1 cards.h

```
1  // cards.h
2
3  #ifndef CARDS_H
4  #define CARDS_H
5
6  //
7  //  WAR card game
8  //
9  //  Described in Chapter 2 of
10 //  Data Structures in C++ using the STL
11 //  Published by Addison-Wesley, 1997
12 //  Written by Tim Budd, budd@cs.orst.edu
13 //  Oregon State University
14 //
15
16 // Updated by Mark Sebern, MSOE
17 // For ISO/ANSI C++, MSVC 6.0, and MSOE coding conventions,
18 // using vector (since this is covered in earlier classes).
19 //
20 // Version 1.01
21 // 29 November 1999
22
23 #include <iostream>
24 #include <vector>
25 using namespace std;
26
27 // Card - a playing card.
28
29 // Define types for card suit and rank.
30 enum suits {diamond, club, heart, spade};
31 // Rank: 1(ace), 2..10, 11..13 (Jack, Queen, King)
32 const unsigned int rank_max = 13;
33
34 class Card
35 {
36 public:
37   // No-arg constructor - default value is Ace of Spades
38   Card();
39
40   // Copy constructor
41   Card(const Card& orig);
42
43   // Constructor specifying card suit and rank
44   // Argument(s):
45   //  sv - suit
46   //  rv - rank (1..13)
47   Card(suits sv, unsigned int rv);
48
49   // Destructor
50   ~Card();
51
52   // Copy assignment operator
```

```
53    Card& operator=(const Card& rhs);
54
55    // Accessors:
56    unsigned int GetRank() const;
57    suits GetSuit() const;
58
59    // Mutators:
60    void SetRank(unsigned int rv);
61    void SetSuit(suits sv);
62
63    // Insert - insert string representation of card into stream.
64    // Argument(s):
65    //  out - output stream.
66    void Insert(ostream& out) const;
67
68  private:
69    // Rank of card (0..13)
70    unsigned int rank;
71    // Suit of card.
72    suits suit;
73  };
74
75  // Insertion operator for Card class.
76  // Argument(s):
77  //  out - stream to output to
78  //  aCard - card to output
79  ostream& operator<<(ostream& out, const Card& aCard);
80
81  // Deck - a deck of cards
82  class Deck
83  {
84  public:
85    // No-arg constructor: creates a full deck.
86    Deck();
87
88    //Copy constructor
89    Deck(const Deck& orig);
90
91    // Destructor
92    ~Deck();
93
94    // Copy assignment operator
95    Deck& operator=(const Deck& rhs);
96
97    // shuffle
98    // Randomize the order of cards in the deck.
99    void shuffle();
100
101   // isEmpty
102   // Test if the deck is empty (has no cards in it)
103   // Returns: true if no cards, false otherwise
104   bool isEmpty() const;
105
106   // draw
```

```
107     // Remove and return a card from the deck.
108     // Precondition(s):
109     //  Deck must not be empty.
110     // Returns: last card from deck.
111     // Postcondition(s):
112     //  Number of cards in deck is reduced by one.
113     Card draw();
114
115   private:
116     // Cards in the deck.
117     vector<Card> cards;
118   };
119
120   // Player - player in a card game.
121   class Player
122   {
123   public:
124     // Main constructor - initialize a player's hand and data.
125     // Argument(s):
126     //  aDeck - deck of cards to draw hand from.
127     // Precondition(s):
128     //  Deck must have enough cards to fill player's hand.
129     // Postcondition(s):
130     //  Player has full hand (see hand_size).
131     //  Deck is reduced by number of cards used to fill hand.
132     //  Player's score is set to zero.
133     Player (Deck& aDeck);
134
135     //Copy constructor
136     Player(const Player& orig);
137
138     // Destructor
139     ~Player();
140
141     // Copy assignment operator
142     Player& operator=(const Player& rhs);
143
144     // draw
145     // Remove and return a card from the player's hand.
146     // Precondition(s):
147     //  Must be at least one card in the hand.
148     // Returns: randomly chosen card from hand.
149     // Postcondition(s):
150     //  Number of cards in player's hand is reduced by one.
151     Card draw();
152
153     // addPoints
154     // Add points to the player's score.
155     // Argument(s):
156     //  howMany - number of points to add to score.
157     // Postcondition(s):
158     //  Player's score is increased by the specified number of points.
159     void addPoints(int howMany);
160
```

```
161    // score (accessor)
162    // Returns: the player's score.
163    int score() const;
164
165    // replaceCard
166    // Add a card back to the player's hand (to replace one previously drawn).
167    // Argument(s):
168    //  aDeck - Deck to draw replacement card from.
169    // Precondition(s):
170    //  Deck must not be empty.
171    // Postcondition(s):
172    //  Number of cards in hand is increased by one.
173    //  Number of cards in deck is decreased by one.
174    void replaceCard(Deck& aDeck);
175
176  private:
177    // No-arg constructor - private since we want to disallow its use.
178    // (No implementation supplied)
179    Player();
180
181  private:
182    // Cards in player's hand.
183    vector<Card> myCards;
184
185    // Player's score.
186    int myScore;
187
188    // Number of cards in a full hand (initialized in cards.cpp).
189    static const hand_size;
190  };
191
192  #endif // CARDS_H
```

2 cards.cpp

```
1  // cards.cpp
2  // For interface description, see cards.h
3
4  //
5  //  WAR card game
6  //
7  //  Described in Chapter 2 of
8  //  Data Structures in C++ using the STL
9  //  Published by Addison-Wesley, 1997
10 //  Written by Tim Budd, budd@cs.orst.edu
11 //  Oregon State University
12 //
13
14 // Updated by Mark Sebern, MSOE
15 // For ISO/ANSI C++, MSVC 6.0, and MSOE coding conventions,
16 // using vector (since this is covered in earlier classes).
17 //
18 // Version 1.01
```

```
19 // 29 November 1999
20 // Modified: taylor@msoe.edu, 11-30-99: Added additional comments
21
22 #include <iostream>
23 #include <algorithm>
24 #include <vector>
25 #include <cassert>
26 using namespace std;
27
28 #include "cards.h"
29 #include "random.h"
30
31 // Mark Sebern, 11-29-99 (based on code by Timothy Budd)
32 // Modified: taylor@msoe.edu, 11-30-99: Added additional comments
33 Card::Card()
34 : rank(1), suit(spade)
35 {
36   // No further initialization needed.
37 }
38
39 // Mark Sebern, 11-29-99 (based on code by Timothy Budd)
40 // Modified: taylor@msoe.edu, 11-30-99: Added additional comments
41 Card::Card  (suits sv, unsigned int rv)
42 : rank(rv), suit(sv)
43 {
44   // No further initialization needed.
45 }
46
47 // Mark Sebern, 11-29-99 (based on code by Timothy Budd)
48 // Modified: taylor@msoe.edu, 11-30-99: Added additional comments
49 Card::Card(const Card& orig)
50 : rank(orig.rank), suit(orig.suit)
51 {
52   // No further initialization needed.
53
54   // We could use "operator=(orig);" here instead of the initializer
55   // list, to avoid having to duplicate code, but in this simple case
56   // it may not be worth it.
57 }
58
59 // Mark Sebern, 11-29-99 (based on code by Timothy Budd)
60 // Modified: taylor@msoe.edu, 11-30-99: Added additional comments
61 Card::~Card()
62 {
63   // No explicit deallocation required.
64 }
65
66 // Mark Sebern, 11-29-99 (based on code by Timothy Budd)
67 // Modified: taylor@msoe.edu, 11-30-99: Added additional comments
68 Card& Card::operator =(const Card& rhs)
69 {
70   // If not self-assignment, assign members.
71   if (this != &rhs)
72   {
```

```
73      rank = rhs.rank;
74      suit = rhs.suit;
75    }
76    return *this;
77  }
78
79  // Mark Sebern, 11-29-99 (based on code by Timothy Budd)
80  // Modified: taylor@msoe.edu, 11-30-99: Added additional comments
81  unsigned int Card::GetRank() const
82  {
83    return rank;
84  }
85
86  // Mark Sebern, 11-29-99 (based on code by Timothy Budd)
87  // Modified: taylor@msoe.edu, 11-30-99: Added additional comments
88  void Card::SetRank(unsigned int rv)
89  {
90    assert ((rv > 0) && (rv <= rank_max));
91
92    rank = rv;
93  }
94
95  // Mark Sebern, 11-29-99 (based on code by Timothy Budd)
96  // Modified: taylor@msoe.edu, 11-30-99: Added additional comments
97  suits Card::GetSuit() const
98  {
99    return suit;
100 }
101
102 // Mark Sebern, 11-29-99 (based on code by Timothy Budd)
103 // Modified: taylor@msoe.edu, 11-30-99: Added additional comments
104 void Card::SetSuit(suits sv)
105 {
106   suit = sv;
107 }
108
109 // Mark Sebern, 11-29-99 (based on code by Timothy Budd)
110 // Modified: taylor@msoe.edu, 11-30-99: Added additional comments
111 void Card::Insert (ostream& out) const
112 {
113   // first output rank
114   switch (rank) {
115     case 1:  out << "Ace";   break;
116     case 11: out << "Jack";  break;
117     case 12: out << "Queen"; break;
118     case 13: out << "King";  break;
119     default:  // output number
120       out << rank; break;
121     }
122
123   // then output suit
124   switch (suit)
125   {
126     case diamond: out << " of Diamonds"; break;
```

```
127       case spade:   out << " of Spades";   break;
128       case heart:   out << " of Hearts";   break;
129       case club:    out << " of Clubs";    break;
130     }
131   }
132
133   // Mark Sebern, 11-29-99 (based on code by Timothy Budd)
134   // Modified: taylor@msoe.edu, 11-30-99: Added additional comments
135   ostream& operator<<(ostream& out, const Card& aCard)
136   {
137     aCard.Insert(out);
138     return out;
139   }
140
141   // Mark Sebern, 11-29-99 (based on code by Timothy Budd)
142   // Modified: taylor@msoe.edu, 11-30-99: Added additional comments
143   Deck::Deck()
144   {
145     // initialize a deck by creating all 52 cards
146     for (int i = 1; i <= rank_max; i++)
147     {
148       Card c1(diamond, i);
149       Card c2(spade, i);
150       Card c3(heart, i);
151       Card c4(club, i);
152       cards.push_back(c1);
153       cards.push_back(c2);
154       cards.push_back(c3);
155       cards.push_back(c4);
156     }
157   }
158
159   // Mark Sebern, 11-29-99 (based on code by Timothy Budd)
160   // Modified: taylor@msoe.edu, 11-30-99: Added additional comments
161   Deck::Deck(const Deck& orig)
162   {
163     // This time, for fun, we'll use the assignment operator.
164     operator=(orig);
165   }
166
167   // Mark Sebern, 11-29-99 (based on code by Timothy Budd)
168   // Modified: taylor@msoe.edu, 11-30-99: Added additional comments
169   Deck::~Deck()
170   {
171     // No explicit deallocation required.
172   }
173
174   // Mark Sebern, 11-29-99 (based on code by Timothy Budd)
175   // Modified: taylor@msoe.edu, 11-30-99: Added additional comments
176   Deck& Deck::operator=(const Deck& rhs)
177   {
178     // If not self-assignment, assign members.
179     if (this != &rhs)
180     {
```

```
181       cards = rhs.cards;
182     }
183     return *this;
184   }
185
186   // Mark Sebern, 11-29-99 (based on code by Timothy Budd)
187   // Modified: taylor@msoe.edu, 11-30-99: Added additional comments
188   void Deck::shuffle()
189   {
190     // Create the function object to generate random numbers.
191     randomInteger randfunc;
192
193     // Invoke STL algorithm to shuffle deck.
194     random_shuffle (cards.begin(), cards.end(), randfunc);
195   }
196
197   // Mark Sebern, 11-29-99 (based on code by Timothy Budd)
198   // Modified: taylor@msoe.edu, 11-30-99: Added additional comments
199   bool Deck::isEmpty() const
200   {
201     return (cards.size() == 0);
202   }
203
204   // Mark Sebern, 11-29-99 (based on code by Timothy Budd)
205   // Modified: taylor@msoe.edu, 11-30-99: Added additional comments
206   Card Deck::draw()
207   {
208     // return one card from the end of the deck
209     assert (!isEmpty());
210
211     // Since "assert" will be disabled in a non-debug build, we default
212     // to the ace of spades in case the deck is empty.
213     Card retval(spade,1);
214
215     // If there are cards in the deck, draw one.
216     if (!isEmpty())
217     {
218       // Copy the last card in the deck.
219       retval = cards.back();
220
221       // Delete the last card in the deck.
222       cards.pop_back();
223     }
224     return retval;
225   }
226
227
228   // Initialize the number of cards in a full hand.
229   const Player::hand_size = 3;
230
231   // Mark Sebern, 11-29-99 (based on code by Timothy Budd)
232   // Modified: taylor@msoe.edu, 11-30-99: Added additional comments
233   Player::Player (Deck& aDeck)
234   : myScore(0)
```

```
235 {
236   // Draw enough cards from the deck to fill the hand.
237   for (int i = 0; i < hand_size; i++)
238   {
239     assert (!aDeck.isEmpty());
240     myCards.push_back(aDeck.draw());
241   }
242 }
243
244 // Mark Sebern, 11-29-99 (based on code by Timothy Budd)
245 // Modified: taylor@msoe.edu, 11-30-99: Added additional comments
246 Player::Player(const Player& orig)
247 : myCards(orig.myCards), myScore(orig.myScore)
248 {
249   // No further initialization needed.
250
251   // We could use "operator=(orig);" here instead of the initializer
252   // list, to avoid having to duplicate code, but in this simple case
253   // it may not be worth it.
254 }
255
256 // Mark Sebern, 11-29-99 (based on code by Timothy Budd)
257 // Modified: taylor@msoe.edu, 11-30-99: Added additional comments
258 Player::~Player()
259 {
260   // No explicit deallocation required.
261 }
262
263 // Mark Sebern, 11-29-99 (based on code by Timothy Budd)
264 // Modified: taylor@msoe.edu, 11-30-99: Added additional comments
265 Player& Player::operator=(const Player& rhs)
266 {
267   // If not self-assignment, assign members.
268   if (this != &rhs)
269   {
270     myCards = rhs.myCards;
271     myScore = rhs.myScore;
272   }
273   return *this;
274 }
275
276 // Mark Sebern, 11-29-99 (based on code by Timothy Budd)
277 // Modified: taylor@msoe.edu, 11-30-99: Added additional comments
278 Card Player::draw()
279 {
280   // return a random card from our hand
281   assert (myCards.size() > 0);
282
283   // Create a function object to generate random number.
284   randomInteger randomizer;
285
286   // Generate a random number to select the card to draw.
287   int removedCard = randomizer(myCards.size());
288
```

```
289     // Copy the chosen card that will be returned.
290     Card drawn_card = myCards[removedCard];
291
292     // Now move the last card in the hand down to "fill" the
293     // postition we just drew from. We do this because it is
294     // difficult to delete an element from the middle of a vector.
295     myCards[removedCard] = myCards.back();
296
297     // Now delete the last card in the deck (the one we just moved).
298     // Note that if we drew the last card, we just moved it to its
299     // same position and then deleted it.
300     myCards.pop_back();
301
302     // Return the drawn card to the caller.
303     return drawn_card;
304 }
305
306 // Mark Sebern, 11-29-99 (based on code by Timothy Budd)
307 // Modified: taylor@msoe.edu, 11-30-99: Added additional comments
308 void Player::addPoints(int howMany)
309 {
310     // add the given number of points to the current score
311     myScore += howMany;
312 }
313
314 // Mark Sebern, 11-29-99 (based on code by Timothy Budd)
315 // Modified: taylor@msoe.edu, 11-30-99: Added additional comments
316 int Player::score() const
317 {
318     return myScore;
319 }
320
321 // Mark Sebern, 11-29-99 (based on code by Timothy Budd)
322 // Modified: taylor@msoe.edu, 11-30-99: Added additional comments
323 void Player::replaceCard(Deck& aDeck)
324 {
325     assert (!aDeck.isEmpty());
326
327     // Add a new card from the deck to replace the last card drawn.
328     myCards.push_back(aDeck.draw());
329 }
```

3 random.h

```
1  // random.h
2  #ifndef RANDOM_H
3  #define RANDOM_H
4
5  //
6  //  WAR card game
7  //
8  //   Described in Chapter 2 of
9  //   Data Structures in C++ using the STL
```

```
10 //  Published by Addison-Wesley, 1997
11 //  Written by Tim Budd, budd@cs.orst.edu
12 //  Oregon State University
13 //
14
15 // Updated by Mark Sebern, MSOE
16 // For ISO/ANSI C++, MSVC 6.0, and MSOE coding conventions,
17 // using vector (since this is covered in earlier classes).
18 //
19 // Version 1.01
20 // 29 November 1999
21
22 // randomInteger - random number generator function object.
23 // This class can be used to produce general random number generators.
24
25 // Note:
26 // Since objects of this class will be used with the "random_shuffle"
27 // STL algorithm, the function call operator must be defined to take
28 // a single argument that is convertible to the "size type" of the
29 // container that is being shuffled.
30
31 class randomInteger
32 {
33 public:
34   // operator()
35   // Generate a random number in a specified range.
36   // Argument(s):
37   //  max - specified range of random numbers
38   // Returns: an integer in the range [0..(max-1)].
39   unsigned int operator () (unsigned int max);
40
41   // Note that this class has no data members.
42   // This is not a restriction on function objects, but in this
43   // case we have no need for any data members to provide the needed
44   // functionality.
45
46   // Since there are no data members, and because of the limited
47   // functionality, we'll let the compiler write the default versions
48   // of:
49   //   no-arg constructor
50   //   copy constructor
51   //   copy assignment operator
52   //   destructor
53   //
54   // If the class were not so simple, it would be better to provide
55   // these functions explicitly.
56 };
57
58 #endif // RANDOM_H
```

4 random.cpp

```
1 // random.cpp
```

```
2
3  //
4  //  WAR card game
5  //
6  //  Described in Chapter 2 of
7  //  Data Structures in C++ using the STL
8  //  Published by Addison-Wesley, 1997
9  //  Written by Tim Budd, budd@cs.orst.edu
10 //  Oregon State University
11 //
12
13 // Updated by Mark Sebern, MSOE
14 // For ISO/ANSI C++, MSVC 6.0, and MSOE coding conventions,
15 // using vector (since this is covered in earlier classes).
16 //
17 // Version 1.00
18 // 29 November 1999
19 // Modified: taylor@msoe.edu, 11-30-99: Added additional comments
20
21 #include <cstdlib>
22 #include "random.h"
23
24 // Mark Sebern, 11-29-99 (based on code by Timothy Budd)
25 // Modified: taylor@msoe.edu, 11-30-99: Added additional comments
26 unsigned int randomInteger::operator () (unsigned int max)
27 {
28   // rand returns a random integer
29   // convert to unsigned to make positive
30   // take remainder to put in range
31   unsigned int rval = rand();
32   return rval % max;
33 }
```

5 war.cpp

```
1  //
2  //  WAR card game
3  //
4  //  Described in Chapter 2 of
5  //  Data Structures in C++ using the STL
6  //  Published by Addison-Wesley, 1997
7  //  Written by Tim Budd, budd@cs.orst.edu
8  //  Oregon State University
9  //
10
11 // Updated by Mark Sebern, MSOE
12 // For ISO/ANSI C++, MSVC 6.0, and MSOE coding conventions,
13 // using vector (since this is covered in earlier classes).
14 //
15 // Version 1.00
16 // 29 November 1999
17
18 #include <iostream>
```

```
19 #include <algorithm>
20 #include <vector>
21 #include <cassert>
22 using namespace std;
23
24 #include "cards.h"
25 #include "random.h"
26
27 void main()
28 {
29   Deck theDeck; // create and shuffle the deck
30   theDeck.shuffle();
31
32   Player player1(theDeck); // create the two
33   Player player2(theDeck); // players
34
35   bool done = false;
36
37   // play until we can't draw enough cards to continue
38   while (!done)
39   {
40     Card card1  = player1.draw();
41     cout << "Player 1 plays " << card1 << endl;
42     Card card2 = player2.draw();
43     cout << "Player 2 plays " << card2 << endl;
44
45     if (card1.GetRank() == card2.GetRank())
46     { // tie
47       player1.addPoints(1);
48       player2.addPoints(1);
49       cout << "Players tie" << endl;
50     }
51     else if (card1.GetRank() > card2.GetRank())
52     {
53       player1.addPoints(2);
54       cout << "Player 1 wins round" << endl;
55     }
56     else
57     {
58       player2.addPoints(2);
59       cout << "Player 2 wins round" << endl;
60     }
61
62     // now replace the cards drawn (if we can)
63     done = theDeck.isEmpty();
64     if (!done) player1.replaceCard(theDeck);
65
66     done = theDeck.isEmpty();
67     if (!done) player2.replaceCard(theDeck);
68   }
69
70   cout << "Player 1 score " << player1.score() << endl;
71   cout << "Player 2 score " << player2.score() << endl;
72 }
```