CS205 C/C++ Programming - Lab Assignment4 Report

Name: 邱逸伦 （Qiu Yilun）

SID: 12013006

Part 1 类和函数介绍

在本次Assignment中主要运用了四个类，分别为Card，Player，ExchangeCard，BigBossCard。其中，Player类用于生成玩家对象；Card类用于生成最为基础的卡牌对象，基础的卡牌对象没有effect函数功能；ExchangeCard类继承了Card基类，用于生成具有相应功能的卡牌对象；BigBossCard类继承了Card基类，用于生成具有相应功能的卡牌对象。

Card:

Card类中有三个成员变量，分别为string类型的name，int类型的attack和defense，分别代表了卡的名字、攻击属性、防御属性。Card有一个三参数的构造器，传入的参数分别为`std::string name, int attack, int defense`，在初始化Card对象时我选用了初始化列表的方式，直接以一行语句`Card::Card(std::string name, int attack, int defense): name(std::move(name)), attack(attack), defense(defense) {}`即可完成创建对象和初始化参数。对于power方法，根据传入的参数`opponentCard`即可计算出相应的power值为`(this->attack - opponentCard.defense / 2) >= 0 ? this->attack - opponentCard.defense / 2 : 0`。此处使用了三元运算符来直接判断是否求出的power结果小于0，若小于0则将结果置为0。重载`<<`运算符只需要根据传入的`std::ostream`对象`os`，执行语句`os << card.name << " " << card.attack << " " << card.defense;`，最后返回`os`对象即可。

Player:

Player类中有三个成员变量，分别为string类型的name，vector类型的deck和hand，分别代表了玩家的名字、拥有卡牌的牌堆、当前的所有手牌。Player有一个两参数构造器，传入的参数为`std::vector<Card>& deck, std::string name`，在初始化对象的参数时，我先尝试使用了`std::move`的方法来初始化name参数，这个方法需要导入`utility`库，其作用为将对象的状态或者所有权从一个对象转移到另一个对象，只是转移，没有内存的搬迁或者内存拷贝，故而一开始使用此方法可以提高利用效率，改善性能。但`std::move`方法也有一个明显的缺点，那就是假设初始化对象时传入一个string类型的参数s，而不是像样例一样每次单独传入一个字符串，如果想要再使用s，会发现原来的s的值应该为迁移再次访问它时它就是一个字符串，这样做虽然提高了效率，但是有着较大的局限性。故而最终我还是选择使用初始化列表的方式执行语句`deck(deck), name(name)`。根据题意，在构造器内部我用一个5次的for循环来每次调用draw方法完成从牌堆抓牌。`draw`方法先是通过deck的`back`方法从deck获取到牌堆顶部的Card，再使用`pop_back`方法来移除在deck里的此Card，最后以hand调用`push_back`方法将先前获取到的Card添加到hand的vector中。`play`方法先通过传入的int参数index获取到了hand中指定索引位置的Card（index索引值始于0），再通过hand调用`erase`方法清除了当前索引的Card元素。其中`erase`方法传入的参数为`hand.begin() + index`，以这样的方式可以获取到hand指定位置的元素。

ExchangeCard:

ExchangeCard继承了Card类，重写了Card类中的virtual方法`effect`方法。其作用在于交换输入参数opponentCard的attack参数和defense参数的值。

BigBossCard:

BigBossCard继承了Card类，重写了Card类中的virtual方法`effect`方法。其作用在于将输入参数Player对象player的deck中的所有Card的attack值和defense值都加上输入参数opponentCard的attack值和defense值，并且移除输入参数Player对象opponent的deck中所有attack值或defense值和opponentCard的attack值或defense值相同的Card。

Part 2 作业拓展和问题陈述

作业拓展：

本次作业需要同时编译多个.cpp文件方可正常运行*main.cpp*程序，为简化程序运行的步骤，我在Assignment4的文件目录下新建了一个*makefile*文件。*makefile*中的代码如下所示：

```
CC = g++
SOURCES = $(wildcard *.cpp)
OBJ = $(patsubst %.cpp, %.o, $(SOURCES))
TARGET = main
LIB = libpair
CXXFLAGS = -c -Wall -O3

$(TARGET) : $(OBJ)
    @mkdir -p lib
    $(CC) -shared -fPIC -o ./lib/$(LIB).so $(OBJ)
    @rm $(OBJ)
    $(CC) -o $(TARGET) -L./lib $(SOURCES) -lpair

%.o : %.cpp
    $(CC) -fPIC $^ $(CXXFLAGS) -o $@

.PHONY : clean
clean:
    rm -rf lib
    rm -rf main
```

当在控制台输入`make`命令时，编译器会在源文件目录下先生成一个名为lib的目录用于存储生成的动态库*libass4.so*文件，并对*card.cpp*，*player.cpp*，*big_boss_card.cpp*，*exchange_card.cpp*和*main.cpp*（主程序）进行编译，此处编译使用了`-Wall`和`-O3`的优化，`-Wall`可以显示编译后的所有警告，而`-O3`虽然会牺牲部分编译速度，但会尽量采用一些优化算法降低代码大小和可执行代码的运行速度。在运行过程也删除了冗余的文件，保证最终生成的只有可执行代码*ass4*和*lib*文件夹。如输入`make clean`命令则会清除生成的可执行代码*ass4*和*lib*文件夹。以下为运行的截图：

```
olin@LAPTOP-ML6O2JNH:~/Code/2022-Spring-C-and-Cpp/Assignment4$ make
g++ -fPIC player.cpp -c -Wall -O3 -o player.o
g++ -fPIC big_boss_card.cpp -c -Wall -O3 -o big_boss_card.o
g++ -fPIC card.cpp -c -Wall -O3 -o card.o
g++ -fPIC main.cpp -c -Wall -O3 -o main.o
g++ -fPIC exchange_card.cpp -c -Wall -O3 -o exchange_card.o
g++ -shared -fPIC -o ./lib/libass4.so  player.o  big_boss_card.o  card.o  main.o  exchange_card.o
g++ -o ass4 -L./lib player.cpp big_boss_card.cpp card.cpp main.cpp exchange_card.cpp -lass4
olin@LAPTOP-ML6O2JNH:~/Code/2022-Spring-C-and-Cpp/Assignment4$ ./ass4
```

最终程序将输出*main.cpp*中的`main`方法运行结果。

问题陈述：

本次作业中最显而易见的问题为vector里的Card对象无法体现多态性。对于这个问题的发现，我先分别在BigBossCard和ExchangeCard的`effect`方法的最后添加了`std::cout<<"The effect of BigBossCard"<<std::endl;`语句和`std::cout<<"The effect of ExchangeCard"<<std::endl;`语句，并将*main.cpp*修改为如下（仅用于测试）：

```cpp
#include <iostream>
#include <vector>

#include "card.h"
#include "exchange_card.h"
#include "player.h"
#include "big_boss_card.h"

using namespace std;

int main() {
    Card card = ExchangeCard("card", 100, 200);
    ExchangeCard card2 = ExchangeCard("exchange card", 100, 300);
    BigBossCard card3 = BigBossCard("name3", 100, 300);
    Card card4 = Card("name4", 200, 400);
    Card card5 = Card("name5", 300, 600);
    Card card6 = Card("name6", 400, 500);
    Card card7 = BigBossCard("boss", 200, 200);

    std::vector<Card> deck = {card, card2, card3, card4, card5, card6, card4, card4, card4, card2, card7};

    std::vector<Card> opponentDeck = {card4, card4, card, card2, card3, card4, card5};

    Player player = Player(deck, "John");
    Player opponent = Player(opponentDeck, "Alice");

    player.play(0).effect(card4, player, opponent);
    player.play(0).effect(card4, player, opponent);

    cout << opponent.deck.size() << endl;
    cout << player.deck.size() << endl;

    player.displayHand();
}
```

最后在命令行使用`make`命令运行了程序，最终程序运行结果如下：

```
2
6
name4 200 400
name4 200 400
name4 200 400
```

可以看出调用`effect`方法时，尽管获取到的Card是子类对象，但实际上也无法调用子类的effect方法，vector 会导致object slicing使派生类的effect被擦掉。

出于练习多态考虑，我另外上传了部分以多态为基准实现的代码，原来的实现代码同样一并上传在文件夹*Assignment4_no_polymorphism*中，但并未修改测试代码*main.cpp*，而接下来的规则阐述适用于实现了多态的代码，并完成了写在*main.cpp*中的测试代码。

Part 3 作业代码拓展和规则阐述

作业代码拓展修改：

在Part 2已提到，本次作业提供的代码是无法体现出多态性的，故而我对原来的代码做了一定的修改。最为主要的修改是将Player类中的deck和hand的变量类型改为了`vector<Card*>`，同时Player中的构造器传入参数也被相应更新为`Player(std::vector<Card*>& deck, std::string name)`。`draw`方法调用deck的`back`方法获取到的变量类型修正为了Card*，`play`方法的返回值也被相应修改为了Card*，如下所示：

```cpp
void Player::draw() {
    Card* c = deck.back();
    deck.pop_back();
    hand.push_back(c);
}

Card* Player::play(int index) {
    Card* c = hand[index];
    hand.erase( position: hand.begin() + index);
    return c;
}
```

根据这样的修改，在`main`方法中调用Card*的`effect`方法，若是子类对象会调用相应的`effect`方法，即可体现多态性。

规则阐述：

根据题意，本次作业的规则是由学生自行设计，结合实际和竞技游戏类型考虑，我设计出的游戏规则如下：

玩家最初始时将拥有一个25张牌的牌堆（deck，包括21张基础的Card，3张ExchangeCard，1张BigBossCard），牌堆中的牌的顺序是随机打乱的。当游戏开始时即玩家被初始化创建出来后，玩家会从牌堆顶中抽取5张牌作为手牌。本游戏是回合制双人竞技游戏，回合数上限为20回合，当超过回合数时，将自动为平局。对于进行游戏的两个Player对象p1和p2，p1将对应一个sum1的值，p2将对应一个sum2的值，每回合将自动生成一个0-4的随机数作为索引，作为Player对象的`play`方法的输入参数，获取到Card对象的指针后，将通过随机数生成一个0或1的数，若是1，则先对p1进行操作，p1先调用`power`方法，并将返回值加到sum1中，再调用p1的`effect`方法，再对p2进行同样的操作；若是2，则先对p2进行上述操作，再对p1进行上述操作。进行完上述操作后，将根据sum1和sum2的值进行胜负判断，若sum1大于等于100且sum2小于100，则胜利者为p1；若sum2大于等于100且sum1小于100，则胜利者为p2；若sum1大于等于100且sum2大于等于100，则需要再进行判断，若sum1大于sum2，则胜利者为p1，若sum2大于sum1则胜利者为p2。不满足上述条件的情况p1和p2将分别调用`draw`方法，在调用前也将先进行牌堆所剩牌数的判断，若有一方牌堆为空，则另一方直接胜利，若同时为空则直接平局，最后继续进行下一个回合或游戏结算。

对于上述规则，我完成了在*main.cpp*中的测试代码。本测试代码以随机数的方式模拟了游戏状态，以语句`srand(unsigned(time(nullptr)));`保证了`rand`方法获取的随机数结果不唯一。以下以初始化p1玩家属性为例，在循环初始化Card*时使用`rand`方法生成了两个1-10的值作为当前Card的attack和defense值，初始化语句如下所示`Card *card = new Card(name, attack, defense)`，其中name是名字字符串，attack和defense各代表了攻击和防御值，每次初始化完Card*后，相应的vector<Card*>都调用`push_back`方法将其加进vector中。对p1中的BigBossCard和ExchangeCard的初始化如下所示：

```
Card *bossCard1 = new BigBossCard("boss1", 10, 10);
Card *exchangeCard1_1 = new ExchangeCard("exchange1_1", 5, 5);
Card *exchangeCard1_2 = new ExchangeCard("exchange1_2", 4, 6);
Card *exchangeCard1_3 = new ExchangeCard("exchange1_3", 10, 4);
```

初始化完后同样将这些Card*加入进对应的vector中。在此处我使用的vector<Card *>的变量名为player1，在添加完所有的元素后，我调用了c++17引入的`shuffle`方法，该方法可以打乱vector中元素的顺序，总体时间复杂度为O(n)，n为vector的size。执行语句如下：`shuffle(player1.begin(), player1.end(), std::mt19937(std::random_device()()));`，最后初始化p1时只需执行语句`Player p1(player1, "player1");`。对于p2，以同样的方式进行初始化。

游戏进行过程在规则中已讲述清楚，代码仅仅实现了上述的规则如下：

```
    double sum1 = 0, sum2 = 0;
    int round = 0;
    Player *winner = nullptr;
    while (++round <= 20) {
        cout << "round " << round << ": " << endl;
        int temp = rand() % 5;
        Card *c1 = p1.play(temp);
        Card *c2 = p2.play(temp);
        cout<<"Card 1: "<<(*c1)<<endl;
        cout<<"Card 2: "<<(*c2)<<endl;
        int dice = rand() % 2;
        if (dice) {
            sum1 += c1->power(*c2);
            c1->effect(*c2, p1, p2);
            sum2 += c2->power(*c1);
            c2->effect(*c1, p2, p1);
        } else {
            sum2 += c2->power(*c1);
            c2->effect(*c1, p2, p1);
            sum1 += c1->power(*c2);
            c1->effect(*c2, p1, p2);
        }
        cout << "sum1=" << sum1 << "  " << "sum2=" << sum2 << endl;
        if (sum1 >= 100 && sum2 < 100) {
            winner = &p1;
            break;
        } else if (sum1 < 100 && sum2 >= 100) {
            winner = &p2;
            break;
        } else if (sum1 >= 100 && sum2 >= 100) {
            if (sum1 > sum2) {
                winner = &p1;
                break;
            } else if (sum2 < sum1) {
                winner = &p2;
                break;
            }
        }
        if (p1.deck.empty() && !p2.deck.empty()) {
            winner = &p2;
            break;
        } else if (!p1.deck.empty() && p2.deck.empty()){
            winner = &p1;
```

```
            break;
        } else if (p1.deck.empty() && p2.deck.empty()) break;
        p1.draw();
        p2.draw();
        delete c1;
        delete c2;
    }
    if (winner == nullptr) cout << "Draw!" << endl;
    else cout << "The winner is " << winner->name << endl;
```

和原来的代码相同，我同样创建了makefile文件在*Assignment4_polymorphism*文件夹中，只需输入`make && ./ass4`命令即可运行程序。某次运行结果输出如下所示：

```
round 1:
Card 1: card1_19 8 5
Card 2: card2_21 6 10
sum1=3  sum2=4
round 2:
Card 1: card1_21 3 4
Card 2: card2_5 4 2
sum1=5  sum2=6
round 3:
Card 1: card1_20 1 1
Card 2: card2_9 8 2
sum1=5  sum2=14
round 4:
Card 1: card1_4 2 5
Card 2: card2_19 9 5
sum1=5  sum2=21
round 5:
Card 1: card1_7 9 7
Card 2: card2_2 10 3
sum1=13  sum2=28
round 6:
Card 1: card1_9 5 8
Card 2: exchange2_1 5 5
sum1=16  sum2=29
round 7:
Card 1: card1_15 5 9
Card 2: card2_18 2 4
sum1=19  sum2=29
round 8:
Card 1: exchange1_3 10 4
Card 2: card2_11 9 10
sum1=24  sum2=36
round 9:
Card 1: card1_5 4 7
Card 2: card2_15 6 10
sum1=24  sum2=39
round 10:
Card 1: card1_11 9 4
Card 2: card2_16 4 8
sum1=29  sum2=41
round 11:
Card 1: exchange1_1 5 5
Card 2: card2_12 6 2
sum1=33  sum2=45
```

```
round 12:
Card 1: exchange1_2 4 6
Card 2: exchange2_2 4 6
sum1=36  sum2=46
round 13:
Card 1: card1_13 6 3
Card 2: card2_8 7 10
sum1=37  sum2=52
round 14:
Card 1: card1_2 4 6
Card 2: exchange2_3 10 4
sum1=41  sum2=59
round 15:
Card 1: card1_10 6 8
Card 2: card2_17 3 3
sum1=46  sum2=59
round 16:
Card 1: card1_17 10 6
Card 2: card2_20 1 3
sum1=55  sum2=59
round 17:
Card 1: card1_12 10 5
Card 2: card2_14 8 9
sum1=61  sum2=65
round 18:
Card 1: card1_3 8 1
Card 2: card2_10 10 1
sum1=69  sum2=75
round 19:
Card 1: card1_14 5 1
Card 2: card2_3 9 5
sum1=72  sum2=84
round 20:
Card 1: card1_18 3 7
Card 2: card2_6 10 3
sum1=74  sum2=91
Draw!
```

上述代码在每个回合结束后对c1和c2指针占用的内存进行删除，防止内存泄漏，在代码的最后同样以以下代码防止内存泄漏：

```
for (auto & i : p1.deck)    delete i;
for (auto & i : p1.hand)    delete i;
for (auto & i : p2.deck)    delete i;
for (auto & i : p2.hand)    delete i;
```

PS：其实在vscode中使用g++编译器是会报warnning信息如下：

```
main.cpp: In function ‘int main()’:
main.cpp:102:16: warning: deleting object of polymorphic class type ‘Card’ which
has non-virtual destructor might cause undefined behavior [-Wdelete-non-virtual-
dtor]
  102 |         delete c1;
      |                ^~
```

```
main.cpp:103:16: warning: deleting object of polymorphic class type ‘Card’ which
has non-virtual destructor might cause undefined behavior [-Wdelete-non-virtual-
dtor]
  103 |         delete c2;
      |                ^~
main.cpp:109:40: warning: deleting object of polymorphic class type ‘Card’ which
has non-virtual destructor might cause undefined behavior [-Wdelete-non-virtual-
dtor]
  109 |     for (auto & i : p1.deck)    delete i;
      |                                        ^
main.cpp:110:40: warning: deleting object of polymorphic class type ‘Card’ which
has non-virtual destructor might cause undefined behavior [-Wdelete-non-virtual-
dtor]
  110 |     for (auto & i : p1.hand)    delete i;
      |                                        ^
main.cpp:111:40: warning: deleting object of polymorphic class type ‘Card’ which
has non-virtual destructor might cause undefined behavior [-Wdelete-non-virtual-
dtor]
  111 |     for (auto & i : p2.deck)    delete i;
      |                                        ^
main.cpp:112:40: warning: deleting object of polymorphic class type ‘Card’ which
has non-virtual destructor might cause undefined behavior [-Wdelete-non-virtual-
dtor]
  112 |     for (auto & i : p2.hand)    delete i;
```

因我最初是在CLion中使用MinGW编译器，在CLion中并不会有警告提示，故而我忽略了这个warning。其实这个warning的出现是因为没有重写父类的析构函数，编译器警告可能释放内存时会出现错误，但实际上Card中并没有需要手动释放内存的指针，故而忽视这个warning并不会出现错误。

Part 4 总结

本次作业聚焦面向对象思想，从题目的意思可以看出出题人的本意是想考察面向对象的多态性，但确实没有考虑好在vector中的变量类型的正确性。本次作业很大程度我是依据自己的理解揣摩出题人的思路完成的，若有一些错误之处烦请指出，也希望老师多多包涵。在完成本次作业时我也对c++的指针特性有了更深的理解，对面向对象的多态性应如何体现更加明了，学会了一些c++随机化的语句、c++的vector变量的各种的方法，也感觉自己去构思一个project是一个很有趣的事。