Compte Rendu TP1 CPOO

Maxime Havez, Gareth Thiveux

INSA de Rennes
4INFO, groupe 2.2

24 septembre 2010

Listing 1 – Fichier en-tête de la classe Carte

```
1  /**
2
3  *
4
5  * \file Carte.h
6
7  * \brief Header file which describes the "Carte" class
8
9  * \author Maxime HAVEZ
10
11 * \author Gareth THIVEUX
12
13 * \date 23/09/10
14
15 *
16
17 */
18
19
20
21
22
23 #ifndef CARTE_H
24
25 #define CARTE_H
26
27
28
29 /**
30
```

```
31 * \enum Couleur
32
33 * \brief Gives the possibilities for Card colors
34
35 */
36
37 enum Couleur {pique,trefle,coeur,carreau};
38
39 /**
40
41 * \enum Valeur
42
43 * \brief Gives the possibilities for Card values
44
45 */
46
47 enum Valeur {as,deux,trois,quartre,cinq,six,sept,huit,neuf,dix,valet,dame,roi};
48
49
50
51
52
53 class Carte {
54
55
56
57 private:
58
59
60
61   //Class Attributes
62
63   Couleur _couleur;
64
65   Valeur _valeur;
66
67   char _proprio;
68
69   Carte* _suc;
70
71
72
73   //Static Class Attributes which are used for the list
74
75   static Carte* teteN;
76
77   static Carte* teteS;
78
79   static Carte* queueN;
```

```
80
81    static Carte* queueS;
82
83
84
85
86
87  public:
88
89
90
91    /**
92
93    * \fn Carte(Couleur c, Valeur v, char p);
94
95    * \brief Constructor with 3 parameters
96
97    *
98
99    * \param[in] c Card color
100
101   * \param[in] v Card value
102
103   * \param[in] p Card owner, represented by a char ('N' ou 'S')
104
105   */
106
107   Carte(Couleur c, Valeur v, char p);
108
109
110
111   /**
112
113   * \fn static Carte* getNTete();
114
115   * \brief allows an access to the top element of N-packet
116
117   *
118
119   * \return the pointer on the first card of N-packet
120
121   */
122
123   static Carte* getNTete();
124
125
126
127   /**
128
```

```
129   * \fn static Carte* getSTete();
130
131   * \brief allows an access to the top element of S-packet
132
133   *
134
135   * \return the pointer on the first card of S-packet
136
137   */
138
139   static Carte* getSTete();
140
141
142
143   /**
144
145   * \fn static Carte* getNQueue();
146
147   * \brief allows an access to the last element of N-packet
148
149   *
150
151   * \return the pointer on the last card of N-packet
152
153   */
154
155   static Carte* getNQueue();
156
157
158
159   /**
160
161   * \fn static Carte* getSQueue();
162
163   * \brief allows an access to the last element of S-packet
164
165   *
166
167   * \return a pointer on the last card of S-packet
168
169   */
170
171   static Carte* getSQueue();
172
173
174
175   /**
176
177   * \fn inline Couleur Carte::couleur() const {return _couleur;}
```

```
178
179   * \brief gives an access to the current card color
180
181   *
182
183   * \return color of the current card
184
185   */
186
187   inline Couleur couleur() const {return _couleur;}
188
189
190
191   /**
192
193   * \fn inline Valeur Carte::valeur() const {return _valeur;}
194
195   * \brief gives an access to the current card value
196
197   *
198
199   * \return value of the current card
200
201   */
202
203   inline Valeur valeur() const {return _valeur;}
204
205
206
207   /**
208
209   * \fn inline char Carte::proprio() const {return _proprio;}
210
211   * \brief gives an access to the current card owner
212
213   *
214
215   * \return a char which represents the current card owner
216
217   */
218
219   inline char proprio() const {return _proprio;}
220
221
222
223   /**
224
225   * \fn inline Carte* Carte::suc() const {return _suc;}
226
```

```
227   * \brief gives an access to the current card successor
228   *
229   * \return a pointer on the successor
230   */
231   inline Carte* suc() const {return _suc;}
232
233
234   /**
235   * \fn bool supAbs(Carte c);
236   * \brief return true if the current element is stronger than called element
237   *
238   * \return a boolean, true if the current element is stronger false esle
239   */
240   bool supAbs(Carte c);
241
242
243   /**
244   * \fn bool egale(Carte c);
245   * \brief compares two cards together and tests their equality
246   *
247   * \return a boolean, true if the elements are equals, else false
248   */
249   bool egale(Carte c);
250
251
252   //methodes d'affichages
253
254
255   /**
```

```
276
277   * \fn void afficher();
278
279   * \brief prints a description of the current card
280
281   */
282
283   void afficher();
284
285
286
287   /**
288
289   * \fn void afficherN();
290
291   * \brief prints the list of cards of owner 'N'
292
293   static void afficherN();
294
295
296
297   /**
298
299   * \fn void afficherN();
300
301   * \brief prints the list of cards of owner 'S'
302
303   */
304
305   static void afficherS();
306
307
308
309   /**
310
311   * \fn void changerProp();
312
313   * \brief puts the first card of the owner at the end of the other player's
          packet
314
315   */
316
317   void changerProp();
318
319
320
321   /**
322
323   * \fn void passerDerriere();
```

```
324
325   * \brief puts the card at the end of the packet
326
327   */
328
329   void passerDerriere();
330
331 };
332
333
334
335 #endif
```

Listing 2 – Classe Carte

```
1  /**
2
3
4
5  *
6
7
8
9  * \file Carte.cpp
10
11
12
13 * \brief file of Carte class (function description)
14
15
16
17 * \author Havez Maxime
18
19
20
21 * \author Thiveux Gareth
22
23
24
25 * \date 23/09/10
26
27
28
29 *
30
31
32
33 */
34
35
```

```
36
37
38
39
40
41 #include "Carte.h"
42
43
44
45 #include <iostream>
46
47
48
49 #include <cstdlib>
50
51
52
53
54
55
56
57 // Cards constuctor
58
59
60
61 Carte::Carte(Couleur c, Valeur v, char p):_couleur(c),_valeur(v),_proprio(p){};
62
63
64
65
66
67
68
69 Carte* Carte::teteN = 0;
70
71
72
73 Carte* Carte::teteS = 0;
74
75
76
77 Carte* Carte::queueN = 0;
78
79
80
81 Carte* Carte::queueS = 0;
82
83
84
```

```
85
86
87
88
89 Carte* Carte::getNTete() { return teteN;}
90
91
92
93 Carte* Carte::getNQueue() { return queueN;}
94
95
96
97 Carte* Carte::getSTete() { return teteS;}
98
99
100
101 Carte* Carte::getSQueue() { return queueS;}
102
103
104
105
106
107
108
109 bool Carte::egale(Carte c) { return _valeur==c.valeur();}
110
111
112
113
114
115
116
117 bool Carte::supAbs(Carte c) { return _valeur>c.valeur();}
118
119
120
121
122
123
124
125 void Carte::afficher() { std::cout << "Carte courante : \n
      "<<_valeur<<"de"<<_couleur<<std::endl; }
126
127
128
129
130
131
132
```

```
133 void Carte::afficherN () {
134
135
136
137   Carte* ec = teteN;
138
139
140
141   while (ec->_suc != queueN ) {
142
143
144
145     ec->afficher();
146
147
148
149     ec=ec->_suc;
150
151
152
153   }
154
155
156
157 }
158
159
160
161
162
163
164
165 void Carte::afficherS () {
166
167
168
169   Carte* ec = teteS;
170
171
172
173   while (ec->_suc != queueS ) {
174
175
176
177     ec->afficher();
178
179
180
181     ec=ec->_suc;
```

```
182
183
184
185     }
186
187
188
189 }
190
191
192
193
194
195
196
197 void Carte::changerProp() {
198
199
200
201     if(_proprio=='N') {
202
203
204
205         _proprio='S';
206
207
208
209         queueS = this;
210
211
212
213         _suc=0;
214
215
216
217     }else{
218
219
220
221         _proprio='N';
222
223
224
225         queueN = this;
226
227
228
229         _suc=0;
230
```

```
231
232
233   }
234
235
236
237 }
238
239
240
241
242
243
244
245 void Carte::passerDerriere() {
246
247
248
249   _proprio=='N' ? queueN=this : queueS=this ;
250
251
252
253   _suc=0;
254
255
256
257 }
```

Listing 3 – Main

```
1  /**
2
3   * \brief Main du TP1.
4
5   *
6
7   * Usage :  .\TP1 < fichier_a_lire
8
9   */
10
11
12
13 #include "Carte.h"
14
15 #include <iostream>
16
17 #include <cstdlib>
18
19
20
```

```
21 int main()
22
23 {
24
25     // Initialisation des paquets/cartes selon la suite des N et S provenant
26
27     // de l'entree standard std::cin (on suppose la redirection de stdin).
28
29     char p[52]; // pour inverser la creation des listes, on lit 52 char.
30
31     for(int i = 0; i < 52; i++)
32
33     {
34
35         // lecture d'une carte
36
37         std::cin >> p[i];
38
39     }
40
41
42
43     for(int icoul = 3; icoul >= 0; icoul--)
44
45     {
46
47         for(int ihaut = 12; ihaut >= 0; ihaut--)
48
49         {
50
51             const char proprietaire = p[icoul * 13 + ihaut];
52
53             if(proprietaire != 'N' && proprietaire != 'S')
54
55             {
56
57                 std::cerr << "Erreur dans l'initialisation : "
58
59                           << "'N' ou 'S' etait attendu"
60
61                           << std::endl;
62
63                 exit(1);
64
65             }
66
67
68
69             Couleur couleur = static_cast<Couleur>(icoul + 1);
```

```
70
71              Valeur valeur = static_cast<Valeur>(ihaut + 1);
72
73
74
75              // Attention : le constructeur doit enchainer la Carte avec la
76
77              // bonne liste.
78
79              //
80
81              // Note : les pointeurs dans les champs statiques de la classe
82
83              // Carte permettront de liberer la memoire a la fin du programme.
84
85              new Carte(couleur, valeur, proprietaire);
86
87          }
88
89      }
90
91
92
93      // Affichages des paquets
94
95      std::cout << "Les paquets de depart sont :" << std::endl;
96
97      std::cout << "Paquet de N :" << std::endl;
98
99      Carte::afficherN();
100
101     std::cout << std::endl;
102
103     std::cout << "Paquet de S :" << std::endl;
104
105     Carte::afficherS();
106
107     std::cout << std::endl;
108
109
110
111     // Boucle principale : le jeu est termine, si une des listes est vide.
112
113     std::cout << "Boucle principale" << std::endl;
114
115     while(Carte::getNTete() != 0 && Carte::getSTete() != 0)
116
117     {
118
```

```
119         Carte* Nt = Carte::getNTete();
120
121         Carte* St = Carte::getSTete();
122
123     std::cout << "Carte de N :";
124
125         Nt->afficher();
126
127     std::cout << "Carte de S :";
128
129         St->afficher();
130
131         std::cout << std::endl;
132
133
134
135         // detection d'une bataille (i.e. les deux cartes ont la meme valeur)
136
137         while(0 != Nt && 0 != St && Nt->egale(*St))
138
139         {
140
141             std::cout << "** bataille ** " << std::endl;
142
143             Nt = Nt->suc();
144
145             if(0 != Nt)
146
147             {
148
149                 // avancer de deux cartes pour N !
150
151                 Nt = Nt->suc();
152
153         std::cout << "Carte de N :";
154
155         Nt->afficher();
156
157             }
158
159             St = St->suc();
160
161             if(0 != St)
162
163             {
164
165                 // avancer de deux cartes pour S !
166
167                 St = St->suc();
```

```
168
169         std::cout << "Carte de S :";
170
171         St->afficher();
172
173             }
174
175         }
176
177         if(0 == Nt)
178
179         {
180
181             // N n'a pas assez de carte : S gagne
182
183             std::cout <<" N : plus de Carte" << std::endl;
184
185             while(0 != Carte::getNTete())
186
187             {
188
189                 Carte* Nec = Carte::getNTete();
190
191                 Nec->changerProp();
192
193             }
194
195         }
196
197         else if(0 == St)
198
199         {
200
201             // S n'a pas assez de carte
202
203             std::cout <<" S : plus de Carte" << std::endl;
204
205             while(0 != Carte::getSTete())
206
207             {
208
209                 Carte* Sec = Carte::getSTete();
210
211                 Sec->changerProp();
212
213             }
214
215         }
216
```

```
217         else if(Nt->supAbs(*St))
218 
219         {
220 
221             // comparaison des dernieres cartes : S a perdu cette bataille
222 
223             std::cout << std::endl;
224 
225             std::cout << " N gagne les cartes :" << std::endl;
226 
227             Carte* Ss = St->suc(); // premiere Carte non gagnee
228 
229             while (Carte::getSTete() != Ss)
230 
231             {
232 
233                 Carte* Sec = Carte::getSTete();
234 
235                 // On met les cartes en question a la fin du paquet N
236 
237                 Sec->changerProp();
238 
239                 Carte* Nec = Carte::getNTete();
240 
241                 Nec->passerDerriere();
242 
243                 // puis on affiche la situation
244 
245                 Sec->afficher();
246 
247                 Nec->afficher();
248 
249                 std::cout << std::endl;
250 
251             }
252 
253                 std::cout << "----" << std::endl;
254 
255         }
256 
257         else
258 
259         {
260 
261             // N a perdu cette bataille
262 
263             std::cout << std::endl;
264 
265             std::cout << " S gagne les cartes :"  << std::endl;
```

```
266
267             Carte* Ns = Nt->suc(); // premiere carte non gagnee
268
269             while(Carte::getNTete() != Ns)
270
271             {
272
273                 Carte* Nec = Carte::getNTete();
274
275                 // On met les cartes en question a la fin du paquet S
276
277                 Nec->changerProp();
278
279                 Carte* Sec = Carte::getSTete();
280
281                 Sec->passerDerriere();
282
283                 // puis on affiche la situation
284
285                 Nec->afficher();
286
287                 Sec->afficher();
288
289                 std::cout << std::endl;
290
291             }
292
293             std::cout << "----" << std::endl;
294
295         }
296
297     }
298
299     std::cout << " >>>>>>>>>>>>> le gagnant de ce jeu est : ";
300
301     if(0 == Carte::getNTete())
302
303     {
304
305         // N a perdu les jeux
306
307         std::cout << 'S' << std::endl;
308
309         Carte* Sec = Carte::getSTete();
310
311         while(0 != Sec)
312
313         {
314
```

```
315             // destruction du paquet de S
316
317             Carte* Ss = Sec->suc();
318
319             delete Sec;
320
321             Sec = Ss;
322
323         }
324
325     }
326
327     else
328
329     {
330
331         // S a perdu les jeux
332
333         std::cout << 'N' << std::endl;
334
335         Carte* Nec = Carte::getNTete();
336
337         while(0 != Nec)
338
339         {
340
341             // destruction du paquet de N
342
343             Carte* Ns = Nec->suc();
344
345             delete Nec;
346
347             Nec = Ns;
348
349         }
350
351     }
352
353     std::cout << "fin de partie" << std::endl;
354
355     return 0;
356
357 }
```