nộp bài làm và được chấm trên **5 testcases** để kiểm tra các lỗi cú pháp, lỗi logic cơ bản có thể có của bài làm sinh viên. Sinh viên được phép nộp bài **vô số lần** ở đây.
- **Bài tập lớn - Đi một nước cờ**: Sinh viên nộp bài làm và được chấm trên **30 testcases** để kiểm tra các trường hợp liên quan đến Nhiệm vụ **Đi một nước cờ**.
- **Bài tập lớn - Kiểm tra chiến thắng**: Sinh viên nộp bài làm và được chấm trên **40 testcases** để kiểm tra các trường hợp liên quan đến Nhiệm vụ **Kiểm tra chiến thắng**.
- **Bài tập lớn - Xem lịch sử trận đấu**: Sinh viên nộp bài làm và được chấm trên **30 testcases** để kiểm tra các trường hợp liên quan đến Nhiệm vụ **Xem lịch sử trận đấu**.

Trong mỗi phần trên, ngoại trừ **Nơi nộp bài thử**, sinh viên có tối đa **10 lần** làm bài. Đối với mỗi lần nộp bài, sinh viên có **10 phút** để nộp code và kiểm tra. Chỉ có lần nhấn "Kiểm tra" đầu tiên là được tính điểm, các lần sau sẽ không được lấy điểm. Kết quả bài làm chỉ hiển thị sau khi bạn nhấn nút "Hoàn thành bài làm". Điểm cao nhất trong các lần làm bài sẽ được lấy làm điểm cho phần đó.

Điểm bài tập lớn của sinh viên được tính theo công thức sau:

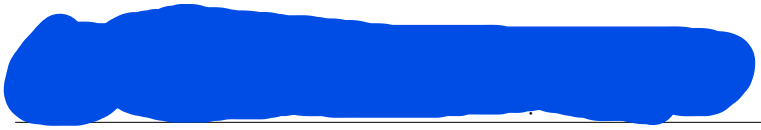
$$\text{Điểm BTL} = \textbf{(30 \%)}\ \text{Nhiệm vụ Đi một nước cờ} + \textbf{(40 \%)}\ \text{Nhiệm vụ Kiểm tra chiến thắng} + \textbf{(30 \%)}\ \text{Nhiệm vụ Xem lịch sử trận đấu}$$

Ví dụ, sinh viên A đạt 10 điểm ở Nhiệm vụ Đi một nước cờ, 7 điểm ở Nhiệm vụ Kiểm tra chiến thắng, 8 điểm ở Nhiệm vụ Xem lịch sử trận đấu. Vậy:

$$\text{Điểm BTL của A} = 10 * 0.3 + 7 * 0.4 + 8 * 0.3 = 8.2$$

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

Sinh viên phải kiểm tra chương trình của mình trên MinGW và nơi nộp bài thử trước khi nộp.



7 Xử lý gian lận

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

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

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

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

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

8 Thay đổi so với phiên bản trước

- Sửa lỗi Ví dụ 2 (isFirstPlayerTurn = false -> true)

——————————HẾT——————————