Blackjack Game

一、游戏规则分析

1、游戏人数：1-7 人

2、工具：一副扑克牌，去掉大小王。

3、点数计算：每张牌都有点数，2 到 10 的牌的点数就是其牌面的数字；J、Q、K 的点数是 10 点；A 有两种算法，1 或者 11。

4、要牌规则：先由玩家要牌，最后到庄家要牌。庄家持牌总点数少于或等于 16 点，则必须要牌；其他情况由玩家决定是否要牌，直到爆牌或停止要牌，则轮到下一个玩家。

5、A 的算法：所有玩家如果 A 算为 11 时总和大于 21，则 A 算为 1，并且对于庄家，若把 A 算为 11 时点数总和大于 16 点且小于 22 点，则把 A 算为 11 点，否则算为 1 点。

6、游戏开始，每人两张牌，其中普通玩家的牌是明牌，而庄家的一张牌是暗牌。然后从第一个玩家开始要牌，当该玩家爆牌或选择不要牌，则轮到下一个玩家，最后轮到庄家。

7、输赢规则：爆牌意思是手中牌点数大于 21 点，爆牌则输。当庄家爆牌，全部玩家都爆牌，则平局。若都没爆牌，则点数最大者赢，若最大点数都为普通玩家，则庄家输，最大点数的普通玩家赢，若最大的点数中有普通玩家也有庄家，则平局。

二、系统 UML 类图(见下一页)

三、类规约说明

该工程共 5 个类，其中共 3 个大类，分别是类 Game21Point，类 Poker，类 Player，具体成员参考上述 UML 类图。

Game21Point 类：

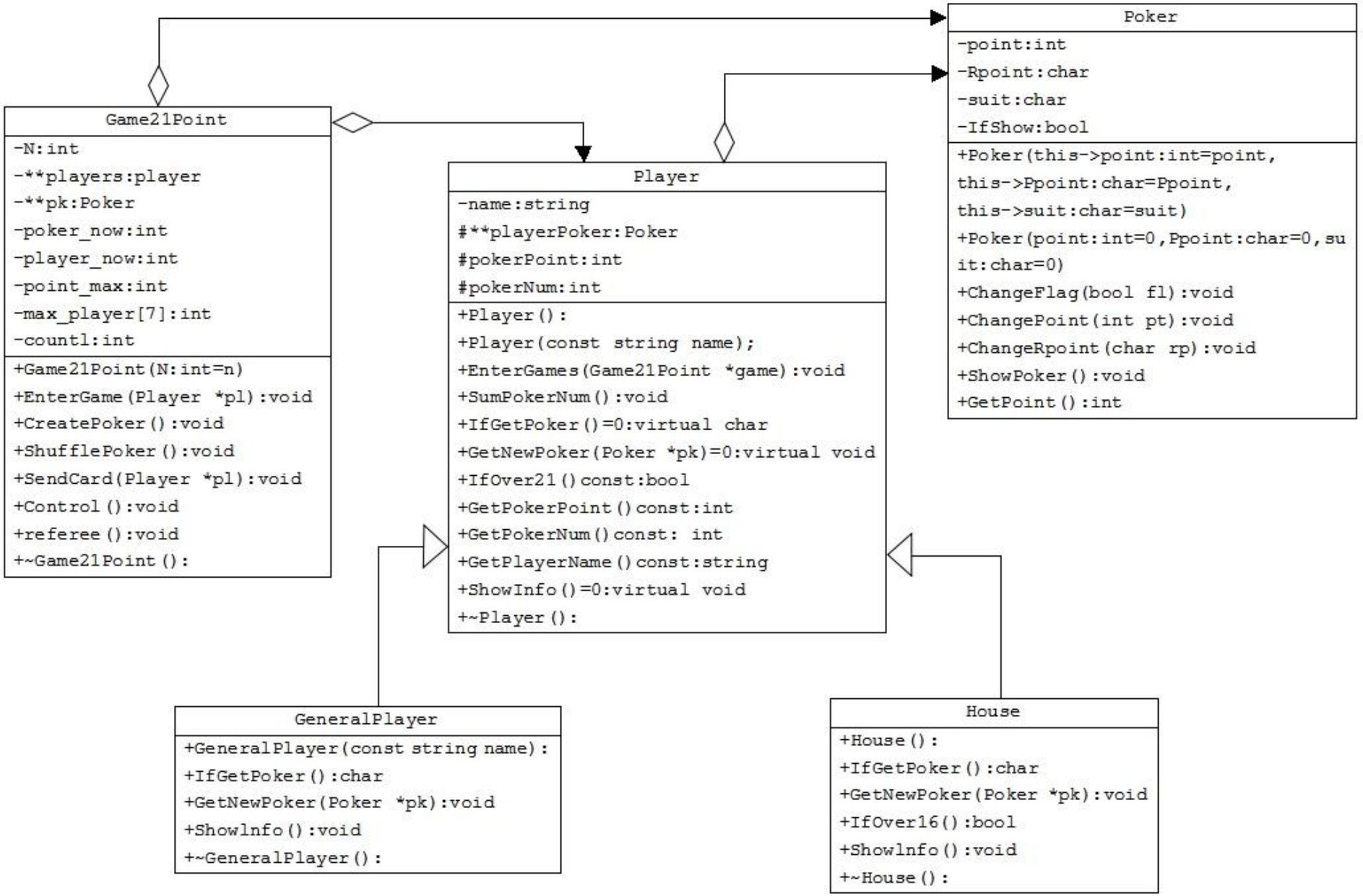
1、游戏系统类，用于洗牌发牌计数等控制游戏工作并判断输赢。

2、在 main 函数会实例化一个名为 game21point 的对象，用于实现游戏的进行。在 main 函数中主要调用 game21point 对象中的 Control()和 referee()函数。当然，也调用 EnterGame(Player *pl)函数，把玩家加入到游戏，方便在 game21point 对象中对玩家进行控制和操作。

3、类中的 Player **players 和 Poker **pk 均为双指针对象，之所以使用双指针是因为 Player 类是个抽象类，无法实例化，所以要双指针，而为了多态的实现，Poker 类也使用双指针而且这两个成员需要动态创建，下面提及的双指针对象都出于此目的。而且这样不会调用构造函数，提高效率，减少内存浪费。

4、类中的 ShufflePoker()函数是洗牌函数，会多次调用 Poker 类的默认复制构造函数，降低了程序的编译和执行效率。

5、类中的私有成员 int max_player[7];//最大点数的玩家序号（数组），因为规定玩家是 1-7 个，所以最大点数的玩家不会超过 7 个。


Poker
-point:int
-Rpoint:char
-suit:char
-IfShow:bool
+Poker(this->point:int=point,
this->Ppoint:char=Ppoint,
this->suit:char=suit)
+Poker(point:int=0,Ppoint:char=0,su
it:char=0)
+ChangeFlag(bool fl):void
+ChangePoint(int pt):void
+ChangeRpoint(char rp):void
+ShowPoker():void
+GetPoint():int
Game21Point
-N:int
-**players:player
-**pk:Poker
-poker_now:int
-player_now:int
-point_max:int
-max_player[7]:int
-countl:int
+Game21Point(N:int=n)
+EnterGame(Player *pl):void
+CreatePoker():void
+ShufflePoker():void
+SendCard(Player *pl):void
+Control():void
+referee():void
+~Game21Point():
Player
-name:string
#**playerPoker:Poker
#pokerPoint:int
#pokerNum:int
+Player():
+Player(const string name);
+EnterGames(Game21Point *game):void
+SumPokerNum():void
+IfGetPoker()=0:virtual char
+GetNewPoker(Poker *pk)=0:virtual void
+IfOver21()const:bool
+GetPokerPoint()const:int
+GetPokerNum()const: int
+GetPlayerName()const:string
+ShowInfo()=0:virtual void
+~Player():
GeneralPlayer
+GeneralPlayer(const string name):
+IfGetPoker():char
+GetNewPoker(Poker *pk):void
+Showlnfo():void
+~GeneralPlayer():
House
+House():
+IfGetPoker():char
+GetNewPoker(Poker *pk):void
+IfOver16():bool
+Showlnfo():void
+~House():

Poker 类

1、扑克牌类，比较简单。私有成员都是扑克牌的基本属性，而共有成员主要用于提供对外接口和修改私有成员，由于游戏规则的特点，部分扑克牌的属性和原本不一样，所以在 Game21Point 类初始化扑克牌的时候需要修改属性。

2、考虑到方便表达，所以私有成员 4 个，一个是点数，一个是花色，一个是牌面字母，一个是是否翻牌。有 16 张扑克牌牌面不是数字（A,J,Q,K），多了一个牌面字母，而是否翻牌主要考虑到庄家和普通玩家有明牌和暗牌的差别。现实中的扑克牌初始时也是暗牌的。

3、在 Game21Point 类中有会创建一个 Poker 的双指针，是一副没有大小王的扑克牌，52 张。

4、在 Player 类的两个派生类中也会创建一个 Poker 的双指针，是玩家手中的牌，动态创建一个有 7 个对象指针的对象数组（按照规则玩家最多拥有 7 张牌）。

5、整个工程会调用 Poker 类的默认复制构造函数，由于不需要深复制，所以不另外编写。

Player 类

1、玩家类，是一个抽象类，派生出了两个类：类 GeneralPlayer 和类 House，分别是普通玩家类和庄家类。用于实现多态。

2、抽象类无法实例化，在 main 函数个 Game21Point 类中分别创建了双指针，用于多态的实现。

3、两个类中函数实现不同的主要有以下函数：

```
virtual char IfGetPoker()=0;//是否要牌
virtual void GetNewPoker(Poker *pk)=0;//抽牌
virtual void ShowInfo()=0;
```

House 类中还多了一个公有成员 `bool IfOver16();`

GeneralPlayer 类

1、公有继承 Player 类，普通玩家类。

2、在 main 函数中会根据用户输入的普通玩家数动态创建 1-7 个对象指针，并调用加入游戏系统的函数。

3、char IfGetPoker()为是否要牌函数，返回类型是 char，即字符'y'和'n'。提示用户是否需要牌，由 Game21Point 类中 Control()函数识别并作出下一步操作。当用户输入'n'时，即不需要继续要牌，此时隐含条件为该玩家仍未爆牌，但由于 A 在停止要牌前系统都先当 1 计算，若输入的是'n'，则在此函数处理手上 A 的牌：所有玩家如果 A 算为 11 时总和大于 21，则 A 算为 1。House 类也做同样的处理。

4、void GetNewPoker(Poker *pk)为抽牌函数，形参是一个 Poker 类的指针，Game21Point 类中 SendCard(Player *pl)函数会调用该函数，并通过传参方式传来一张牌。该函数首先把传过来的扑克牌记录成为玩家手中的扑克牌，然后改变该牌的翻牌属性为明牌，最后调用计算玩家手中牌的点数并玩家手中牌自增。House 类的功能类似，但步骤上有差别。

5、void ShowInfo()为打印基本信息函数，包括名字，手中的牌，和总点数，带有格式化输出。开局发牌两张后，Game21Point 类中 Control()函数会调用各玩家的该函数，每次调用 SendCard(Player *pl)函数也会调用该函数。

House 类

1、公有继承 Player 类，庄家类。

2、在 main 函数中动态创建完普通玩家对象指针后，会动态创建一个 House 指针，并调用加入游戏系统的函数。

3、bool IfOver16()为手中点数是否大于 16 点的函数，由于庄家持牌总点数少于或等于 16 点，则必须要牌。在此函数中对 A 的处理为：若把 A 算为 11 时点数总和大于 16 点，则把 A 算为 11 点，否则算为 1 点。

4、char IfGetPoker()要牌函数，拥有和 GeneralPlayer 类中要牌函数一样的功能外，还会先调用自身的 fOver16()函数，如果点数小于或等于 16 点，则直接返回'y'提示系统发牌，之

后再执行和 GeneralPlayer 类中该函数一样的操作。

5、void GetNewPoker(Poker *pk)功能上和 GeneralPlayer 类一样，但是由于庄家得到的第一（0）张牌，默认对外不可见即暗牌，所以如果判断到这是传过来的第一张扑克牌，那么就不同修改其明暗属性，但之后的牌都要修改。其他操作和 GeneralPlayer 类中该函数一样。

6、void ShowInfo()和 GeneralPlayer 类中该函数不一样的地方是庄家要牌时，第一张牌是不可见的，但之后这张牌就可见了，换句话说，第一次调用该函数时就可以修改第一张牌的翻牌属性，并且第一次调用该函数时是不需要打印总点数的，因为有张牌是不可见。

四、系统测试方案

由于发牌具有随机性，所以无法控制所需要情况的直接出现，但预定需要出现的情况如下：

1、1 个玩家的情况，名字为 Mike

2、2 个玩家的情况，名字分别为 Mike 和 Job

3、7 个玩家的情况，名字分别为 ya, yb, yc, yd, ye, yf, yg。

4、1 个普通玩家赢或庄家赢。

5、普通玩家和庄家同为最大点数，此时平局。

6、所有玩家都爆牌，此时平局。

7、2 个或以上普通玩家赢的情况，此时玩家赢。

8、有普通玩家手上有 A 这张牌，且他赢了。

9、庄家手上有 A 这张牌，且庄家赢了。

10、上述 9 种情况可以不退出游戏一直选择再来一局，直到所有情况出现才选择结束游戏。

需要注意的地方有：

1、点数计算是否正确。每个玩家选择不继续要牌后会打印一次，这时候要尤其注意有 A 的情况是否正确处理。

2、最大点数的记录是否正确。轮到每个玩家要牌前会打印提示一次当前最大点数（第一个要牌玩家除外），若前面玩家都爆牌，则最大点数为 0.

3、庄家自动发牌是否正确，尤其是有 A 的情况。

五、调试结果

1、语法上的错误

声明一个类的时候，成员变量在声明的时候其实还没有分配内存空间，所以是不能初始化的，要在实例化的时候初始化与赋值，其方式是调用构造函数。内嵌对象则无法直接调用带参的构造函数。例如：

```
1  #include<iostream>
2  using namespace std;
3
4  class A{
5  public:
6      A(int i):i(i){};
7  private:
8      int i;
9  };
10
11 class B{
12     A a(2);
13 };
14
15 int main(){
16     B b;
17     return 0;
18 }
19
```

Line	Message
12	error: expected identifier before numeric constant
12	error: expected ',' or '...' before numeric constant
	=== Build failed: 2 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ===

若把第 12 行的改成：A a，则编译结果如下：

Line	Message
	In constructor 'B::B()':
11	error: no matching function for call to 'A::A()'
11	note: candidates are:
6	note: A::A(int)
6	note: candidate expects 1 argument, 0 provided

由于只要人为写了构造函数，系统就不再自动生成默认构造函数，需要自己写上。

改法有两种：

```
1  #include<iostream>
2  using namespace std;
3
4  class A{
5  public:
6      A(int i):i(i){};
7      A(){};
8  private:
9      int i;
10 };
11
12 class B{
13     A a;
14 };
15
16 int main(){
17     B b;
18     return 0;
19 }
20
```

←加上一个默认构造函数。

加上一个 B 类的构造函数对内嵌对象 a 进行初始化。→

```
1  #include<iostream>
2  using namespace std;
3
4  class A{
5  public:
6      A(int i):i(i){};
7  private:
8      int i;
9  };
10
11 class B{
12 public:
13     B(int j):a(j){};
14 private:
15     A a;
16 };
17
18 int main(){
19     B b(2);
20     return 0;
21 }
22
```

2、算法上的错误

最大点数玩家序号和个数记录有误，算法上存在逻辑性错误。错误在 Control（）函数中。

调试办法：在适当位置打印相关变量

```
34      for(;player_now<N;player_now++){
35          if(player_now != 0)
36              cout<<"                    Now, the max point is "<<point_max<<endl;
37          players[player_now]->ShowInfo();
38          while(1){
39              if(players[player_now]->IfGetPoker()  == 'y'){
40                  SendCard(players[player_now]);
41                  players[player_now]->ShowInfo();
42                  if(players[player_now]->IfOver21()){
43                      cout<<players[player_now]->GetPlayerName()<<" busts."<<endl;
44                      break;
45                  }
46              }
47              else{
48                  if(player_now != N-1){
49                      if(players[player_now]->GetPokerPoint() > point_max)
50                          point_max = players[player_now]->GetPokerPoint();
51                      else if(players[player_now]->GetPokerPoint() == point_max)
52                          count1++;
53                      max_player[count1] = player_now;                cout<<max_player[count1]<<endl;
54                  }
55                  break;
56              }
57          }
58      }
```

只要一个普通玩家既没有爆牌，且选择停止要牌，则无论如何都记录成为最大点数玩家序号。

调试办法：适当打印变量

①上述截图中逻辑上，若一个普通玩家既没有爆牌，且选择停止要牌，那么这时候要判断是否最大点数，是，则记录，并且检索是否第一个玩家最大点数。然后记录最大点数玩家序号。

②但是只要一个普通玩家既没有爆牌，且选择停止要牌，则无论如何都记录成为最大点数玩家序号，并且容易数组造成越界。

③开始时程序偶尔会异常报错强制结束。调试时打印变量发现变量并非一个数字，而是一串无规律的类似于地址的数字，于是发现在此数组可能越界了。

④改正方法可以把第 53 行代码发到两个“else if”分支中，并用大括号扩起来。

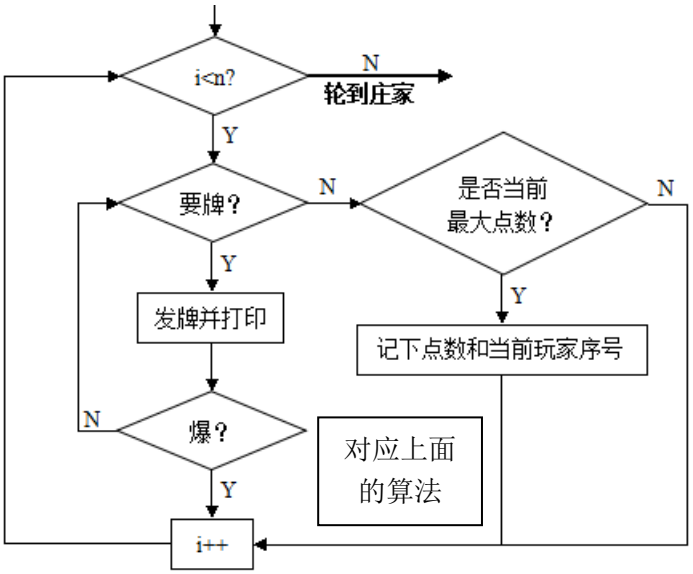
```
47              else{
48                  if(player_now != N-1){
49                      if(players[player_now]->GetPokerPoint() > point_max){
50                          point_max = players[player_now]->GetPokerPoint();
51                          max_player[count1] = player_now;
52                      }
53                      else if(players[player_now]->GetPokerPoint() == point_max){
54                          count1++;
55                          max_player[count1] = player_now;
56                      }
57                  }
58                  break;
59              }
```

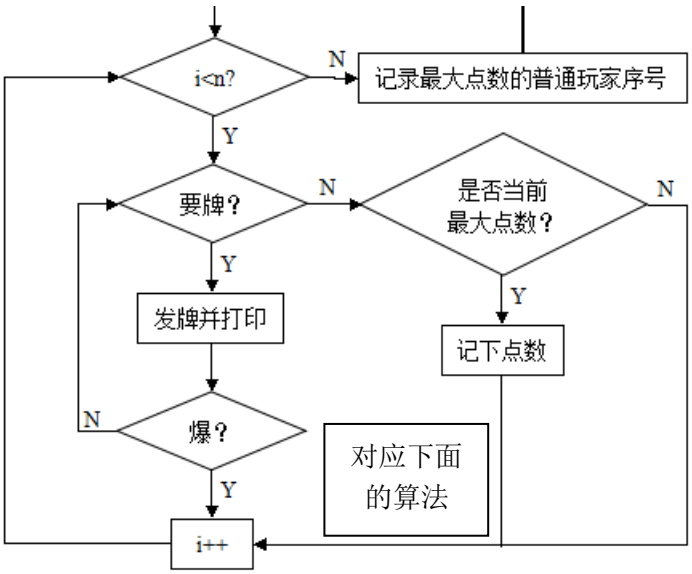
但是这样还有一个逻辑错误：若恰巧之前最大点数的玩家有两个，假设序号为 1 和 2，此时出现一个新的最大的，序号为 3，那么执行后系统记录的最大点数玩家仍有两个，序号分别为 1 和 3 显然是错误的。
①该错误的发现非通过测试测出，是大家研究函数时细心发现的。
②解决办法：在第 50 行后加入一句：count1=0;

流程图微调避免错误

之前流程图及其算法：

```
34      for(;player_now<N;player_now++){
35          if(player_now != 0)
36              cout<<"                    Now, the max point is "<<point_max<<endl;
37          players[player_now]->ShowInfo();
38          while(1){
39              if(players[player_now]->IfGetPoker()  == 'y'){
40                  SendCard(players[player_now]);
41                  players[player_now]->ShowInfo();
42                  if(players[player_now]->IfOver21()){
43                      cout<<players[player_now]->GetPlayerName()<<" busts."<<endl;
44                      break;
45                  }
46              }
47              else{
48                  if(player_now != N-1){
49                      if(players[player_now]->GetPokerPoint() > point_max){
50                          point_max = players[player_now]->GetPokerPoint();
51                          max_player[count1] = player_now;
52                      }
53                      else if(players[player_now]->GetPokerPoint() == point_max){
54                          count1++;
55                          max_player[count1] = player_now;
56                      }
57                  }
58                  break;
59              }
60          }
```



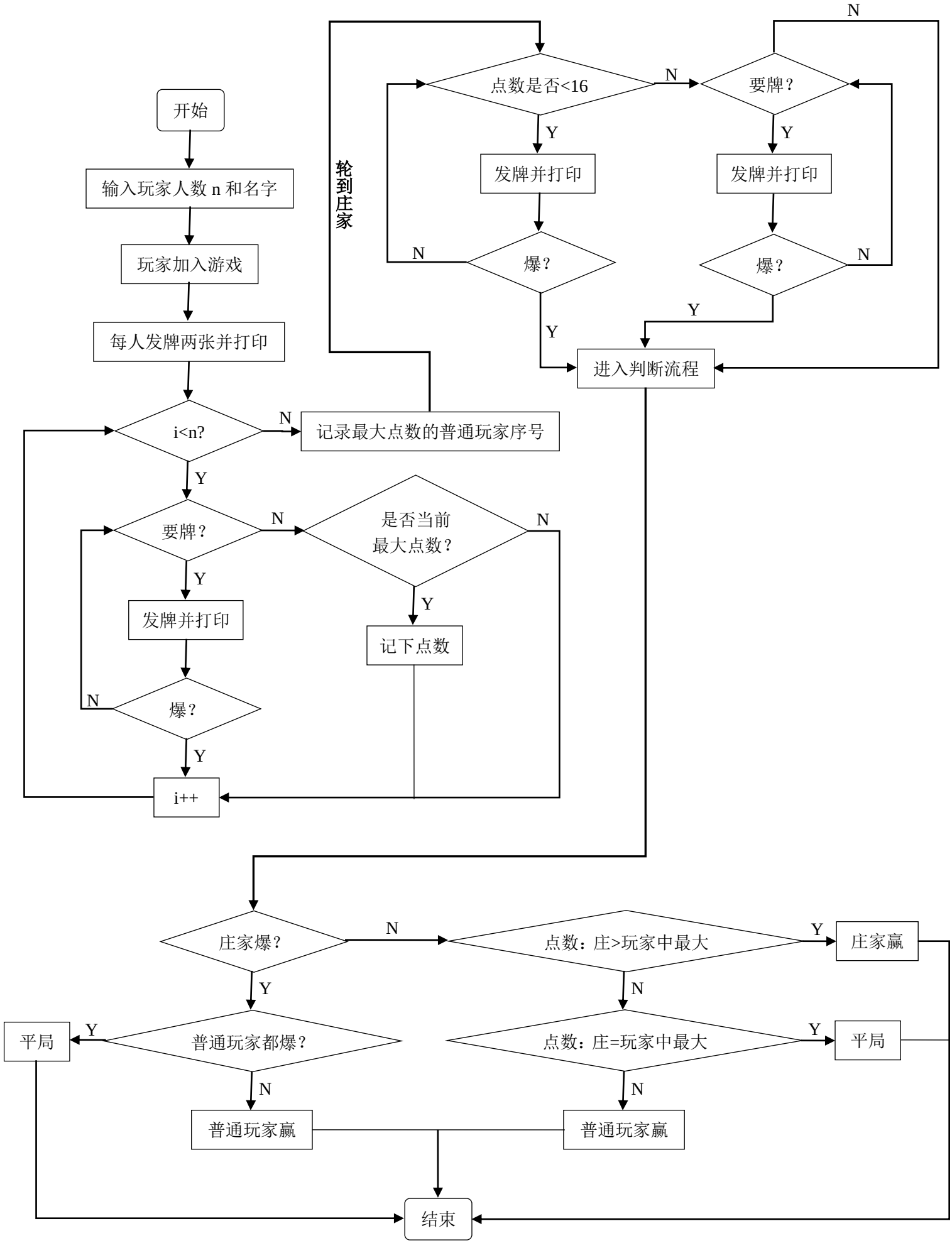



```
for(;player_now<N;player_now++){
    if(player_now != 0)//提示用户当前最大点数，给是否要牌的决定作一个参考
        cout<<"                    Now, the max point is "<<point_max<<endl;
    players[player_now]->ShowInfo();
    //玩家过多的时候由于屏幕大小有限，之前打印的信息可能要滚上去才能看见，这样给用户带来麻烦，所以打印一次

    while(players[player_now]->IfGetPoker()  == 'y'){
        //如果要牌，则发牌并打印
        SendCard(players[player_now]);
        players[player_now]->ShowInfo();
        //判断是否爆牌，爆牌则打印信息并跳出循环，轮到下一个玩家
        if(players[player_now]->IfOver21()){
            cout<<players[player_now]->GetPlayerName()<<" busts."<<endl;
            break;
        }
    }
    //玩家停止要牌后，判断是否为当前最大点数，是，则记录最大点数到系统，不是则继续轮流下一个玩家，本操作不处理庄家
    if(!players[player_now]->IfOver21() && player_now != N-1)
    {
        cout<<"Now, "<<players[player_now]->GetPlayerName()<<"'s point is "<<players[player_now]->GetPokerPoint()<<endl;
        //打印信息是为了对a的处理结果进行显示
        if(players[player_now]->GetPokerPoint() > point_max)
            point_max = players[player_now]->GetPokerPoint();
    }
    cout<<endl;
}
//所有玩家轮流完后，记录最大点数的玩家序号，不处理庄家
for(player_now = 0;player_now < N-1;player_now++){
    if(players[player_now]->GetPokerPoint() == point_max){
        max_player[count1] = player_now;
        count1++;
    }
}
```

小组讨论最终决定微调算法原因是最初的流程图思路是没错，但是实现起来比较麻烦，因为遇到循环和判断合在一起并多种这样的双重结构嵌套的情况，我们已学知识中的分支和循环结构并没有一个语句能直接实现，在编写代码是就遇到一个问题，for 循环的用处很大，但却没有 if...else...语句的选择功能，于是刚开始强制用 while（1）这种方法先循环后判断，但是这个循环结构明显写得没有任何意义，有点强词夺理的意味，于是决定改变算法。把统计最大点数玩家序号这一工作放到大家都停止要牌后进行，这样就少了一层嵌套，而且也符合现实。两种算法的不同点在于对最大点数玩家序号记录这个工作上，前者是一个玩家结束统计一次，后者是所有玩家结束再进行统计。后者算法实现起来比前者容易且出错率低。

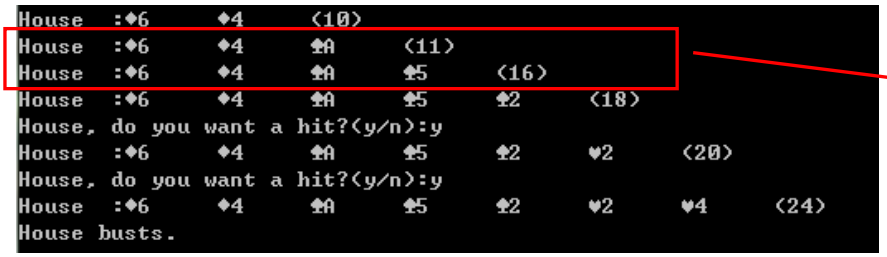
附完整流程图：


开始
输入玩家人数 n 和名字
玩家加入游戏
每人发牌两张并打印
i<n?
N
记录最大点数的普通玩家序号
轮到庄家
Y
要牌？
N
是否当前
最大点数？
N
Y
发牌并打印
记下点数
爆？
N
Y
i++
点数是否<16
N
要牌？
Y
Y
发牌并打印
发牌并打印
爆？
N
爆？
N
Y
Y
进入判断流程
庄家爆？
N
点数：庄>玩家中最大
Y
庄家赢
Y
N
平局
Y
普通玩家都爆？
点数：庄=玩家中最大
Y
平局
N
N
普通玩家赢
普通玩家赢
结束

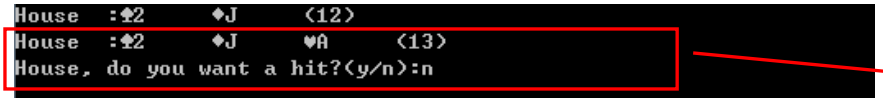
庄家对 A 的处理逻辑上有误

主要是在庄家 IfOver16()函数中逻辑错误

①发现过程：小组成员测试发现错误



正常情况下，当庄家手中牌为 6、4、A 时，这时候 A 应该当 11 算这样庄家点数为 21>16，系统不需要为其发牌，但此时系统为他自动发牌了。



把 A 当 11 算的时候点数超过 21 而当 1 没超过的 16 的情况，应该算是小于 16 点的情况，系统会自动为其发牌。但此时系统不为他自动发牌。

错误代码如下：

```
51  /*
52  庄家持牌总点数少于或等于16点，则必须要牌
53  若把a算为11时点数总和大于16点，则把a算为11点，否则算为1点。
54  */
55  bool House::IfOver16(){
56      int i = 0;
57      while(playerPoker[i]->GetPoint() == 1 && i < pokerNum){
58          if(pokerPoint + 10 > 16)
59              return true;//最多也就有一张a能看成11，所以检索到有a便可以处理并结束程序
60          i++;
61      }
62      if(pokerPoint > 16)
63          return true;
64      else
65          return false;
66  }
```

注意到进入 while 循环的条件有俩，一是当前检索的牌为 A，而是当前检索的牌序号没超过手中牌的数量。这样如果第 1 张牌不是 A，那么就不会进入循环。上述情况是第 3 张牌为 A，所以没能进入循环，A 任然当 1 处理。

同时如果把 A 当 11 算的时候点数超过 21 而当 1 没超过的 16 的情况，应该算是小于 16 点的情况，系统会自动为其发牌。但按照上述算法，该情况会不自动发牌。

改正如下：把两个条件分离。并完善条件:是 16<pokerPoint<=21

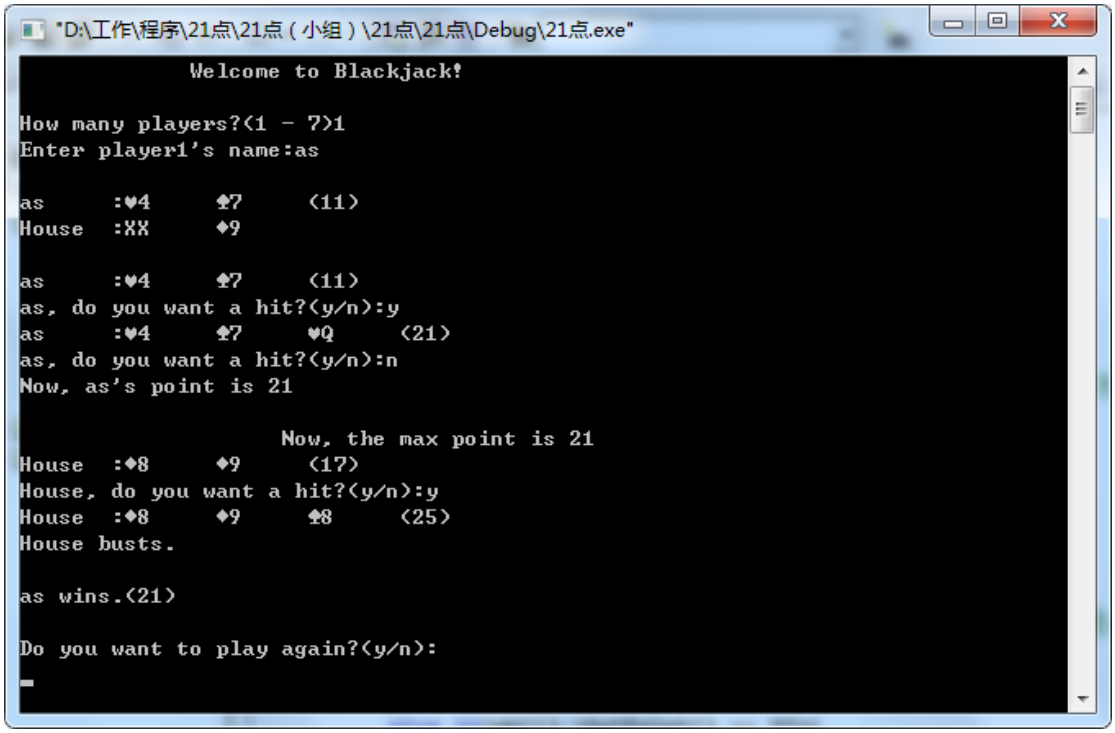
```
55  bool House::IfOver16(){
56      for(int i = 0; i < pokerNum; i++)
57          if(playerPoker[i]->GetPoint() == 1)
58              if(pokerPoint + 10 > 16 && pokerPoint + 10 <= 21)
59                  return true;//最多也就有一张a能看成11，所以检索到有a便可以处理并结束程序
60      if(pokerPoint > 16)
61          return true;
62      else
63          return false;
64  }
```

除了上述错误，还有一些粗心大意的错误。

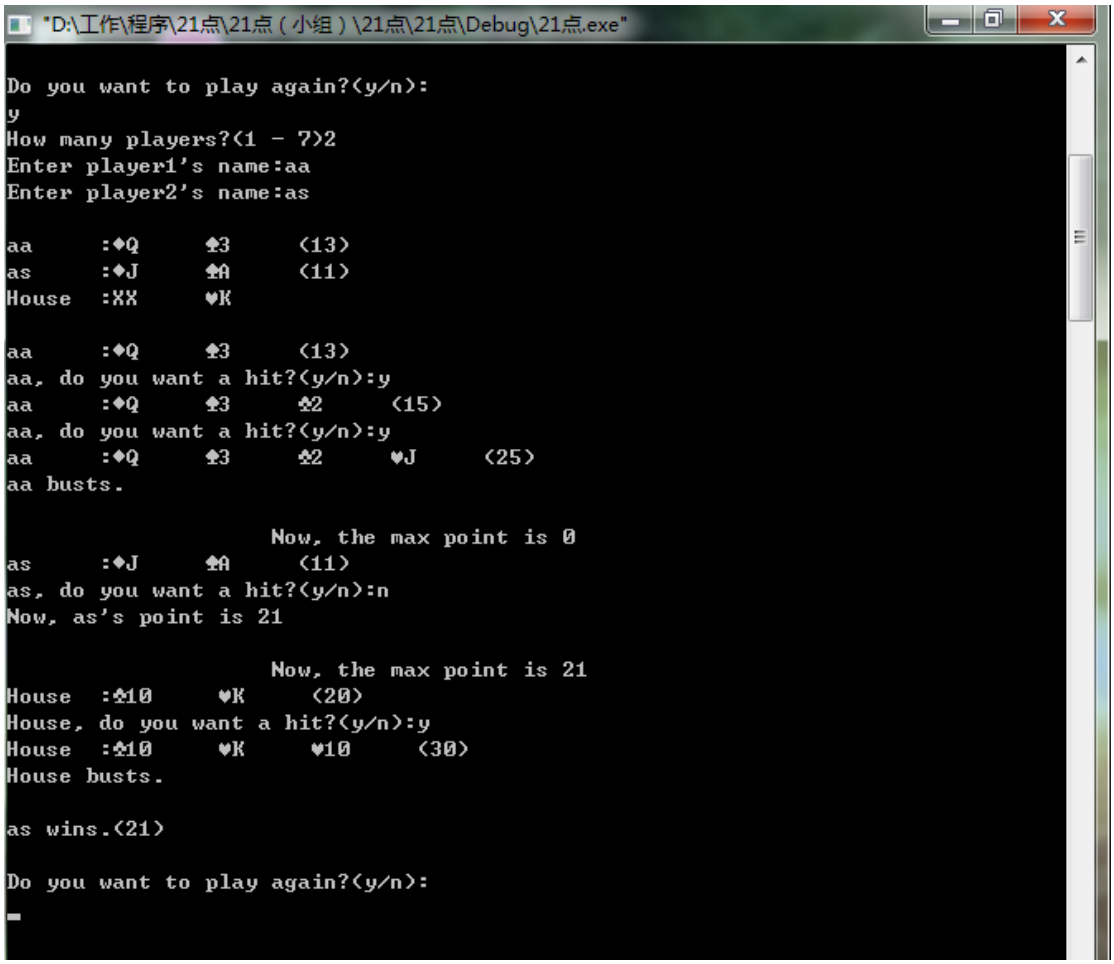
一方面是小组里相互检查代码时发现，一方面是测试的时候发现执行结果不对，然后加入相关打印语句进行测试。

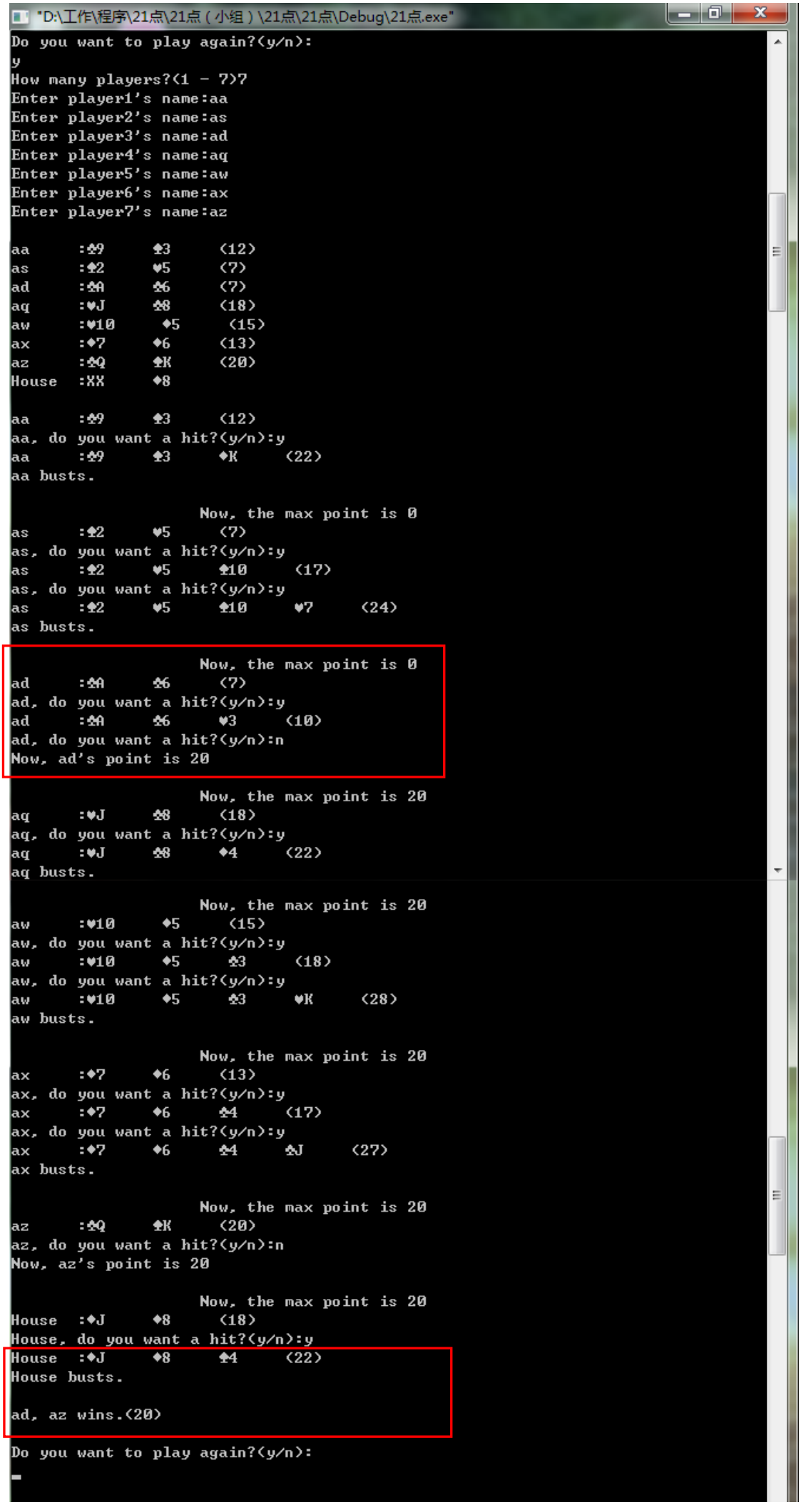
六、程序运行结果示例

①
玩家数：1
结局：玩家赢



②
玩家数：2
结局：其中一个玩家赢





③

玩家数：7

结局：其中两个玩家赢。

说明：

1、玩家 ad 有 A，且牌没爆，能正确处理。

2、两个玩家 ad 和 az 同时获得最大点数，同时赢，显示正确。

④

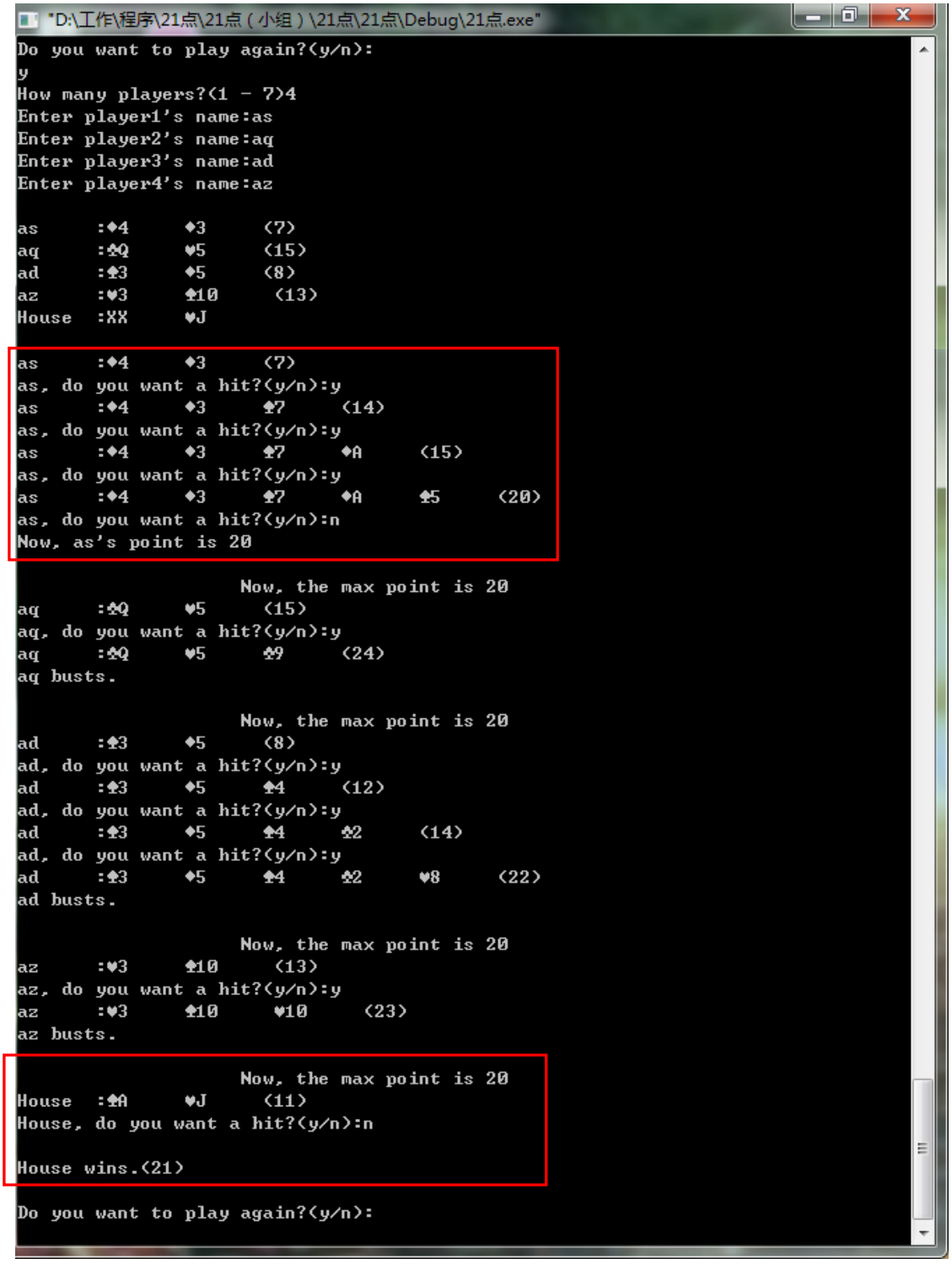
玩家数：4

结局：庄家赢

说明：

1、玩家 as 有 A，且牌没爆，能正确处理。

2、庄家 A 也能正确处理，使得庄家不需要系统自动为其发牌并以 21 点赢。



⑤

玩家数：4

结局：平局

说明：玩家 q 中的 A 处理正确。