

# Object Oriented Programming

## Programming Assignment 3

Due: 2017/04/26

### ★Announcement★

1. 제출 기한은 4월 26일 자정까지입니다. LMS 시스템에서는 제출자가 몰리면 업로드가 안 될 수 있으니, 유의해주시기 바랍니다.
2. 과제는 늦게 제출하면 이유를 불문하고 0점입니다.
3. 채점은 Visual Studio 2015 환경에서 이루어집니다. 파일을 업로드하실 때, 작업하신 환경에 있는 프로젝트 폴더를 그대로 압축해서 올려주십시오. 그리고 폴더명은 ProblemN\_학번의 형태로 만들어주십시오. 프로젝트 폴더를 압축하지 않고 cpp 파일만 업로드하거나, Visual Studio 2015 환경에서 컴파일 되지 않을 경우 **큰 감점**이 있습니다.  
예시) Problem1\_20162939, Problem2\_20162939
4. C++에서 배운 내용을 토대로 작성하셔야 합니다. C++로 구현 가능한 내용을 C로 구현했을 경우에는 감점으로 처리합니다.  
예시) scanf, printf, fprintf, fscanf, malloc, free
5. 문서를 꼼꼼하게 읽고 문서에 나온 method들은 input argument와 output result를 문서에 맞게 작성해주세요.

## 1번 문제: Pokemon GO!

최근 한국에 출시되어 큰 인기를 끌었던 Pokemon GO! 그 게임의 Gym 대결을 간단하게 구현해본다. 포켓몬은 불, 물, 풀, 3개의 속성 중 한 개의 속성을 가진다. 불 포켓몬은 물 포켓몬에 약하고, 물 포켓몬은 풀 포켓몬에 약하며, 풀 포켓몬은 불 포켓몬에 약하다.

각 포켓몬은 공격력, 방어력, 체력을 가진다. 포켓몬이 서로 싸울 때 각 턴마다 포켓몬은 (상대 포켓몬의 공격력) - (내 포켓몬의 방어력) 만큼의 데미지를 입는다. 방어력이 공격력보다 같거나 높은 경우에는 1씩 데미지를 입는다. 먼저 체력이 떨어진 포켓몬이 진다.

불 포켓몬이 물 포켓몬과 싸울 때, 불 포켓몬의 방어력이 0.8배로 감소하고 물 포켓몬의 공격력이 1.5배로 증가한다. 마찬가지로 물 포켓몬이 풀 포켓몬과 싸울 때, 물 포켓몬의 방어력이 0.8배로 감소하고 풀 포켓몬의 공격력이 1.5배로 증가하며 풀 포켓몬이 불 포켓몬과 싸울 때, 풀 포켓몬의 방어력이 0.8배로 감소하고 불 포켓몬의 공격력이 1.5배로 증가한다.

### Requirements

#### 1. Pokemon class

- A. 공통적으로 가지고 있어야 할 공격력, 방어력, 체력을 지닌다. 또한, `type(string)`으로 타입을 저장한다.
- B. 다음과 같은 method를 가진다.
  - i. `bool battle(Pokemon*)` - 다른 포켓몬과 전투를 벌여 승리한 경우 `True`, 패배한 경우 `False`를 return한다.

#### 2. FirePok class, GrassPok, WaterPok

- A. `Pokemon class`를 상속받는다.
- B. `bool battle(Pokemon*)`을 각 속성끼리의 전투에 맞게 정의한다.

#### 3. Input and Output

- A. `input.txt`에는 다음과 같이 적혀있다.

```
2
Grass 10 5 20
Fire 8 3 30
Water 9 5 15
Water 8 4 20
```

- B. 첫 줄에는 전투를 몇 번 벌이는지가 적혀있다. 그 이후부터는 어떤 포켓몬끼리 전투를 벌이는지가 적혀있다. 속성 공격력 방어력 체력 순서로 입력된다.
- C. 위의 input의 경우, 풀 속성의 포켓몬(공격력 10, 방어력 5, 체력이 20)과 불

속성의 포켓몬(공격력 8, 방어력 3, 체력이 30)이 전투를 벌이고, 물 속성의 포켓몬(공격력 8, 방어력 4, 체력 20)과 물 속성의 포켓몬(공격력 9, 방어력 5, 체력 15)가 전투를 벌인다.

D. output.txt는 다음과 같이 출력된다.

|   |
|---|
| 2 |
| 1 |

E. 첫 번째 포켓몬이 이길 경우 1, 두 번째 포켓몬이 이길 경우 2, 두 포켓몬이 동시에 쓰러질 경우 0이 각 줄에 출력된다.

#### 채점 기준

1. Accuracy - 30 pts
2. Implementation - 20 pts
  - A. Pokemon class - 10 pts
  - B. Other class - 10 pts
3. Comment - 10 pts

## 2번 문제

tvN의 인기 서바이벌 프로그램 <더 지니어스>에서 나온 게임을 직접 구현한다. 십이장기는 가로 4칸, 세로 3칸으로 이루어진 판에서 이루어지는 장기 게임이다.



위와 같은 판에서 이루어진다. 각 플레이어의 앞에 있는 3개의 칸은 각 플레이어의 진영이 된다. 처음에 시작할 때는 다음과 같이 시작한다.



각 플레이어는 4개의 패를 위와 같이 가지고 시작한다. 각 패는 다음과 같은 역할을 한다.

1. 왕(王): 이 게임에서 지켜야 하는 패. 상하좌우 대각선까지 총 8방향으로 움직일 수 있다. 플레이어 진영의 가운데에 위치한다.
2. 장(將): 상하좌우 4방향으로 움직일 수 있다. 플레이어 진영의 오른쪽에 위치한다.
3. 상(相): 대각선 4방향으로 움직일 수 있다. 플레이어 진영의 왼쪽에 위치한다.
4. 자(子): 앞으로만 움직일 수 있다. 원래 게임에서는 상대 플레이어의 진영에 도달할 경우 후(候)로 바뀌어 사용할 수 있지만 이번 과제에서는 구현하지 않는다. 왕의 앞에 위치한다.

각 플레이어는 자신의 차례에 다음 두 가지 행동 중 하나를 선택해 행동할 수 있다.

1. 자신의 패 중 하나를 선택해 한 칸을 움직인다.
  2. 자신이 포로로 잡은 패 하나를 자신의 패로 사용하기 위해 내려놓는다. 단, 상대 플레이어의 진영이나 이미 패가 있는 곳에는 놓을 수 없다.
- 서로 차례를 번갈아 가면서 진행해 다음 두 가지 상황 중 하나의 상황일 때 게임이 종료된다.

1. 어떤 플레이어의 왕이 포로로 잡힌다. 상대 왕을 포로로 잡은 플레이어가 승리한다.
2. 어떤 플레이어의 왕이 상대 진영에 들어가 한 턴을 버틴다. 상대 진영으로 들어간 플레이어가 승리한다.

더 자세한 룰은 [https://youtu.be/\\_YS2iAOTJPw](https://youtu.be/_YS2iAOTJPw)를 참조한다.

## Requirements

### 1. Board class

- A. 장기판을 구현한다. `Piece* piece[3][4]` 를 이용해 각 패를 표현한다. 패가 놓여있지 않은 곳은 `NULL`을 저장하도록 하고, 각 패는 아래와 같이 정의하도록 한다.
- B. 다음과 같은 method는 필수로 구현한다.
  - i. `Board()` - 판을 초기화 한다. 처음에 시작할 때의 배치로 `Piece`를 배치한다. 이 때 `Piece` 들을 동적 할당하며 다른 부분에서 필요한 경우 여기서 할당한 `Piece` 들을 활용한다.
  - ii. `printBoard()` - 십이장기판을 다음과 같이 출력한다.

|    |  |    |  |    |  |    |
|----|--|----|--|----|--|----|
| BR |  |    |  |    |  | RG |
| KR |  | PR |  | PG |  | KG |
| RR |  |    |  |    |  | BG |

### 2. Piece class

- A. 각 패들이 상속 받는 base class. 현재 어디에 있는지 위치 `pos_x(int)`, `pos_y(int)`를 저장한다. 또한 어떤 팀인지를 결정하는 `team(bool)`를 저장한다.  
단, 왼쪽 맨 위를 0, 0으로 생각한다. x축이 가로축(4칸), y축이 세로축(3칸)으로 생각한다. Team은 2개가 있으며 Red팀과 Blue팀으로 나뉜다.
- B. 각 패들은 다음과 같은 method를 가져야 한다.
  - i. `virtual bool isPossibleMove(Board*, int _x, int _y) -` (`_x`, `_y`)로 이동 가능한지를 판단해 출력한다. 해당 말이 움직일 수 있는 좌표이고 해당 좌표에 상대 말이 있거나 비어있을 경우 이동이 가능하다.
  - ii. `virtual void printPossibleLocation(Board*) -` 이동 가능한 모

든 좌표를 출력한다.

- iii. `virtual void printPieces()` - 각 패를 출력한다. Rook은 R, Bishop은 B, Pawn은 P, King은 K로 출력한다. 추가적으로 Red Team인 경우 뒤에 R, Green Team인 경우 G를 출력한다. 즉, Red Team의 Rook은 RR로 출력한다.

### 3. Rook class(將), Bishop class(相), Pawn class(子), King class(王)

- A. 각 종류의 패를 나타내는 class로 Piece class를 상속 받는다.
- B. 위에서 언급한 method들을 각 종류에 맞게 구현하여야 한다.

### 4. User class

- A. 어떤 팀인지를 저장하는 `team(bool)`, 현재 판에 놓여있는 패를 저장하는 `onBoardPiece(Piece*[7])` 및 포로로 잡은 패 `heldPiece(Piece*[7])`을 저장하고 있다.
- B. User Class는 다음과 같은 method를 필수로 가진다. 기타 포로로 잡은 패, 놓여있는 패를 추가하고 삭제하는 것은 알아서 구현한다.
  - i. `printUser()` - User가 현재 가진 패와 포로로 잡고 있는 패를 다음과 같이 출력한다.

```
Green team
Lived piece:  [ RG ]  [ KG ]  [ BG ]
Held piece:   [ PR ]  [ BR ]
```

### 5. Controller class

- A. 게임을 control 하는 class를 작성한다. Controller는 게임 판을 저장하는 `board(Board*)`, 2명의 User를 나타내는 `userA`와 `userB (User*)`, 그리고 현재 누구의 차례인지를 가르키는 `team(bool)`를 가진다. 또한 몇 번째 차례인지를 저장하는 `cnt(int)`도 가진다.
- B. Controller class는 다음과 같은 method를 가진다.
  - i. `Controller()` - board를 초기화하고, `userA`와 `userB`를 맨 처음 게임의 상태로 초기화 한다.
  - ii. `~Controller()` - 사용한 모든 동적할당을 해제한다.
  - iii. `turn()` - 턴을 진행한다. 턴은 다음과 같이 진행한다.
    - 1. 누구의 차례인지 출력한다.
    - 2. 가장 먼저 패를 움직일 지, 포로로 잡은 패를 판에 새로 놓을 지를 결정한다.

```
Red Team Turn

1. Move the lived pieces
2. Use the held pieces
>>
```

3. 패를 움직일 때는 패를 선택하는 메뉴를 주어준다.

```
0:  [ KR ]
1:  [ RR ]
>>
```

4. 움직일 패를 선택하면 어디로 움직일지 좌표를 입력 받는다.

```
King: possible location to move
(1, 2)
(1, 1)
(1, 0)
(0, 0)
>>
```

5. 입력한 좌표로 움직일 수 있을 경우 움직이고 턴을 넘긴 후 1을 리턴한다.

6. 입력한 좌표로 움직일 수 없는 경우 “Impossible location”을 출력한 후 턴을 넘기지 않고 1을 리턴한다.

```
>> 3 1
Impossible Location
10th turn
```

7. 포로로 잡은 패를 판에 놓을 경우 놓을 패를 선택하는 메뉴를 주어준다.

8. 판에 놓을 패를 선택하면 어디에 놓을지 좌표를 입력 받는다.

```
0:  [ PR ]
1:  [ BR ]
>> 0
>> 1 2
```

9. 입력한 좌표에 놓을 수 있는 경우 패를 놓고 턴을 넘긴 후 1을 리턴한다.

10. 입력한 좌표에 놓을 수 없는 경우 “Impossible location”을 출력한 후 턴을 넘기지 않고 1을 리턴한다.

```
>> 3 2
Impossible Location
10th turn
```

- iv. `printStatus()` - status를 출력한다. 먼저 몇 번째 차례인지 출력한 후, UserA와 UserB의 상태를 출력하고, 그 아래에 현재 판을 출력한다.

```

10th turn
Red team
Lived piece:  [ KR ]  [ RR ]
Held piece:   [ PG ]

Green team
Lived piece:  [ RG ]  [ KG ]  [ BG ]
Held piece:   [ PR ]  [ BR ]



|    |  |    |    |
|----|--|----|----|
|    |  |    |    |
| KR |  |    | KG |
| RR |  | RG | BG |


```

- v. `isEnded()` - 게임이 종료되는 상태인지를 확인한다. 종료되지 않는 상태인 경우 0을 return한다. 종료되는 상황인 경우, UserA가 이겼을 때는 1을 UserB가 이겼을 경우에는 2를 return한다. 승리한 팀의 승리 메시지 출력한다.

```

>>>
Rook: possible location to move
(1, 2)
(2, 1)
>> 1 2
Green Team Win!!

```

6. Main Function은 주어진다. 주어지는 Main function을 실행시켰을 때 정상적으로 게임이 작동하여야 한다. Main function은 수정하지 말 것! (수정 시 감점)
7. 각 Piece는 여러 번 동적 할당 받지 않아야 한다. Board의 Constructor에서 처음으로 동적 할당한 8개의 Piece를 User와 Board에서 함께 사용한다. (추가적으로 동적 할당을 할 경우 감점)

#### 채점 기준

1. Accuracy - 50 pts
2. Implementation - 75 pts
  - A. Board class - 10 pts
  - B. Piece class - 5 pts
  - C. King, Bishop, Rook, Pawn class - each 10 pts
  - D. User class - 10 pts
  - E. Controller class - 10 pts
3. Comment - 15 pts