```
 1: //modified and completed by Kalli Bonin
 2: //Assignment #9 - Question 1 - CardHand Class
 3:
 4: #include <iostream>
 5: #include <stdlib.h>     // rand
 6: #include <time.h>       // time
 7: #include <fstream>
 8:
 9: using namespace std;
10:
11: enum Suit {NONE, CLUBS, HEARTS, DIAMONDS, SPADES};
12:
13: class Card
14: {
15: public:
16:     int value;
17:     Suit suit;
18:
19:     //default constructor to ensure empty hand
20:     Card()
21:     {
22:         value = 0;
23:         suit = NONE;
24:     }
25: };
26:
27: class CardHand
28: {
29: private:
30:     Card hand[52];
31:     int handSize;
32:     void SortCards();
33: public:
34:     int GetHandSize() const;
35:     bool AddCard(Card new_card);
36:     int BetterHand(CardHand otherHand);
37:     void PrintHand(ostream & out);
38:
39:     CardHand()
40:     {
41:         handSize = 5;
42:     }
43:
44:     CardHand(int handSize0)
45:     {
46:         //check to see if it exceeds max size or min size
47:         if (handSize0 > 52 || handSize0 < 1)
48:             handSize = 5;
49:         else
50:             handSize = handSize0;
```

```
51:     }
52: };
53:
54: bool DealCards(CardHand hands[], int num_hands)
55: {
56:     // check if there are enough cards (only 52 cards available)
57:     int cards_needed = 0;
58:     for (int hand_index = 0; hand_index < num_hands; hand_index++)
59:     {
60:         cards_needed += hands[hand_index].GetHandSize();
61:     }
62:     if (cards_needed > 52)
63:         return false;
64:
65:     const int NUM_CARDS = 52;
66:     Card deck[NUM_CARDS];
67:     int current_value = 1;
68:     Suit current_suit = CLUBS;
69:
70:     // create the deck of cards
71:     for (int card_index = 0; card_index < NUM_CARDS; card_index++)
72:     {
73:         deck[card_index].value = current_value;
74:         deck[card_index].suit = current_suit;
75:
76:         current_value++;
77:         if (current_value > 13)
78:         {
79:             switch (current_suit)
80:             {
81:                 case CLUBS: current_suit = HEARTS;
82:                     break;
83:                 case HEARTS: current_suit = DIAMONDS;
84:                     break;
85:                 case DIAMONDS: current_suit = SPADES;
86:                     break;
87:                 default: break;
88:             }
89:             current_value = 1;
90:         }
91:     }
92:
93:     // seed the random number generator
94:     srand(time(NULL));
95:
96:     // randomly pull cards from the deck and assign them to each hand
97:     for (int hand_index = 0; hand_index < num_hands; hand_index++)
98:     {
99:         for (int card_index = 0; card_index < hands[hand_index].GetHandSize();
100:         card_index++)
```

```
101:         {
102:             Card assign_card;
103:             assign_card.value = 0;
104:             assign_card.suit = NONE;
105:
106:             while (assign_card.value == 0)
107:             {
108:                 int try_card = rand() % 52;
109:
110:                 if (deck[try_card].value > 0)
111:                 {
112:                     assign_card.value = deck[try_card].value;
113:                     assign_card.suit = deck[try_card].suit;
114:                     deck[try_card].value = 0;
115:                     deck[try_card].suit = NONE;
116:                 }
117:             }
118:
119:             hands[hand_index].AddCard(assign_card);
120:         }
121:     }
122:
123:     return true;
124: }
125:
126: int main()
127: {
128:     const int NUM_HANDS = 4;
129:
130:     //calls default constructor so there will be 5 cards in each of the 4 hands
131:     CardHand player[NUM_HANDS];
132:
133:
134:     DealCards(player, NUM_HANDS);
135:
136:     //create and check output file
137:     ofstream fout("hands_list.txt");
138:     if (!fout)
139:     {
140:         cout << "Could not open file.";
141:         return EXIT_FAILURE;
142:     }
143:
144:     //print all four hands
145:     for (int i = 0; i < NUM_HANDS; i++)
146:         player[i].PrintHand(fout);
147:
148:     //comparing hands
149:     int best = 0;
150:     for (int i = 1; i < NUM_HANDS; i++)
```

```
151:     {
152:         if (player[i].BetterHand(player[best]) == 1)
153:             best = i;
154:     }
155:
156:     fout << "Player " << best+1 << " has the best hand." << endl;
157:
158:     return 0;
159: }
160:
161: int CardHand::GetHandSize() const
162: {
163:     return handSize;
164: }
165:
166: bool CardHand::AddCard(Card new_card)
167: {
168:     //check within the bounds of the hand
169:     for (int i = 0; i < handSize; i++)
170:     {
171:         //ensure space is empty
172:         if (hand[i].value == 0 && hand[i].suit == 0)
173:         {
174:             hand[i].value = new_card.value;
175:             hand[i].suit = new_card.suit;
176:             return true;
177:         }
178:     }
179:
180:     //if it doesn't find space in hand
181:     return false;
182: }
183:
184: void CardHand::SortCards()
185: {
186:     //sort by ascending value
187:     for (int pass = 0; pass < handSize - 1; pass++)
188:     {
189:         int minValue = hand[pass].value;
190:         Suit minSuit = hand[pass].suit;
191:         int minIndex = pass;
192:
193:         for (int check = pass+1; check < handSize; check++)
194:         {
195:             if (hand[check].value > minValue)
196:             {
197:                 minValue = hand[check].value;
198:                 minSuit = hand[check].suit;
199:                 minIndex = check;
200:             }
```

```
201:
202:
203:         }
204:
205:         hand[minIndex].value = hand[pass].value;
206:         hand[minIndex].suit = hand[pass].suit;
207:
208:         hand[pass].value = minValue;
209:         hand[pass].suit  = minSuit;
210:
211:     }
212:
213:     //sort by ascending suit
214:     int curValue = 1;
215:
216:     for (int pass = 0; pass < handSize - 1; pass++)
217:     {
218:         int minValue = hand[pass].value;
219:         Suit minSuit = hand[pass].suit;
220:         int minIndex = pass;
221:         int check = pass + 1;
222:
223:         while (hand[check].value == minValue)
224:         {
225:             if (hand[check].suit > minSuit)
226:             {
227:                 minValue = hand[check].value;
228:                 minSuit = hand[check].suit;
229:                 minIndex = check;
230:             }
231:
232:             check++;
233:         }
234:
235:         hand[minIndex].value = hand[pass].value;
236:         hand[minIndex].suit = hand[pass].suit;
237:         hand[pass].value = minValue;
238:         hand[pass].suit = minSuit;
239:     }
240:
241: }
242:
243: int CardHand::BetterHand(CardHand otherHand)
244: {
245:     //this function takes into account the value of the card
246:     //the hand with the highest sum of card values has the better hand
247:     int thisSum = 0, otherSum = 0;
248:
249:     for (int i = 0; i < handSize; i++)
250:     {
```

```
251:         thisSum += (*this).hand[i].value;
252:         otherSum += otherHand.hand[i].value;
253:     }
254:
255:     if (thisSum > otherSum)
256:         return 1;
257:     else if (thisSum < otherSum)
258:         return -1;
259:     else
260:         return 0;
261: }
262:
263: void CardHand::PrintHand(ostream & out)
264: {
265:     SortCards();
266:
267:     out << "------------------------ start of hand ---------------------------"
268:         << endl;
269:
270:     for (int i = 0; i < handSize; i++)
271:     {
272:         //use switch-case to assign letter values to face cards
273:         switch (hand[i].value)
274:             {
275:                 case 1:
276:                     out << "A";
277:                     break;
278:                 case 11:
279:                     out << "J";
280:                     break;
281:                 case 12:
282:                     out << "Q";
283:                     break;
284:                 case 13:
285:                     out << "K";
286:                     break;
287:                 default:
288:                     out << hand[i].value;
289:                     break;
290:             }
291:
292:         //use switch-case to assign suit string
293:         switch (hand[i].suit)
294:             {
295:                 case CLUBS:
296:                     out << " CLUBS" << endl;
297:                     break;
298:                 case HEARTS:
299:                     out << " HEARTS" << endl;
300:                     break;
```

```
301:                 case DIAMONDS:
302:                     out << " DIAMONDS" << endl;
303:                     break;
304:                 case SPADES:
305:                     out << " SPADES" << endl;
306:                     break;
307:                 default:
308:                     out << " NONE" << endl;
309:                     break;
310:             }
311:     }
312:
313:
314:     out << "------------------------ end of hand ---------------------------"
315:         << endl;
316:
317: }
```