

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,quatre,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 *
230
231 * \return a pointer on the successor
232
233 */
234
235 inline Carte* suc() const {return _suc;}
236
237
238
239 /**
240
241 * \fn bool supAbs(Carte c);
242
243 * \brief return true if the current element is stronger than called element
244
245 *
246
247 * \return a boolean, true if the current element is stronger false else
248
249 */
250
251 bool supAbs(Carte c);
252
253
254
255 /**
256
257 * \fn bool egale(Carte c);
258
259 * \brief compares two cards together and tests their equality
260
261 *
262
263 * \return a boolean, true if the elements are equals, else false
264
265 */
266
267 bool egale(Carte c);
268
269
270
271 //methodes d'affichages
272
273
274
275 /**

```

```

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 */
294
295 static void afficherN();
296
297
298
299 /**
300
301 * \fn void afficherN();
302
303 * \brief prints the list of cards of owner 'S'
304
305 */
306
307 static void afficherS();
308
309
310
311 /**
312
313 * \fn void changerProp();
314
315 * \brief puts the first card of the owner at the end of the other player's
    packet
316
317 */
318
319 void changerProp();
320
321
322
323 /**

```

```

324
325     * \fn void passerDerriere();
326
327     * \brief puts the card at the end of the packet
328
329     */
330
331     void passerDerriere();
332
333 };
334
335
336
337 #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 constructor
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     // Initialisation des paquets/cartes selon la suite des N et S provenant
25     // de l'entree standard std::cin (on suppose la redirection de stdin).
26
27     char p[52]; // pour inverser la creation des listes, on lit 52 char.
28
29     for(int i = 0; i < 52; i++)
30     {
31         // lecture d'une carte
32
33         std::cin >> p[i];
34     }
35
36
37     for(int icoul = 3; icoul >= 0; icoul--)
38     {
39         for(int ihaut = 12; ihaut >= 0; ihaut--)
40         {
41             const char proprietaire = p[icoul * 13 + ihaut];
42
43             if(proprietaire != 'N' && proprietaire != 'S')
44             {
45                 std::cerr << "Erreur dans l'initialisation : "
46
47                     << "'N' ou 'S' etait attendu"
48
49                     << std::endl;
50
51                 exit(1);
52             }
53
54
55
56
57
58
59
60
61
62
63
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 enchaîner la Carte avec la
76         // bonne liste.
77
78         //
79
80         // Note : les pointeurs dans les champs statiques de la classe
81         // Carte permettront de libérer la mémoire à la fin du programme.
82
83         new Carte(couleur, valeur, proprietaire);
84
85     }
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             // avancer de deux cartes pour N !
149
150             Nt = Nt->suc();
151
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                 // avancer de deux cartes pour S !

```



```

166         St = St->suc();
167
168     std::cout << "Carte de S :";
169
170     St->afficher();
171
172     }
173
174 }
175
176 if(0 == Nt)
177 {
178
179     // N n'a pas assez de carte : S gagne
180
181     std::cout << " N : plus de Carte" << std::endl;
182
183     while(0 != Carte::getNTete())
184     {
185
186         Carte* Nec = Carte::getNTete();
187
188         Nec->changerProp();
189
190     }
191
192 }
193
194 else if(0 == St)
195 {
196
197     // S n'a pas assez de carte
198
199     std::cout << " S : plus de Carte" << std::endl;
200
201     while(0 != Carte::getSTete())
202     {
203
204         Carte* Sec = Carte::getSTete();
205
206         Sec->changerProp();
207
208     }
209
210 }
211
212 }
213
214

```

```

215     }
216
217     else if(Nt->supAbs(*St))
218     {
219
220         // comparaison des dernieres cartes : S a perdu cette bataille
221
222         std::cout << std::endl;
223
224         std::cout << " N gagne les cartes :" << std::endl;
225
226         Carte* Ss = St->suc(); // premiere Carte non gantee
227
228         while (Carte::getSTete() != Ss)
229         {
230
231             Carte* Sec = Carte::getSTete();
232
233             // On met les cartes en question a la fin du paquet N
234
235             Sec->changerProp();
236
237             Carte* Nec = Carte::getNTete();
238
239             Nec->passerDerriere();
240
241             // puis on affiche la situation
242
243             Sec->afficher();
244
245             Nec->afficher();
246
247             std::cout << std::endl;
248
249         }
250
251         std::cout << "----" << std::endl;
252     }
253
254     else
255     {
256
257         // N a perdu cette bataille
258
259         std::cout << std::endl;
260
261     }
262
263

```

```

264
265         std::cout << " S gagne les cartes : " << std::endl;
266
267         Carte* Ns = Nt->suc(); // premiere carte non gantee
268
269         while(Carte::getNTete() != Ns)
270         {
271
272             Carte* Nec = Carte::getNTete();
273
274             // On met les cartes en question a la fin du paquet S
275
276             Nec->changerProp();
277
278             Carte* Sec = Carte::getSTete();
279
280             Sec->passerDerriere();
281
282             // puis on affiche la situation
283
284             Nec->afficher();
285
286             Sec->afficher();
287
288             std::cout << std::endl;
289
290         }
291
292         std::cout << "----" << std::endl;
293
294     }
295
296 }
297
298
299 std::cout << " >>>>>>>>>> le gagnant de ce jeu est : ";
300
301 if(0 == Carte::getNTete())
302 {
303
304     // N a perdu les jeux
305
306     std::cout << 'S' << std::endl;
307
308     Carte* Sec = Carte::getSTete();
309
310     while(0 != Sec)
311
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         // destruction du paquet de N
341
342         Carte* Ns = Nec->suc();
343
344         delete Nec;
345
346         Nec = Ns;
347
348     }
349
350 }
351
352 std::cout << "fin de partie" << std::endl;
353
354 return 0;
355
356
357 }

```