

```

1  /*
2
3  --SYSTEME DE RECONNAISSANCE DE CHRONIQUES--
4
5  Code developpe pour le projet tuteuré "SURVEILLANCE DES PERSONNES DANS L'HABITAT INTELLIGENT" (2018)
6  par Valentine Bellet, Benjamin Blum, Amélie Meyer, Ninni Stenkula et Amaury Vignolles
7
8  Compilation : g++.exe -Wall -fexceptions -g -std=c++14
9  IDE : Code::Blocks 16.01
10
11 Fonctionnement : le fichier .exe doit être à la racine de /src et /txt.
12
13 */
14
15 #include <iostream>
16 #include <string>
17 #include <vector>
18 #include <list>
19 #include "chronique.h"
20 #include "event.h"
21 #include "sequence.h"
22 #include "parser.h"
23 #include "parser_chr.h"
24 #include "parser_evt.h"
25 #include "interface.h"
26 using namespace std;
27
28
29 /* VARIABLES GLOBALES */
30
31 int HEURE_COURANTE = 0;
32 std::list<event> MES_EVENTS ;
33 std::list<chronique> MES_CHRONIQUES ;
34 std::vector<elmt_sequence> MA_SEQUENCE ;
35
36
37 /* FUNCTIONS */
38
39 chronique check_validation(chronique chro, event e) {
40
41     // mise a jour de chronique dans la liste des chroniques quand on a trouvé le bon event
42     // dans la liste des events (recherche par nom)
43     std::vector<contrainte> list_contraintes;
44     list_contraintes = chro.get_contraintes();
45
46     // indice represente l'indice du tableau de contrainte
47     int indice = (chro.get_contraintes_total()) - (chro.get_contraintes_restantes());
48
49     /* Affectation des conditions générales */
50
51     // TYPE 1/2 : temp sur la condition sur le nom
52     bool cond_nom = ((list_contraintes[indice].get_event().get_label()) == (e.get_label()));
53
54     // TYPE 2 : temp sur la condition sur le temps
55     bool cond_temps = ((list_contraintes[indice].get_time()) >= (e.get_last_h_occur()));
56
57     // TYPE 3 : temp sur la condition sur le temps pour le no event
58     // >> valeur de la dernière validation de la contrainte précédente
59     int last_valid = 0; // dans le cas où il n'y a pas de contrainte précédente
60     if (indice >= 1) {
61         string nom_last = list_contraintes[indice-1].get_event().get_label();
62         for(list<event>::iterator last_ev=MES_EVENTS.begin(); last_ev!=MES_EVENTS.end(); last_ev++){
63             if ( ((*last_ev).get_label()) == nom_last) {
64                 if ((*last_ev).get_occured()) { // sécurité : ne pas faire d'appel sur un pointeur null
65                     last_valid = (*last_ev).get_last_h_occur();
66                 }

```

```

67     }
68 }
69 }
70 int contr_temps = list_contraintes[indice].get_time();
71 // >> ajout de la contrainte de temps et comparaison avec l'heure courante
72 // >> vrai si la contrainte est validée
73 bool cond_temps_no_event = (HEURE_COURANTE >= (last_valid+contr_temps));
74
75 switch(list_contraintes[indice].get_type()){
76
77     case 1: // A pour t(A) quelconque
78
79         if (cond_nom) {
80             // contrainte validée, on passe à la contrainte suivante
81             int nb = chro.get_contraintes_restantes();
82             chro.set_contraintes_restantes(nb -1);
83         }
84         break;
85
86     case 2: // A arrive avant T après le dernier event
87
88         if (cond_nom && cond_temps) {
89
90             // contrainte validée, on passe à la contrainte suivante
91             int nb = chro.get_contraintes_restantes();
92             chro.set_contraintes_restantes(nb-1);
93         }
94         break;
95
96     case 3: // Pas d'évènement A depuis le dernier élément pendant un temps T
97
98         if (cond_temps_no_event) {
99
100             // contrainte validée, on passe à la contrainte suivante
101             int nb = chro.get_contraintes_restantes();
102             chro.set_contraintes_restantes(nb-1);
103
104         } else { // si l'évènement non voulu apparait : remise à zéro de la chronique
105
106             chro.set_contraintes_restantes(chro.get_contraintes_total());
107
108         }
109         break;
110
111     case 4: // A déclenché par une chronique
112
113         // parcours de toute la liste des chroniques
114         for(list<chronique>::iterator chr_ev=MES_CHRONIQUES.begin(); chr_ev!=MES_CHRONIQUES.end(); chr_ev++) {
115
116             // on regarde si la chronique dont dépend notre chronique
117             // a déjà été validée (chercher par nom et bool à 1)
118             string nom_contr = (list_contraintes[indice].get_event().get_label());
119             bool cond_chr_de_chr = ( nom_contr == ((*chr_ev).get_nom()) ) && ( (*chr_ev).get_event() );
120             if ( cond_chr_de_chr ) {
121
122                 // contrainte validée, on passe à la contrainte suivante
123                 int nb = chro.get_contraintes_restantes();
124                 chro.set_contraintes_restantes(nb-1);
125
126                 // validation de l'évènement associé à la chronique
127                 for(list<event>::iterator eve=MES_EVENTS.begin(); eve!=MES_EVENTS.end(); eve++) {
128
129                     // mise a jour de l'evenement dans la liste des events quand on a trouvé le bon
130                     event
131
132                     // dans la liste des events (recherche par nom)

```

```

131         bool condition = ((*eve).get_label()) == (chro.get_nom());
132         if ( condition ) {
133             (*eve).set_occured(true);
134             (*eve).set_h_event(HEURE_COURANTE);
135             (*eve).set_occurence((*eve).get_nb_occurence()+1);
136
137         }
138
139     }
140 }
141 }
142 break;
143
144 case 5: // A n'apparaît jamais
145
146     bool cond_never; // condition : l'événement n'a jamais eu lieu
147
148     // recherche de l'événement dans la liste des événements pour savoir s'il a eu lieu
149     for(list<event>::iterator ev=MES_EVENTS.begin();ev!=MES_EVENTS.end();ev++){
150         if ( ((*ev).get_label()) == list_contraintes[indice].get_event().get_label() ) {
151             cond_never = !((*ev).get_occured());
152         }
153     }
154
155     if (cond_never) {
156
157         // contrainte validée, on passe à la contrainte suivante
158         int nb = chro.get_contraintes_restantes();
159         chro.set_contraintes_restantes(nb-1);
160
161     }
162     break;
163
164
165 default:
166     cout<<"Erreur : type de contrainte inconnu"<<endl;
167
168 }
169
170 // toutes les contraintes ont été validées
171 if ((chro.get_contraintes_restantes()) == 0 ) {
172
173     chro.validation_chronique();
174
175     // gestions des chroniques de chroniques :
176     // remise à zero de event_verify (pour les chroniques de présence)
177     if (chro.get_nom()=="PresenceSDB_Personne") {
178         std::list<chronique>::iterator it;
179         for(it=MES_CHRONIQUES.begin();it!=MES_CHRONIQUES.end();it++){
180             if ((*it).get_nom()=="PresenceSalon_Personne"){
181                 (*it).set_event(false);
182             }
183             if ((*it).get_nom()=="PresenceChb_Personne"){
184                 (*it).set_event(false);
185             }
186         }
187     }
188     if (chro.get_nom()=="PresenceSalon_Personne") {
189         std::list<chronique>::iterator it;
190         for(it=MES_CHRONIQUES.begin();it!=MES_CHRONIQUES.end();it++){
191             if ((*it).get_nom()=="PresenceSDB_Personne"){
192                 (*it).set_event(false);
193             }
194             if ((*it).get_nom()=="PresenceChb_Personne"){
195                 (*it).set_event(false);
196             }

```

```

197     }
198 }
199 if (chro.get_nom()=="PresenceChb_Personne") {
200     std::list<chronique>::iterator it;
201     for(it=MES_CHRONIQUES.begin();it!=MES_CHRONIQUES.end();it++){
202         if (((*it).get_nom()=="PresenceSalon_Personne"){
203             (*it).set_event(false);
204         }
205         if (((*it).get_nom()=="PresenceSDB_Personne"){
206             (*it).set_event(false);
207         }
208     }
209 }
210
211 }
212
213 return chro;
214 }
215
216
217 /* MAIN */
218
219 int main(){
220
221     /** Initialisation **/
222
223     // recuperation des événements
224     MES_EVENTS = parser_evenements();
225
226     // recuperation des chroniques
227     MES_CHRONIQUES = parser_chroniques();
228
229     // recuperation de la sequence
230     MA_SEQUENCE = parser();
231
232
233     /** Algorithme de lecture de la sequence **/
234
235     for(vector<elmt_sequence>::iterator vec=MA_SEQUENCE.begin(); vec!=MA_SEQUENCE.end();vec++) {
236
237         // mise a jour de l'heure courante
238         HEURE_COURANTE = (*vec).get_date();
239
240         // mise a jour de la liste d'event
241         list<event>::iterator eve;
242         for(eve=MES_EVENTS.begin(); eve!=MES_EVENTS.end(); eve++) {
243
244             // mise a jour de l'evenement dans la liste des events quand on a trouvé le bon event
245             // dans la liste des events (recherche par nom)
246             bool condition = (((*eve).get_label()) == ((*vec).get_label()));
247             if ( condition ) {
248                 (*eve).set_occured(true);
249                 (*eve).set_h_event((*vec).get_date());
250                 (*eve).set_occurence((*eve).get_nb_occurence()+1);
251
252                 // mise a jour de la liste des chroniques
253                 for(list<chronique>::iterator chr=MES_CHRONIQUES.begin();chr!=MES_CHRONIQUES.end();chr++){
254
255                     (*chr)=check_validation((*chr),(*eve));
256
257                 }
258             }
259         }
260     }
261 }
262

```

```

263     }
264
265     /* Affichage de toutes les chroniques à chaque itération */
266     //interface();
267
268 }
269
270
271
272 /***** PROCEDURES DE TEST *****/
273
274
275 // TEST DE FONCTIONNEMENT : AFFICHAGE DE LA SEQUENCE LUE
276
277 /*
278 cout<<"La sequence : "<<endl;
279 afficheur_sequence(MA_SEQUENCE);
280
281 */
282
283
284
285 // TEST DE FONCTIONNEMENT : AFFICHAGE DES EVENEMENTS LUS
286
287 /*
288 cout<<"Les evenements : "<<endl;
289 afficheur_liste_evt(MES_EVENTS);
290
291 */
292
293
294 // TEST DE FONCTIONNEMENT : AFFICHAGE DES CHRONIQUES LUES
295
296 /*
297 cout<<"Les chroniques : "<<endl;
298 afficheur_liste_chr(MES_CHRONIQUES);
299 */
300
301
302
303 return 0;
304 }

```