双城之战

在遥远的符文之地，两座截然不同的城市之间弥漫着紧张的对峙。皮尔特沃夫（简称皮城），繁荣的进步之城，充满机械与科技的辉煌；祖安，这座地底迷雾之城，却散发着炼金术与混沌的气息。两地的矛盾由来已久，皮城坚信秩序与创新，祖安则追求自由与不羁。

某一天，一场巨大的动荡打破了微妙的平衡——一件足以颠覆符文之地力量格局的海克斯核心被两方同时发现。为了争夺这一关键技术，皮城与祖安派出了最精锐的代表，通过卡牌决斗展开这场较量，胜者将决定核心的归属。这场战斗，不仅关乎两座城市的命运，也将影响符文之地未来的走向。

游戏规则

游戏目标

- 两名玩家分别代表皮城与祖安。
- 每位玩家的目标是通过合理使用卡牌，将对手的血量降至 0。

游戏设置

玩家设置：

- 每位玩家初始血量为H_1, H_2。
- 每轮只能打出一张卡牌。
- 玩家可以从手牌中选择卡牌，目标可以是对手或自己（例如治疗效果）。

卡牌设置：

- 卡牌分为 **普通卡牌** 和 **特殊卡牌**：
- **普通卡牌**：
- 海克斯炸弹(**Hex Bomb, HXB**)：攻击力为 $Attack$，直接对目标玩家造成伤害。
- 微光(**Shimmer, SH**)：恢复力为 $Heal$，恢复目标玩家血量，但不能超过初始血量。
- **特殊卡牌**：
- 时间冻结装置（**Time Freezing Equipment, TFE**）：使目标玩家跳过下一轮行动。
- 炼金护盾(**Alchemical Shield, AS**)：使玩家免疫一次伤害（治疗不受影响）。

游戏流程

1. **初始化：**

- 两名玩家创建角色（皮城或祖安），初始血量设定为 H_1, H_2，并带有种类的手牌（数量无限）。

2. **轮流行动：**

• 出牌阶段：玩家从手牌中选择 1 张卡牌，并指定目标玩家（对方或自己）使用。

• 普通卡牌直接结算伤害或恢复。

• 特殊卡牌产生附加效果，如眩晕对方或为自己生成护盾。

• 结算阶段：计算卡牌效果（例如减少血量、恢复血量或触发护盾等）。

3. **游戏结束：**

• 当一名玩家的血量降至 0，游戏结束，另一玩家获胜。

作为召唤师的你，肩负着维护符文之地平衡的重任，请按照规则搭建起这场巅峰对决的竞技平台，见证皮城与祖安为荣耀与未来展开的终极较量！

你要做的：

我们已经提供了代码框架以及必要的注释说明，你需要按照题目要求以及自己的理解完善对应代码，使整个程序可以顺利进行。

输入：

第一行与第二行，为两位玩家的昵称S_1, S_2（字符串）以及初始血量H_1, H_2（$0 \leq \max\{H_1, H_2\} \leq 10000$）。

接下来四行，会依次输入四种卡牌（分别为海克斯炸弹，微光，时间冻结装置，炼金护盾）的唯一编号$ID_i \in \{1, 2, 3, 4\}$，名称字符串$Name_i \in \{\text{HXB}, \text{MG}, \text{SH}, \text{AS}\}$，以及对应数值$0 \leq Value_i \leq 10000$, 其中特殊卡牌的数值固定为1。

接下来若干行代表玩家轮流出出牌，格式为`S_x ID_y S_z`，代表玩家S_x对玩家S_z使用了编号为ID_y的卡牌. (**注意，如果某玩家本轮动作应该跳过，则你需要忽略该输入操作**)

注意：特殊卡牌的效果不会叠加

输出：

经过上述对战后，如果有一方获胜，你需要打印出获胜方，如果没有，你需要打印出双方剩余血量（按照输入顺序）。

如果对战中途即有一方获胜，则需要忽略后续操作，直接输出结果。

具体的，如果对战双方为`Jinx`和`Caitlyn`，假如玩家`Jinx`获胜，你需要打印

```
Jinx wins!
```

如果无人胜利，你需要打印出双方剩余血量和特殊卡牌的效果（如果有），例如

```
Jinx 6
Caitlyn 6
```

Caitlyn被施加了护盾并且还在生效：

```
Jinx 6
Caitlyn 6 (Shield Active)
```

Caitlyn被施加了时间冻结装置并且还在生效：

```
Jinx 6
Caitlyn 6 (Stunned)
```

示例：

输入：

```
Jayce 20
Ekko 20
1 HXB 5
2 MG 3
3 SH 1
4 AS 1
3
Jayce 1 Ekko
Ekko 3 Ekko
Jayce 2 Jayce
```

输出：

```
Jayce 20
Ekko 15 (Stunned)
```

代码模板

我们提供了基本的代码模板，请你完成代码中的TODO部分。

```cpp
#include <iostream>
#include <sstream>
#include <string>
#include <unordered_map>
#include <memory>
#include <vector>

using std::unordered_map;
using std::string;
using std::cout;
using std::cin;
using std::endl;
using std::min;
using std::max;
using std::shared_ptr;
using std::make_shared;

```

```cpp
class Card {
public:
    string name;
    int value;

    Card(const string& name, int value) : name(name), value(value) {}

    virtual void use(class Player& user, class Player& target) = 0;

    virtual ~Card() = default;
};

class AttackCard : public Card {
public:
    AttackCard(const string& name, int value) : Card(name, value) {}

    void use(Player& user, Player& target) override;
};

class HealCard : public Card {
public:
    HealCard(const string& name, int value) : Card(name, value) {}

    void use(Player& user, Player& target) override;
};

class StunCard : public Card {
public:
    StunCard(const string& name, int value) : Card(name, value) {}

    void use(Player& user, Player& target) override;
};

class ShieldCard : public Card {
public:
    ShieldCard(const string& name, int value) : Card(name, value) {}

    void use(Player& user, Player& target) override;
};

class Player {
public:
    string name;
    int hp;
    int initialHp;
    bool shieldActive = false;
    bool stunned = false;

    Player(const string& name, int initialHp) : name(name), hp(initialHp), initialHp(initialHp) {}

    void takeDamage(int damage) {
```

```
        // TODO
    }

    void heal(int amount) {
        // TODO
    }

    void activateShield() {
        // TODO
    }

    void applyStun() {
        // TODO
    }

    void resetTurnStatus() {
        // TODO
    }

    bool isDefeated() const {
        // TODO
    }
};

class Game {
private:
    Player playerA, playerB;
    unordered_map<int, shared_ptr<Card> > cardLibrary;

public:
    Game(const string& playerAName, int hpA, const string& playerBName, int hpB)
        : playerA(playerAName, hpA), playerB(playerBName, hpB) {}

    void addCardToLibrary(int cardId, const string& name, int value) {
        // TODO
    }

    void playTurn(Player& currentPlayer, Player& opponent, int cardId, Player&
target) {
        if (currentPlayer.stunned) {
            currentPlayer.resetTurnStatus();
            return;
        }

        auto it = cardLibrary.find(cardId);
        it->second->use(currentPlayer, target);

        if (opponent.isDefeated()) {
            this->displayPlayerStatus();
            exit(0);
        }
    }
```

```
120
121     void displayPlayerStatus() {
122         // TODO
123     }
124
125     void processAction(const string &currentPlayerName, int cardId, const string
    &targetPlayerName) {
126         Player& currentPlayer = (currentPlayerName == playerA.name) ? playerA :
    playerB;
127         Player& opponent = (currentPlayerName == playerA.name) ? playerB : playerA;
128         Player& target = (targetPlayerName == playerA.name) ? playerA : playerB;
129
130         playTurn(currentPlayer, opponent, cardId, target);
131     }
132 };
133
134 void AttackCard::use(Player& user, Player& target) {
135     // TODO
136 }
137
138 void HealCard::use(Player& user, Player& target) {
139     // TODO
140 }
141
142 void StunCard::use(Player& user, Player& target) {
143     // TODO
144 }
145
146 void ShieldCard::use(Player& user, Player& target) {
147     // TODO
148 }
149
150 int main() {
151
152     // TODO
153
154     return 0;
155 }
```