CS205 C/C++ Program Design Assignment4

Name: 陈逸飞

Student ID：12010502

Part-1 Task analysis

This assignment required us to construct a card game which based on the power of different cards.

The rules are as follow:

There are two players. Each player has a hand of cards and a deck(5 each in the beginning), and at the start of each round, each player draws a random card from their deck. Cards have a name, an attack value, and a defense value. Each round, each player chooses one card to play from their own hands. The card with the higher *power* wins the round.

What's more there are special effect cards for the game, which make the game more interesting.

The structure of this assignment as below:

```
assignment4
`-- cardgame
|-- big_boss_card.cpp
|-- big_boss_card.h
|-- card.cpp
|-- card.h
|-- exchange_card.cpp
|-- exchange_card.h
|-- main.cpp
|-- player.cpp
`-- player.h
```

In total there are four files that need our efforts.

1.1 Card

The first step is to make cards. Each cards have following **public** member：

Variables:

• *name* - name of the card

• *attack* - attack value of the card

• *defense* - defense value of the card

Functions:

• *Card* - Constructor

• *power* - return the power value

• *effect* - the effect of the card, a card has no effect by default

• *operator«* - format card in format

power is the rule to compete the power of player's card and opponent card's.

```
powerValue = (this->attack) - (opponentCard.defense) / 2;
```

What's more, the **effect** of cards also need to be mentioned that, since **effect** is considered as a **virtual function** and regluar cards have no effects at all, so in **card.cpp** we don't have to do anything with this function.

The operator<< is just to able to print the value of name, attack and defense.

1.2 Player

The second step is to create player

Variable：

• *deck* - the deck of the player, which is a card vector, the back of the vector is the top of the deck, player draws card from the top of the deck. We make sure that deck always have enough cards to draw.

• *name* - name of the player

• *hand* - cards in player’s hand, players draw cards from deck to hand

Functions：

• *Player* - constructor, when a player is initialized, he will draw 5 cards from deck

• *draw* - draw a card from deck to hand

• *play* - return the card in hand of given index, we make sure the index is in range

• *displayHand* - display cards in hand, one line for each card

The **constructor Player** need to be especially care, because there is rule that each player have five cards in hand.

```
Player::Player(std::vector<Card>& deck, std::string name)
{
    this->name = name;
    this->deck = deck;
    //A hand of cards means five cards???
    for (int i = 0; i <= 5; i++)
    {
        this->draw();
    }

}
```

The most interesting part in this section is that the useage of , I learnd several functions of in order to better control apply this undetermined array.

draw and **play** actually have common logic, to move and erase the card in the vector.

I test several functions to achieve my target.

```
void Player::draw()
{
    Card PlayerCard = deck.at(deck.size()-1);
    //this->hand.at(deck.size()) = deck.at(deck.size()-1);
    deck.pop_back();

    //hand.push_back(PlayerCard);
    hand.emplace_back(PlayerCard);  //New way
}

Card Player::play(int index)
{
    Card PlayedCard = hand[index];
    this->hand.erase(hand.begin() + index);
    return PlayedCard;
}
```

1.3 Effect cards

The effect cards are mainly based on the function `virtual void effect(Card& opponentCard, Player& player, Player& opponent);`, So when we inherited the class **card** in below classes, we need to rewite the **effect**. and declare this in them.(**virtual functions needs to be in the derived class**)

1.3.1 ExchangeCard

```
class ExchangeCard : public Card
{
public:
    ExchangeCard(/* args */);
    ExchangeCard(std::string, int attack, int defense);

    void effect(Card& opponentCard, Player& player, Player& opponent);
    ~ExchangeCard();
};
```

> ExchangeCard - effect: exchange opponent card's attack and defense (add exchange_card.
>
> cpp and exchange_card.h)

The task here is really vague, I spoke with several fellow students, the target of exchange is unclear. I consider it as only a operate to opponent's card, which is the one that on the playboard. Here, I just use **swap** in namespace std.

```
void effect(Card& opponentCard, Player& player, Player& opponent)
{
    //Exchange the values of attack power and defense power of the opponent
played card

    //regular way
    /* int Temp_Change;
    Temp_Change = opponentCard.attack;
    opponentCard.attack = opponentCard.defense;
    opponentCard.defense = Temp_Change; */

    //Simple way
    swap(opponentCard.attack,opponentCard.defense);
}
```

1.3.2 BIgBossCard

```
class BigBossCard : public Card
{
public:
    BigBossCard(/* args */);
    BigBossCard(std::string, int attack, int defense);

    void effect(Card& opponentCard, Player& player, Player& opponent);
    ~BigBossCard();

};
```

> BigBossCard - effect: adds the attack and defense of the opponent's card to all cards in the player's deck, then removes all cards in the opponent's deck that share an attack or defense stat with the opponent's card (add big_boss_card.cpp and big_boss_card.h)

This Big_Boss_card is more dim than the one above. According to the description, there are two move in this effect.

The first step is to add the attack and defense of the opponent's card to all cards in the player's deck(The task file use **adds** , this is very dim that since this represent the frequency of doing this step). I just add the value n times(n = `player.deck.size()`).

```
    //add attack and defense of the opponent's card to all card's in the
player's deck
    for (int i = 0; i < player.deck.size(); i++)
    {
        player.deck[i].attack += opponentCard.attack;
        player.deck[i].defense += opponentCard.attack;
    }
```

Then we were required to removes all cards in the opponent's deck that share an attack or defense stat with the opponent's card. I see this means that we destroy every cards in opponent deck that has same value of attack or defense with opponent card on the play board.

```
    //remove all the card's in the opponent's deck that share an attack or
defense value of the opponent's card
    for (int i = 0; i < opponent.deck.size(); i++)
    {
        if (opponent.deck[i].attack == opponentCard.attack ||
            opponent.deck[i].defense == opponentCard.defense)
        {
            opponent.deck.erase(opponent.deck.begin() + i);
        }
    }
```

However, the first version above is not quite correct, since everytime we remove one of the cards from the opponent's deck, the location of that card will be filled with the following cards, which are unable to fullfilled the requirement of **BigBossCard**. Therefore, I learned from **CC** my roommates about the method of using map and **queue**. So we first create a **map** to copy the value of the cards(it's actually mapping those values).

```
    for(int i = 0; i < player.deck.size(); i++)
    {
        player.deck[i].attack += opponentCard.attack;
        player.deck[i].defense += opponentCard.defense;
        A[player.deck[i].attack] = 1;
        A[player.deck[i].defense] = 1;
    }
```

Next we use a queue to manage the the opponent's deck, when ever the card effect was been activated, that card will be **push**.

Then it will **erase** it in the queue, **begin()** is returns an iterator to the first element of the queue, and **front()** is returning the reference type of the first element. And we will **pop** it out of the queue.

```
for(int i = 0; i < opponent.deck.size(); i++)
        if(A[opponent.deck[i].attack] || D[opponent.deck[i].defense])
        {
            deck_queue.push(i);
        }

    while(deck_queue.size())
    {
        opponent.deck.erase(opponent.deck.begin() + deck_queue.front());
        deck_queue.pop();
    }
```

Part-2 Conclusion and thinking

This assignment greatly inhance my understanding of class, specially virtual class.

Virtual is an important keyword in C++ object-oriented mechanism.
In the Base Derived class, you can override the Base virtual function by rewriting it.
When the pointer point of the Base class points to an object of the Derived class,
The call to point's print function actually calls the Derived print function instead of the Base print function. This is the embodiment of polymorphism in object orientation.

Overloaded functions must be in one class; Overriding functions must be in different classes that have inherited relationships. Overridden functions must have the same function name, arguments, and return values. Overloaded functions must have the same function name but different arguments. The whole point of having different parameters is to tell which function the program is calling at the time the function is called.
Overridden functions must be preceded by the keyword Virtual; Overloading has nothing to do with virtual and does not affect the operation of overloading.

The second interest part is the functions to operate and stack.

> https://blog.csdn.net/qq_38196982/article/details/119136

1.push_back()

在vector容器尾部添加一个元素，用法为：

```
arr.push_back(val);
arr.emplace_back()
```

C++11新增，功能与push_back相同，向vector容器尾部添加一个元素，用法为：

```
arr.emplace_back(value);
```

3.区别

可以发现push_back()与emplace_back()在用法上没有区别，主要的区别在于底层实现的机制不同。
push_bakc()添加元素时，首先会创建这个元素，然后再将这个元素拷贝或移动到容器中(如果是拷贝的话，事后会自行销毁之前创建的这个元素)；emplace_back()添加元素时，则是直接在尾部创建这个元素，省去了拷贝或移动元素的过程。

```
void Player::draw()
{
    Card PlayerCard = deck.at(deck.size()-1);
    //this->hand.at(deck.size()) = deck.at(deck.size()-1);
    deck.pop_back();

    //hand.push_back(PlayerCard);
    hand.emplace_back(PlayerCard);  //New way
}
```

Last but not the least, about the function **effect**.

我们知道要实现运行时的多态, 必须在基类中声明和定义相关的虚函数, 并在派生类中重新实现基类中的虚函数.
当编译器见到这种继承层次结构的时候, 编译器将为定义了虚函数的基类和覆盖了基类虚函数的派生类分别创建一张虚函数表(VFT), 通过编译器的编译, 基类和派生类的代码中都将有自己的虚函数表.
为这些类创建实例化对象时, 会在实例化的对象中插入一个指向对应的虚函数表的指针 (VFT*), 该指针通常在存放在对象的开头区域.
而虚函数表(VFT)就是一个其数据类型为函数指针的静态数组, 每一个函数指针指向对应类中的一个虚函

数.

必须是基类指针来调用基类虚函数，不能是传值，因为传值的话会造成对象的切除, 切除派生类对象相对于基类对象多出来的部分, 造成最终调用的函数是基类的函数, 而不是我们想要的派生类中的覆盖版本.

这也是effect card the main.cpp 中没有测试成功的原因。

Writing in the last assignment report.

Thanks TA and SA for making assignments and project,as well as answer my questions. CS205 made me learnd a lot, I'm trully grateful for this.

何须多虑盈亏事，终归小满胜万全——曾国藩