项目四：21 点

截止日期：2024 年 12 月 1 日

~~1. 动机~~

为您提供实现抽象数据类型（ADTs）、使用接口（抽象基类）以及使用接口/实现继承的经验。

~~2。引言~~

在这个项目中，我们将实现一个简化版的纸牌游戏，称为 21 点，**有时也被称为 Blackjack**。这是一个相对简单的游戏，用一副标准的 52 张的扑克牌来玩。有两个参与者，一个发牌者和一名玩家。玩家一开始有存款，游戏以被称为“**手”的回合进行**。

在每一手开始时，玩家决定在这一手押注多少。押注金额可以是介于某一**最低**允许押注额和玩家总赌注额之间的任何数额，*包括两者*。

打赌之后，发牌者总共发四张牌：第一张正面朝上给玩家，第二张正面朝上给自己，第三张正面朝上给玩家，第四张朝下给自己。发牌者朝下的一张牌被称为“**洞牌**”。

然后，玩家查看自己的牌，计算总分。2 至 10 的牌按其面值计分；每张面牌（杰克、皇后、国王）也计 10 分。A 牌的价值要么是 1 分，要么是 11 分——以对玩家更有利的方式计分（即尽可能接近 21 分，且不超过 21 分）。如果总分中包含一张被算作 11 分的 A 牌，则总分被称为“软总分”，否则被称为“硬总分”。

游戏首先由玩家开始，然后由发牌者开始。玩家的目标是构建一手尽可能接近 21 且不超过 21 的手牌——后者被称为“爆牌”，爆牌的玩家会输掉手牌，且无需迫使发牌者继续发牌。只要玩家认为再要一张牌会有帮助，玩家就会“击中”——向发牌者要另一张牌。每张额外的牌都是**正面朝上发出的**。这个过程要么在玩家决定“站立”——不要牌时结束，要么在玩家爆牌时结束。请注意，玩家手中有两张牌时可以站立；在一手牌中，一张牌不必击中。

如果玩家拿到一张 A 加上任何一张 10 或面值卡片（杰克、皇后、国王），玩家的手牌就被称为“自然 **21 点**”，玩家的赌注会以 3 比 2 的比例支付，而不查看发牌者的牌。换句话说，如果玩家押注 10，如果拿到自然 21 点，玩家将赢得 15（即他的赌注会增加 15）。请注意，由于我们使用的是整数，对于 3/2 的赔付，您必须稍微小心。例如，如果押注 5 且拿到自然 21 点，赔付将是 7，因为在整数运算中，(3*5)/2 等于 7 。

如果玩家既没有爆牌也没有拿到自然 21 分，游戏就轮到发牌者。发牌者**必须**击牌，直到其总分大于或等于 17 分（硬 17 分或软 17 分），或者爆牌。如果发牌者爆牌，玩家就赢了。否则，会对比两人的总分。如果发牌者的总分更高，玩家的赌注金额就会从其银行账户中扣除。如果玩家的总分更高，其银行账户就会增加赌注金额。如果总分相等，银行账户金额不变；这被称为“平局”。

请注意，这是该游戏的一种非常简化的形式：我们不拆分对子牌、不允许加倍押注或购买保险等等。（http://en.wikipedia.org/wiki/Blackjack）

III. 编程作业

您将为这个项目提供四个独立抽象的一个或多个实现：一副牌、一个 21 点手牌、一个 21 点玩家和一个游戏驱动程序。本规范中引用的所有文件都位于 Canvas 上的“Projects/Project-4”文件夹中。

您可以将它们复制到您的私人目录空间，但**不得以任何方式对其进行修改**。这将有助于确保您提交的项目能够正确编译。

1. （表示“时间”）甲板 ADT

您的首要任务是实现以下表示一副牌的抽象数据类型：

```
class DeckEmpty { // An exception type
};

const int DeckSize = 52;

class Deck {

    // A standard deck of 52 playing cards---no jokers

    Card deck[DeckSize]; // The deck of cards
    int next; // The next card to deal
```

公共的；公开的

```
    Deck();
    // EFFECTS: constructs a "newly opened" deck of cards: first the
    // spades from 2 to A, then the hearts, then the clubs, then the
    // diamonds. The first card dealt should be the 2 of Spades.

    void reset();
    // EFFECTS: resets the deck to the state of a "newly opened" deck
    // of cards.

    void shuffle(int n);
    // REQUIRES: "n" is between 0 and 52, inclusive.
    // MODIFIES: this
    // EFFECTS: cut the deck into two segments: the first "n" cards,
    // called the "left", and the rest called the "right". Note that
    // either right or left might be empty. Then, rearrange the deck
    // to be the first card of the right, then the first card of the
    // left, the 2nd of right, the 2nd of left, and so on. Once one
    // side is exhausted, fill in the remainder of the deck with the
    // cards remaining in the other side. Finally, make the first
    // card in this shuffled deck the next card to deal. For example,
    // shuffle(26) on a newly-reset() deck results in: 2-clubs,
    // 2-spades, 3-clubs, 3-spades ... A-diamonds, A-hearts.
    //
    // Note: if shuffle is called on a deck that has already had some
    // cards dealt, those cards should first be restored to the deck
    // in the order in which they were dealt, preserving the most
    // recent post-shuffled/post-reset state. After shuffling, the
    // next card to deal is the first one in the deck.

    Card deal();
    // MODIFIES: this
    // EFFECTS: deals the "next" card and returns that card.
    // If no cards remain, throws an instance of DeckEmpty.

    int cardsLeft();
    // EFFECTS: returns the number of cards in the deck that have not
    // been dealt since the last reset/shuffle.
};
```

“Deck ADT”在“deck.h”中定义。“Deck ADT”依赖于以下“Card”类型:

```
    enum Suit {
        SPADES, HEARTS, CLUBS, DIAMONDS
    };

    extern const char *SuitNames[DIAMONDS+1];

    enum Spot {
```

```
        TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE, TEN,
        JACK, QUEEN, KING, ACE
    };

    extern const char *SpotNames[ACE+1];

    struct Card {
        Spot spot;
        Suit suit;
    };
```

这在 card.h 中声明，由 card.cpp 实现，并被 deck.h 包含。文件 card.cpp 为您定义了 SpotNames 和 SuitNames，因此 SuitNames[HEARTS] 的值为“心形”，以此类推。

请注意，关键字“extern”告知编译器存在一个全局变量（并且在某处已定义），以便它能够编译使用该全局变量的任何代码。在链接步骤中，链接器会将对全局变量的引用解析为其中一个已编译文件中该全局变量的唯一定义。

要求您将这个抽象数据类型的实现放入一个名为 deck.cpp 的文件中。

~~2.（表示“时间”）~~手部界面

您的第二项任务是实现以下表示 21 点手牌的抽象数据类型:

```
struct HandValue {
    int  count;    // Value of hand
    bool soft;     // true if hand value is a soft count
};

class Hand {
    // OVERVIEW: A blackjack hand of zero or more cards

    // Note: this really is the only private state you need!
    HandValue curValue;

public:

    Hand();
    // EFFECTS: establishes an empty blackjack hand.

    void discardAll();
    // MODIFIES: this
```

```
    // EFFECTS: discards any cards presently held, restoring the state
    // of the hand to that of an empty blackjack hand.

    void addCard(Card c);
    // MODIFIES: this
    // EFFECTS: adds the card "c" to those presently held.

    HandValue handValue() const;
    // EFFECTS: returns the present value of the blackjack hand.  The
    // count field is the highest blackjack total possible without
    // going over 21.  The soft field should be true if and only if at
    // least one ACE is present, and its value is counted as 11 rather
    // than 1.  If the hand is over 21, any value over 21 may be
    // returned.
    //
    // Note: the const qualifier at the end of handValue means that
    // you are not allowed to change any member variables inside
    // handValue. Adding this prevents any accidental change by you.
};
```

“手”抽象数据类型在 hand.h 中定义。“手”抽象数据类型依赖于“卡”的类型，并包含 card.h。要求您将此抽象数据类型的实现放在名为 hand.cpp 的文件中。

3. 玩家界面

您的第三项任务是实现两个不同的 21 点玩家。玩家接口如下:

```
class Player {
    // A virtual base class, providing the player interface

public:
    virtual int bet(unsigned int bankroll,
                    unsigned int minimum) = 0;
    // REQUIRES: bankroll >= minimum
    // EFFECTS: returns the player's bet, between minimum and bankroll
    // inclusive

    virtual bool draw(Card dealer,           // Dealer's "up card"
                    const Hand &player) = 0; // Player's current hand
    // EFFECTS: returns true if the player wishes to be dealt another
    // card, false otherwise.

    virtual void expose(Card c) = 0;
    // EFFECTS: allows the player to "see" the newly-exposed card c.
    // For example, each card that is dealt "face up" is expose()d.
    // Likewise, if the dealer must show his "hole card", it is also
```

```
    // expose()d.  Note: not all cards dealt are expose()d---if the
    // player goes over 21 or is dealt a natural 21, the dealer need
    // not expose his hole card.

    virtual void shuffled() = 0;
    // EFFECTS: tells the player that the deck has been re-shuffled.
};
```

玩家抽象数据类型（Player ADT）在 player.h 中定义。玩家抽象数据类型取决于手牌类型，并包含 hand.h。您需要从这个接口中实现两个不同的派生类。

第一个派生类是简单玩家，它采用一种简化版的 21 点基本策略进行游戏。简单玩家总是下最低允许的赌注，并根据以下规则以及玩家是“硬**数**”**还是“软数**”来决定击牌或停牌：

如果玩家有“硬**数**”，即他手牌的最佳总分将 A（如果有的话）算作 1 而不是 11，或者他的手牌中不含 A，则适用第一组规则。

- 如果玩家的手牌总和为 11 或更少，他总是击中。
- 如果玩家的手牌总和为 12，当发牌者亮出的牌为 4、5 或 6 时，玩家就站立；否则玩家就击牌。
- 如果玩家的手牌总分在 13 到 16 之间（含 13 和 16），当发牌者显示的牌是 2 到 6 之间（含 2 和 6）时，玩家就站立；否则玩家就击牌。
- 如果玩家的手牌总和达到 17 或更大，他就总是站着。

如果玩家有“软**数**”——**其**最佳总分包含一张价值 11 的 A 时，适用第二组规则。（请注意，一手牌绝不会把两张 A 都算作 11 **分**——那会是 22 分爆牌。）

- 如果玩家的手牌总和为 17 或更少，他总是要牌。
- 如果玩家的手牌总和为 18，当发牌者亮出的牌为 2、7 或 8 时，玩家就站立；否则玩家就击牌。
- 如果玩家的手牌总和达到 19 或更大，他总是站立。

注意：简单玩家对于**暴露**和随机事件不做任何处理。

第二个派生类是计数玩家。除了采用基本策略外，这种玩家还会对牌进行计数。计数背后的直觉是，当牌组中花牌（价值 10 分）的数量多于小数字牌时，牌组对玩家有利。反之亦然。

计数玩家会持续记录他看到的牌数。每次他看到（通过 expose（） 方法）一张 10、杰克、皇后、国王或王牌时，他就从计数中减去 1。每次他看到一张 2、3、4、5 或 6 时，他就给计数加 1。当他看到牌组时

shuffled（） 时，计数器会重置为零。每当计数器为 +2 或更大，**并且他**有足够的资金（**大于或等于最小赌注的两**倍）时，计数玩家会押两倍于最小赌注的金额，**否则**（包括计数器大于或等于 +2 但资金小于最小赌注两倍的情况），他会押最小赌注。计数玩家不应不必要地重新实现简单玩家的任何方法。

这两个玩家的代码必须在一个名为 player.cpp 的文件中实现。您还必须在 player.cpp 文件中声明这两个玩家各自的静态全局实例。最后，您应该在 player.cpp 文件中实现以下“访问”函数，这些函数返回这两个全局实例的指针。

```
extern Player *get_Simple();
extern Player *get_Counting();
```

4.（表示“时间”）驱动程序

最后，您要实现一个驱动程序，该程序能够利用您对上述抽象数据类型的实现来模拟这个版本的心跳游戏。

要求您将这个驱动程序的实现放入一个名为 blackjack.cpp 的文件中。

驱动程序在运行时，会接收三个参数：

```
<bankroll> <hands> [simple|counting]
```

第一个参数是一个整数，表示玩家的初始赌注。第二个参数是模拟中要玩的最大手数。**您可以假设用户输入的这两个整数是正数（≥1），并且在上限10000以内**。最后一个参数是两个字符串“简单”或“计数”中的一个，表示在模拟中使用哪种玩家。

例如，假设您的程序名为 blackjack 。它可以通过在终端中输入来调用：

```
./blackjack 100 3 simple
```

然后，您的程序模拟简单玩家用初始资金 100 玩 3 手牌。

司机首先洗牌。要洗牌，您**随机选择 13 到 39 之间的 7 个**切牌位置，用每个切牌位置对牌组进行洗牌。我们提供了一个头文件 rand.h 和一个实现文件 rand.cpp，其中定义了一个函数来提供这些随机切牌位置。每次洗牌时，首先进行宣告：

```
cout << "Shuffling the deck\n";
```

并公布这**七个**切点中的每一个：

```
cout << "cut at " << cut << endl;
```

那么一定要通过 shuffle（） 告知玩家。

注意：在进行任何进一步操作之前，您应该始终打印与初始洗牌相对应的消息。

我们假设最低赌注为 5。那么，只要玩家的资金大于或等于 5 这个最低赌注，并且还有牌局未结束：

- 宣布手部：

  ```
  cout << "Hand " << thishand << " bankroll " << bankroll << endl;
  ```

  其中变量“thishand”表示手牌编号，**从 1 开始**。

- 如果剩余的牌严格少于 20 张，则如上所述重新洗牌。请注意，这仅在每手牌开始时发生。即使牌数少于 20 张，在一手牌进行期间也不会发生。

- 向玩家询问赌注并宣布：

  ```
  cout << "Player bets " << wager << endl;
  ```

- 发四张牌：一张正面朝向玩家，一张正面朝向发牌者，一张正面朝向玩家，一张背面朝向发牌者。使用 cout 输出正面朝上的牌。例如：

  ```
  Player dealt Ace of Spades
  Dealer dealt Two of Hearts
  ```

  使用 SpotNames 和 SuitNames 数组来完成这个，并且一定要向玩家展示任何正面朝上的卡片。

· 如果玩家拿到自然 21 分，立即向玩家支付其押注金额的 3/2。请注意，由于我们使用的是整数运算，对于 3/2 的赔付您得稍微小心处理。例如，如果押注 5 分且拿到自然 21 分，赔付金额为 7 分，因为在整数运算中 (3*5)/2 的结果是 7。在这种情况下，宣布玩家获胜：

```
cout << "Player dealt natural 21\n";
```

· 如果玩家未拿到自然 21 分，让玩家出牌。抽取牌直至玩家选择停牌或爆牌。如上文所述，宣布并展示每张发出的牌。

· 公布玩家的总分

```
cout << "Player's total is " << player count << endl;
```

其中变量 player_count 是玩家手牌的总值。如果玩家爆牌，就表明这一点：

```
cout << "Player busts\n";
```

从赌资中扣除赌注，接着进行下一手。

· 如果玩家尚未破产，宣布并揭露发牌者的底牌。例如：

```
Dealer's hole card is Ace of Spades
```

（注意：如果玩家爆牌或被发到了自然 21 分，则隐藏牌不会被亮出。）

· 如果玩家尚未破产，就亮出发牌者的手牌。发牌者必须击牌，直至达到 17 分或破产。如上所述进行宣告并亮出每张牌。

· 宣布经销商的总分

```
cout << "Dealer's total is " << dealer count << endl;
```

其中变量 dealer_count 是发牌者手牌的总值。如果发牌者爆牌，就表明这一点。

```
cout << "Dealer busts\n";
```

从赌资中扣除赌注，然后继续下一手。

· 如果发牌者和玩家都没有破产，比较总分并宣布结果。根据情况增加、减少或保持赌注总额不变。

```
cout << "Dealer wins\n";
cout << "Player wins\n";
cout << "Push\n";
```

· 如果玩家的赌注大于或等于 5 的最小赌注，并且还有手牌剩余待发，那么就继续玩下一手牌（即从第一个要点“宣布手牌”重新开始）。

最后，当玩家要么钱太少无法下最低赌注，要么已用完分配的手牌时，宣布结果：

```
cout << "Player has " << bankroll
     << " after " << thishand << " hands\n";
```

其中变量“thishand”是当前手牌编号。在初始赌注低于最低赌注的特殊情况下，我们有 thishand = 0，因为玩家尚未玩过任何手牌。此外，在这种特殊情况下，在打印玩家的状态之前，仍应宣布初始洗牌。

IV. 实施要求与限制

· 您可以 #include <iostream>、<iomanip>、<string>、<cstdlib> 和 <cassert>。不得包含其他系统头文件，并且不得调用任何其他库中的任何函数。
· 只有在指定的情况下才应进行输出。
· 您可能不能使用“goto”命令。
· 在驱动程序中您可能不能有任何全局变量。您可以在类实现中使用全局状态，但它必须是静态的并且（对于两个玩家除外）是常量。
· 您可以假定函数调用符合其公布的规格说明。这意味着您无需进行错误检查。然而，在协同测试您的代码时，您可以使用 assert（） 宏进行防御性编程。

五、源代码文件与编译

在我们的 Canvas 资源中，“Project-4-Related-Files.zip”里有五个头文件（card.h、deck.h、hand.h、player.h 和 rand.h）以及两个 C++ 源文件（card.cpp 和 rand.cpp）。

您应该将这些文件复制到您的工作目录中。**请勿**对其**进行修改**！

您还需要编写另外四个 C++ 源文件：deck.cpp、hand.cpp、player.cpp 和 blackjack.cpp。它们在上文已有讨论，并总结如下：

deck.cpp：您的牌组抽象数据类型（ADT）的实现hand.
cpp：您的手牌抽象数据类型（ADT）的实现player.cpp：
您的两个玩家抽象数据类型（ADT）的实现blackjack.
cpp：您的模拟驱动程序

在您编写完这些文件之后，您可以在终端中输入以下命令来编译程序：

```
g++ -Wall -o blackjack blackjack.cpp card.cpp deck.cpp hand.cpp
player.cpp rand.cpp
```

这将在您的工作目录中生成一个名为“blackjack”的程序。为了确保助教的程序能成功编译，您应该将您的源代码文件命名为与上述指定完全一致的名称。

VI. 测试

对于这个项目，您应该为所有的抽象数据类型（ADT）的实现编写单独的、有针对性的测试用例。对于这些抽象数据类型，确定其实现所需的特性。然后，对于这些特性中的每一个：

- 确定该实现必须展现出的**具体**行为。
- 编写一个程序，在与该抽象数据类型（ADT）的实现相链接时，测试该行为的存在/缺失。

例如，如果您在 Deck 的实现中识别出两种行为，那么您将会有两个文件，每个文件测试一种行为。您可以按如下方式给它们命名：

```
deck.case.1.cpp
```

deck.case.2.cpp

您这个项目的测试用例被视为“验收测试”。您的手/牌 ADT 的测试（每个测试都包含一个 main（） 函数）在编译程序时应该与 card.cpp（您可以假定它是正确的）以及您可能不正确的 hand.cpp/deck.cpp 链接。您的玩家 ADT 的测试（每个测试都包含一个 main（） 函数）在编译程序时应该与 card.cpp、一个您假定为正确的已经测试过的 hand.cpp 以及您可能不正确的 player.cpp 链接。

您的测试用例必须根据对 Hand/Deck/Player 方法的调用的结果来决定 Hand/Deck/Player 抽象数据类型（ADT）是正确还是不正确。如果您的用例认为 Hand/Deck/Player 是正确的，那么 main（） 函数应该返回 0。如果您的用例认为 Hand/Deck/Player 不正确，那么 main（） 函数应该返回除 0 以外的任何值（通常使用 -1 来表示失败）。不要将您的测试用例的输出与正确/错误的实现进行比较。相反，在 Linux 中运行您的程序时查看其返回值，以查看您是否根据测试是否在您所测试的 ADT 的实现中发现错误而返回了正确的值。

在 Linux 系统中，您可以通过输入来查看程序的返回值。

```
echo $?
```

在运行程序之后立即。您可能还会发现向输出添加错误消息会有所帮助。

以下是一个代码示例，用于测试一个假设的“整数相加”函数（在 addInts.h 中声明），并带有一个“预期”测试用例：

```
// Tests the addInts function
#include "addInts.h"
int main() {
    int x = 3;
    int y = 4;
    int answer = 7;
    int candidate = addInts(x, y);
    if (candidate == answer) {
        return 0;
    } else {
        return -1;
    }
```

}

您应该编写一组具有不同特定错误的“手牌/牌组/玩家”的实现，并进行测试以识别不正确的代码。

我们提供了一个测试源文件 example.cpp 的示例，它测试了 Deck 抽象数据类型（ADT）的 shuffle（int） 方法。

我们还提供了一个简单的输出示例，由正确的牌组、手牌、简单玩家和驱动程序生成。它被称为 sample.txt。要测试您的整个项目，请在您的程序编译后，在 Linux 终端中输入以下内容：

```
./blackjack 100 3 simple > test.out
diff test.out sample.txt
```

如果 diff 程序报告有任何差异，那么您就遇到了一个漏洞。

7。提交与截止日期

你应该提交四个源代码文件：deck.cpp、hand.cpp、player.cpp和blackjack.cpp。（你不需要为这个项目提交Makefile。）这些文件应该通过在线评分系统以tar文件的形式提交。有关提交的详细信息，请查看助教的公告。截止日期是2024年12月1日晚上11点59分。

VIII. 评分

你的项目将根据三项标准进行评分：

1. 功能正确性
2. 实施限制
3. 一般风格

功能正确性是通过运行各种测试用例来确定的，并与我们的参考解决方案进行检查。我们将对实现约束进行评分，以查看您是否满足所有的实现要求和限制。一般风格是指的易读性。

通过这种方式，教学助理能够阅读并理解您的程序，以及您代码的整洁性和优雅性。例如，大量的代码重复会导致一般风格方面的扣分。