```
1       //
2       //  p3.h
3       //  p3
4       //
5       //  Created by James Francis II on 10/5/14.
6       //  Copyright (c) 2014 James Francis II. All rights reserved.
7       //

8       #ifndef p3
9       #define p3
10      #include <iostream>
11      #include <fstream>
12      #include <cstdlib>
13      #include <sstream>
14      #include <string>
15      #include <algorithm>
16      #include <vector>
17      #define MAX_HAND_SIZE 13
18      using namespace std;
19      class Card{
20
21      public:
22          Card();
23          Card(string token);
24          int getValue();
25          char getSuit();
26          bool operator==(const Card &t);
27
28
29      private:
30          int value;
31          char suit;
32          void setValue(char c);
33          void setSuit(char f);
34          bool isSuitValid(char f);
35          bool isValueValid(char f);
36
37      };

38      class Hand{
39
40      public:
41          Card myHand[20];
42          string Clubs[MAX_HAND_SIZE];
43          string Diamonds[MAX_HAND_SIZE];
44          string Hearts[MAX_HAND_SIZE];
45          string Spades[MAX_HAND_SIZE];
46          int clubsCounter;
47          int diamCounter;
```

```
    48          int heartCounter;
    49          int spadeCounter;
    50          Hand();
    51          Hand(string line);
    52          void displayHand();
    53          int scoreHand();
    54
    55      private:
    56          void mySort();
    57          void setCard(int n, string token);
    58          int isHandValid();
    59          void clearHand();
    60          void setToSuitArray(Card temp);
    61          void assignToClubs(int v);
    62          void assignToDiamonds(int v);
    63          void assignToHearts(int v);
    64          void assignToSpades(int v);
    65          void mySuitSort(string array[]);
    66      };



cat -b Bridge.cpp
     1      /*
     2       PROGRAM NAME: Program 3: BRIDGE
     3
     4       PROGRAMMER:   James Francis
     5
     6       CLASS:        CSC 331.001, Fall 2014
     7
     8       INSTRUCTOR:   Dr. Robert Strader
     9
    10       DATE STARTED: September 5, 2014
    11
    12       DUE DATE:     September 7, 2014
    13
    14       PROGRAM PURPOSE:
    15       This file contains the class definitions for Hand and Card objects.
    16
    17       VARIABLE DICTIONARY:
    18       myHand[]: an array used to house references to Card objects
    19       clubsCounter: integer used to count the number of diamonds in a hand
    20       diamCounter: integer used to count the number of diamonds in a hand
    21       heartCounter: integer used to count the number of diamonds in a hand
    22       spadeCounter: integer used to count the number of diamonds in a hand
    23       item: String used to contain the current Value and Suit of a Card
    24
    25
    26       ADTs: Arrays
```

```
27
28       FILES USED: NONE
29
30
31       SAMPLE INPUTS: NONE
32       SAMPLE OUTPUTS: NONE
33
34       ----------------------------------------------------------------------*/

35      #include "p3.h"
36      Hand::Hand(){
37          //-------------------------------------------------------------------------------
38          //Default Hand Constructor
39          //-------------------------------------------------------------------------------
40
41      }

42      Hand::Hand(string line){
43          //-------------------------------------------------------------------------------
44          // Hand constructor that catches errors thrown by invalid hands
45          //-------------------------------------------------------------------------------
46
47          clubsCounter = 0;
48          diamCounter = 0;
49          heartCounter = 0;
50          spadeCounter = 0;
51          try{
52              stringstream linestream(line);
53              string item;
54              int i = 0;
55              while (getline(linestream, item, ' '))
56              {
57                  // "Tokenizes" the input line from the calling code
58                  // Each token is a potential Card
59                  // supposed to contain a Value followed by a Suit
60
61                  if (i< MAX_HAND_SIZE+1) {
62                      setCard(i, item);
63                      setToSuitArray(myHand[i]);
64                      i++;
65                  }
66
67              }
68
69              displayHand();
70
71
72          }
73          catch (int e){
```

```
74              if (e == 1) {
75                  cout<<"Error in setting the Suit for a card, hand is invalid."<<endl<<endl;
76                  clearHand();
77
78              }
79              if (e == 2) {
80                  cout<<"Error in setting the Value for a card, hand is invalid."<<endl<<endl;
81                  clearHand();
82
83              }
84              if (e == 3){
85                  cout << "Too few cards in this hand, hand is invalid."<<endl<<endl;
86                  clearHand();
87
88              }
89              if (e == 4){
90                  cout << "Too many cards in this hand, hand is invalid."<<endl<<endl;
91                  clearHand();
92
93              }
94              if (e == 5){
95                  cout << "Duplicate Cards Detected in Hand, hand is inavlid."<<endl<<endl;
96                  clearHand();
97
98              }
99
100
101
102         }
103
104     }

105     void Hand::setCard(int n, string token){
106         //-----------------------------------------------------------------------------
107         // Populates the value at myHand[n] with a Card(Value,Suit)
108         //-----------------------------------------------------------------------------
109         myHand[n] = Card(token);
110
111     }

112     void Hand::displayHand(){
113         //-----------------------------------------------------------------------------
114         // Displays the current hand, this method checks hand validity and throws
115         // errors if the hand is too short, too long, or duplicates exist
116         // Displays each suit's
117         //-----------------------------------------------------------------------------
118
119         if (isHandValid()==3) {
120             clearHand();
```

```
121             throw 3;
122 
123         }
124         if (isHandValid()==4) {
125             clearHand();
126             throw 4;
127 
128         }
129         if (isHandValid()==5) {
130             clearHand();
131             throw 5;
132         }
133         if (isHandValid() == 6){
134             cout<<endl;
135 
136             printf("%s", "CLUBS:   ");
137             mySuitSort(Clubs);
138             for (int i = 0; i < MAX_HAND_SIZE; i++) {
139                 if (Clubs[i].length() != 0) {
140                     printf(" %s ",Clubs[i].c_str());
141                 }
142             }
143             cout<<endl;
144             printf("%s", "DIAMONDS:");
145             mySuitSort(Diamonds);
146 
147             for (int i = 0; i < MAX_HAND_SIZE; i++) {
148                 if (Diamonds[i].length() != 0) {
149                     printf(" %s ",Diamonds[i].c_str());
150                 }
151             }
152             cout<<endl;
153             printf("%s", "HEARTS:  ");
154 
155             for (int i = 0; i< MAX_HAND_SIZE; i++) {
156                 mySuitSort(Hearts);
157                 if (Hearts[i].length() != 0) {
158                     printf(" %s ",Hearts[i].c_str());
159                     mySuitSort(Hearts);
160 
161                 }
162             }
163             cout<<endl;
164             printf("%s", "SPADES:  ");
165 
166             for (int i = 0; i< MAX_HAND_SIZE; i++) {
167                 mySuitSort(Spades);
168 
169                 if (Spades[i].length() != 0) {
```

```
170                     printf(" %s ",Spades[i].c_str());
171                     mySuitSort(Spades);
172 
173                 }
174 
175             }
176 
177             cout<<"\n\nPoints = "<<scoreHand()<<endl;
178 
179             cout<<endl;
180             clearHand();
181         }
182     }
183     int Hand::isHandValid(){
184         //-----------------------------------------------------------------------------
185         // Checks the current hand and returns values associated with illegal hands
186         //-----------------------------------------------------------------------------
187 
188         int j = 0;
189         while (myHand[j].getSuit() != '\0') {
190 
191             //Counts the number of Cards in the current hand
192             j++;
193 
194         }
195         if (j< MAX_HAND_SIZE) {
196 
197             //Returns a 3 to the calling code if too few cards are present in the hand
198             return 3;
199         }
200         if (j > MAX_HAND_SIZE){
201 
202             //Returns a 4 to the calling code if too many cards are present in the hand
203             return 4;
204         }
205 
206         for(int i = 0; i < MAX_HAND_SIZE; i++){
207             for(int j = i+1; j < MAX_HAND_SIZE; j++){
208 
209                 //Returns a 5 to the calling code if any duplicate cards are encountered
210                 if (myHand[i] == myHand[j]){
211                     return 5;
212                     break;
213                 }
214             }
215         }
216 
217         return 6;
218     }
```

```
219     int Hand::scoreHand(){
220         //-----------------------------------------------------------------------------
221         // Scores the current hand
222         //-----------------------------------------------------------------------------
223 
224         int worth = 0;
225         for (int i = 0; i < MAX_HAND_SIZE; i++){
226             if (myHand[i].getValue() == 27) {
227                 //Adds 3 to the worth of the hand if a King is encountered
228 
229                 worth+=3;
230             }
231             if (myHand[i].getValue() == 26) {
232                 //Adds 1 to the worth of the hand if a Jack is encountered
233 
234                 worth+=1;
235             }
236             if (myHand[i].getValue() == 33) {
237                 //Adds 2 to the worth of the hand if a Queen is encountered
238 
239                 worth+=2;
240             }
241             if (myHand[i].getValue() == 17) {
242                 //Adds 4 to the worth of the hand if an Ace is encountered
243 
244                 worth+=4;
245             }
246         }
247 
248 
249         //Void Scoring Begins
250 
251         if (clubsCounter==0) {
252             worth+=3;
253         }
254         if (diamCounter==0) {
255             worth+=3;
256         }
257         if (heartCounter==0) {
258             worth+=3;
259         }
260         if (spadeCounter==0) {
261             worth+=3;
262         }//End Void Scoring
263 
264 
265         //Single Scoring Begins
266 
```

```
267          if (clubsCounter==1) {
268              worth+=2;
269          }
270          if (diamCounter==1) {
271              worth+=2;
272          }
273          if (heartCounter==1) {
274              worth+=2;
275          }
276          if (spadeCounter==1) {
277              worth+=2;
278          }//End Singleton Scoring
279
280
281          //Doubleton Scoring Begins
282
283          if (clubsCounter==2) {
284              worth+=1;
285          }
286          if (diamCounter==2) {
287              worth+=1;
288          }
289          if (heartCounter==2) {
290              worth+=1;
291          }
292          if (spadeCounter==2) {
293              worth+=1;
294          }//End Doubleton Scoring
295
296
297          //Long Scoring Begins
298
299          if (clubsCounter>5) {
300              worth+= (clubsCounter-5);
301          }
302          if (diamCounter>5) {
303              worth+= (diamCounter-5);
304          }
305          if (heartCounter>5) {
306              worth+=(heartCounter-5);
307          }
308          if (spadeCounter>5) {
309              worth+=(spadeCounter-5);
310          }//End Long Scoring
311
312
313          return worth;
314      }
```

```
315      void Hand::clearHand(){
316          //-----------------------------------------------------------------------------
317          // Clears the references to any objects within the myHand array
318          // Assigns null values to each suit array.
319          //-----------------------------------------------------------------------------
320
321          for(int i = 0; i< MAX_HAND_SIZE; i++){
322              myHand[i] = *new Card();
323              Clubs[i] = '\0';
324              Diamonds[i] = '\0';
325              Hearts[i] = '\0';
326              Spades[i] = '\0';
327
328          }
329      }

330      void Hand::mySuitSort(string arr[]){
331          //-----------------------------------------------------------------------------
332          // Bubble Sort that orders all the values within an Array in Descending order
333          //-----------------------------------------------------------------------------
334
335          string temp;
336
337          for(int i = 0; i < MAX_HAND_SIZE; i++)
338          {
339              for(int j = 0;j < MAX_HAND_SIZE - 1; j++)
340              {
341                  if(arr[j] < arr[j+1])
342                  {
343                      //we need to swap
344                      temp = arr[j];
345                      arr[j] = arr[j+1];
346                      arr[j+1] = temp;
347                  }
348                  if(arr[j+1]=="A"){
349                      temp = arr[j];
350                      arr[j] = arr[j+1];
351                      arr[j+1] = temp;
352                  }
353                  if(arr[j+1]=="K" && arr[j]=="Q"){
354                      temp = arr[j];
355                      arr[j] = arr[j+1];
356                      arr[j+1] = temp;
357                  }
358
359              }
360          }
361      }
```

```
362     Card::Card(){
363         //-------------------------------------------------------------------------
364         // Default Card Constructor
365         //-------------------------------------------------------------------------
366         value = -1;
367         suit = '\0';
368 
369     }

370     Card::Card(string token){
371         //-------------------------------------------------------------------------
372         // Card Constructor that accepts a string containing: value and suit
373         //-------------------------------------------------------------------------
374         value = -1;
375         suit = '\0';
376         //cout<<"Creating Card: "<<token[0]<<" "<<token[1]<<endl;
377         setValue(token[0]);
378         setSuit(token[1]);
379 
380     }

381     //-------------------------------------------------------------------------
382     // --------------BEGIN GETTERS AND SETTERS-----------------------------------
383     //-------------------------------------------------------------------------
384     int Card::getValue(){
385         return value;
386     }
387     char Card::getSuit(){
388         return suit;
389     }
390     void Card::setValue(char c){
391 
392         if (isValueValid(c)) {
393 
394             value = c -'0';
395         }
396         else throw 2;
397     }
398     void Card::setSuit(char f){
399         if (isSuitValid(f)) {
400             suit = f;
401         }else throw 1;
402 
403     }
404     //-------------------------------------------------------------------------
405     // --------------END GETTERS AND SETTERS-------------------------------------
406     //-------------------------------------------------------------------------
```

```
407     bool Card::operator==(const Card &t){
408         //-----------------------------------------------------------------------------
409         // Overloaded == operator for the Card class, returns true if current card and
410         // the card passed reference to a card are equal
411         //-----------------------------------------------------------------------------
412 
413         if (value == t.value && suit == t.suit) {
414             return true;
415         }
416 
417         return false;
418     }

419     bool Card::isSuitValid(char f){
420         //-----------------------------------------------------------------------------
421         // Accepts a character representing a Card's suit, if the suit is valid true
422         // is returned, else false is returned
423         //-----------------------------------------------------------------------------
424 
425 
426         switch (f) {
427             case 'C':
428                 return true;
429                 break;
430             case 'S':
431                 return true;
432                 break;
433             case 'H':
434                 return true;
435                 break;
436             case 'D':
437                 return true;
438                 break;
439             default:
440                 return false;
441                 break;
442         }
443     }

444     bool Card::isValueValid(char f){
445         //-----------------------------------------------------------------------------
446         // Accepts a character representing a Card's value, if the valus is valid true
447         // is returned, else false is returned
448         //-----------------------------------------------------------------------------
449 
450         switch (f) {
451             case '2':
452                 return true;
```

```
453                 break;
454             case '3':
455                 return true;
456                 break;
457             case '4':
458                 return true;
459                 break;
460             case '5':
461                 return true;
462                 break;
463             case '6':
464                 return true;
465                 break;
466             case '7':
467                 return true;
468                 break;
469             case '8':
470                 return true;
471                 break;
472             case '9':
473                 return true;
474                 break;
475             case 'T':
476                 return true;
477                 break;
478             case 'J':
479                 return true;
480                 break;
481             case 'Q':
482                 return true;
483                 break;
484             case 'K':
485                 return true;
486                 break;
487             case 'A':
488                 return true;
489                 break;
490 
491             default:
492                 return false;
493                 break;
494         }
495     }

496     void Hand::setToSuitArray(Card temp){
497         //------------------------------------------------------------------------
498         // Accepts a valid Card object representing a newly created Card object
499         // Assigns the card value to the first free index in the appropriate Suit array
500         //------------------------------------------------------------------------
```

```
501
502         switch (temp.getSuit()) {
503             case 'C':
504                 assignToClubs(temp.getValue());
505                 break;
506             case 'D':
507                 assignToDiamonds(temp.getValue());
508                 break;
509             case 'H':
510                 assignToHearts(temp.getValue());
511                 break;
512             case 'S':
513                 assignToSpades(temp.getValue());
514                 break;
515             default:
516                 break;
517         }
518     }

519     void Hand::assignToClubs(int v){
520         //-----------------------------------------------------------------------------
521         // Assigns a String representation of v to the first free index to the Clubs array
522         //-----------------------------------------------------------------------------
523
524         clubsCounter++;
525
526         if(v<=9){
527             int i = 0;
528             while (Clubs[i].length() != 0) {
529                 i++;
530             }
531             string value;
532             ostringstream convert;
533             convert << v;
534             value = convert.str();
535             Clubs[i] = value;
536         }
537         else if(v == 36){
538             int i =0;
539             while (Clubs[i]!="\0") {
540                 i++;
541             }
542             Clubs[i] = "10";
543         }
544         else if(v == 27){
545             int i =0;
546             while (Clubs[i]!="\0") {
547                 i++;
548             }
```

```
549             Clubs[i] = "K";
550         }
551         else if(v == 26){
552             int i =0;
553             while (Clubs[i]!="\0") {
554                 i++;
555             }
556             Clubs[i] = "J";
557         }
558         else if(v == 33){
559             int i =0;
560             while (Clubs[i]!="\0") {
561                 i++;
562             }
563             Clubs[i] = "Q";
564         }
565         else if(v == 17){
566             int i =0;
567             while (Clubs[i]!="\0") {
568                 i++;
569             }
570             Clubs[i] = "A";
571         }
572     }
573     void Hand::assignToDiamonds(int v){
574         //-----------------------------------------------------------------------------
575         // Assigns a String representation of v to the first free index to the Diamonds array
576         //-----------------------------------------------------------------------------
577 
578         diamCounter++;
579 
580         if(v<=9){
581             int i = 0;
582             while (Diamonds[i].length() != 0) {
583                 i++;
584             }
585             string value;
586             ostringstream convert;
587             convert << v;
588             value = convert.str();
589             Diamonds[i] = value;
590         }
591         else if(v == 36){
592             int i =0;
593             while (Diamonds[i]!="\0") {
594                 i++;
595             }
596             Diamonds[i] = "10";
597         }
```

```
598         else if(v == 27){
599             int i =0;
600             while (Diamonds[i]!="\0") {
601                 i++;
602             }
603             Diamonds[i] = "K";
604         }
605         else if(v == 26){
606             int i =0;
607             while (Diamonds[i]!="\0") {
608                 i++;
609             }
610             Diamonds[i] = "J";
611         }
612         else if(v == 33){
613             int i =0;
614             while (Diamonds[i]!="\0") {
615                 i++;
616             }
617             Diamonds[i] = "Q";
618         }
619         else if(v == 17){
620             int i =0;
621             while (Diamonds[i]!="\0") {
622                 i++;
623             }
624             Diamonds[i] = "A";
625         }
626     }
627     void Hand::assignToHearts(int v){
628         //-----------------------------------------------------------------------------
629         // Assigns a String representation of v to the first free index to the Hearts array
630         //-----------------------------------------------------------------------------
631 
632         heartCounter++;
633         if(v<=9){
634             int i = 0;
635             while (Hearts[i].length() != 0) {
636                 i++;
637             }
638             string value;
639             ostringstream convert;
640             convert << v;
641             value = convert.str();
642             Hearts[i] = value;
643         }
644         else if(v == 36){
645             int i =0;
646             while (Hearts[i]!="\0") {
```

```
647                 i++;
648             }
649             Hearts[i] = "10";
650         }
651         else if(v == 27){
652             int i =0;
653             while (Hearts[i]!="\0") {
654                 i++;
655             }
656             Hearts[i] = "K";
657         }
658         else if(v == 26){
659             int i =0;
660             while (Hearts[i]!="\0") {
661                 i++;
662             }
663             Hearts[i] = "J";
664         }
665         else if(v == 33){
666             int i =0;
667             while (Hearts[i]!="\0") {
668                 i++;
669             }
670             Hearts[i] = "Q";
671         }
672         else if(v == 17){
673             int i =0;
674             while (Hearts[i]!="\0") {
675                 i++;
676             }
677             Hearts[i] = "A";
678         }
679     }
680     void Hand::assignToSpades(int v){
681         //-----------------------------------------------------------------------------------
682         // Assigns a String representation of v to the first free index to the Spades array
683         //-----------------------------------------------------------------------------------
684 
685         spadeCounter++;
686 
687         if(v<=9){
688 
689             int i = 0;
690             while (Spades[i].length() != 0) {
691                 i++;
692             }
693             string value;
694             ostringstream convert;
695             convert << v;
```

```
   696              value = convert.str();
   697
   698              Spades[i] = value;
   699          }
   700          else if(v == 36){
   701              int i =0;
   702              while (Spades[i]!="\0") {
   703                  i++;
   704              }
   705              Spades[i] = "10";
   706          }
   707          else if(v == 27){
   708              int i =0;
   709              while (Spades[i]!="\0") {
   710                  i++;
   711              }
   712              Spades[i] = "K";
   713          }
   714          else if(v == 26){
   715              int i =0;
   716              while (Spades[i]!="\0") {
   717                  i++;
   718              }
   719              Spades[i] = "J";
   720          }
   721          else if(v == 33){
   722              int i =0;
   723              while (Spades[i]!="\0") {
   724                  i++;
   725              }
   726              Spades[i] = "Q";
   727          }
   728          else if(v == 17){
   729              int i =0;
   730              while (Spades[i]!="\0") {
   731                  i++;
   732              }
   733              Spades[i] = "A";
   734          }
   735      }



cat -b BridgeTest.cpp
     1      /*
     2       PROGRAM NAME: Program 3: BRIDGE
     3
     4       PROGRAMMER:   James Francis
     5
```

```
CLASS:        CSC 331.001, Fall 2014

INSTRUCTOR:   Dr. Robert Strader

DATE STARTED: September 5, 2014

DUE DATE:     September 7, 2014

PROGRAM PURPOSE:
This class implements the Hand class and tests the data with a file supplied by the Instructor

VARIABLE DICTIONARY:
file: ifstream from the input file
line: String used to store the current line of data from the ifstream
currentHand: used to represent the current potential bridge hand to be score



ADTs: none

FILES USED: prog3.dat


SAMPLE INPUTS: (first 6 lines of prog3.dat)

2C QD TC AD 6C 3D TD 3H 5H 7H AS JH KH
3C 4C 2D AC QC 7S 7C TD 9C 4D KS 8D 6C
2C 3C KC JC 4C 8C 7C QC AC 5C 9C 6C TC
5H 3S 4D KC 9S 3D 4S 8H JC TC 8S 2S 4C
2S 5D 6S 8S 9D 3C 2H TH
2H 6D %S 8S 7S 4D 3H 4S KS QH JH 5C 9S

SAMPLE OUTPUTS:

CLUBS:    6  2  10
DIAMONDS: A  Q  3  10
HEARTS:   K  J  7  5  3
SPADES:   A

Points = 16


CLUBS:    A  Q  9  7  6  4  3
DIAMONDS: 8  4  2  10
HEARTS:
SPADES:   K  7

Points = 15

```

```
55
56       CLUBS:    A  K  Q  J  9  8  7  6  5  4  3  2  10
57       DIAMONDS:
58       HEARTS:
59       SPADES:
60
61       Points = 27
62
63
64       CLUBS:    K  J  4  10
65       DIAMONDS: 4  3
66       HEARTS:   8  5
67       SPADES:   9  8  4  3  2
68
69       Points = 6
70
71       Too few cards in this hand, hand is invalid.
72
73       Error in setting the Value for a card, hand is invalid.
74
75       No more hands, File is closed
76
77       ------------------------------------------------------------------*/



78      #include "p3.h"

79      int main(int argc, const char * argv[]) {

80          ifstream file("../instr/prog3.dat");
81          string line;
82          while (!file.eof())
83          {
84              getline(file, line);
85              Hand currentHand = Hand(line);
86          }
87          file.close();
88          cout<<"No more hands, File is closed"<<endl;

89          return 0;
90      }
```